

Ubiquitous Annotation Visualization – Concept and Rapid Prototyping Framework

Von der Fakultät für Mathematik, Informatik und
Naturwissenschaften der RWTH Aachen University zur Erlangung
des akademischen Grades eines Doktors der Naturwissenschaften
genehmigte Dissertation

vorgelegt von

Diplom-Geoinformatiker
Marc Jentsch

aus Bünde

Berichter: Prof. Dr. Matthias Jarke
Prof. Dr. Alberto Giretti

Tag der mündlichen Prüfung: 19. Februar 2015

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek verfügbar.

Abstract

Ubiquitous computing is about blurring the boundaries between virtuality and reality. Computers and virtual content are merged with the real world. For instance, ambient displays use the entire physical environment as interface to digital information or augmented reality systems show virtual content over real world objects. Gaining knowledge of commonalities among these technologies would reduce their development effort. This thesis investigates the ubiquitous computing domain regarding similarities in visualization methods. The identification and classification of the most important approaches reveal a recurring concept: Real world objects are visually augmented by virtual information. We call this novel umbrella concept *ubiquitous annotation visualization (ubiAV)*.

As advantage of knowing of commonalities among these visualization methods, application developers can choose from similar visualization options for their application. But they do not always know which approach fits best for a particular application. Hence, if they want to evaluate a visualization's effect on a particular application, they have to try different ubiquitous annotation visualization approaches. This can be time-consuming and cumbersome because the application might need to be adapted to each visualization. Moreover, a developer might lack the knowledge how to implement the visualization, and getting to know the technology takes time. As a consequence, she might refrain from comparing and instead choose a maybe suboptimal visualization arbitrarily, which could decrease the quality of the user experience.

This thesis specifies the UbiVis software framework, which aims at facilitating this kind of rapid prototyping. The framework's architecture separates the application logic and visualization into encapsulated components and defines a standard interface between them. This allows developers to exchange and evaluate visualizations without having to adapt the application logic. A standard procedure defines how developers can quickly and easily apply the framework. So, lengthy familiarization with the technology is not necessary. This thesis also specifies a technical environment so that UbiVis can be applied in practice.

UbiVis intends that visualization components are encapsulated as exchangeable libraries. If the libraries are freely available, developers can apply a visualization for their application without knowing implementation details. A major contribution of this thesis is the delivery of a set of libraries for supporting the most important ubiquitous annotation visualization approaches. We also specify a concept for extending this initial set of libraries. This specification allows every application developer to add further libraries, which opens the framework to support new ubiquitous annotation visualization technologies.

In order to validate whether UbiVis supports rapid prototyping of ubiquitous annotation visualization applications, we develop several examples in different domains. Then, we show how the provided libraries can be used for exchanging ubiquitous annotation visualization approaches without having to change application code. We further demonstrate the extensibility of the set of libraries by developing a new library within the rules of the framework and by applying it to the developed applications. Finally, we evaluate in a set of practical user workshops to which degree the framework is easy to apply.

Zusammenfassung

Ubiquitous Computing lässt die Grenze zwischen Virtualität und Realität verschwimmen. Computer und virtuelle Inhalte verschmelzen mit der echten Welt. Beispielsweise nutzen Ambient Displays die gesamte physische Umgebung als Schnittstelle zu digitalen Informationen oder Augmented Reality Systeme überlagern Objekte der realen Welt mit virtuellen Inhalten. Das Wissen über Gemeinsamkeiten dieser Technologien würde ihren Entwicklungsaufwand verringern. Diese Arbeit untersucht die Ubiquitous Computing Domäne auf Ähnlichkeiten in Bezug auf Visualisierungsmethoden. Die Identifizierung und Klassifizierung der wichtigsten Ansätze deckt ein häufig verwendetes Konzept auf: Physische Objekte werden mit virtuellen Informationen angereichert. Wir nennen dieses neuartige Sammelkonzept *Ubiquitous Annotation Visualization (UbiAV)*.

Ein Vorteil des Wissens über Gemeinsamkeiten dieser Visualisierungsmethoden ist, dass Anwendungsentwickler aus ähnlichen Visualisierungsoptionen für ihre Anwendung wählen können. Sie wissen aber nicht immer, welcher Ansatz sich für die jeweilige Anwendung am besten eignet. Daher müssen sie verschiedene UbiAV Ansätze ausprobieren, wenn sie deren Effekt auf die jeweilige Anwendung beurteilen möchten. Das kann umständlich und zeitaufwändig sein, sobald eine Anwendung für jede Visualisierung angepasst werden muss. Ferner fehlt den Entwicklern ggf. das nötige Fachwissen, um die Visualisierung zu implementieren und die Einarbeitung in die Technologie benötigt Zeit. Daher wählen sie eventuell per Zufall eine ggf. suboptimale Visualisierung aus statt einen Vergleich durchzuführen, was die Qualität der Nutzererfahrung reduzieren kann.

Die vorliegende Arbeit spezifiziert das UbiVis Software Framework, welches dieses Rapid Prototyping vereinfachen möchte. Die Architektur trennt die Anwendungslogik und die Visualisierung in gekapselte Komponenten und definiert eine Standardschnittstelle zwischen ihnen. Somit können Entwickler die Visualisierung austauschen, ohne die Anwendungslogik anpassen zu müssen. Eine Standardprozedur definiert, wie Entwickler schnell und einfach das Framework einsetzen können, ohne dass eine langwierige Einarbeitung notwendig wäre. Da die technische Umgebung des Frameworks ebenfalls in dieser Arbeit spezifiziert ist, kann es direkt praktisch angewandt werden.

UbiVis sieht Visualisierungskomponenten als gekapselte, austauschbare Bibliotheken vor. Sofern diese frei verfügbar sind, können Entwickler eine Visualisierung für ihre Anwendung einsetzen, ohne etwas über deren Implementierung zu wissen. Als wichtigen Beitrag dieser Arbeit stellen wir Bibliotheken bereit, welche die wichtigsten UbiAV Ansätze bedienen. Ferner spezifizieren wir ein Konzept des Ausbaus dieser initialen Bibliotheken. Es ermöglicht jedem Entwickler, dem Framework weitere Bibliotheken hinzuzufügen und öffnet es damit der Möglichkeit, neue Visualisierungstechnologien zu unterstützen.

Um zu validieren, ob UbiVis das Rapid Prototyping von UbiAV Anwendungen unterstützt, werden mehrere Beispiele in unterschiedlichen Domänen entwickelt. Wir zeigen dann, wie die bereitgestellten Bibliotheken für den Austausch von Visualisierungsansätzen genutzt werden können. Dabei muss der Anwendungscode nicht verändert werden. Des Weiteren demonstrieren wir die Erweiterbarkeit der initialen Bibliotheken. Dazu erstellen wir innerhalb der Regeln des Frameworks eine neue Bibliothek, die wir auf die Beispielprogramme anwenden. Abschließend ermitteln wir in mehreren praxisorientierten Nutzerseminaren, inwiefern das Framework einfach anzuwenden ist.

Acknowledgements

While writing this thesis, I received mental support and feedback of numerous people to whom I would like say thank you here.

I very much appreciate the comprehensive and constructive feedback of my advisor, Prof. Dr. Matthias Jarke. Besides being very busy in general, I know about a huge number of other theses being supervised by you simultaneously. Therefore, your detailed advices deserve my high esteem. Thank you.

I also thank Prof. Dr. Alberto Giretti for straightforwardly being my co-advisor. The good atmosphere in the project work with you positively influenced this thesis.

My gratitude goes to the head of my working group at Fraunhofer FIT, Dr. Markus Eisenhauer. Your integrity and fairness enabled me to combine my dissertation with our daily work. The discussions about earlier drafts of this document provided valuable input. In addition, without the encouragement of you and Dr. Andreas Zimmermann I would never have started to write my dissertation thesis. Thank you, Andreas, for structuring my unordered ideas and helping me to find my scope.

The whole User-Centered Ubiquitous Computing group has played an important role for this work. My colleagues provided precious feedback in doctoral colloquia or debates in the office. Several of my work results evolved in our mutual project efforts. Your participation made the evaluation of my framework successful. I would like to single out Dr. René Reiners and Jonathan Simon for several fruitful discussions and Ferry Pramudianto for the productive work environment in our shared office.

An important experience during which I could contribute to this thesis was my stay at the HITLab NZ of the University of Canterbury in Christchurch. Thank you, Prof. Dr. Mark Billingham, Dr. Leif Oppermann and again Dr. Markus Eisenhauer for making this possible.

I can always rely on my family, my wife Kathrin Wünnemann, my parents Karola and Friedrich-Wilhelm Jentsch and my sister Kathrin Jentsch. Thank you, for standing by me all my life. A special thanks goes to my daughter Merle Jentsch for giving me joy, probably without knowing it.

Last but not least, I express my deepest thanks to my beloved wife Kathrin Wünnemann. The constructive dinner discussions with you saved me from several dead ends I had come to. You never got tired of proof-reading. Finally, your encouragement and mental support were the foundation of this thesis.

Contents

Abstract	3
Zusammenfassung	5
Acknowledgements	7
List of Figures	15
List of Tables	19
List of Abbreviations	21
Conventions	25
1 Introduction	27
1.1 Visualization in Ubiquitous Computing	28
1.2 Problem Description	29
1.3 Thesis Statement	29
1.4 Body of Methods	30
1.5 Contribution	31
1.6 Thesis Structure	32
2 Definitions and State of the Art	35
2.1 Definitions	36
2.1.1 Ubiquitous Computing	36
2.1.2 Mixed Reality	37
2.1.3 Augmented Reality	38
2.1.4 Ambient Displays	39
2.1.5 Ubiquitous Annotation Visualization	40
2.1.6 Concepts' Interrelation	41
2.1.7 Rapid Prototyping	43
2.1.8 Software Framework	44
2.2 Related Development Support Tools	45

Contents

2.2.1	Visualization Support	45
2.2.2	Development Support Tools for Rapid Prototyping of Pervasive Applications	46
2.2.3	Development Support Tools Which Separate the Interaction Layer	47
2.2.4	Development Support Tools for Rapid Prototyping of Visual Output	49
2.2.5	Discussion	52
2.3	Conclusion	53
3	Classification of ubiAV Approaches	55
3.1	Ubiquitous Annotation Visualization Systems	55
3.1.1	Visualization Through Ambient Displays	56
3.1.2	Mobile Applications	57
3.1.3	Augmented Reality	58
3.2	Discussion	59
3.2.1	Annotated Objects	59
3.2.2	Annotations	60
3.2.3	Compositions	61
3.2.4	Visualization Capabilities	62
3.2.5	Location Relationship	62
3.3	Adapted Mixed Reality Classification	63
3.3.1	Classes of Ubiquitous Annotation Visualization	64
3.3.2	Challenges	65
3.3.3	Reproduction Fidelity and Extent of Presence Metaphor	66
3.4	Classifying Related Work	67
3.4.1	Location-Aware, Real World Systems	67
3.4.2	Location-Unaware, Real World Systems	67
3.4.3	Location-Aware, Virtual World Systems	69
3.4.4	Location-Unaware, Virtual World Systems	69
3.5	Conclusion	69
4	Conceptual UbiVis Framework	71
4.1	Design Decisions	72
4.2	Specification of the UbiVis Framework	74
4.2.1	The Architecture Guide by [Rozanski and Woods, 2011]	74
4.2.2	Functional View	76
4.2.3	Information View	79
4.2.4	Deployment View	83
4.3	Procedure of Rapid Prototyping	84
4.4	Procedure of Developing a new Library	85
4.5	Conclusion	87
5	Technical UbiVis Framework	89

5.1	Requirements	90
5.2	Middleware	91
5.2.1	Amigo	92
5.2.2	LinkSmart	92
5.2.3	GSN	93
5.2.4	Aspire	93
5.2.5	RTI Connext	94
5.2.6	aWESoME	95
5.2.7	Discussion	95
5.3	LinkSmart Details	96
5.4	Applying LinkSmart to UbiVis	97
5.4.1	Communication to Smartphones	99
5.5	Development View	103
5.6	Conclusion	106
6	UbiVis Libraries	109
6.1	UbiLens	109
6.1.1	Features	111
6.1.2	Configuration Options	112
6.2	UbiMap	114
6.2.1	Features	115
6.2.2	Configuration Options	116
6.3	UbiLight	117
6.3.1	Features	118
6.3.2	Configuration Options	118
6.4	UbiBoard	119
6.4.1	Features	120
6.4.2	Configuration Options	121
6.5	Conclusion	121
7	Validation	123
7.1	Microphone Application	124
7.1.1	Analyze Configuration of Visualizations	124
7.1.2	Analyze Data Space of Physical Objects and Annotations	124
7.1.3	Configure Visualization	126
7.1.4	Develop Application Logic Including Calls of VisualizationProxy	128
7.2	Conclusion	133
8	Application Examples	135
8.1	Energy Efficiency Application	135
8.1.1	Domain Description and Requirements	136
8.1.2	Concept	136
8.1.3	Implementation using UbiVis Framework and Libraries	136

Contents

8.2	Comparing Visualization Technologies of the EE Application	146
8.2.1	Setup	148
8.2.2	Results	149
8.3	eTriage Application	152
8.3.1	Domain Description and Requirements	152
8.3.2	Concept	153
8.3.3	Implementation using UbiVis Framework and Libraries	155
8.4	Comparing Visualization Technologies of the eTriage Application	165
8.4.1	Study Setup	165
8.4.2	Results	167
8.5	Conclusion	170
9	Evaluation	171
9.1	UbiTorch	172
9.1.1	Features	173
9.1.2	Applying to Energy Efficiency and eTriage	176
9.1.3	Summary	181
9.2	Quantitative Usability Assessment	181
9.2.1	System Usability Scale	181
9.2.2	Setup	182
9.2.3	Results	183
9.3	Pragmatic and Hedonic Quality	183
9.3.1	AttrakDiff	183
9.3.2	Setup	184
9.3.3	Results	185
9.4	Finding Attributes	188
9.4.1	User Experience Questionnaire	188
9.4.2	Setup	189
9.4.3	Results	189
9.5	Qualitative Usability Assessment Through Observation	193
9.5.1	Observation	193
9.5.2	Setup	193
9.5.3	Results	194
9.6	Qualitative Usability Assessment Through Discussion	194
9.6.1	Focus Group	195
9.6.2	Setup	195
9.6.3	Results	195
9.7	Conclusion	197
10	Conclusion	201
10.1	Contributions	201
10.1.1	Definition, Classification and Generalization of Ubiquitous Annotation Visualization Systems	201

10.1.2	Specification of a Conceptual and Technical Framework for Performing Rapid Prototyping of UbiAV Approaches	202
10.1.3	Provision of Visualization Libraries and Practical Application of the Framework	203
10.1.4	Validation and Evaluation of the Framework	204
10.2	Future Work	205
Own Publications		207
Bibliography		211
Appendix A EE Questionnaire		233
Appendix B Study - Presentation		235
Appendix C Study - Handout		245
C.1	What is UbiVis?	245
C.2	The Visualization Libraries	247
C.2.1	UbiLens	247
C.2.2	UbiMap	249
C.2.3	UbiLight	251
C.2.4	UbiBoard	252
C.3	How do I apply UbiVis?	254
C.4	Exemplary Walkthrough	255
C.4.1	Example Application	255
C.5	The Programming Task	257
C.6	Development Instructions	258
Appendix D System Usability Scale		261
Appendix E AttrakDiff		263
Appendix F User Experience Questionnaire		267

List of Figures

2.1	Simplified representation of a "virtuality continuum" [Milgram and Kishino, 1994]	37
2.2	Weather information is shown for the region the handheld device is currently pointing at [Fitzmaurice, 1993].	39
2.3	The augmented reality system NaviCam presents information about the currently viewed video tape [Rekimoto, 1995].	42
2.4	A virtual 3D conferencing space is augmented by video images of the participants [Kauff and Schreer, 2002].	42
2.5	The environmental display shows how the office looks like when then specified rule is active: It is raining and a weekday morning, so switch on the northern lamp [Beckmann and Dey, 2003].	47
2.6	(a) Physical model with AR marker (b) augmented as a game phone [Park et al., 2009]	48
2.7	The use of a digital camera is simulated by UbiWise in a virtual 3D environment [Barton and Vijayaraghavan, 2003].	50
2.8	A set of self-made output components for iStuff [Ballagas et al., 2003] . .	51
2.9	A mobile phone is augmented by sensor hardware [Ballagas et al., 2007]. .	52
3.1	Representation of the "Extent of World Knowledge (EWK)" dimension [Milgram and Kishino, 1994]	65
3.2	The Power-Aware Cord shows how much energy the connected lamp consumes [Gustafsson and Gyllenswärd, 2005].	66
4.1	The different views are related to each other and so, together describe the software structure (adapted from [Rozanski and Woods, 2011]).	77
4.2	The functional view defines components and their interactions.	78
4.3	The class diagram shows the static data structure of the ubiAV design space.	79
4.4	The adjusted class diagram shows the static data structure of UbiVis. . .	81
4.5	Information flow: Trigger visualization	82
4.6	Information flow: Configuration	82
4.7	The deployment view shows to which hardware nodes the software components are distributed.	83
5.1	Instantiation of UbiVis concept using LinkSmart components	98

List of Figures

5.2	The class diagram shows involved components for the communication between LinkSmart and smartphones.	100
5.3	Process flow: Start RSMB and connect	101
5.4	Process flow: Stop RSMB	102
5.5	Process flow: Connect smartphone to RSMB	102
5.6	Process flow: Disconnect smartphone from RSMB	103
5.7	Process flow: Connection to RSMB is lost	103
5.8	Process flow: Subscribe, publish, notify	104
6.1	UbiLens	110
6.2	Configuration options of UbiLens	113
6.3	UbiMap outdoor	114
6.4	UbiMap indoor	115
6.5	Configuration options of UbiMap	116
6.6	UbiLight	118
6.7	Configuration options of UbiLight	119
6.8	UbiBoard	120
6.9	Configuration options of UbiBoard	121
7.1	Overview of the combined configuration options of UbiMap and UbiBoard. The source library of each particular option is indicated by the colors.	125
7.2	Data structure of the sample application	129
7.3	Deployment of the sample application	130
7.4	Interaction of the components of the sample application	131
7.5	Message flow for the sample application	131
7.6	UbiMap visualizes the input volume of a microphone	133
7.7	UbiBoard visualizes the input volume of the microphones	133
8.1	A Plugwise Circle measures the current power consumption of the connected appliance [Plugwise B.V., 2014].	137
8.2	Overview of the combined configuration options of UbiLens, UbiMap, UbiLight and UbiBoard. The source library of each particular option is indicated by the colors.	139
8.3	Deployment of the energy efficiency application	143
8.4	Interaction of components of the energy efficiency application	144
8.5	Information flow inside the energy efficiency application	145
8.6	Visualizing a lamp's power consumption via UbiLens	146
8.7	Visualizing energy saving potential of devices in the room via UbiMap	147
8.8	Visualizing energy saving potential of devices in the room via UbiLight	147
8.9	Visualizing the power consumption of nearby devices via UbiBoard	148
8.10	A traditional paper triage tag is attached to a patient and contains the most important information about her [Metttag.com, 2014].	153
8.11	The electronic triage tag is attached to a colored bracelet, which closes automatically when snapped against an arm.	154

8.12	Overview of the combined configuration options of UbiLens, UbiMap and UbiBoard. The source library of each particular option is indicated by the colors.	156
8.13	Deployment of the eTriage application	161
8.14	Interaction of components of the eTriage application	162
8.15	Information flow inside the eTriage application	162
8.16	Visualizing location and vital values of a patient via UbiLens	163
8.17	Visualizing patient information on the emergency site via UbiMap	163
8.18	Visualizing patient information on the emergency site via UbiBoard	164
8.19	The study setup is visualized by UbiMap.	166
8.20	The study setup is visualized by UbiLens.	166
9.1	A projector reveals annotations when scanning physical objects by pointing at them.	173
9.2	Configuration options of UbiTorch	175
9.3	Visualizing an appliance’s power consumption via UbiTorch	180
9.4	Visualizing patient information via UbiTorch	180
9.5	The percentile ranks of all analyzed evaluations of [Sauro, 2011] associated with SUS scores and letter grades	184
9.6	Portfolio with average values of the dimensions PQ and HQ and the confidence rectangle of UbiVis	185
9.7	Mean values of the four AttrakDiff dimensions for UbiVis	186
9.8	Mean values of each word pair associated with UbiVis. The colors illustrate of which word pairs each dimension is calculated from.	187
9.9	UEQ scores of UbiVis are visualized by black diamonds. Blue lines indicate standard deviations. Green is a positive evaluation, yellow a neutral evaluation and red a negative evaluation.	190
9.10	UbiVis results in relation to benchmark	191
9.11	Mean values of word pairs associated with UbiVis. Colors indicate to which category a pair belongs as follows. Orange: Attractiveness. Green: Perspicuity. Purple: Novelty. Yellow: Stimulation. Dark Blue: Dependability. Light Blue: Efficiency. Grey: Own added pairs.	192
C.1	Component structure and interface methods	246
C.2	The class diagram shows the static data structure of UbiVis.	246
C.3	UbiLens	247
C.4	Configuration options of UbiLens	248
C.5	UbiMap indoor	249
C.6	Configuration options of UbiMap	250
C.7	UbiLight	251
C.8	Configuration options of UbiLight	252
C.9	UbiBoard	253
C.10	Configuration options of UbiBoard	253
C.11	Example instantiation of microphone application	257

List of Tables

3.1	Classification of ubiquitous annotation visualization technologies	68
5.1	Overview of which requirements are met by which middleware. The headlines refer to requirements as follows. HP - Heterogeneous Protocols. LC - Loose Coupling. DS - Distributed Structure. RCD - Resource Constrained Devices. St - Standards. Ma - Maturity. FA - Free Availability. .	95
6.1	The initial UbiVis libraries support each ubiquitous annotation visualization class.	110
8.1	Participants' answers. The first two columns show arithmetic mean and variance of the overall ratings of the system, 5 being best and 1 being worst. The last two columns show pros and cons for each visualization with the number of similar answers in brackets.	150

List of Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
API	Application Programming Interface
AR	Augmented Reality
AV	Augmented Virtuality
CO ₂	Carbon Dioxide
COM port	Communications port
DDS	Data-Distribution Service
DVD	Digital Video Disc or Digital Versatile Disc
EU	European Union
FAST	Features from Accelerated Segment Test
FP6	Framework Programme 6
FP7	Framework Programme 7
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human-Computer Interaction
hex	hexadecimal
HMD	Head-Mounted Display
HQ	Hedonic Quality
HTML	HyperText Markup Language
HUD	Head-up Display
ID	Identifier
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol

List of Abbreviations

J2ME	Java 2 Platform, Micro Edition
LCD	Liquid-Crystal Display
LED	Light-Emitting Diode
LGPL	GNU Lesser General Public License
MAC address	Media Access Control address
MQTT	Message Queuing Telemetry Transport
MR	Mixed Reality
NAT	Network Address Translation
OS	Operating System
OSGi	Open Services Gateway initiative
PC	Personal Computer
PDA	Personal Digital Assistant
PQ	Pragmatic Quality
RFID	Radio-Frequency Identification
RW	Real World
SAR	Spatially Augmented Reality
SIFT	Scale-Invariant Feature Transform
SoA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SUS	System Usability Scale
TCP	Transmission Control Protocol
TUI	Tangible User Interface
TV	Television
ubiAV	Ubiquitous Annotation Visualization
ubicomp	Ubiquitous Computing
UEQ	User Experience Questionnaire
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

List of Abbreviations

VPN	Virtual Private Network
VRD	Virtual Retinal Display
VW	Virtual World
WGS84	World Geodetic System 1984
WSDL	Web Services Description Language
XML	Extensible Markup Language

Conventions

To improve the readability of the thesis, we use the following conventions:

- Technical terms or jargon are set in *italics* when they appear for the first time.
- Referenced source code or implementation structures are set in a `monospace font`.
- URLs are set in a `sanf serif font`.
- Neutral persons are referred to by using the female pronoun (e.g., "she" instead of "she or he").
- References to other work can be used as subjects or objects. In these cases, the reference is not added to the end of the sentence again. This avoids redundancies in sentences such as: Jentsch et al. present work about ubiquitous computing [Jentsch et al., 2011a].
- If a referenced system does not have a name, the referenced work is used for it.

Chapter 1

Introduction

In his famous paper "The Computer for the 21st Century" [Weiser, 1991], Mark Weiser describes his vision of how computer systems will change in the nearer future. These computers are more integrated in the real world up to the extent that they cannot be recognized as computers anymore; the user perceives the environment as intelligent. Such a *ubiquitous computing* system consists of many distributed subsystems, which are connected in a network. It acts autonomously and context-aware. Intelligence, which in traditional computing systems is required from the user, is transferred to the ubiquitous computing system.

This article has given rise to the research field ubiquitous computing. Researchers explore one or several aspects of Weiser's vision in more detail and derive further research questions. Some existing research fields merge with ubiquitous computing. By displaying virtual content over images of the real world, *augmented reality* brings both worlds closer to each other [Caudell and Mizell, 1992]. *Wearable computing* tries to make computational devices wearable [Mann, 1997]. The exploration of Weiser's ideas also leads to new subsequent research fields. For example, *context-aware computing* tries to detect and react upon the user's context [Schilit et al., 1994]. *Ambient intelligence* is about computational intelligence, which is integrated into the user's environment [Ducatel et al., 2001]. The *Internet of Things* is a vision about real world objects being connected in an Internet-like network [Gershenfeld et al., 2004]. Early papers about these research areas define scopes and visions. The first decade of the new century then produces a huge number of scientific prototypes to demonstrate how particular ubiquitous computing concepts can be put into practice, such as [Hinckley et al., 2000, Dietz and Leigh, 2001, Patel et al., 2007, Consolvo et al., 2008]. Lately, [Abowd, 2012] claims that ubiquitous computing is not a niche research topic anymore, but "its ideas and challenges spread throughout most of computing thought today." Research has evolved so far that several concepts are available in commercial products.

Some aspects of the ubiquitous computing vision have been introduced in daily life and used for different domains. For example, a modern car consists of several interlinked sensors and computing units, which the driver does not perceive as computer systems but as part of the car. Using the parking assistance does not require interacting with a

computer system; users only drive their car as they would normally do. The car is aware of its location and the surroundings and manages this information without the need for explicit interaction of the driver.

The number of scientific and commercial implementations has increased the demand for tool support of ubiquitous computing application development. Software frameworks or libraries, which ease the implementation of several aspects of ubiquitous computing, have tried to satisfy the demand. Since the field of ubiquitous computing is manifold, the implementation support of some facets is not addressed yet. Also, the tools lack of usability for developers who are not familiar with all ubiquitous computing aspects [Abowd, 2012].

1.1 Visualization in Ubiquitous Computing

As stated before, one core aspect of the ubiquitous computing vision is implicit human-computer interaction. The concept of calm computing implies that computers disappear [Weiser and Brown, 1996]. Disappearance does not necessarily mean that computers are hidden. They might still be seen but are perceived as physical objects instead of computers. The idea is to make interacting with a computer system easier for the user. When interacting with a physical object, the user resorts to the comprehension of properties about the object that she learned her whole life. For example, [Fitzmaurice et al., 1995] use physical artifacts called *bricks*, which are coupled to virtual objects for virtual manipulation. In this case, users know about the physical behavior of a *brick* in time and space. The degree of explicit interaction with the computer system is reduced making human-computer interaction more seamless.

Calm computing implies a change of visual output. It is not restricted to a classic computer monitor anymore. Instead, it is diverse and integrated into the real world. Smartphones, tablet computers and large public displays are ubiquitous. Everyday objects which do not have any kind of displays are used as visualization device [Gustafsson and Gyllenswärd, 2005]. Also, visualization technologies are integrated in clothes (*wearable computing*) [Mann, 1997] or in the environment (*ambient displays*) [Wisneski et al., 1998]. For instance, LED lights are incorporated into buildings to highlight objects [Rogers et al., 2010]. Existing stationary technologies present virtual information about real world objects in their neighborhood [Jahn et al., 2010].

Besides the change of visualization devices, new visualization methods occur, which can be used in many divergent ways. For example, the classical *augmented reality* [Milgram et al., 1994] approach uses a head-mounted display as “information lens” [Fitzmaurice, 1993] through which the real world is examined while it is augmented by virtual overlays. A miniaturization of projectors allows for augmentation of real world objects by projecting digital information onto them [Raskar et al., 2004].

As foreseen in the ubiquitous computing vision, the mentioned visualization approaches help to connect physical objects more deeply to the computer system than in traditional computing. A recurring concept is that physical objects in the environment are associated with electronic information [Fitzmaurice, 1993].

1.2 Problem Description

In contrast to traditional computing systems, ubiquitous application developers can choose from a plethora of disparate technologies, methods and concepts for visualization. Deciding on which technique to use for a single application is difficult because a summarizing analysis of similarities among the approaches does not exist.

To make matters worse, the visualization approaches work differently well for each single use case. So, application developers face the problem that they do not know which approach suits their particular use case best. In order to evaluate their effect, developers need to test contrasting visualization modes for their application. This is a common technique for testing different interaction modalities, for which effects on the application are unknown [Ballagas et al., 2005, Greaves and Rukzio, 2008, Lorenz and Jentsch, 2010].

It can be a time consuming process for many reasons:

The application developer needs to implement the technical control of all visualization technologies she wants to compare. These technologies can be complex and the application developer might not have knowledge about all of them. Becoming acquainted to a number of unknown technologies is time consuming. If the developed technology turns out not to fit to the use case, it is discarded right after the evaluation. This is an imbalance of effort and benefit.

Besides the implementation of a visualization technology, an application developer might need to change the actual program logic for every visualization. Each augmentation approach might require a dissimilar control concept to which the application has to adjust. This can lead up to a complete reimplementaion of the whole application for every compared visualization approach.

Apart from increased implementation efforts, application developers might lack the knowledge how to compare different visualization approaches. The single process steps how to exchange visualization technologies are not obvious. A developer needs to analyze the activities first and thus loses time. Also, certain process steps might be missed or conducted in the wrong order. The developer might recognize this late and thus waste time.

In addition to expenditure, developers who want to compare different visualization approaches face a high level of complexity. The complexity results from a high number of possibilities how visualization in ubiquitous computing can be conducted. Each individual approach can require complex knowledge about a particular technology. Development can be more frustrating when one needs to become acquainted with a complex topic. Resulting applications might be error-prone.

1.3 Thesis Statement

This thesis states that a summarizing concept of visualization approaches in ubiquitous computing reduces the time effort and complexity of ubiquitous application development. As an advantage, such a concept can be supported by a conceptual and technical frame-

work for developers who want to rapidly prototype different visualization approaches for their application. From the technical point of view, the framework supports the developer during the implementation of program logic and visualization technology. If less code has to be written and less familiarization with new technology is needed, development time is reduced. Complexity is reduced if the application developer only has to make herself familiar with the framework and refrain from becoming acquainted with a large number of technical details about visualization approaches. Consequently the handling of the framework itself must be simple in order to benefit from hiding complexity.

From the conceptual point of view, the framework suggests a standard process, which application developers can follow instead of having to analyze the process steps by themselves. Additionally, the framework ensures that application logic does not have to be changed when exchanging a visualization approach. Both properties result in a reduced development time and lower complexity because the developer has to consider fewer issues.

1.4 Body of Methods

We analyze visualization approaches which associate physical objects and digital information since there is no summarizing concept of this aspect available. For this, we survey existing approaches for prioritizing and identifying the most important ones. Moreover, the survey helps to understand the processes of such techniques. Then, we classify the approaches in order to find similarities among them. As a result, we define the novel umbrella concept *ubiquitous annotation visualization (ubiAV)* as systems which visualize digital information that is meaningfully related to a physical object.

For supporting the development of ubiquitous annotation visualization systems, we create the new *UbiVis* software framework within the scope of this thesis. We specify a conceptual framework that allows rapidly generating ubiAV components and exchange them in a given application. Additionally, a workflow for the development process is conceptualized. By setting the technical and procedural rules for an application developer during the technical implementation, rapid prototyping is enabled. A second workflow for adding new visualization technologies ensures UbiVis's extensibility.

In order to make the framework practically applicable, we furthermore specify its technical environment. It is based on the results of a discussion of middleware solutions; the middleware takes care of networking issues. Part of the technical environment is a provision of library support for the most important annotation visualization methods according to the survey.

The validation and evaluation of UbiVis is based on the technical instantiation. For validating whether its requirements are met, we implement applications and exchange the visualizations according to the development workflow. We evaluate the framework through a collection of user studies, qualitative and quantitative assessments.

1.5 Contribution

The ubiAV concept and the UbiVis framework, which are developed within the scope of this thesis, provide support for ubiquitous application developers who want to evaluate a certain kind of visualization approaches for their application. The developers do not have to concentrate on the development of the actual visualization technology. Instead, a developer implements a particular ubiquitous computing application and applies a set of provided ubiquitous annotation visualization technologies to it.

For achieving this, the main contributions of this thesis are:

Definition, Classification and Generalization of Ubiquitous Annotation Visualization Systems

As a foundation for the specification of the framework, the term ubiquitous annotation visualization is defined and classified into a set of related concepts. The relevance of that novel point of view is corroborated by a survey of ubiquitous annotation visualization approaches. It leads to the identification and manageable classification of the most important approaches. Similarities among the approaches are generalized and common challenges are highlighted.

Specification of a Conceptual and Technical Framework for Performing Rapid Prototyping of Ubiquitous Annotation Visualization Approaches and Conceptualization of How to Extend the Framework

Being based on the generalization, the UbiVis framework's structural and procedural concept enables developers to quickly exchange output techniques so that visualization variants can then be easily compared to each other. It specifies concept- and technology-wise how the exchange can be performed without having to change application code. Additionally, a step-by-step tutorial is serving as a standard development process instruction, which allows developers to apply the framework.

All classes of ubiquitous annotation visualizations are supported by UbiVis. Each visualization is encapsulated in a library. Another step-by-step tutorial enables application developers to extend the framework with new libraries. This ensures that further visualization technologies can be integrated into the framework's rules. It makes UbiVis applicable for current and future visualization approaches.

The specification of UbiVis's technical environment allows for its practical applicability. Technologies for the important issues communication, networking, event handling and interfacing are integrated. Development rules and conventions ensure a robust implementation process.

Provision of Visualization Libraries and Practical Application of the Framework

An initial set of libraries allows application developers to start implementing different ubiquitous annotation visualization possibilities in a rapid prototyping procedure. When

developing their application within UbiVis's rules, they can configure the needed libraries for the utilization in their application. Hence, they do not have to deal with visualization issues. These initial libraries also act as templates for each ubiAV class. Two example applications in different domains corroborate the framework's broad applicability in practice.

Validation and Evaluation of the Framework

A validation shows that UbiVis fulfills its main purpose: Enabling rapid prototyping of ubiquitous annotation visualization applications. The defined standard process is suitable for this rapid prototyping. Likewise, the extension process suits the creation of new libraries.

A usability and user experience evaluation shows that the validated features make UbiVis preferable to rapid prototyping without framework. The framework is perceived as simple to use and learn, time-saving and facilitating. The general usability evaluation is positive.

1.6 Thesis Structure

The contributions are worked out as follows.

We start by defining the necessary terms for this work in Section 2.1. As second foundation, Section 2.2 investigates related work on tools that support developers to build ubiquitous computing applications. This highlights trends and gaps in current research.

In Section 3.1, we survey works that perform ubiquitous annotation visualization and present an ensuing discussion in Section 3.2. This corroborates the relevance of the ubiAV concept and identifies main trends. It allows us to structure the field by defining a classification in Section 3.3. At the same time, we identify common practices among the approaches. This helps us find requirements for the technical specification of a supporting framework. In Section 3.4, the surveyed systems are classified, substantiating our classification's validity.

The conceptual specification of the UbiVis framework follows in Chapter 4. In Section 4.1, essential decisions about the architecture design are taken based on the knowledge gathered in the previous chapters. After the actual specification in Section 4.2, in Section 4.3, we conceptualize a standard workflow for applying the framework. The conceptual specification is complemented in Section 4.4 by presenting a standard procedure for extending the framework with a new visualization library.

The specification of UbiVis is completed by the technical concept in Chapter 5. The technical framework uses a middleware for lower level issues such as networking and event management. Several middleware solutions are discussed in Section 5.2, based on requirements which are elaborated in Section 5.1. Details about the finally chosen LinkSmart middleware are introduced in Section 5.3. In Section 5.4, we show how

LinkSmart is used to apply the concepts of Chapter 4. The technical framework is finalized in Section 5.5 by discussing development rules.

The UbiVis libraries for the development of visualization technologies are presented in Chapter 6. Each of the identified main visualization approaches is supported by at least one library.

We continue in Chapter 7 by validating whether the UbiVis framework supports rapid prototyping of ubiquitous annotation visualization applications. For this, we develop an application using UbiVis and connect two of the initial libraries to it without changing the code.

In Chapter 8, we develop two more complex example applications. We connect the initial visualization technologies to the applications and compare them in user studies. This demonstrates UbiVis's practical applicability, lets us gain practical experience with the framework and substantiates the validation's findings.

In Chapter 9, we complete UbiVis's validation and evaluate its usability and user experience. In Section 9.1, we create an additional library and integrate it in the two sample applications in order to demonstrate UbiVis's extensibility. The remaining sections assess usability aspects. For quantitative usability assessment, we perform a programming task and subsequent standard questionnaires analysis in Section 9.2, Section 9.3 and Section 9.4. At the same time, some qualitative usability assessments are conducted by these questionnaires. It is intensified by an observation session in Section 9.5 and focus group discussion in Section 9.6.

We conclude and provide an outlook to future work in Chapter 10.

Chapter 2

Definitions and State of the Art

In classical desktop computing, visualization usually takes place at the computer screen. There are widely accepted interaction modes, such as the WIMP paradigm [Williams, 1984] and well-known representations, such as the desktop metaphor [Smith et al., 1982]. However, in ubiquitous computing systems, it is not predetermined how and where visualization takes place because of a plethora of possible paradigms, concepts and technologies. This chapter provides an understanding of visualization in ubiquitous computing by presenting the main approaches which have been discussed in detail in related literature. Meanwhile, we find a commonality among these known approaches that has not been carved out yet. So, in Section 2.1.5, we define a novel sub-scope inside the research area, which is the foundation of the discussions throughout the rest of this thesis. In Section 2.1.6, the interrelation of the existing and the novel concepts is highlighted. Since the terms introduced in this chapter are coining the research addressed in this thesis, later chapters will refer to these definitions.

A ubiquitous computing application developer can choose from several different visualization alternatives. Since there is no standard which alternative serves best for the particular use case, she often has to try out several visualization modes. *Rapid prototyping* can be used for this if the time for deciding on one final visualization alternative shall be decreased. Since the framework introduced in this thesis aims at supporting rapid prototyping for ubiquitous annotation visualization, we introduce the term rapid prototyping in Section 2.1.7 and apply it to the domain. The term will recur throughout the thesis, especially when we describe the design of the conceptual framework in Chapter 4 and its validation in Chapter 7.

Since the way of developing ubiquitous computing applications and their visualizations differs from classical desktop computing, frameworks and other support tools have been developed that aim at supporting ubiquitous computing application developers. We discuss the term framework and related definitions in Section 2.1.8 because they are not consistently defined and can be used in several contexts. This is a prerequisite for Section 2.2 where existing support tools for the development of ubiquitous computing applications with a focus on visualization are surveyed. The survey shows trends but also gaps in current research. We explain which functionality is currently not covered

by existing support tools. The UbiVis framework aims at closing this gap.

2.1 Definitions

In order to provide the necessary background knowledge about the topics investigated in this thesis, we review definitions about visualization approaches in ubiquitous computing. Additionally, we define the term ubiquitous annotation visualization for approaches which visualize digital information that is meaningfully related to a physical object because such an umbrella term does not yet exist. By highlighting the interrelation of this novel definition and the existing ones, we clarify the scope of ubiquitous annotation visualization and differentiate between it and the related concepts.

2.1.1 Ubiquitous Computing

In Weiser's initial vision [Weiser, 1991], he talks about the departure from using traditional computer terminals. Weiser presents three classes of devices, which he predicts to replace computer terminals. The device classes are associated to existing non-computational tools, which largely exist in work environments. They are differentiated most notably by their size and style of usage.

Weiser describes *tabs* as "inch-scale machines that approximate active Post-it notes". Users employ several of these devices and they can even be wearable. A single tab usually performs only one or a small amount of tasks and within a project a huge amount of them is used. Consequently, a tab's display is intended to carry information about one task which is represented by an icon in workstation computers.

Pads are characterized as "foot-scale ones that behave like a sheet of paper". They seem to have similarity to a laptop but are used quite different. Instead of being a personal device that has to be carried everywhere, pads are described as "analogous to scrap paper that can be grabbed and used anywhere".

Boards are "yard-scale displays that are equivalent of a blackboard". A board is typically integrated in a room and serves several people for collaboration purposes.

Weiser's first article on ubiquitous computing has given rise to a new research area and his initial vision has matured to a plethora of concepts and ideas. Now, the five core qualities of the ubiquitous computing vision according to [Poslad, 2009] are:

1. Distributed systems: Ubiquitous computing systems consist of several smaller distributed subsystems which are interlinked. The distribution is transparent for the user, thus appearing as a single system to the user.
2. Implicit human-computer interaction: Human interfaces of ubiquitous computing systems are hidden or integrated in the real world. So, users rather interact with the real world than with a computer system. In the optimal case, "they do not even realize to interact with a computerized system but the system understands it as input." [Schmidt, 2000]

- Real objects are any objects that have an actual objective existence
- Virtual objects are objects that exist in essence or effect, but not formally or actually

This definition includes that real objects can be observed directly or resynthesized on a display after being sampled. So, for example, a ball in a video recording is still a real object. On the contrary, for being able to view a virtual object, it has to be simulated.

In an iteration of this thesis, we replaced the term real object by the term physical object because we found out that readers understand this better. So, within the scope of this thesis, we use physical objects in the same sense as [Milgram and Kishino, 1994] define real objects. Hence, physical objects have a physical shape but also images or videos of them are still called physical objects.

2.1.3 Augmented Reality

[Milgram et al., 1994] based their definition of augmented reality on the use of specific technologies. AR systems started by mainly using *head-mounted displays (HMD)* [Sutherland, 1968] - displays that users wear on their head. HMDs can display 3-dimensional objects, which change according to the user's head movements. Later, HMDs were miniaturized to the size of glasses [Lukowicz et al., 2001]. *Virtual retinal displays VRD* create images by scanning low power laser light directly onto the retina [Viirre et al., 1998]. They are integrated in glasses and are thus a miniaturized HMD. *Head-up displays (HUD)* are usually integrated in means of transportation. For instance, this semi-transparent display can be located in the field of view of a pilot and show flight-related information [Dopping-Hepenstal, 1981].

A common way of implementing AR systems is the *information lens* technique [Fitzmaurice, 1993]. It makes use of handheld display widgets. To achieve a spatial matching, a widget displays the data when it is placed on top of the object (cf. Figure 2.2). A special kind of information lens is when the widget is semi-transparent. Virtual information and real-world objects look merged to the user. This is often referred to as *magic lens* or *see-through* technique [Bier et al., 1993].

[Milgram et al., 1994] distinguish between *optical see-through* and *video see-through*. Optical see-through systems use a (semi-)transparent displays device on which virtual objects are synthesized. Reality is directly observed. Video see-through systems capture and resynthesize reality in front of a user's eye. So, the user has the feeling of directly observing reality although it is resynthesized.

In contrast to see-through techniques, "in *spatially augmented reality (SAR)*, the user's physical environment is augmented with images that are integrated directly in the user's environment, not simply in their visual field. For example, the images might be projected onto real objects using digital light projectors, or embedded directly in the environment with flat panel displays." [Raskar et al., 1998]. Projector-based systems can either utilize fixed projectors [Raskar et al., 1999], head-worn projectors [Rolland et al., 1998] or handheld projectors [Beardsley et al., 2005].



Figure 2.2: Weather information is shown for the region the handheld device is currently pointing at [Fitzmaurice, 1993].

[Azuma, 1997] introduces a more generalized and commonly used definition. He defines augmented reality systems by these characteristics:

1. Combines real and virtual
2. Interactive in real time
3. Registered in 3D

This definition is not technology-specific. It excludes virtual overlays on live video when it is not combined with the real world in 3D. However, it allows video see-through systems.

2.1.4 Ambient Displays

Tabs, pads and boards still have similarity to workstation computer monitors. They mainly differ in size, mobility and style of usage. But [Weiser and Brown, 1996] also state that visualization in ubiquitous computing is not restricted to monitor-like devices. They describe the concept of calm computing as an as important quality of ubiquitous computing. Visualization of ubiquitous systems shall be taken away from the computer screen and integrated into the environment. Thus, the visualization stays unobtrusively in the periphery of a user's attention but is able to get in the center of attention as well.

The idea of integrating everyday things with computer systems is focus in the field of *tangible user interfaces (TUI)*, initiated by [Ishii and Ullmer, 1997]. In this context, [Wisneski et al., 1998] describe that "*Ambient Displays* take a broader view of display than the conventional GUI, making use of the entire physical environment as an interface to digital information." The definition is not limited to visual phenomena but also includes other sensory perception such as sound, smell or temperature.

Ambient displays can be designed in different approaches. The environmental space around the user, e.g. the room's walls, can be augmented. In contrast, single artifacts of the environment can be used as displays. [Wisneski et al., 1998] present two exemplary ambient displays. Each of them is representing a different approach. The ambientROOM is a mini-office installation. Human activity in a work area is measured and represented by a pattern of illuminated patches projected onto a wall. As an example of a standalone ambient display, they are the attempt to transport information through airflow. Pinwheels with electrical motors are mounted at the ceiling. Their rotation speed changes according to information flow. The Pinwheel flow can be mapped to patterns of solar wind as information source for an atmospheric scientist.

2.1.5 Ubiquitous Annotation Visualization

In Chapter 3.1, we will show that there are many works in the field of ubiquitous computing which combine digital information and physical objects in a particular way. The visualized information is meaningfully related to the physical object. Often, the digital data represents additional facts about the object which is presumed to be unknown to the user. Technology is used with the intension to extend the user's perception of the object.

Since there is no umbrella term for all systems which include this central characteristic, we will create and define the term *ubiquitous annotation visualization (ubiAV)* systems as follows.

A ubiquitous annotation visualization (ubiAV) system visualizes digital information that is meaningfully related to a physical object in the user's context.

"Physical object in the user's context" means that the user knows about the actual objective existence of the object. For instance, the user of Wikitude World Browser [Wikitude GmbH, 2014] sees the annotated object by see-through view and, hence, knows about its actual existence. As contrary example, we refer to a football which is advertised on Ebay¹. Ebay also visualizes digital information, such as price or brand, about this physical object. But although a bidder might know footballs, she usually does not know this concrete football. Hence, Ebay is not doing ubiquitous annotation visualization.

As the word "ubiquitous" suggests, using ubiquitous computing technology is mandatory for ubiAV systems. So, Ebay is also not classified as ubiquitous annotation visualization system because it is traditionally accessed via desktop computer.

Within this scope, we will call the digital information which is meaningfully related to a physical object an *annotation*. In traditional context, an annotation is "a note by way of explanation or comment added to a text or diagram" [Oxford University Press, 2014]. So, annotating a text aims at changing the reader's understanding of the text by providing additional facts about it. Analogously, physical objects are annotated by

¹<http://www.ebay.com>

digital information with the aim of changing a user's understanding of the object in the context of ubiquitous computing.

According to the transcriptivity theory, an annotation is a transcript [Jäger et al., 2008]. It makes facts perceivable for a viewer and thus augments the meaning of a viewed object. The information about the object already existed before it was visualized but the viewer probably did not know about it.

There are also systems which present the annotation in another than the visual modality. In this work, we only aim at visualization, which is also the most commonly used modality. Definitions for other modalities or a definition of *ubiquitous annotation representation systems* in general can be easily derived from the definition above.

2.1.6 Concepts' Interrelation

In the following, we will explain the interrelation of the presented concepts with ubiquitous annotation visualization.

Often, augmented reality is used for ubiquitous annotation visualization. A common use case for augmented reality systems is to overlay digital information about the currently viewed physical object (as example see Figure 2.3, taken from [Rekimoto, 1995]). But augmented reality and ubiquitous annotation visualization are no synonyms. For instance, many mobile museum guides like Kore [Bombara et al., 2003] display texts or images about a nearby exhibit on a handheld device. They are no AR systems according to Section 2.1.3. But they can be classified as ubiAV since the visualized digital information is meaningfully related to a physical object in the user's context. On the other hand, augmented reality can be used for presenting terrestrial navigation systems (e.g. [Thomas et al., 1998]). However, a navigation instruction is no annotation about a particular physical object and hence, the system is not ubiAV.

Similar to augmented reality, augmented virtuality can be used for ubiquitous annotation visualization but not every augmented virtuality system is a ubiAV system. In an image-guided neurosurgery use case, [Paul et al., 2005] integrate real objects - the operative field - into a virtual world, i.e. the 3D multimodal scene which includes preoperative images of the patient. In this setting, the preoperative images represent digital information that is meaningfully related to a physical object - the patient. Hence, this augmented virtuality system is performing ubiquitous annotation visualization. However, the VIRTUE system is not performing ubiquitous annotation visualization because no additional digital information about any real object is visualized [Kauff and Schreer, 2002]. This augmented virtuality system creates a virtual 3D conferencing space in which video images of participants are integrated (cf. Figure 2.4).

The concepts of ambient displays and ubiquitous annotation visualization systems also overlap but do not completely cover each other. One of the first ambient display installations, the Live Wire [Weiser and Brown, 1996], is doing ubiquitous annotation visualization. The moving speed of a whirling string, which is mounted in the ceiling, is indicating the amount of network traffic of a nearby Ethernet cable. Here, the visualized digital information is the amount of network traffic, which is meaningfully related to a



Figure 2.3: The augmented reality system NaviCam presents information about the currently viewed video tape [Rekimoto, 1995].

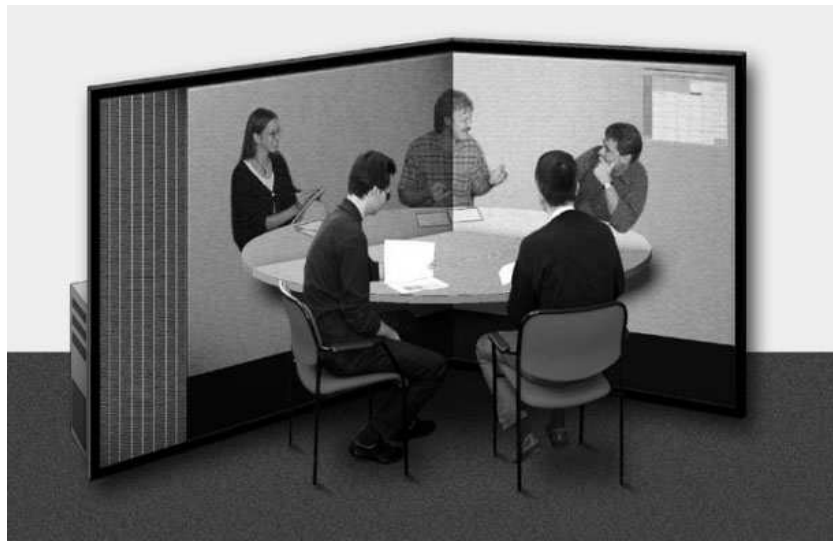


Figure 2.4: A virtual 3D conferencing space is augmented by video images of the participants [Kauff and Schreer, 2002].

physical object, the Ethernet cable. The installation would be interesting for network

administrators, so the Ethernet cable is in the context of the user's work. However, DynaWall is an example of an ambient display which is not used for ubiAV [Streitz et al., 1998]. DynaWall provides an interactive electronic wall in a project room. Project teams collaborate on large information structures. But the information is not related to any physical object.

As we explained in Section 2.1.5, a system is not classified as ubiAV if it is not using ubiquitous technology. So, ubiAV systems are always ubiquitous computing systems.

2.1.7 Rapid Prototyping

Prototypes are "an approximation of a product (or system) or its components in some form for a definite purpose in its implementation" [Chua et al., 2010]. For example, prototypes can be used for understanding a problem better, for testing and proving concepts or for communicating ideas. For achieving this, a prototype can be created as a first design of a part of a product. It is commonly understood that a prototype's benefits lead to a reduction of overall efforts, although developing it costs extra efforts during the creation of a product. Nevertheless, the aim is to keep the extra efforts as low as possible. Hence, tools have arisen for creating prototypes more rapidly.

Rapid prototyping exists in different domains such as fabrication, design or software development. In software development, "definitions of the term 'rapid prototyping' are often implicit and highly diverse, and there is often a great disparity between stated methodology and engineering practice." [Overmyer, 1991] The prototype's common goal is to project realism of a certain aspect of the software system. Often, this aspect is user oriented because the purpose of building prototypes is to test them on users for gaining feedback from them. As difference to the final product, prototypes miss or neglect other facets, such as safety or reliability [Dix et al., 2004].

According to [Dix et al., 2004], there are three main ways of prototyping:

- **Throw-away.** The prototype is built for gaining knowledge from testing it. Then, the prototype is discarded so that it will not be part of the final product.
- **Incremental.** There is an overall design which partitions the product into separate components. Each component is built as a prototype and the final product is released as merged prototypes.
- **Evolutionary.** Each prototype is not discarded but serves as basis for the next iteration. So, each prototype is an improvement of the previous iteration. The last prototype is the final product.

A category of prototypes which is of special interest for the human computer interaction domain is user interface prototypes [Bäumer et al., 1996]. The focus is on the user interface while the functionality is neglected. The purpose is to demonstrate how the system behaves from the user's point of view in order to investigate user reactions. This can be achieved by prototypes in different maturity levels. Presentations only show the snapshots of a graphical user interface. They are completely human controlled.

Functional prototypes demonstrate parts of the user interface including some automatic functionality. Pilots are close to a product and can be practically applied.

User interface prototyping is often applied in ubiquitous computing. In this domain, emphasis is put on usability and other HCI qualities (cf. Section 2.1.1). Hence, often user interfaces are tested through prototypes. Throw-away, incremental or evolutionary prototyping approaches can all be applied with user interface prototypes.

”The first major difficulty in evaluating a ubicomp system is simply having a reliable system to evaluate. The technology used to create ubicomp systems is often on the cutting edge and not well understood by developers” [Abowd and Mynatt, 2000]. Hence, developers have a great need for support of prototyping ubiquitous computing applications. A framework can take over the task of handling the technological parts.

The design process is about generating different solutions, elaborating them and deciding on the ones worth pursuing [Laseau, 2000]. So, a framework which supports rapid prototyping of ubiquitous computing applications allows for fast build and easy exchange of user interface prototypes. Applied to ubiquitous annotation visualization systems, the following requirements must be met:

1. When exchanging the visualization, the prototype’s application code must not change.
2. The visualization technology must not be developed by the user of the framework.

2.1.8 Software Framework

Frameworks exist in diverse domains and on different comprehension levels, e.g., legal frameworks, the Microsoft .NET framework or frameworks of thought. A common understanding of a *software framework* is a ”technique to reuse both design and code” [Johnson, 1997]. Design refers to software architectural concepts such as the definition of layers, structure, information flow and their interrelation. Beyond that, the control flow is often dictated by the software framework while framework users extend, implement and write specific functionality. Metaphors generalize circumstances to higher levels in order to make conceptual distinctions and organize ideas. For code reuse, software frameworks usually deliver concrete components, such as libraries, snippets or interfaces.

The design part of a software framework is often called *conceptual framework* [Dey et al., 2001] while the code aspect is referred to as *toolkit* or *technical framework* [Chou, 2003]. Unfortunately, the terms are not always consistently used. For instance, [Hong and Landay, 2001] perceive frameworks only from the design reuse point of view. On the contrary, [Holleis, 2008] calls his framework toolkit although it includes design decisions such as a component-based architecture or programming abstractions. Partially, even further terms overlap the understanding of frameworks, e.g., library, template, schema, infrastructure or metaphor [Brown, 1996, Hong and Landay, 2001, Truong and Abowd, 2004].

For the design of our own software framework UbiVis, concepts and practices of all these tools might be useful. In order to address them all, irrespective of potentially

inconsistent use of terms in related literature, we understand software frameworks in a broader sense and refer to the above described concepts as *development support tools*.

2.2 Related Development Support Tools

In this section, we present frameworks and toolkits which support application developers in building ubiquitous computing applications. We focus on works which aim at visualization support for rapid prototyping of ubiquitous computing applications or are related to the topic of this thesis in other respects. In this sense, we start with development tools that support visualization technologies which are commonly used for ubiAV. A presentation of development support tools which focuses on rapid prototyping of ubiquitous applications follows. A central feature of our framework is the separation of the visualization component (cf. Design Decision 1 in Section 4.1). So, development support tools which likewise separate the interaction layer are explored. Finally, we take a look on tools which aim at the same goal as we do: Support rapid prototyping of visual output for ubiquitous computing applications. Existing gaps in this field are highlighted and explain the motivation for creating our own framework.

2.2.1 Visualization Support

There are tools available which support the development of a single visualization technology. A few of them which support important ubiquitous annotation visualization concepts are presented in this section.

In general, desktop computer based rendering engines can be used for some ubiquitous computing technologies like *boards* as well. Rendering engines create images from models. The most commonly used ones are OpenGL [Shreiner et al., 2013] and Direct3D [Microsoft, 2014a]. Alternatively, web based engines help creating graphical output such as WebGL [Marrin, 2011], Flash [Adobe Systems, 2014] or Silverlight [Microsoft, 2014b].

An important task for ubiquitous computing applications is the identification of real world objects. A common way is the use of a visual marker which is attached to the object. These markers are usually 2D black and white patterns, which are captured by a camera and then decoded using image analysis. [Rohs and Gfeller, 2004] present a technique to identify and decode markers using a consumer mobile phone.

A common tool for creating AR applications is the ARToolkit [Lamp, 2014], initiated by [Kato and Billinghurst, 1999]. It creates see-through visualization, which is also based on marker tracking. The library takes over the calculation of the camera's viewpoint by tracking markers in the video frame. It then places computer-generated content over the marker.

The Mobile AR Framework [Billinghurst, 2014] is a successor of ARToolkit. It helps creating AR applications on smartphones with a focus on the Android platform. The framework consists of three major components. The Outdoor AR Library is an Android library that provides support for the fundamental AR technologies. Developers can create stand-alone Android AR applications, focus on the actual application development

and leave fundamental technology handling to the library. For instance, the library grants access to all needed sensors. It displays the camera's video image and creates graphical overlays. For this, it manages the graphical objects including their positions and graphical processing. Additionally, for each screen touch event, the library checks which graphical object was touched. The management of mobile AR content is also provided by the second component, Outdoor AR Server. It is a server to which mobile AR applications can connect in order to download that content. The third component, the Computer Vision based Tracking Library, provides tracking support based on computer vision for mobile AR applications. Being robust and effective, it is optimized for running on mobile platforms.

2.2.2 Development Support Tools for Rapid Prototyping of Pervasive Applications

As we discussed in Section 2.1.7, the ubiquitous computing domain has great need for testing different alternatives of an application. Thus, several tools support rapid prototyping for ubicomp so that alternatives can be built faster.

A framework for the domain of context-aware applications is based on a separation of concerns approach [Dey et al., 2001]. It provides context widgets similar to GUI widgets. A widget hides the complexity of used technology for contextual input, which often consists of implicit merged input from a plethora of sensors. In addition, a widget abstracts context information so that every application is only notified about relevant context. Finally, widgets provide a usable and customizable building block of context. Developers rapidly prototype applications by configuring and attaching widgets to each other. However, changing a widget often requires to also change the rest of the application - a circumstance that our own framework avoids.

SiteView creates a simulated environment in which the user can check out a prototype of a rule-based pervasive application [Beckmann and Dey, 2003]. A SiteView setup can be controlled in two ways (cf. Figure 2.5). General conditions can be set by placing RFID tags (interactors) on sensors (condition composers). Location-based rules are specified by placing the interactor on a floor plan, under which another RFID reader is located (world-in-miniature). The resulting rules are displayed on a laptop (rules display). A large screen visualizes how the environment looks like when the rule is active (environmental display). Conducting user evaluations would be complicated because of the prototypes' abstract simulation level.

Topiary supports rapid prototyping of location-based pervasive applications [Li et al., 2004]. Its tools allow painting GUI windows and specifying rules when which window has to appear. A developer can stick together test application flows in a storyboard tool. The framework exports to J2ME 1.1 compatible code, thus the GUI can be tried on a range of PDAs and phones. However, these window GUIs are widely similar to desktop ones and are not typical pervasive application visualizations from a modern perspective.

VisualRDK is a graphical high level SDK [Weis et al., 2007]. It aims at rapid prototyping for pervasive applications. The user creates program logic in a drag&drop GUI. This source can generate a prototype and a debug version. The debug version is running

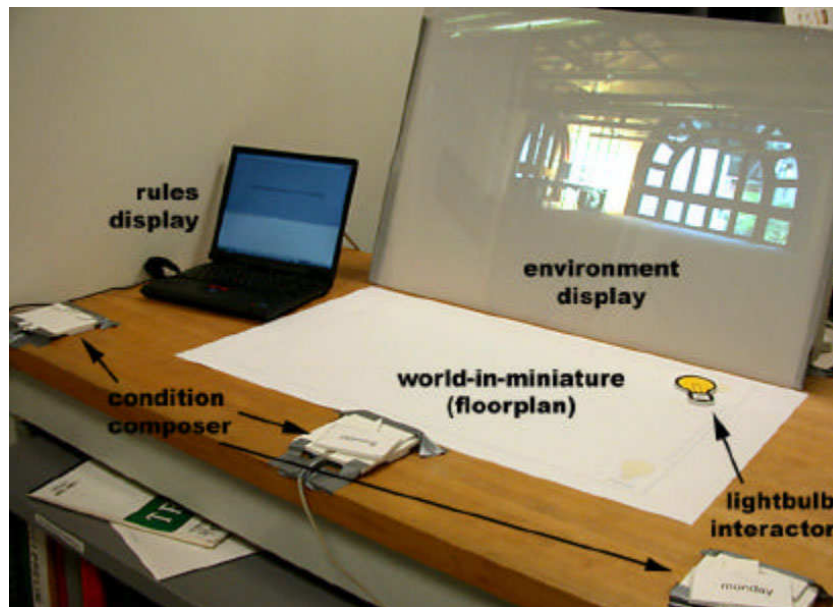


Figure 2.5: The environmental display shows how the office looks like when then specified rule is active: It is raining and a weekday morning, so switch on the northern lamp [Beckmann and Dey, 2003].

on a centralized PC and remote controlling other devices. The prototype version is truly distributed and therefore close to a product regarding architecture. VisualRDK provides a small set of features and supported hardware. This set cannot be extended by users. The framework’s focus is on application logic and interaction of devices. Visual output is only touched upon.

Another technique for rapid prototyping of a new physical device’s design is presented by [Park et al., 2009]. A physical model is augmented in an AR environment in order to simulate tangible interfaces (cf. Figure 2.6). Real functionality of a device cannot be tested using this approach.

[Pramudianto et al., 2013a] allow developers to virtually model real world objects and to link them to implementation technologies. Their tool generates Java code for a given group of technologies. Developers can then extend the code stubs by interfacing with the virtual objects. This enables them to build full applications without specific technology knowledge. Visual output is not focused; the tool is limited to textual output to consoles or databases.

2.2.3 Development Support Tools Which Separate the Interaction Layer

A main concept of our work is that output components are separated from the application logic (cf. Design Decision 1 in Section 4.1). This is a common approach for software

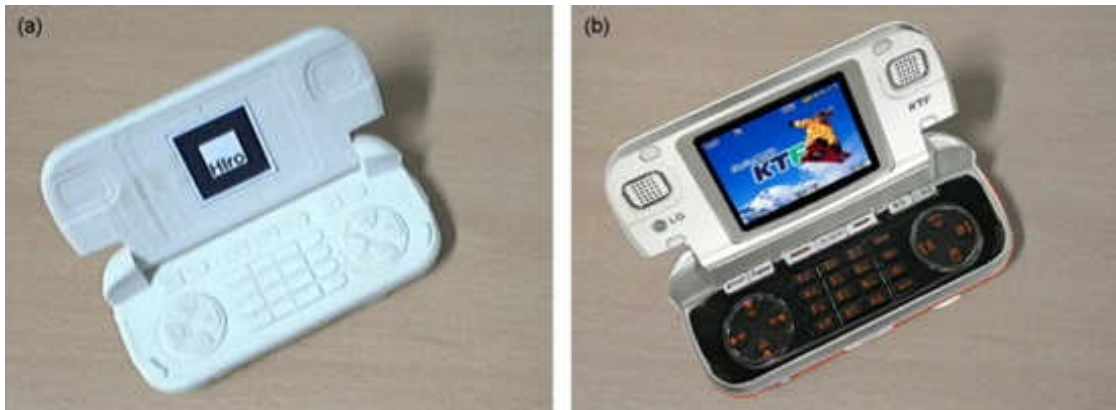


Figure 2.6: (a) Physical model with AR marker (b) augmented as a game phone [Park et al., 2009]

systems and mostly implemented using the model-view-controller architecture pattern [Reenskaug et al., 1996]. [Ballagas, 2007] and [Lorenz, 2009] make use of this approach in order to create a framework for supporting rapid prototyping of ubiquitous computing applications. Both these focus on input technologies. [Ballagas, 2007] provides an architectural framework, which allows for exchanging self-made interaction devices for a given application. The interfaces subscribe to events. They are able to adapt themselves to general commands because the commands have been mapped to device-specific instructions. The framework of [Lorenz, 2009] facilitates the control of ambient services by different commercial input devices. It "extends the idea of separating the user interface from the application logic by defining virtual or logical input devices physically separated from the services to control".

[Rukzio, 2006] takes a look at physical mobile interactions. Those are interactions of mobile computing devices with physical objects. The PMIF framework of [Rukzio, 2006] supports the rapid creation of physical mobile interaction applications. However, it does not include a method for changing interaction techniques in a given application.

Papier-Mâché is a toolkit for the development of tangible user interfaces [Klemmer et al., 2004]. Events from RFID sensors, image recognition systems or barcode scanners are abstracted to objects. This way, a physical event that is activated by a TUI can be associated to a digital content or action. The association is set using a simple mapping mechanism. So, this toolkit deals with the concept of associating physical objects and digital information and it makes use of an event-based architecture. However, it supports input devices and cannot be applied to output related frameworks which support rapid prototyping. In Papier-Mâché, the association between digital object and physical information can be simply mapped. For example, a scan of a certain RFID tag which is attached to a pen triggers the playback of an audio file. The scan of the same pen using image recognition delivers much more information about the pen which is not being used for the mapping. Creating a common interface for output devices is more difficult

since the broader number of possibilities is on the output side. While heterogeneity of information on the input side may just be ignored and investigation may be restricted to the atomic property of an ID, this heterogeneity must be handled by a framework which aims at supporting rapid prototyping for ubiquitous annotation visualization output.

2.2.4 Development Support Tools for Rapid Prototyping of Visual Output

The frameworks MOCA [Roman et al., 2000] and Aura [Sousa and Garlan, 2002] support rapid prototyping of visual output for desktop based visualization devices. In the MOCA architecture, programming logic and representation are again separated [Roman et al., 2000]. A service advertises itself via XML document. A UI is then generated technology dependently, downloading the XML before. For instance, the abstract XML description can be distilled into an HTML page or it can be represented by a Java application.

The framework Aura implements an architecture which defines user tasks as first class entities that are described as abstract services [Sousa and Garlan, 2002]. Service suppliers also have an abstract description. Aura now tries to find service suppliers for the requested services of the task when resources change, e.g., because the user is moving to a different environment. The architecture is not restricted to a particular resource so that it is also applied for visual output.

The Real World Interfaces library facilitates the usage of physical objects for visual output [McCrickard et al., 2003]. It provides an API for hiding complicated details of lower level programming languages. In addition, it handles hardware issues, such as error detection or interface blocking. A physical object is virtually represented through a proxy, acting as interface to the device for the programmer. The RWI library is limited to $X10^2$ devices. Rapid exchange of visualizations is not possible due to the lack of a common interface for all device proxies.

The aim of UbiWise is similar to the one of this thesis: Support the exploration of user interfaces and interactions of devices in ubiquitous computing applications [Barton and Vijayaraghavan, 2003]. For this, UbiWise provides a virtual 3D environment for which ubiquitous computing devices can be modeled (cf. Figure 2.7). The environment itself can be designed with the framework and interaction rules for the devices can be defined. Hence, UbiWise lets the user integrate new technologies but not in the real world.

The INCA infrastructure allows for capturing, storing, transducing and again accessing of media data [Truong and Abowd, 2004]. Its Accessor module provides a standardized interface to the data which every output component can request. However, INCA is initially restricted to media data. Output technologies themselves are not provided by the toolkit and must be developed from scratch.

d.tools allows for rapid prototyping of physical UIs with a focus on system design [Hartmann et al., 2006]. It provides a number of input devices that can be combined and also supports the creation of output components that incorporate a small LCD display.

²[http://en.wikipedia.org/wiki/X10_\(industry_standard\)](http://en.wikipedia.org/wiki/X10_(industry_standard))



Figure 2.7: The use of a digital camera is simulated by UbiWise in a virtual 3D environment [Barton and Vijayaraghavan, 2003].

Designers place physical controllers, sensors and output devices on form prototypes. The corresponding behavior is authored in a software workbench. d.tools does not support the actual development process and is also not open for a general set of devices.

[Mori et al., 2004] support more steps of the application design process. At first, tasks need to be modeled. From these, system tasks are derived. Then, UI components for fulfilling the system tasks are identified for every device class. The outcome of every step is an XML file from which concrete UIs are automatically generated.

The EIToolkit supports rapid prototyping of pervasive applications by providing an architecture for connecting various input and output technologies [Holleis, 2008]. Each technology is connected by a particular stub, which acts as driver that translates from technology-specific code to EIToolkit messages. Although not specialized on exchanging visual output for a given application, the framework can be used to connect different output technologies to an application, as long as stubs which understand the same messages exist.

Several frameworks apply the model-view-controller pattern [Reenskaug et al., 1996] to generate device-specific visual interfaces [Roman and Campbell, 2002, Berti et al., 2004, Stirbu, 2010]. Developers can create abstract models of the views with the help of the frameworks' toolkits. The frameworks translate the models into concrete implementations that are adapted for the particular devices. The frameworks do not support ubiquitous annotation visualization as they do not consider the connection of virtual and real world. In addition, the supported devices of the earlier frameworks are outdated.

2.2. Related Development Support Tools

Reverse engineering tools like [Bandelloni et al., 2008] try to analyze an existing user interface for a given platform and derive a general task model. This can again be used to adapt the user interface to other platforms. The general idea behind this procedure is to find an abstract representation for a user interface and to derive concrete implementations. Our UbiVis framework will make use of this idea.

The work which conforms most to our work is done in the scope of the interactive workspaces project [Johanson et al., 2002]. It features the iRoom, a workspace environment which consists of several interconnected ubiquitous computing appliances like smartboards, remote interactable lights or handheld devices. ICrafter is the framework for dynamically generating UIs for these appliances [Ponnekanti et al., 2001]. The Interface Manager, a central instance, is delivering the UI based on the requesting appliance. A template for each appliance is generating code (e.g. HTML), which is rendered on the appliance.

iStuff is a toolkit supporting physical input and output components (cf. Figure 2.8) [Ballagas et al., 2003]. An application in iStuff generates an event which is put on an event heap. Events must be mapped to devices, which can be done with the help of a graphical tool. The devices generate output and input based on this mapping and the events they receive from the event heap. iStuff also introduces several supported gadgets, most of them self-built. An extended mobile version of iStuff [Ballagas et al., 2007] is used for fast prototyping of existing mobile phones which are augmented by external sensors (cf. Figure 2.9). The mapping of application events to phone+sensor events is done via background application, which is provided by iStuff mobile.

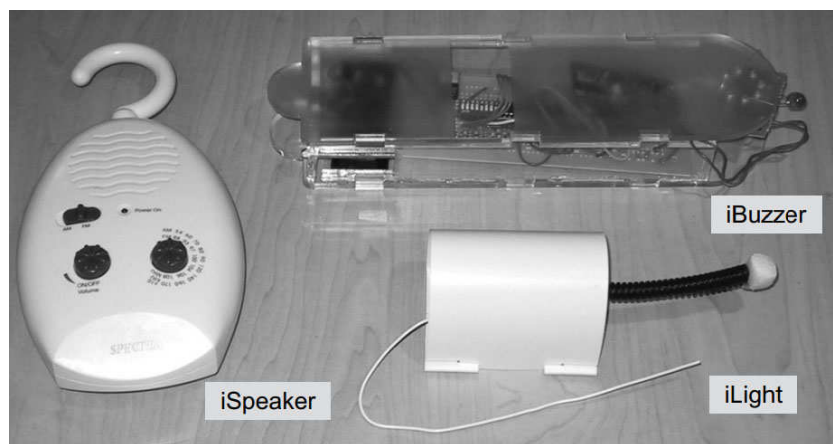


Figure 2.8: A set of self-made output components for iStuff [Ballagas et al., 2003]

The work of [Greenberg and Fitchett, 2001] is similar to the interactive workspaces project. They present *phidgets*, which is a short term for *physical widgets*. Like a widget, which hides implementation details for the programmer, phidgets package physical input and output devices. [Greenberg and Fitchett, 2001] let physical designers create the hardware device and make it available to software developers via an event-based API.



Figure 2.9: A mobile phone is augmented by sensor hardware [Ballagas et al., 2007].

2.2.5 Discussion

The visualization tools in Section 2.2.1 are commonly used in the scope of other frameworks for the actual visualization task. Accordingly, the Mobile AR Framework [Billinghurst, 2014] will be used for creating the UbiLens visualization in the scope of our framework (cf. Section 6.1).

Rapid prototyping support for pervasive applications is often provided by choosing from a set of pre-defined functionality (cf. Section 2.2.2). So, the assistance of these frameworks is restricted to this limited functionality. However, our aim is to leave the developer the complete freedom to implement everything possible in a standard programming language.

In general, most of the presented development support tools cannot be used for rapid prototyping of ubiAV as they address a different domain. Though, the survey gives us insight about successful concepts for the design of a framework, such as the separation of concerns or the ability for developers to extend framework code.

Separating the input layer from the rest of the application is a promising approach in several development support tools (cf. Section 2.2.3). But as the example of Papier-Mâché shows, the concepts cannot be exactly transferred to separating the output layer [Klemmer et al., 2004].

The tools presented in Section 2.2.4 support the rapid exchange of visualization interfaces for ubiquitous computing applications. However, most of them do not support ubiquitous post-desktop visualization devices. Additionally, the tools only provide a restricted set of functionalities. So, the scope of applications that can be implemented with these tools is limited. Often, the tools are restricted to a certain aspect so that the developed prototypes miss important factors for user evaluations. In this sense, [Park et al., 2009] simulate haptic user experience. UbiWise simulates interaction experience but lacks of haptics [Barton and Vijayaraghavan, 2003]. Development support is missing for d.tools [Hartmann et al., 2006]. EIToolkit lacks of a defined interface, which makes the visualization exchange highly dependent on the existence of stubs that understand

the same commands [Holleis, 2008].

Phidgets and iStuff support the rapid exchange of post-desktop interface devices for an application [Greenberg and Fitchett, 2001, Ballagas et al., 2003]. But they still do not take the integration of virtual information in real world objects into consideration, which is an important aspect of the ubiquitous computing vision. The aim of this work is to develop a framework that supports rapid exchange of modern output devices, which merge the virtual world into the real world.

iStuff and Phidgets support devices which combine input and output for the system in the same way as widgets do for graphical user interfaces. However, our work aims at creating support for output devices which are not connected to the input in order to make the framework more flexible.

At the same time, while iStuff focuses on the creation of new interface gadgets, UbiVis is trying to establish a framework which already supports the main ubiquitous output devices. A main contribution is the delivery of an initial set of libraries which support the most important ubiquitous annotation visualization approaches. Ideally, a library for a new output technology has to be written only once and is then available to all developers who are using the framework. These libraries can be enhanced by everyone so that after some time there will be a huge set of libraries from which a developer can choose. This makes rapid prototyping faster than in the presented environments where the output technology still has to be developed.

iStuff only helps to develop an interaction way which is specific for a single combination of application and device. UbiVis, on the other hand, provides a general output style which is applicable for a broad range of applications.

The majority of the presented development support tools was developed in the early 2000s. This first wave of frameworks already provided good support for the technology at that time. The possibility of ubiquitous annotation visualization by a plethora of technologies has become mature in the late 2000s, e.g., autofocused miniaturized projectors [MicroVision Inc., 2014], projectors that are integrated into smartphones [Greaves et al., 2008], augmented reality on modern smartphones [Schmalstieg and Wagner, 2009], or cheap miniaturized microcontrollers [Arduino, 2014, Raspberry Pi Foundation, 2014]. This is why the need for a framework for supporting rapid prototyping of ubiquitous annotation visualization has not been addressed by other frameworks.

2.3 Conclusion

The aim of this section was twofold. It provided an overview of visualization approaches in ubiquitous computing and of development support tools in this scope.

The discussion of ubiquitous computing and related research areas serves as foundation for the further investigations. Its outcome defines the scope of this thesis and aligns our work with existing research. Ubiquitous computing deals with the departure from traditional computer terminals. This leads to new challenges in the user interface design for developers. For instance, often, there is no classical display anymore, so other ways

of visualization must be found. As another example, interaction in ubicomp is meant to be implicit and the user may even not realize that she interacts with a computerized system. This challenges the application developer to still make sure that the user perceives information output from the system. New best practices must emerge, but they might again be dependent on the particular use case.

Two common approaches in ubicomp are augmented reality and ambient displays. They represent standard practices for the problem how to deal with the deeper integration of virtual and real world in ubicomp systems. Still, both comprise several possibilities for implementation. Thus, a deeper analysis of existing approaches is needed.

The definition of ubiquitous annotation visualization determines which approaches have to be considered in this deeper analysis, which will follow in Chapter 3. The definition creates a novel point of view on the ubicomp domain, which drives the investigations throughout the rest of the thesis. The interrelation with the existing concepts shows that we cannot simply sum up all AR and ambient display systems to ubiAV because none of the concepts is completely covered by the other.

We investigated how developers are already supported in these visualization aspects for ubiquitous computing. Support for rapid prototyping of visualization approaches is of special interest because this allows application developers to efficiently try out and assess different visualization alternatives for their system. The application of the term rapid prototyping to ubiquitous annotation visualization drove our survey.

The terms framework, toolkit and related definitions are not consistently used. However, a lot of these development support tools imply useful concepts for supporting rapid prototyping of ubiAV applications, such as user interface separation and event-based communication. Separating the user interface layer from the application logic facilitates the UI exchange. It can be applied by specifying an abstract representation of the user interface that is implemented by concrete interfaces. An event-based communication between application logic and user interface implies the definition of a software interface and a mechanism how to exchange UI components. These concepts can be adopted for designing our own framework.

However, full support for rapid prototyping of ubiAV is not given by any existing development support tool. They limit the freedom of standard programming languages by pre-defined sets of functionality. Often, development support tools are restricted to certain aspects of the prototypes. Most of them do not support solely the output layer. The tools do not support ubiquitous annotation visualization in particular or even neglect ubiquitous visualization devices at all.

Since there is no appropriate support for rapid prototyping of ubiquitous annotation visualization approaches, we need to build our own framework. In order to know which requirements it must meet, we survey ubiAV approaches in related literature in the next chapter.

Chapter 3

Classification of Ubiquitous Annotation Visualization Approaches

This chapter starts with surveying ubiquitous annotation visualization systems in Section 3.1. The broad availability of the concept in related works corroborates the relevance of this novel point of view on ubicomp systems. At the same time, the survey and the ensuing discussion in Section 3.2 conveys a sense of ubiAV's application. Finally, analyzing the state of the art identifies concepts which have to be supported by the UbiVis framework.

Since our framework cannot support every single surveyed ubiAV method in a separate way, we need to find an abstraction layer which groups similar approaches. This leads to a manageable number of framework requirements. So in Section 3.3, we define a classification to which ubiAV approaches can be categorized. For this, we adapt the related mixed reality classification of [Milgram and Kishino, 1994] to ubiquitous annotation visualization. The classification process improves the understanding of the field. In addition, concentrating on finding similarities among the approaches helps to structure the knowledge about the domain. At the same time, we identify a set of common challenges.

The approaches are classified in Section 3.4 in order to substantiate the applicability of the classification to the presented systems. In addition, this exemplification deepens the understanding of classes and ubiAV approaches.

3.1 Ubiquitous Annotation Visualization Systems

The visualization of ubiquitous computing applications differs in many factors from the visualization of workstation computers. Ubiquitous computing is characterized by three device classes tabs, pads and boards (see Section 2.1.1). They have become more and more sophisticated in commercial applications. Today, researchers usually use smart-

phones as tab-class instances for ubiquitous computing applications. They provide the computational power and display resolution to visualize text, images and video in high quality. At the same time, current smartphones offer important other features for being interesting as ubiquitous computing device, like networking capabilities, sensing and extensibility.

Tablet computers are usually used in current research as implementation of pad-class devices. They provide the same technical features as smartphones but at a larger size. The similarity between smartphones and tablet computers has become clear since the main operating systems, Android and iOS, are available for both classes.

High resolution projectors or large-scale LED screens are common instances of board-class devices in work environments today. In contrast to smartphones and tablet computers, not all installations are interactable in the sense of ubiquitous computing. Many installations just replace a normal workstation monitor and are controlled via mouse and keyboard. Therefore, often pads or tabs are used for interacting with the board, however, this combination also brings new interaction problems [Müller et al., 2008].

3.1.1 Visualization Through Ambient Displays

As first example of an ambient display which conducts ubiquitous annotation visualization, [Weiser and Brown, 1996] present the installation of the Dangling String. It is a long plastic string hanging from a small electric motor mounted in the ceiling. The moving speed of the motor is connected to the amount of network traffic of a nearby Ethernet cable. Thus, the string is indicating network usage through whirling and noise.

The first two examples of Ishii's group are Waterlamp and Pinwheels [Dahley et al., 1998]. The Waterlamp is a lamp which is shining from below through a water surface. On the water, ripples can be created through actuators. The results are patterns projected onto the ceiling. Pinwheels is the attempt to transport annotations through airflow. Pinwheels with electrical motors are mounted at the ceiling. Their rotation speed changes according to information flow. In both cases, the technologies were first constructed and afterwards, the researchers investigated possible use cases. The use cases were matched to the metaphors. For example, heartbeats of a significant other were associated to the Waterlamp. More flow-like events were used for Pinwheels. Patterns of solar wind mapped into patterns of Pinwheels might be interesting for an atmospheric scientist.

Newer research tries to design this kind of visualization technology more unobtrusive and investigate how it influences human behavior. [Rogers et al., 2010] install two groups of colored spheres in an atrium. Based on the usage of elevator and usage of stairs, the groups are moved up or down. After the installation, more people took the stairs instead of taking the lift than before the installation. Further examples of ambient displays which are used for ubiquitous annotation visualization are presented by [Holstius et al., 2004, Pettersson, 2004, Messeter and Molenaar, 2012, Müller et al., 2012, Šimbelis et al., 2014].

Another approach is to integrate monitor-like visualization areas into the environment. This can be a projector, which displays activity information about a coffee room

3.1. Ubiquitous Annotation Visualization Systems

to an office wall, like the NESSIE system does [Prinz, 1999]. [McCarthy et al., 2001] create the term *peripheral displays* for displays which are used for “content that is not directly related to one’s primary activities”. They present the Outcast system, a flat-panel monitor embedded in a cubicle wall. It displays annotations about a nearby office worker which addresses her colleagues, such as calendar data or an infrared badge based location. As another example, the Hermes Door Displays are located at office doors [Cheverst et al., 2003]. Office owners can leave messages for visitors when they are out of office. Further examples of using peripheral displays for visualizing data about nearby physical objects are described by [Greenberg and Rounding, 2001, Russell et al., 2002, Bartram et al., 2010, Lee and Dey, 2014]

The works which have been presented so far introduce everyday things into an environment where those things would usually not be found. A different approach is to extend the visualization capabilities of things in their natural environment. For example, [Mynatt et al., 2001] populate a digital picture frame with icons that indicate the status of a remote relative. The Power-Aware Cord is a power cord which is glowing according to how much electricity is consumed [Gustafsson and Gyllenswård, 2005]. This approach makes the integration of technology into the real world even more seamless.

The systems do not necessarily have to use light emitting devices. For example, the CoDine system prints text messages on toast using eatable materials such as chocolate cream or peanut cream [Wei et al., 2011]. The food depositing component is installed at a dining table and transmits information about a remote relative in order to create the feeling of dining together. Further examples of enhanced everyday things which are used for ubiAV in their natural environment are described by [Arroyo et al., 2005, Paulos and Jenkins, 2005, Kalnikaite et al., 2011].

3.1.2 Mobile Applications

For sure, the first prototypes of Weiser’s new device classes were not used to visualize digital annotations of physical objects. One of the first systems that introduces this method is Cyberguide [Abowd et al., 1997]. Cyberguide is a tourist guide on a PDA. It uses icons on a map to reference real world objects and it describes the visited locations. Using a digital map for representing the real world in ubiquitous computing applications is a straight forward approach because the basic concept is well known to the user from traditional maps. There are a lot of other projects which make use of that approach, e.g., [Burigat et al., 2005, De Carolis et al., 2009].

Another approach to establish a connection to physical objects is to present pictures of them on the digital device. The interactive museum guide HIPPIE shows thumbnail images of the exhibition objects which are currently explained [Oppermann and Specht, 2000]. HIPPIE tries to localize the user and presents information about the exhibition object the user is looking at.

Another common way to connect physical objects to digital information is mobile tagging [Holmquist, 2006]. Barcodes, which have digital information encoded, are attached to a physical object. They can be scanned with a smartphone’s camera and

decoded by an app. This lets, e.g., the physical timetable at a bus stop be augmented with real-time data about the next bus arrival. Other examples of mobile tagging are described by [Lange et al., 1998, Kanev et al., 2007, Balestrini et al., 2014].

Maps and photos or icons are the most common ways to link digital information on a mobile screen to the physical object to which this annotation belongs. There are other, less often, used connections like naming or indexing. *XP-Iris* [Antenna International, 2013] makes use of a common museum guide's approach. Index notes are attached to the exhibits. A user enters the index to a portable device in order to receive information about the exhibit.

Mobile devices do not necessarily have to be hand-held. An example is the MICA system, which assists warehouse workers when picking goods [Prause et al., 2010]. A tablet PC is mounted to a hand pallet truck, which is moved by a worker through the warehouse. As soon as an article is put to the trolley, it is recognized by RFID and the article information is displayed on the tablet screen. The work process implies that the visualization device is moving with the user although it is not hand-held.

Smartwatches are wrist-worn small displays including a processor and some sensors such as acceleration sensor or gyroscope. They shall replace traditional watches as they offer more dynamic and interactive visualizations. [Bieber et al., 2013] display the heart rate on the smartwatch when the user holds it to her chest. The acceleration sensor is sensible enough to recognize heartbeat patterns.

3.1.3 Augmented Reality

A straight forward way of matching digital information to physical objects is to present the virtual information spatially close to the real-world object. In the optimal case, virtual information and real-world object merge. Augmented reality is often used for implementing this approach.

The first system which employs AR to visualize annotations about a real world object is presented by [Bajura et al., 1992]. They use a small video camera in front of a conventional HMD for visualizing live ultrasound echography data on a pregnant woman. [Viirre et al., 1998] describe the same use case for their virtual retinal display (VRD). [Paul et al., 2005] integrates the operative field into a preoperative image view. Further projects using head-worn devices for ubiquitous annotation visualization are [Liarokapis, 2005, Kerr et al., 2011].

In parallel to HMD-based approaches, AR was implemented for handier visualization devices. [Rekimoto, 1995] presents the first AR system which uses a palmtop-sized video see-through device. In an example application, the name and date of video material is displayed when watching the video tape (cf. Figure 2.3). The Invisible Train prototype makes use of a commercially available PDA [Wagner et al., 2005]. Using video see-through, a virtual train is displayed over a real wooden miniature railroad track. Today, see-through augmented reality systems are common for smartphones and commercial versions such as Wikitude World Browser [Wikitude GmbH, 2014] or Layar

[Layar, 2014] are available.

An alternative to see-through systems is using a projector to project digital information directly on physical objects. The Everywhere Displays use a rotating mirror to steer the light from a projector mounted on a room's ceiling. It is applied for creating different surfaces on objects in the room [Pinhanez, 2001]. The system is explored for several use cases like advertisement, interactive information, gaming or navigation support. The Fluid Beam system extends the approach by replacing the projector + mirror unit with a steerable projector [Spasova, 2004]. While the configuration of Everywhere Displays only allows a projection range of a cone, Fluid Beam enables projection in almost every direction. Further projects using fixed projectors for ubiquitous annotation visualization are described by [Underkoffler, 1997, Butz and Krüger, 2006, Xiao et al., 2013].

When smaller projectors became available, the first systems aiming at mobile projectors appeared. The advantage of most of those systems is that they are not restricted to one instrumented environment. iLamps is the first investigation of mobile projectors mainly focusing on creating distortion-free projections [Raskar et al., 2003]. Its successor RFIG Lamps shows a use case where objects are augmented by mobile projectors in a warehouse scenario [Raskar et al., 2004]. Food products that are close to expiry date are highlighted. Sixth Sense is equipping a wearable camera and projector to augment any real world object [Mistry and Maes, 2009]. For example, a newspaper article can be augmented by a video. Further projects using mobile projectors for ubiquitous annotation visualization are [Cao and Balakrishnan, 2006, Cowan et al., 2010, Song et al., 2010, Lee et al., 2011, Molyneaux et al., 2012]

A different approach is to use a fixed projector and move objects to the projection field. For example, the Bonfire system can project a graph of cups of coffee consumed when a coffee cup is moved to the projection area [Kane et al., 2009]. FACT projects annotations on a paper document when the document is moved to the projection area [Liao et al., 2010]. Further projects using fixed projectors for augmenting objects which are moved to their projection field are [Molyneaux and Gellersen, 2009, Molyneaux et al., 2012].

3.2 Discussion

In this section, we regard the presented ubiquitous annotation visualization systems from different points of view. This highlights common concepts and problems and improves the understanding of the field.

3.2.1 Annotated Objects

UbiAV systems annotate real world objects. These objects correspond to the description of real world entities by [Kindberg et al., 2002], i.e., they can be subdivided into people, places and things. People are other persons who the user wants to interact with or about whom she wants to get information. Places are locations with boundaries such as

a house or a room. The boundaries can be invisible and fuzzy, e.g., the boundaries of a bus stop. Things are all other physical world objects like appliances, light switches or paintings.

These instances of real world objects are important for end users. The users are potentially interested in getting information about these objects. The instances interrelate. Places contain people and things and sometimes also other places. People can carry and use things.

Ubiquitous annotation visualization systems follow [Kindberg et al., 2002]’s classification and annotate people [Bajura et al., 1992], places [Holmquist, 2006] or things [Antenna International, 2013]. The definition of things is broad. It may incorporate streamed objects such as water [Arroyo et al., 2005]. Or, they are invisible so that the user must know about their existence [Weiser and Brown, 1996].

Since places can contain other real world objects, it might be unclear if the place itself or, e.g., a person in the place is being annotated. For instance, the Clouds project visualizes how many people are using an elevator [Rogers et al., 2010]. In this case, the annotated object is a place - the elevator. People themselves are not annotated; their number is just the information used for the annotation. Often, this differentiation is not important for the user. For example, Cyberguide annotates pubs with ratings [Abowd et al., 1997]. A pub is a place whereas the ratings might also refer to people (friendliness of waiters) or things (tastiness of beer) inside the pub. However, the user will get the idea how the rating annotation is linked to the pub.

The same uncertainty might occur if people are annotated who carry things. If an annotation is projected onto a person who carries a cup (e.g. using Sixth Sense [Mistry and Maes, 2009]), the user must find out whether the person or the cup is annotated. However, in most cases the annotation should clearly fit better to one of the two alternatives.

There are other cases where uncertainty is clarified through content. An annotated object can also symbolize another physical object. For example, the digital family portrait [Mynatt et al., 2001] actually annotates a photo but the photo represents a person. The annotation reports about physical health and relationships status and thus can only be about the person.

Summarized, ubiquitous annotation visualization systems annotate a broad range of objects. Often, annotated objects interrelate but the content usually clarifies to which object a particular annotation belongs.

3.2.2 Annotations

Most of the presented ubiAV systems are used to reveal invisible information about a physical object to the user. These are often sensor measurable values like heartbeats [Dahley et al., 1998] or energy consumption [Gustafsson and Gyllenswärd, 2005]. Sometimes, the measurements are aggregated to higher level values, such as health or activity [Mynatt et al., 2001]. Contrary examples of annotations which cannot be measured with a sensor are messages [Wei et al., 2011] or background information [Oppermann and Specht, 2000].

Rarely, annotations represent values which the user could find out by watching the physical object without the ubiAV system. These annotations are usually processed information. This aims at making accessing the information easier or more comfortable for the user. As an example, the Clouds project shows the percentage of people who preferred the elevator over the stairs [Rogers et al., 2010]. Clouds' users could theoretically watch and count, but the Clouds system aggregates this information and allows accessing it without preconditions.

UbiAV systems need to analyze which data type the annotation represents. Not every ubiAV technology is capable of visualizing every data type (cf. Section 3.2.4). The presented ubiAV systems mainly use numeric [Weiser and Brown, 1996], textual [Rekimoto, 1995] or image [Viirre et al., 1998] annotations. Other types of information are usually composed of these three main data types, e.g., calendar data [McCarthy et al., 2001] or cumulative data about a warehouse article [Prause et al., 2010]. Occasionally, video [Wagner et al., 2005, Mistry and Maes, 2009] or boolean [Raskar et al., 2004] annotations are used.

3.2.3 Compositions

If a place consists of other smaller places, it might be a problem for the user to identify whether the overall place or a sub-place is annotated (cf. Section 3.2.1). Generalized, this problem occurs if an annotated object is composed of sub-objects. Natural persons consist of sub-parts if we consider arms, legs etc. as such. More clearly, things can be made up of sub-things, similar to places. For example, RFIG Lamps annotates articles in a rack [Raskar et al., 2004]. So, potentially, the annotation could be about the articles or about the whole rack. But similar to the discussion in Section 3.2.1, this uncertainty is again clarified through the annotation content. The visualization of expiration dates only makes sense for the individual articles and not for the whole rack.

An annotated object can be atomic but represent several physical objects. For instance, the power-aware cord itself can be clearly identified as atomic object [Gustafsson and Gyllenswärd, 2005]. However, the annotation is about one or more connected appliances. The setup makes it clear to the user to which objects the annotation belongs.

As described in Section 3.2.2, annotations are often composed of several smaller units. This can mean that the transmitted information is visualized in several ways or that it consists of several smaller information pieces. For instance, Bonfire shows the number of consumed cups of coffee as numbers and as graph [Kane et al., 2009]. An annotation about an exhibition object in HIPPIE can, e.g., consist of information about the object's material, its creator and the creator's intention [Oppermann and Specht, 2000]. The information pieces and visualization ways are bunched together in the annotation.

Systems which transmit further graphical elements besides the annotations must ensure that a clear boundary of the annotation exists. This way, the user can clearly identify the annotation. For example, the public display system Outcast presents the annotation in a frame that separates the annotation from the navigational and design

elements [McCarthy et al., 2001].

3.2.4 Visualization Capabilities

In Section 3.2.2, the main data types of annotations are identified. Now, we check whether the used ubiAV technologies are capable of visualizing them.

Many ubiAV systems are able to present the same graphical elements as a computer screen. Projector-based systems or public displays are examples. Smartphones or smartwatches are smaller instances with the same graphical computing power. Thus, these visualization devices are capable of presenting numbers, texts, images and videos.

Especially the ambient display systems often do not use computer screen-like devices. These systems can be restricted in their visualization capabilities. For instance, CoDine cannot present moving objects such as video [Wei et al., 2011]. The dangling string is only capable of transmitting information through different dangling speed [Weiser and Brown, 1996]. This way, it is possible to transmit numeric data. Although users do not comprehend exact numbers, they perceive rough classes of speed. Also, they are able to compare dangling speeds at different points in time.

The visualization capabilities of computer screen-like devices match the identified data types of annotations (cf. Section 3.2.2). But problems occur if those annotations shall be visualized with the less powerful technologies. For instance, transmitting text through the dangling string in a way that is understandable for users is puzzling.

Sometimes, also the physical objects are represented by the ubiAV system. The visualization capabilities of the system also influence how this representation can be conducted. For example, the option of symbolizing the real world object through an image or icon (e.g. [Abowd et al., 1997]) is not available for systems that cannot visualize images.

3.2.5 Location Relationship

Often, the physical object, the user and the visualization device have a spatial connection with one another. It is used for establishing a logical link between the real world object and the annotation, which is visualized on the device. In this setup, the visualization device acts as connecting element between the real world object and the annotation. [Rukzio, 2006] calls such an element a pervasive *mediator*.

The ubiAV systems in our survey use the following ways to establish that link:

- Physical object and mediator are physically connected to each other [Gustafsson and Gyllenswärd, 2005].
- The user brings the physical object to the mediator [Prause et al., 2010].
- The user brings the mediator to the physical object [Bieber et al., 2013].
- The user moves the mediator into her line of sight on the physical object [Rekimoto, 1995]. For this case and also in general, it is useful to mark physical objects, e.g.,

3.3. Adapted Mixed Reality Classification

through barcodes [Holmquist, 2006]. This helps the user to distinguish between annotated and non-annotated physical objects.

- The user points the mediator onto the physical object [Mistry and Maes, 2009].
- A user action triggers the mediator to establish the link [Spassova, 2004].
- The current locations of the user and all annotated objects are presented on the mediator [Abowd et al., 1997]. The user has to match this information to the real world.

Finally, there are systems which do not establish a spatial link between the physical object and the annotation [Antenna International, 2013].

Generalized, we distinguish systems as follows:

- The user explicitly moves the physical object [Liao et al., 2010] or the mediator [Wagner et al., 2005].
- The user indirectly moves the mediator [Pinhanez, 2001]. Although none of the surveyed systems move physical objects upon user actions, this approach is also imaginable.
- The mediator and the physical object are connected [Gustafsson and Gyllenswärd, 2005].

Which approach is used is heavily dependent on whether the user can or is allowed to move the mediator and physical objects. The following setups can occur:

- The physical object may be moved; the mediator may not [Kane et al., 2009].
- The mediator may be moved; the physical object may not [Oppermann and Specht, 2000]. Often, the mediator is closely connected to the mobility of the user [Viirre et al., 1998].
- Both, physical object and mediator may be moved [Mistry and Maes, 2009].
- None of them may be moved. This setup would exclude most of the presented approaches.

3.3 Adapted Mixed Reality Classification

As the discussion in Section 3.2 shows, there are many points of view from which the field of ubiquitous annotation visualization systems can be classified. In this section, we develop the classification which we consider as basis for the UbiVis framework. Two dimensions with two options each are identified, which leads to a room of four different classes. The UbiVis libraries will support each class with at least one technology.

3.3.1 Classes of Ubiquitous Annotation Visualization

The classes are adapted from the classification of mixed reality by [Milgram and Kishino, 1994] (cf. Section 2.1.2). Mixed reality covers ubiquitous annotation visualization. Therefore, [Milgram and Kishino, 1994]’s taxonomy is also valid for ubiAV. We update it to the current state of the art according to the knowledge gained in the analysis in Section 3.1.

Virtuality Continuum

[Milgram and Kishino, 1994]’s first dimension for classification is the virtuality continuum (cf. Figure 2.1). A mixed reality system is classified according to the extent of real or virtual environment. [Milgram and Kishino, 1994] describe this dimension as continuum because they expect ”as technology progresses, it may eventually become less straightforward to perceive whether the primary world being experienced is in fact predominantly ’real’ or predominantly ’virtual’”. However, for all systems presented in the previous chapter, the predominant environment can be clearly determined.

We harmonize the dimension for the application of ubiAV to the two discrete classes *real world* and *virtual world*. They correspond to what is labeled as augmented reality and augmented virtuality in Figure 2.1 but use a different terminology in order to avoid conflicts with other perceptions of augmented reality. Classifying an instance to one of two discrete classes is often easier than classifying it on a continuous scale. When we match an instance to one of a set of classes, we only have to decide to which class it fits best. For the virtuality continuum, however, we must also specify where exactly on the continuum a system must be classified. Since no metric is provided, this can be difficult to determine. We can only classify an instance compared to another one.

So, for ubiAV, we only have two discrete virtuality classes. A ubiquitous annotation visualization system is classified as real world (RW)-based if the primary world being experienced is predominantly real. It is classified as virtual world (VW)-based if the primary world being experienced is predominantly virtual. [Milgram and Kishino, 1994] are mostly referencing virtual 3D spaces when talking about virtual worlds. However, our understanding of virtual worlds is less focused on 3D spaces. As the presented systems show, a common approach is to model the world and use maps or window-like GUIs as virtual world.

Extent of World Knowledge

[Milgram and Kishino, 1994] call their second dimension *extent of world knowledge*. It refers to how much knowledge about the world is modeled. For example, geometry, location data or contextual attributes of real world objects might be modeled in a system.

The original dimension is again specified as continuum. We adapt it to ubiquitous annotation visualization systems analogously to the virtuality continuum by harmonizing it to two discrete classes. There are again subcases highlighted, labeled ”Where” and ”What” (cf. Figure 3.1). ”Where” refers to cases where data about locations is available

3.3. Adapted Mixed Reality Classification

to the system. "What" refers to cases where the system has knowledge about objects. By definition, an annotation represents knowledge about an object, so all ubiAV systems belong to the "What" subcase.

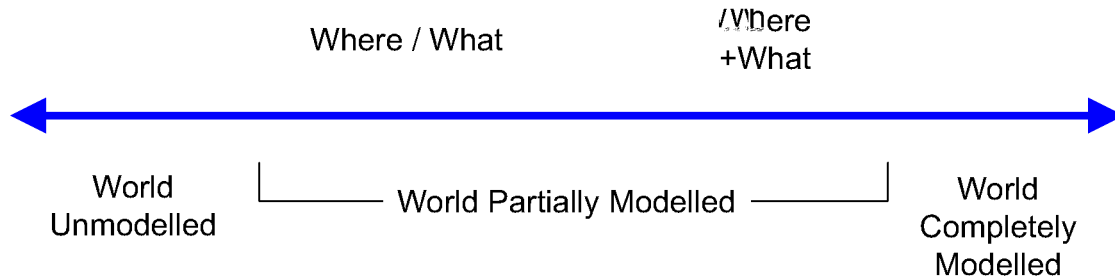


Figure 3.1: Representation of the "Extent of World Knowledge (EWK)" dimension [Milgram and Kishino, 1994]

So, ubiquitous annotation visualization systems can still be distinguished by their knowledge about objects' locations. A ubiquitous annotation visualization system is classified as *location-aware* if the annotated object's location is modeled. A ubiquitous annotation visualization system is classified as *location-unaware* if the annotated object's location is not modeled or the model is not used for annotation visualization.

3.3.2 Challenges

Real world ubiquitous annotation visualization systems face different challenges than virtual world systems. A major challenge for RW ubiAV systems is to visualize digital information in the real world so that the user understands to which object the information belongs. For VW ubiAV systems, an important issue is to represent the real object in the virtual world in a way the user can identify the real object. The matching of information and object in the virtual world tends to be easier than in RW systems because there are no physical constraints how the visualization can be arranged. To summarize, for both worlds, an important challenge is to integrate the element from the other world.

Location-aware ubiAV systems usually have a powerful possibility to express to which physical object a particular piece of information belongs. Ubiquitous visual augmentation systems need to link digital information to a physical object in a way that is understandable for the user. The straight forward link is location (cf. Section 3.2.5). If the digital information is visualized close to a physical object from the user's viewpoint, she should understand which information belongs to which object. This also includes approaches where the digital information is visualized on top of the physical object. The straight forward approach to establish the proximity of information and object is to perform the visualization in the three-dimensional space next to or on top of the object. But it is also possible to place the visualization in the line of sight of the user and object. So, the user perceives proximity of information and object although the proximity might

not be given from other perspectives.

Location-unaware ubiAV systems rarely have this powerful possibility. If the location of the visualization is not used for establishing the link, the connection needs to be on a logical level. For instance, a physical object can be represented by a digital photo or icon. Another example are indexes which are placed close to physical objects and digitally repeated.



Figure 3.2: The Power-Aware Cord shows how much energy the connected lamp consumes [Gustafsson and Gyllenswärd, 2005].

Some location-unaware ubiAV systems use location for linking information to an object although they don't have knowledge about the object's location. For example, the Power-Aware Cord [Gustafsson and Gyllenswärd, 2005] visualizes energy consumption information close to the consumer. The cord does not have knowledge about the consumer's location. The location connection is established by the system setup (cf. Figure 3.2).

3.3.3 Reproduction Fidelity and Extent of Presence Metaphor

The remaining two dimensions of [Milgram and Kishino, 1994]'s taxonomy are "reproduction fidelity" and "extent of presence metaphor". We do not adapt them for ubiAV because we think that the issues which they are dealing with are less relevant for ubiAV systems than the above described dimensions. Reproduction fidelity and extent of presence metaphor both treat the topic of realism in mixed reality displays. Reproduction fidelity refers to the image quality of the synthesizing display. It ranges from simple wireframes to photorealistic graphic rendering. Extent of presence metaphor refers to the extent to which the user feels present within the displayed scene.

These two dimensions are difficult to adjust for ubiquitous annotation visualization technologies that do not make use of light emitting displays and graphics. This is the case, e.g., for ambient displays like the Dangling String [Weiser and Brown, 1996]. The string does not reproduce the visualization of a physical object. Hence, a reproduction fidelity cannot be determined. Analogously, the extent to which the user feels present within the displayed scene cannot be determined because there is no scene displayed.

3.4 Classifying Related Work

In this section, the systems presented in Section 3.1 are classified according to the taxonomy in Section 3.3.1. Table 3.1 shows the results in a matrix. Hereafter, we explain which work is placed where and why.

3.4.1 Location-Aware, Real World Systems

Often, the location of real world objects is modeled in AR systems. Knowing the object's location and the user's location makes 3D registration of virtual content in the real world possible. For example, Wikitude World Browser [Wikitude GmbH, 2014] and Layar [Layar, 2014] estimate the current line of sight of the user using the mobile phone's GPS module, compass and other sensors. Thanks to the knowledge about absolute locations of objects of interest, these applications can determine which objects are in the line of sight. Information about these objects is overlaid. For the Everywhere Displays [Pinhanez, 2001] and Fluid Beam [Spassova, 2004], the location of objects in the room is modeled so that the projector knows onto which locations to project information. Also for the mentioned use cases for Virtual Retinal Displays [Viirre et al., 1998] and [Lim et al., 1999], object's locations are modeled.

3.4.2 Location-Unaware, Real World Systems

Other AR systems also need location data about the real object in order to visualize information at the right place. But the location data can be derived at runtime instead of being modeled before. For example, [Bajura et al., 1992] senses the location of HMD and ultrasound transducer using motion tracker and infers the annotated object's and user's locations. Also NaviCam [Rekimoto, 1995], Sixth Sense [Mistry et al., 2009], RFIG Lamps [Raskar et al., 2004], Invisible Train [Wagner et al., 2005], Bonfire [Kane et al., 2009] and FACT [Liao et al., 2010] sense or derive the annotated object's location.

Ambient displays that utilize everyday objects are real world systems as well. As we explained in Section 3.3.1, the Power-Aware Cord is an example for a system which links information to an object through location although the object's location is not modeled [Gustafsson and Gyllenswärd, 2005]. Often, the connection to the annotated object is established via non-location references. For example, the annotation of Digital Family Portrait is about the person on the photo [Mynatt et al., 2001]. Other ambient display installations, e.g. Waterlamp and Pinwheels [Dahley et al., 1998], require the user to know which particular object is being annotated. CoDine [Wei et al., 2011], Clouds

Chapter 3. Classification of ubiAV Approaches

	Real World	Virtual World
Location-Aware	<p>Virtual Retinal Displays [Viirre et al., 1998] [Lim et al., 1999]</p> <p>Everywhere Displays Projector [Pinhanez, 2001] Fluid Beam [Spasova, 2004] Layar [Layar, 2014] Wikitude World Browser [Wikitude GmbH, 2014]</p>	<p>Cyberguide [Abowd et al., 1997]</p> <p>HIPPIE [Oppermann and Specht, 2000] Outcast [McCarthy et al., 2001]</p> <p>Kore [Bombara et al., 2003] [Paul et al., 2005]</p>
Location-Unaware	<p>[Bajura et al., 1992]</p> <p>NaviCam [Rekimoto, 1995] Dangling String [Weiser and Brown, 1996] Waterlamp and Pinwheels [Dahley et al., 1998] Digital Family Portrait [Mynatt et al., 2001] RFIG Lamps [Raskar et al., 2004] Power-Aware Cord [Gustafsson and Gyllenswård, 2005] Invisible Train [Wagner et al., 2005] Bonfire [Kane et al., 2009] Sixth Sense [Mistry et al., 2009] FACT [Liao et al., 2010] Clouds [Rogers et al., 2010] CoDine [Wei et al., 2011]</p>	<p>NESSIE [Prinz, 1999]</p> <p>BlueBoard [Russell et al., 2002] Hermes Door Displays [Cheverst et al., 2003] Mobile Tagging Systems [Holmquist, 2006] MICA [Prause et al., 2010]</p> <p>XP-Iris [Antenna International, 2013] [Bieber et al., 2013]</p>

Table 3.1: Classification of ubiquitous annotation visualization technologies

[Rogers et al., 2010] and Dangling String [Weiser and Brown, 1996] are also real world systems that do not model object's location.

3.4.3 Location-Aware, Virtual World Systems

Virtual worlds can be created as mobile device applications. A common approach for ubiAV systems on mobile devices is to reference real objects on a map. Cyberguide [Abowd et al., 1997] models the object's location in order to place its representation at the right position. Modeling location can also be necessary for mobile applications which do not incorporate maps. HIPPIE [Oppermann and Specht, 2000] and Kore [Bombara et al., 2003] determine the user's current line of sight by a positioning infrastructure. The location of annotated objects must be modeled in order to present an annotation depending on the line of sight.

A virtual world with reference to real objects can also be displayed on fixed displays. Outcast models location data of annotated objects [McCarthy et al., 2001]. It can show location information about the annotated owner based on her infrared badge. [Paul et al., 2005] model the location of the preoperative images in order to place them correctly in the virtual 3D scene.

3.4.4 Location-Unaware, Virtual World Systems

The more common case is that peripheral displays do not model location data. For example, the Hermes Door Displays show annotations about the owner of the office where they are situated [Cheverst et al., 2003]. For the visualization in NESSIE, the location of the coffee room is irrelevant [Prinz, 1999].

Also mobile ubiAV applications do not need to model location data in order to reference annotated objects. XP-Iris only needs to model indexes of exhibits [Antenna International, 2013]. MICA shows information about the object which is currently scanned [Jentsch et al., 2009]. [Bieber et al., 2013]'s smartwatch knows that it is held to the heart by analyzing the acceleration sensor. Similar examples of location-unaware virtual world systems are mobile tagging systems [Holmquist, 2006].

3.5 Conclusion

This chapter gave insight into the ubiquitous annotation visualization domain. Weiser's vision of changing from terminal based computer systems to mobile and situated systems is put into practice. The presented ubiAV systems come from different research fields in ubiquitous computing, such as ambient displays, mobile applications and augmented reality. This shows the broad availability of the concept in existing works. Every research field again comprises several different ways of performing ubiAV. This makes the domain broad; it is difficult for a framework to address every single approach.

Organizing the domain by relevant criteria helps to find commonalities and to focus on the important aspects. Possible structure elements are annotations, annotated objects, the capabilities of visualization devices or the spatial relationship of the involved

entities. The interrelation of these classes reveals additional challenges. For example, the incompatibility of the data types of annotations and the visualization capabilities of mediators demand solutions.

We decided to use a manageable grouping scheme that is based on the taxonomy of [Milgram and Kishino, 1994]. By adjusting the taxonomy, we detected four classes of ubiAV systems that structure the design space of our framework. Harmonizing the taxonomy from a continuum to discrete classes makes it more manageable and also decreases fuzziness. [Milgram and Kishino, 1994]’s understanding of some aspects was updated and applied to the surveyed ubiAV concepts so that the classification can be applied to the novel point of view. For example, [Milgram and Kishino, 1994] mainly mean 3D spaces when talking about virtual worlds, but the concept can be applied to mobile maps as well. By classifying the presented systems according to the adjusted taxonomy, we showed its applicability to these ubiAV systems.

Our classification helps to identify common problems and solutions among elements of the same class. So, if a framework supports the instance of one class, there is good chance that it addresses problems and solutions of other instances of that class as well. Hence, the goal of our framework is to support at least one instance of each identified class.

In Chapter 2 and Chapter 3, we outlined the necessary knowledge to specify the UbiVis framework, which supports developers in rapid prototyping of ubiAV applications. We sketched out general domain knowledge about ubiquitous annotation visualization. We elaborated the requirements for a framework to support rapid prototyping of ubiAV. We presented best practices of related frameworks. We conveyed a sense of current ubiAV systems’ characteristics. We made aware which kind of digital information is used and how it is treated. Finally, we provided an overview of recurring problems and corresponding solution approaches. All this knowledge will be used to develop the UbiVis framework in the upcoming chapters.

Chapter 4

Conceptual UbiVis Framework

UbiVis supports developers of ubiquitous annotation visualization systems to exchange visualizations, thus enabling rapid prototyping. Before specifying the framework, we take essential design decisions, which are driven by the existing and novel concepts elaborated in Chapter 2 and Chapter 3. The classification process revealed common approaches and challenges among existing ubiAV systems. As a first step towards UbiVis's specification, we review these commonalities and link them to the framework's objectives. The objectives are derived from the discussion about ubiquitous annotation visualization systems and rapid prototyping in the previous chapters. Some design decisions lead to consequential reflections, which demand further decisions. The design decisions are worked out in Section 4.1.

The actual framework specification in Section 4.2 is based on these design decisions. It follows the viewpoints and views concept of [Rozanski and Woods, 2011]. The specification consists of several aspects. The system is structured and individual components' functionalities and responsibilities are defined. This also leads to interface agreements. Next, data structure and flow are elaborated. The third major cornerstone of the specification is the deployment configuration.

In Section 4.3, we develop a standard procedure how to apply the UbiVis framework. The process is particularized as a step-by-step tutorial. Having specified an architecture and instructions how to apply it, enables developers to instantiate the framework. We will use this standard procedure in order to apply the framework architecture to a concrete example application in subsequent chapters.

UbiVis aims at supporting a broad domain. It is difficult for a single supplier to deliver a library for each conceivable visualization technology. Thus, a main feature of the UbiVis framework is the possibility to extend it with new visualization libraries. The development of another standard procedure for the generation of fresh libraries completes the framework specification in Section 4.4. This is again a step-by-step tutorial. So, every developer may add new technology support to the framework and make it publicly available. This concept allows for extensibility of current and future technologies.

4.1 Design Decisions

In this section, we take architectural design decisions about the UbiVis framework. Each decision is based on the analyses in Chapter 2 and Chapter 3.

Design Decision 1: Defined Interface for all Visualization Components

In Section 2.1.7, we elaborated that for rapid prototyping of ubiAV systems, we want application code to remain unchanged when visualization is exchanged. This can be achieved by the following design:

The visualization is encapsulated as an own component of the system. We define an interface for a method that triggers the actual visualization. Every visualization component has to implement that interface.

So, an application which uses the UbiVis framework can call that interface and all visualization components of the framework understand it. Hence, visualizations can be exchanged without having to change the core application code.

Design Decision 2: Two Interface Parameters

The interface that is mentioned in Design Decision 1 needs parameters which have to be passed by the application logic to the visualization component. By definition (cf. Section 2.1.5), a ubiquitous annotation visualization annotates a physical object with digital information. Hence, a call which triggers such a visualization will need two parameters: A reference to the physical object and the digital information to be used for the annotation.

Design Decision 3: Data Types

The most important requirement for the reference to the physical object is to be unique because it only serves to unambiguously identify the object. Hence, any classic basic primitive data type suffices as physical object ID.

In Section 3.2.2, we identified numeric values, texts, images as being the main data types of the digital information of the ubiquitous annotation visualization systems in our survey. Boolean information and videos are further possible information types. We restrict the data type of the digital information within our framework to numeric values, texts and images, since we regard these as the atomic types. The other information formats can be transformed to these atomic types while numerics, texts and images cannot be transformed to the other without losing main characteristics. An animation or video can be accomplished by several succeeding visualization calls that are consisting of image information. Boolean information is a restriction of the numeric space to the values 0 and 1.

Numeric values could be represented as texts but by doing this we would lose the possibility to use the inherent characteristic of numeric values to be positioned on an ordinal scale. A numeric value always includes some meaning since it can be compared on an ordinal scale. However, some texts only represent a nominal scale. These texts

would be just visualized without interpreting its meaning by most of the visualization libraries while numeric values are likely to control the visualization. As an example of a location-unaware, real world system, CoDine [Wei et al., 2011] prints textual information on toasts. The print's format is the same for every text. However, the Power-Aware Cord [Gustafsson and Gyllenswärd, 2005], another similar location-unaware, real world system, uses numeric data to control the visualization's format. The higher the value of the digital information - the consumed electrical power of a connected device - is, the faster the visualization moves.

In the same way, texts could be represented as images. But images need much more memory. In addition, conversion between image and text or numeric is cumbersome and slow. So, for performance reasons, we keep the distinction between images and the more basic data types.

Design Decision 4: Configuration

Annotations are treated differently by every visualization approach. For instance, numbers, texts and images can be presented on a public display straight forward. However, an installation of ambient light needs to represent the information in light patterns. Hence, it does not have the possibility to visualize a string in textual form. So, a visualization component needs to have the possibility to transform given digital information of a data type defined in Design Decision 3 into an appropriate format for the particular component (cf. Section 3.2.4). For this purpose, we give developers a configuration possibility to specify how to handle a certain kind of information for the particular application. This will mainly be a mapping of particular digital information to another form of data which suits the visualization component.

The same concept applies to the physical object's ID. Every visualization approach may need to derive different knowledge from such an ID. For instance, location-aware ubiquitous annotation visualization approaches might extract the location of the physical object from the ID. Other approaches might use the physical object's ID to identify an image of this physical object. We use the same configuration file as for the digital information. Hence, the application developer can map the physical object's ID, e.g., to location coordinates.

Design Decision 5: Modeling Data Space

The particular instantiations of physical objects and digital information vary from application to application. It is the developer's task to model them. The libraries still determine which input variables are needed for the configuration but do not specify their format. This way, the framework stays compatible for different kind of data models. Since an application developer decides what the input variables look like, she can cluster the data space to her own preferences without being restricted by the framework. The applicability of the framework is still guaranteed.

Design Decision 6: Configuration Ranges

It would be cumbersome to map each individual physical object's ID of the data space to a particular attribute value. That is why we also allow specifying ID ranges. So, the application developer can assign a range of IDs to a specific attribute value. Since the way for assigning IDs is decided by the application developer, she can cluster them in a way that makes it easiest for her to map ranges of IDs to a specific attribute value. We proceed analogously for mapping annotation IDs.

Design Decision 7: Compositions

As pointed out in Section 3.2.1, annotated objects can consist of other physical objects. This can lead to uncertainty for users which object is annotated. However, our survey shows that this uncertainty is usually clarified through the annotation content (cf. Section 3.2.3). As a consequence, we restrict physical objects and annotations to atomic instances.

4.2 Specification of the UbiVis Framework

In this section, we specify the architecture of the UbiVis framework according to the design decisions we took in Section 4.1. We base our architecture specification on the viewpoints and views concept of [Rozanski and Woods, 2011]. They provide a commonly used compendium of methods for architecture specification. It extends the definitions of [Institute of Electrical and Electronics Engineers Standards Association, 2000] and thus is based on an official standard. Hence, the specification scheme is likely known and understood by a huge number of application developers.

4.2.1 The Architecture Guide by [Rozanski and Woods, 2011]

Since a software architecture consists of many different functional features and quality properties, it is not possible to properly describe all of them in one single model. Either, this would make it too complex to find and understand the aspects that a particular observer needs. Independent aspects might become intertwined. Or, if complexity is reduced, there is the risk to miss details of an aspect, which might lead to the audience making wrong assumptions.

As a consequence, [Rozanski and Woods, 2011] handle the different aspects of a software architecture by a set of separate but interrelated views. Each view deals with a separate quality but collectively, they define the whole system. The authors define: "A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders."

For each view, [Rozanski and Woods, 2011] define viewpoints. A viewpoint is a description of a view which is also proposing templates and models. So, a viewpoint can be described as a standard for a view. Architecture designers do not have to define

what belongs to the view but can build on reusable knowledge. Having standards for each view also improves the understandability of it. [Rozanski and Woods, 2011] define: "A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views."

We use the following viewpoints from the set of viewpoints provided by [Rozanski and Woods, 2011].

Functional Viewpoint

The functional view is often seen as the cornerstone of most architecture specifications. It structures a system into single elements and defines their functionalities and responsibilities. The view additionally describes the interactions between the elements including exposed interfaces. In the same way, interactions and interfaces to external systems are specified. A functional element is any software module, data store or external entity which has particular responsibilities and exposes well-defined interfaces for its connection to other elements. Interfaces are defined by input and output parameters and the nature of interaction needed to invoke the operation. Interactions between components are indicated by connectors. For our specification, we use a UML component diagram [Object Management Group, 2014], which is mainly used to particularize a functional view.

Information Viewpoint

The main purpose of an information system is to manipulate and distribute data. The information view describes how this is done and how data is managed and stored. First, the static data entities and their relationships to each other are modeled. UML class models can be used for this purpose. The modeled classes are the data entities including attributes and associations are the static relationships among them. Optionally, the behavioral aspects of a system can be omitted by not specifying the methods. Besides static information structure, the way information flows inside the system is part of the information view. It specifies by which components data is accessed and modified. Every model that can specify which data is transferred between which components, including a direction, is suitable for illustrating the information flow. One example is a UML sequence diagram.

Development Viewpoint

This view focuses on the implementation time. This includes the definition of design constraints or standards to ensure technical integrity. Other examples, that might be worth to specify, are code structures, used external tools or how logging should be done. The development view does not want to interfere too much with the implementers' liberty. So, only those aspects should be specified that make the system easier to understand and use. For example, the risk of duplication of efforts is reduced by identifying standard

approaches or by stimulating technical coherence. Development views can be modeled using diagrams, e.g., a module structure is often depicted as a UML component diagram. However, for logging of a medium-sized set of constraints, a simple text approach suffices. This often consists of a definition of the common processing required across elements, standard design procedures and what common software to use and how to use it.

Deployment Viewpoint

The deployment view stresses the physical environment that the system is intended to run in. This includes the description of technical nodes and other hardware elements as well as a mapping of software units to the runtime environment. Third-party software which is running on a hardware node may be of interest; this is often especially true for the operating system. The deployment view reveals technological compatibilities and communication between hardware nodes, e.g., requirements for a network connection. A UML deployment diagram can be used to model this view. It shows computing nodes with software elements inside and the network links between the nodes.

Concurrency and Operational Viewpoint

The concurrency view and the operational view are left out for the specification of the UbiVis framework. The concurrency view is dealing with the questions which threads of the systems can be executed concurrently and how this is controlled. Since there are no concurrent elements in the UbiVis framework, this view is skipped.

The operational view is dealing with maintenance issues, once a system is deployed from the project period to operations. Since UbiVis aims at supporting rapid prototyping, the applications which are built with the help of the framework are not intended to reach operations. Hence, the operational view is ignored.

Viewpoints Relationships

The presented views focus on different aspects of the software system. Each individual view comprises details that might be of interest for a specific stakeholder or situation in the development process. Figure 4.1 shows how the views are interrelated and how they build the architecture of the specified software structure together.

The software structure is defined by the functional view, information view and concurrency view, each of them detailing another aspect. The deployment view defines the deployment of the software structure while the operation of this deployment is particularized by the operational view. The development view focuses the implementation time, i.e., defining constraints for this.

4.2.2 Functional View

In the following sections, the before described architecture specification process is applied for the UbiVis framework. Figure 4.2 gives the overview of the UbiVis framework's

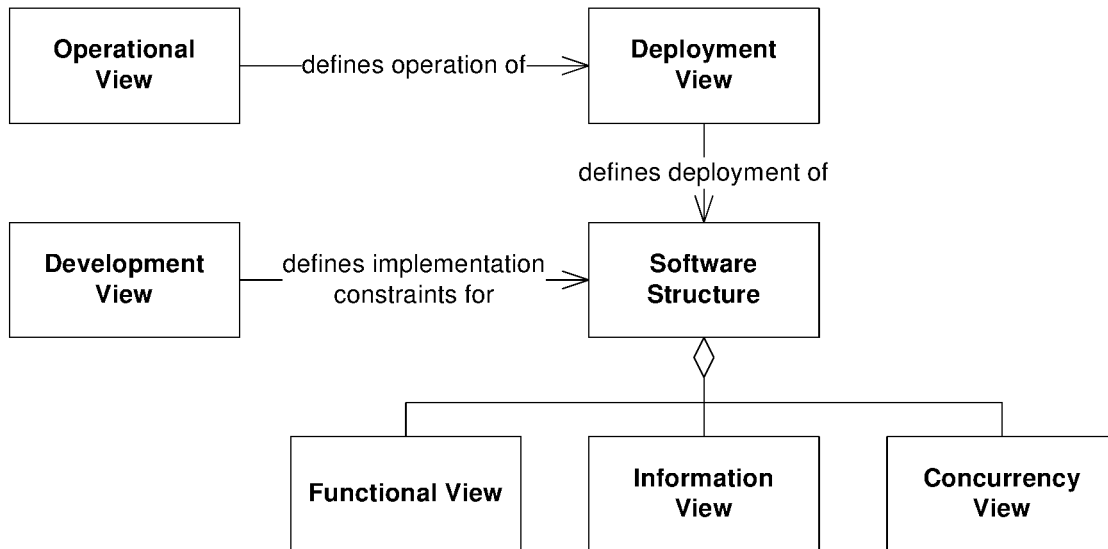


Figure 4.1: The different views are related to each other and so, together describe the software structure (adapted from [Rozanski and Woods, 2011]).

functional view. Its design is discussed in this section.

Main design decisions taken in Section 4.1 are: The visualization inside the UbiVis framework must be designed as an own component and must not be part of any application logic. It must implement a defined interface through which the visualization is triggered. When triggering the visualization, a parameter tuple shall be passed. We declare the `visualize(visualizationTuple:VisualizationTuple)` method to be this interface.

The actual visualization can be performed theoretically on any technology which can be integrated into a computer system. Accessing and integrating this technology is often the most difficult piece for a ubiquitous application developer because it is not part of her normal work. It shall thus be hidden in the visualization library. We cut off a visualization proxy component from the actual visualization unit for underlining the necessity of dealing with the connection to a potentially unfamiliar technology inside the library. The idea is that the visualization proxy is running as a standard technology part which can be easily integrated by a common ubiquitous application developer. The visualization proxy is implementing the interface and the only element the developer has to interact with. It is then responsible for controlling the actual visualization component and forwarding the visualization tuple according to the call of the `visualize(visualizationTuple:VisualizationTuple)` method. For this, the visualization part provides a technology specific visualization method. Both units, visualization proxy and visualization, together build a visualization library.

As pointed out in Section 4.1, the application developer shall have the possibility to configure the visualization component. We introduce a configuration file for this purpose

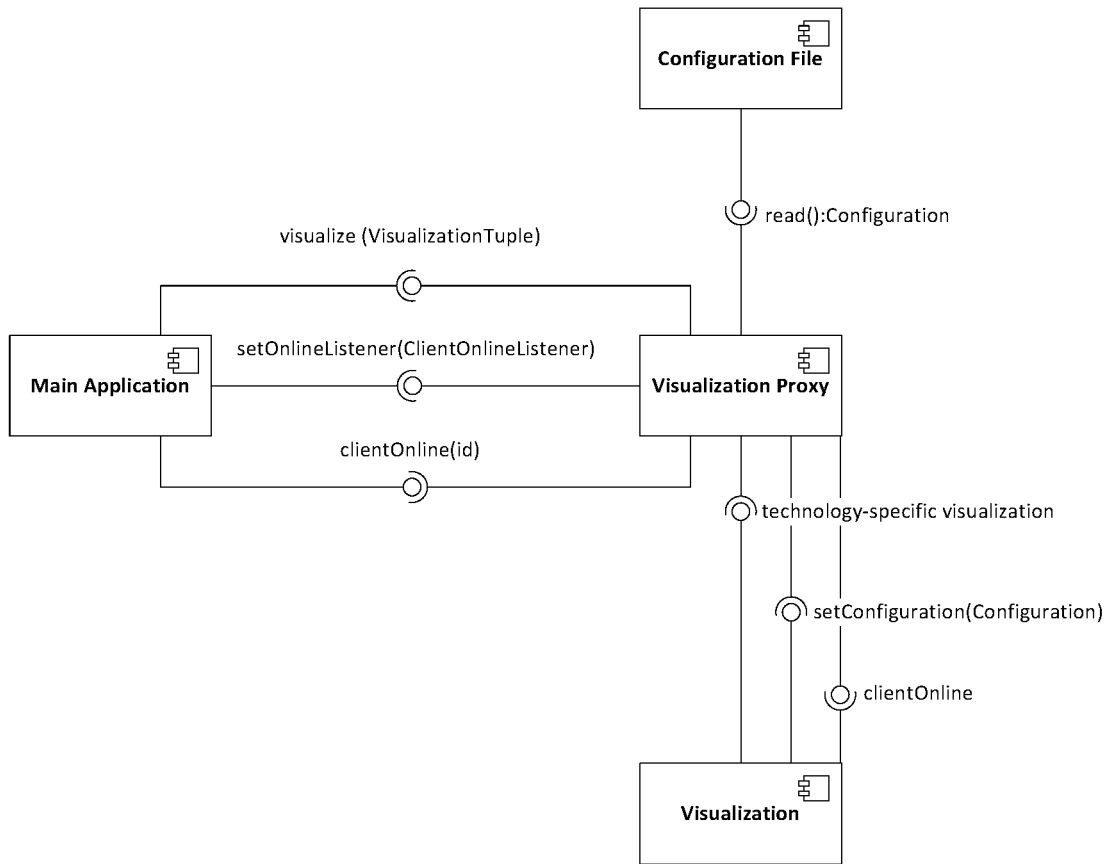


Figure 4.2: The functional view defines components and their interactions.

because it can be easily externalized from the visualization library. Also from the point of view of the library, a configuration file is easy to manage compared to other input possibilities because file handling is supported by many programming languages. The configuration file is connected to the visualization proxy since this element is intended to be the interface for the application developer. Since the visualization proxy is always running on the same standard technology, all visualization proxies treat the configuration in the same way. This makes configuration easier for an application developer. At start time, the visualization proxy reads the configuration file and forwards it to the visualization component.

For some visualization technologies it may take some time to start. So, it is not ready when the main application connects to the visualization proxy. Hence, the main application needs a mechanism to know when the visualization is active. Otherwise, it misses the information whether the visualization call can be executed. For this purpose, the visualization proxy acts as a notifier for a registered main application. The visualization proxy provides a method `setOnlineListener(ClientOnlineListener)` to which

the main application can register. The main application has to implement this `ClientOnlineListener` interface, which requires implementing a `clientOnline(id)` method. This method is called from the visualization proxy when it receives a `clientOnline` notification from the visualization component. The visualization proxy must then apply an ID, which enables the main application to identify the visualization unit that has become active.

4.2.3 Information View

The information view consists of several models. A UML class diagram shows the static data structure of the ubiAV design space (cf. Figure 4.3). It is simplified to the purpose of the UbiVis framework (cf. Figure 4.4). UML sequence diagrams are used for visualizing the information flow (cf. Figure 4.5 and Figure 4.6).

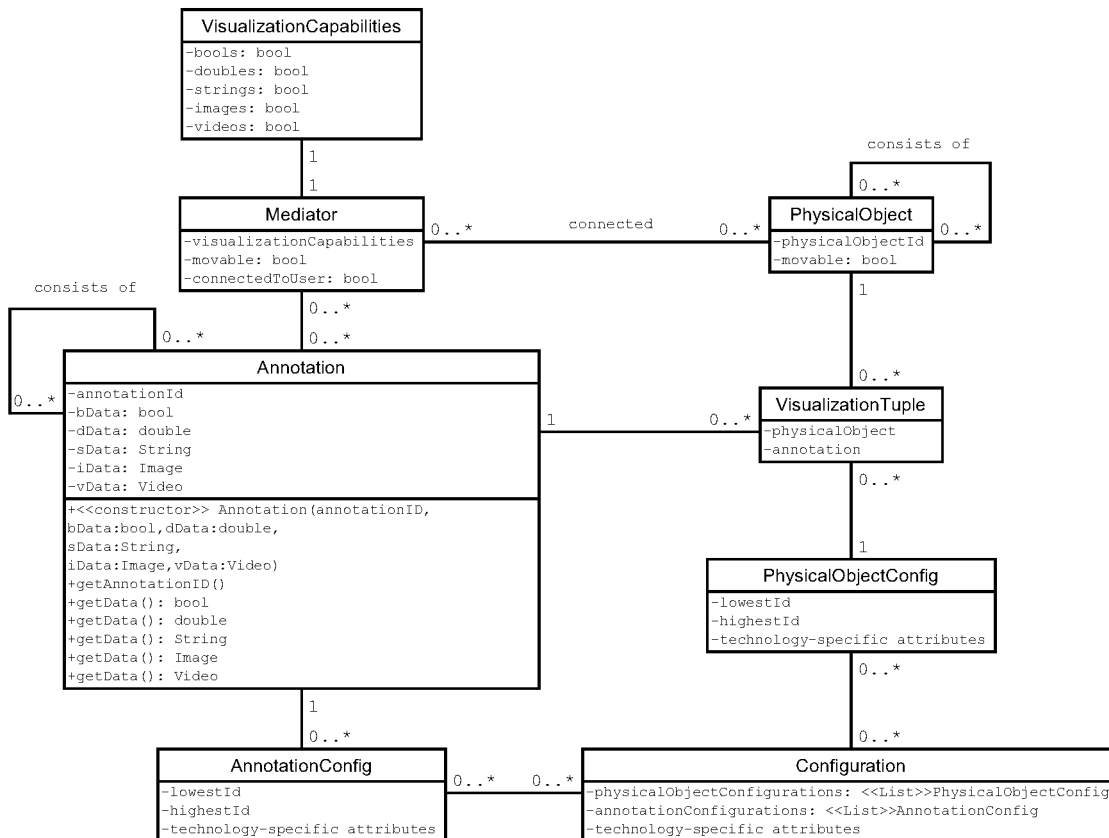


Figure 4.3: The class diagram shows the static data structure of the ubiAV design space.

Design Decision 2 (cf. Section 4.1) demands that a call of the defined interface’s visualization trigger needs a reference to the annotated physical object and to the digital

information which constitutes the annotation. In Section 4.2.2, we named this trigger method `visualize(visualizationTuple:VisualizationTuple)`. So, the `VisualizationTuple` class is the container of the two references.

A physical object is identified through a unique ID. The `movable` attribute indicates whether the user can or is allowed to move the object or not (cf. Section 3.2.5). According to Section 3.2.1 and Section 3.2.3, a physical object can consist of other physical objects. So, a `PhysicalObject` instance can have references to other instances of that class.

As pointed out in Section 3.2.2, the annotation can be a boolean, number, text, image or video. We deal with this by adding five data attributes to the `Annotation` class, one for each type. Along with this comes an own getter method for each type and a constructor holding all five attributes. When creating an `Annotation` object in the main application, each data attribute can be set. If a data attribute shall not be filled, a null object must be passed.

Some visualization components might expect a particular data type. If that data type is not filled, the visualization might not work for the particular application. But the visualization component can offer the application developer to map one data type to another using the configuration to overcome this problem.

Analogously to physical objects, an annotation can consist of other annotations (cf. Section 3.2.2). Consequently, an `Annotation` instance can also have references to other instances of that class.

It is unusual to specify concrete data types, getter and constructor methods in the information view, but since it is an important aspect for this specification, we disregard this convention for the `Annotation` class. The data type `double` is used for the numeric data attribute since it comprises all other numeric primitive data types.

The `Mediator` class represents the device on which the annotation is visualized. It is capable of visualizing the data types indicated by `VisualizationCapabilities` (cf. Section 3.2.4). Its properties determine how the mediator can be used to establish the logical link between the physical object and the annotation (cf. Section 3.2.5). Hence, the mediator's mobility is modeled, analogously to the `PhysicalObject` class. In addition, it is modeled whether the mediator is connected to the user and to which physical objects it is connected.

Design Decision 4 (cf. Section 4.1) implies to provide the possibility to configure a visualization library by mapping an annotation or a physical object to technology-specific attributes. It shall also be allowed to map a range of physical objects or annotations (cf. Design Decision 6). This purpose is served by the classes `PhysicalObjectConfig` and `AnnotationConfig`. A `PhysicalObjectConfig` consists of a range of physical object's ID, which are indicated by the highest and lowest ID that belongs to the range. Additionally, it consists of a number of technology-specific attributes. This way, a group of physical objects is associated to technology specific attribute values. Single physical objects can be configured by setting the lowest ID equal to the highest ID.

In order to be able to configure a range of annotations, we introduce a unique identifier to the `Annotation` class, which we call `annotationId`. The `AnnotationConfig` class works analogously to `PhysicalObjectConfig`. An instance also selects a range of

4.2. Specification of the UbiVis Framework

Annotation objects by specifying the highest and lowest `annotationId`. A number of technology-specific attributes is associated to this.

The Configuration class is the container for all `PhysicalObjectConfig` classes and `AnnotationConfig` classes of a visualization library. Further technology-specific attributes, which are not depending on a particular physical object or annotation, are also kept in this class. These can be, e.g., global library settings such as port configurations or the used positioning technology. The attribute structure of the configuration file should fit to the structure of the Configuration class so that mapping is easy.

The presented data model contains the comprehensive collection of different ubiAV concepts and dimensions. Some aspects are irrelevant for the framework core because they have to be managed in the particular visualization libraries. Moreover, we take design decisions to simplify the model to the most important aspects. The resulting adjusted data model which we use for UbiVis is shown in Figure 4.4.

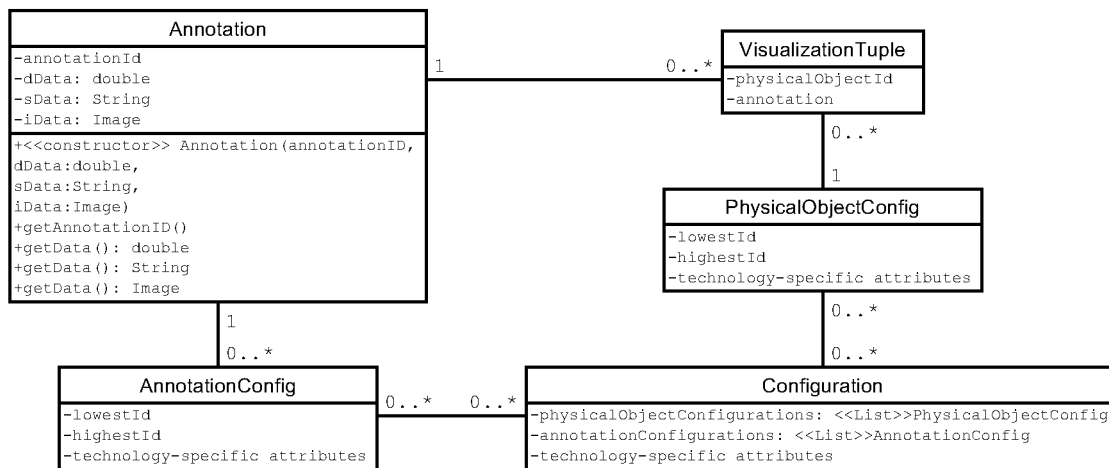


Figure 4.4: The adjusted class diagram shows the static data structure of UbiVis.

The question how to establish the logical link between physical object and annotation must be answered by each individual visualization approach (cf. Section 3.2.5). Thus, the modeling of the mobility and connectivity of the mediator and physical objects is not needed for the framework core. Furthermore, the mediators' visualization capabilities do not need to be modeled in the framework core because it can be handled through the configuration possibilities (cf. Design Decision 4 in Section 4.1). The mentioned classes can still be used by the visualization libraries but we remove them for the data model of the framework core.

As pointed out in Design Decision 3 (cf. Section 4.1), we restrict the modeled data types of the Annotation class to `double`, `String` and `Image` because they are the most important ones. Moreover, the other types can be transformed to these types. Design Decision 7 removes the necessity to model compositions of physical objects and anno-

tations. Since the `PhysicalObject` class then only consists of the `physicalObjectId` attribute, we dismiss the class and integrate the attribute into `VisualizationTuple`.

`VisualizationTuple` and `Configuration` are the two classes which are passed between the system components. They refer to the other modeled classes, i.e., these classes are implicitly passed as well.

The main application creates a visualization tuple to specify which physical object shall be annotated by which annotation. It passes the tuple to the visualization proxy when calling the `visualize(visualizationTuple:VisualizationTuple)` method. The tuple is then forwarded to the visualization component in the technology-specific visualization call. This information flow is depicted in Figure 4.5.

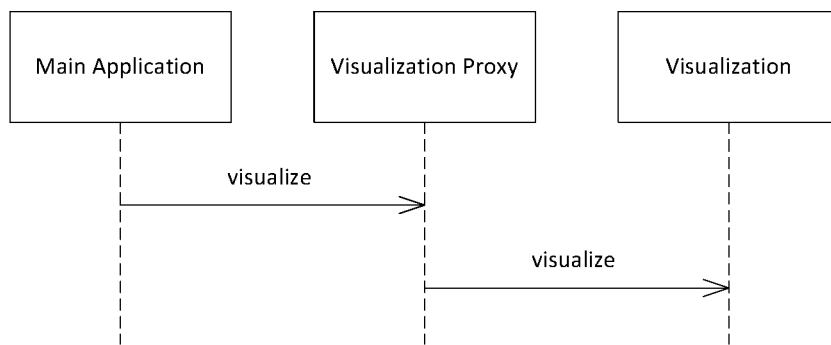


Figure 4.5: Information flow: Trigger visualization

The configuration is read from the configuration file by the visualization proxy. In this step, its content is translated to a `Configuration` object. This object is sent to the visualization component where it is applied. The information flow is shown in Figure 4.6.

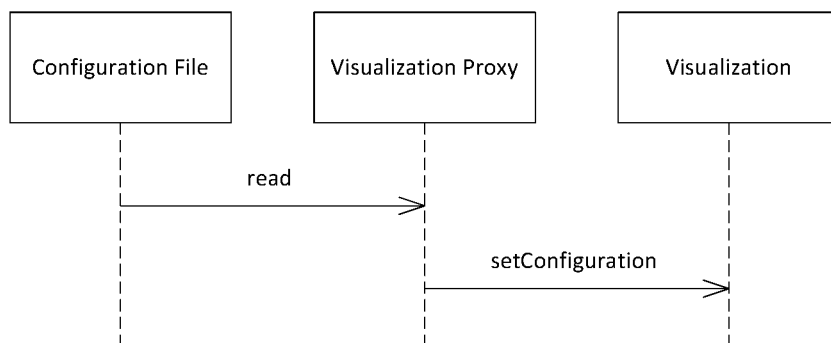


Figure 4.6: Information flow: Configuration

4.2.4 Deployment View

The UbiVis framework covers a lot of different visualization technologies. The used hardware differs from library to library. Hence, the deployment view can only specify a few mandatory and an additional set of optional hardware components. The latter ones are used by some libraries while other libraries do not need them. Dashed lines are used for representing these optional components in Figure 4.7. UbiVis can also be extended by future visualization technologies (cf. Section 4.4). However, potential future hardware cannot be foreseen here.

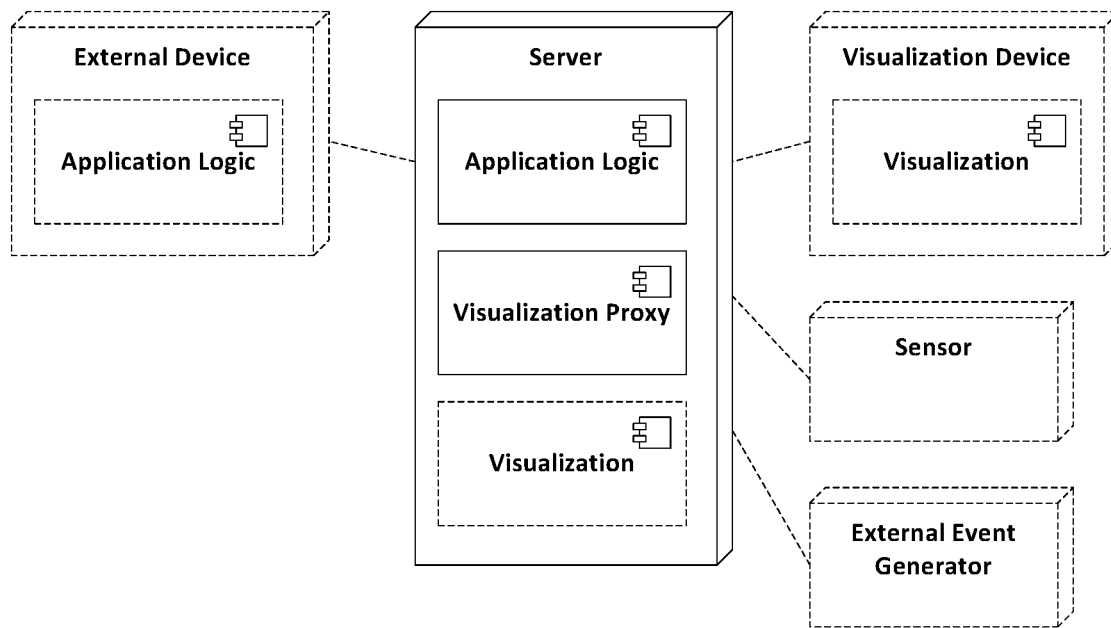


Figure 4.7: The deployment view shows to which hardware nodes the software components are distributed.

According to Section 4.2.2, the only mandatory component is a server where standard technology can be applied, which a common ubiquitous application developer can handle. The server holds the visualization proxy and that part of the application logic which connects to the proxy. It is possible that other parts of the main application are not running on the server but on external devices instead. Likewise, the visualization component can be hosted on the server but on an external visualization device as well. Several UbiVis applications might react on external sensors or external events that trigger visualizations.

The development view is part of Chapter 5 because that chapter describes a concrete technical environment of the framework. As a consequence, implementation details are discussed there. The implementation constraints of the development view provide higher

impact in that chapter than to the conceptual definitions here.

4.3 Procedure of Rapid Prototyping

To supplement the system architecture, we give instructions for a standard procedure how a ubiquitous application developer can apply the UbiVis framework. Having a fixed process to follow is intended to ease the application of the framework. So, the procedure helps to exchange ubiAV technologies rapidly. The standard procedure consists of the following four parts:

1. Analyze configuration of visualizations
2. (optional) Analyze data space of physical objects and annotations
3. (optional) Configure visualizations
4. Develop application logic including calls of VisualizationProxy

1. Analyze configuration of visualizations

Before connecting a visualization technology to the application, the technology must be configured. Each visualization library can provide a configuration file for mapping physical objects' IDs and digital information to technology-specific attributes. So, the first step for a user of the UbiVis framework is to check the configuration files of each visualization library which she wants to apply. This will generate a list of what mappings are possible for the particular set of visualization libraries.

2. (optional) Analyze space of physical objects and annotations

Step 1 has revealed whether physical objects' IDs or digital information must be mapped to technology-specific attributes. If no mapping is required, Step 2 can be skipped. Otherwise, this next step is to analyze which kind of physical objects and annotations the application can expect. This enables to model the data as it is demanded in Design Decision 5 (cf. Section 4.1). It can include several aspects. If the libraries require mapping annotations' IDs to technology-specific attributes, the application developer must determine a format for these IDs. The application developer must be able to access the IDs in order to refer to them in the configuration. At the same time, the system must access the ID so that it can pass it to the constructor of the visualization tuple. Furthermore, the application developer might want to group or classify the IDs in some way.

Additionally, it can be necessary to set particular primitive data types of annotation objects, which would not be set by the application logic otherwise. For example, the application logic could actually only process image information but a visualization library

4.4. Procedure of Developing a new Library

might require to have the numeric annotation attribute set as well. Similar considerations have to be made for the space of the physical objects' IDs. Often, such an ID is mapped to specific attributes of a visualization library. Since the application developer decides what the concrete instantiations of physical objects look like, she can also determine how these IDs are clustered. So, when a group of physical objects or annotations shall be mapped to the same attribute value, their corresponding IDs should be inside a collective range.

3. (optional) Configure visualizations

The next step is to do the actual configuration according to what was analyzed in the first two steps. The configuration file of each library might need to be edited. Besides the required mappings, it might also be necessary to set some global attributes which are not dependent on physical objects or annotations. In principle, it is possible that the visualization libraries do not have to be configured at all. In this case, Step 3 can be skipped.

4. Develop application logic including calls of VisualizationProxy

Finally, the actual development of the application logic can start. This may mean to integrate sensors or other external event sources which determine how objects are annotated. It definitely includes integrating visualization proxies and triggering the visualization by calling the defined `visualize(visualizationTuple:VisualizationTuple)` interface. For passing the visualization tuple parameters to the visualization proxies, the tuple objects must be created according to the analysis of Step 2.

4.4 Procedure of Developing a new Library

A main feature of the UbiVis framework is its extensibility. The framework is designed in a way that any application developer can implement a library for an additional visualization technology. After providing this library to the public, it can be applied by every UbiVis user in the same way as the initial libraries. In this section, we provide another standard procedure for the generation of a UbiVis library in order to ease its development, similar to what we did in Section 4.3. This procedure comprises the following four parts:

1. Define configuration options
2. (optional) Set up configuration file
3. Develop visualization component
4. Develop visualization proxy

1. Define configuration options

The first step is to analyze which parameters a user of the library shall set. These can be either global parameters, parameters depending on physical objects or parameters depending on annotations. Global parameters are all parameters which are not depending on individual physical objects or annotations. For instance, a visualization library might need some port settings to be defined. A common example for a parameter depending on physical objects is the location of such an object. Finally, parameters depending on annotations often arise when a visualization technology is not capable of dealing with a certain annotation data type and needs some conversion rules.

There is a good chance that the library developer becomes aware of some parameters not earlier than when implementing the actual visualization component. Of course, these parameters can still be included in a later step. However, it is recommended trying to define all configuration options before the implementation because analyzing the necessary data space may also reveal incompatibilities of the library to the UbiVis approach. If this leads to canceling the library development, no implementation effort is wasted.

2. (optional) Set up configuration file

If Step 1 revealed that the library requires no configuration, this step can be skipped. Otherwise, the parameters analyzed in Step 1 must be entered to the configuration file. Each individual parameter must get its own substructure in the file. Global configuration options can just be listed. However, parameters depending on physical objects or annotations must apply a structure where the parameter is mapped to a range of physical objects' IDs or annotations' IDs.

Since the principle structure of each configuration file is the same, it is easiest to copy a configuration file from an existing library, enhance or reduce it as necessary and change parameter names.

3. Develop visualization component

The actual component which is controlling the visualization technology is implemented. It needs to provide the actual technology access and the trigger of visualizing annotations. Additionally, a visualization component needs to provide a mechanism for notifying the visualization proxy when it is active, as we have discussed in Section 4.2.2. It must also be able to understand configuration information, which it receives from the visualization proxy.

4. Develop visualization proxy

As last step, the visualization proxy must be implemented in the given standard technology. It will control the visualization component and must incorporate a set of concepts,

which we discussed in Section 4.2. It must implement the `visualize(visualization-Tuple:VisualizationTuple)` interface and trigger the visualization in the visualization component. It must also incorporate the configuration file, be able to read it, convert it to `Configuration` objects and forward these objects to the visualization component. Additionally, it must provide an interface for the notification that the client is active. After receiving this information from the visualization component, the proxy forwards it to registered listeners. For this, it must provide a unique identifier of the visualization library.

4.5 Conclusion

We analyzed the commonalities of ubiAV systems, linked them to UbiVis's goal and derived design decisions for its architecture. The resulting specification offers a series of benefits. As a main feature, the concept allows application developers to use ubiAV technologies without the need of technical knowledge about it. That is because the actual visualization technologies are encapsulated in libraries, which just have to be configured but not edited by the developer. The possibility to configure visualization components ensures the compatibility between the framework and different kinds of visualization technologies. The architecture allows extending the framework with more visualization libraries, thus addressing further than an initial set of visualization approaches. Rapid prototyping of ubiAV approaches is enabled as libraries can be easily exchanged without having to change code. This allows for quickly comparing visualizations for a given application. A majority of data types can be handled as they can be transformed to the supported atomic types. At the same time, the number of data types that must be handled remains clear and a good system performance is ensured. The data space can be determined by application developers and is not restricted by the framework. There is also support for the important group of devices which cannot display text or images on screen-like devices, e.g., ambient light. This is done through the mapping ability in the library configuration. This method additionally effects that different visualization technologies can derive different information from the same data. Developers model data themselves instead of being bound to a predefined data model. This enhances the framework's flexibility regarding the information view.

The framework specification is based on a standard procedure. This makes it easier for developers to understand and adjust it if necessary. The procedure specifies different views for each single purpose. This implies that every relevant piece of information for that purpose is defined. Whereas, information is hidden which is irrelevant for that purpose because it would only confuse the reader.

Furthermore, a four step instruction set provides a standard procedure how developers can apply the framework for rapid prototyping of ubiAV systems. It reduces complexity because the developer does not need to come up with an own procedure. It allows developers to apply the framework without knowing details about ubiquitous annotation visualization and without having completely understood its architecture.

A second standard procedure guides the application developer to extend the frame-

work by adding a further visualization library. This is another precondition for UbiVis to be extended by technologies beyond an initial set of visualization libraries. Also future technologies can be supported. Analogously to the application standard procedure's benefits, providing a new library does not require having completely understood the architecture specification or being an expert in ubiquitous annotation visualization. Developers can simply follow the four step process. This opens the possibility of extending UbiVis to a broader range of developers, and thus increases the probability that the framework is extended.

Chapter 5

Technical UbiVis Framework

Chapter 4 specified the concept how developers can perform rapid prototyping of ubiquitous annotation visualization. The concept is not bound to a particular technical implementation yet. Such a technical framework is needed before concrete software can be developed with UbiVis. At the same time, it is a precondition for creating the initial visualization technology libraries because they have to be developed in a concrete technical environment. In this chapter, we define the technical framework for applying the UbiVis concept. Technical and conceptual framework together add up to the UbiVis framework. This combination supports implementing software for the intended use cases.

We investigate whether middleware can handle many of the arising technical issues. These issues need to be solved by the technical framework but are not on our focus. We base our technical framework on the middleware so that it concentrates on the ubiAV aspects instead of having to care about the lower level problems. At the same time, basing UbiVis on an approved technology promises a good level of stability. For finding the optimal underlying middleware, in Section 5.1, we specify the requirements based on the knowledge gained in the previous chapters. In Section 5.2, we discuss potential middleware solutions with regard to these requirements.

The outcome is to build the technical framework upon the LinkSmart middleware [Eisenhauer et al., 2009]. In Section 5.3, we detail its components to convey an understanding of its concepts. In Section 5.4, we show how the LinkSmart components are used to apply UbiVis's concepts. In order not to limit the technical framework to LinkSmart's borders, we also provide a standard way of communication between LinkSmart and non-LinkSmart components.

We finalize the technical framework by a set of conventions and development rules, illustrated in Section 5.5. This serves as additional standard which developers can follow. The standard decreases inconsistencies that would occur if incompatible ways of implementation were used. Also, it facilitates understanding the code of other developers. The conventions and rules are presented as development view, which also completes the procedure of [Rozanski and Woods, 2011].

5.1 Requirements

Before we can specify the technical framework, we need to analyze which requirements it must meet. The requirements arise from the conceptual framework and the knowledge about ubiquitous annotation visualization systems which we have outlined in the previous chapters.

Heterogeneous Protocols

Ubiquitous annotation visualization can be conducted in many divergent ways, which involves a diverse collection of visualization technologies. The difference in potential approaches is huge, e.g., augmented reality on a smartphone and visualization through everyday objects seem to have little common ground. As one characteristic, the technologies make use of a heterogeneous set of protocols, which UbiVis aims to support. So, the technical framework needs to be able to manage these dissimilar protocols. Ideally, it can harmonize them and provide a generic way to integrate every existing and future protocol.

Loose Coupling

The conceptual framework specifies several components. Hence, the technical framework must allow encapsulation in components. The central feature of UbiVis is to exchange the visualization component. Thus, the technical framework must intend a loose coupling of parts so that single elements can be exchanged. Ideally, this should be possible at runtime so that the visualization exchange can happen without the need of restarting the system.

Distributed Structure

UbiAV systems often require software components not to be at a central hardware component. This is due to the distributed nature of involved hardware, such as mobile phones or sensors. Accordingly, the conceptual framework comprises a distributed software structure (cf. Section 4.2.4). The technical framework must manage the resulting networking issues, such as connection and address management. In consequence, parts of the technical framework must run on different network nodes, resulting in a distributed structure itself.

Resource Constrained Devices

Due to the distributed structure, the technical framework must run on different devices. Since we are addressing the ubiComp domain, these devices can be resource constrained. To summarize, the technical framework must support and be able to run on low resource devices.

Standards

The more visualization libraries are available, the more useful UbiVis will be. Each new library allows addressing an additional visualization approach and thus additional developers' needs. The conceptual framework foresees application developers to provide fresh libraries and make them available to the public. So, the development threshold should be minimized in order to encourage more developers to provide a fresh visualization library. The threshold indicates how difficult it is to start using a system [Myers et al., 2000]. Basing the technical framework on a standard programming language and standard protocols lowers the threshold since developers do not have to learn new standards.

Maturity

An unstable technical framework would decrease the chance that developers use the framework, due to frustration with bugs and crashes. Hence, an underlying technology should provide a high level of stability. This lowers the framework's threshold and increases the probability of library extensions. The stability can be estimated, e.g., by the number of years the technology has matured and by checking whether there is continuous development of it going on.

Free Availability

Our final requirement in order to lower the threshold and to increase the probability of library extensions is free availability. If the application of the technical framework costs money, fewer developers will make use of it. As a consequence, the technical framework should not be based on other tools that cause costs.

Basing UbiVis on standards, maturing it and making it freely available not only lowers the threshold for developers who provide new libraries. It also lowers the threshold for the application of the framework.

5.2 Middleware

A lot of the requirements elaborated in Section 5.1 represent comprehensive challenges for the technical framework. For example, handling heterogeneous protocols, networking and supporting resource constrained devices comprise several subproblems. Each of these requirements necessitates a bunch of technical solutions. But the addressed problems are well known from the Internet of Things domain so that middleware which provides ready-made solutions might exist [Delicato et al., 2013]. In this section, we review a set of middleware projects for finding out which one can handle the addressed problems. In Section 5.2.7, we compare the middleware projects so that we can base the technical framework upon the best fitting one. The technical framework can then focus on the remaining, ubiAV related problems.

5.2.1 Amigo

Amigo is a middleware for ambient intelligence, developed in an EU project of the same name¹. As a key issue, it addresses the problem of turning heterogeneous devices into an integrated environment [Vallée et al., 2005]. For this, Amigo services are introduced [Thomson and Georgantas, 2007]. An Amigo service is a web service that encapsulates a device. It handles the proprietary device protocol. Amigo services are parsed by the INMIDIO middleware, a part of Amigo, and translated into a target protocol [Amigo Consortium, 2007]. The modular approach leads to a loosely coupled architecture.

The Amigo service concept allows for integration of heterogeneous devices in a distributed environment. Though, particular support for the arising networking issues is not provided. Resource constrained devices are addressed as well. However, the middleware itself is not intended to run on low resource machines.

Besides being based on web services, Amigo uses two major standard frameworks: .NET and OSGi. However, for some components, deployment is only supported for Linux [Amigo Consortium, 2007]. Windows, as standard OS, is not explicitly supported.

Code and documentation is comprehensively available for free download². However, after the end of the project, the code was no longer maintained; the last commit is dated February 2008. For the reason of not being maintained and not being up to date, we consider Amigo being not mature.

5.2.2 LinkSmart

LinkSmart has been developed under the lead of Fraunhofer FIT in the FP6 European project "Hydra"³. The resulting middleware is available as open source software⁴. After Hydra's end, LinkSmart has been matured and further enhanced by a team of 24 core developers and further co-workers. Developers are supported via online documentation, bug tracker, discussion forum and feature request form. LinkSmart is employed as core technology in more than ten EU projects.

LinkSmart combines a Service-oriented Architecture (SoA) and a Model Driven Architecture [Eisenhauer et al., 2009]. Every physical device can be controlled via web service, irrespective of its network interface technology. The service can either be embedded in the device or a proxy can host the service instead. LinkSmart's networking features interconnect the devices and thus provide interoperability on a syntactic level. Devices and services can be discovered using attribute based search mechanisms. The discovery resorts to XML based service descriptions [Jahn et al., 2009]. Device and network dependent details are hidden from the user. This provides interoperability on a semantic level.

The LinkSmart components are loosely coupled and can be distributed over different machines. So, there is no need for central servers or a main, complex application, which

¹<http://www.hitech-projects.com/euprojects/amigo/index.htm>

²<https://gforge.inria.fr/projects/amigo/>

³<http://www.hydramiddleware.eu/>

⁴<http://sourceforge.net/projects/linksmart/>

makes LinkSmart ideal for the ubiquitous computing domain.

The components are managed as OSGi bundles [OSGi Alliance, 2014]. The OSGi framework defines an API on top of a Java environment. It provides improved modularity and dynamic adding, update and deletion of bundles at runtime. Dependencies between bundles are automatically released. Alternatively, each LinkSmart component can be developed in the .NET framework standard.

5.2.3 GSN

The Global Sensor Network (GSN) middleware connects distributed sensor environments. The highly modular architecture allows users to deploy it on various hardware platforms [Aberer et al., 2006]. When deploying it to resource constrained devices, such as a PDA, only a subset of modules is used. In addition, the modular architecture allows users to exchange devices at runtime.

GSN employs wrappers and virtual sensors as abstraction layer to integrate heterogeneous sensors [Salehi, 2010]. A wrapper translates from the particular protocol into the standard GSN data model. As the higher level abstraction layer, virtual sensors represent configurable units in human readable declarative forms. Since this is not restricted to hardware sensors, also non-sensor ubicomp devices can be integrated as virtual sensors. However, the support is restricted to data input devices.

A GSN user only has to configure virtual sensors via XML files. One configuration option is the connection setting for mediating remote sensors. [Salehi, 2010] also describes a technique for dealing with complex network infrastructures comprising a NAT.

We regard GSN as being mature with the following considerations. The GSN project started in 2005 [GSN-Team, 2009]. Since then, the middleware is used as core technology in over ten EU or Swiss funded projects. It is available⁵ under GNU General Public License (GPL) version 2. In 2014, the source code has still been continuously updated. The GitHub project serves as central contact point for users. Meanwhile, about 100 wrappers are provided. Developers are supported by an online documentation, a mailing list and a bug tracking system.

The middleware requires only knowledge of standard languages, such as Java, XML and SQL.

5.2.4 Aspire

The development of the Aspire middleware began in a FP7 European project with the same name⁶. The resulting software is released^{7,8} under the free LGPL license.

Aspire is structured into components [Gama et al., 2011]. For example, a hardware abstraction layer encapsulates all code that is used to manage a particular device. This includes translating from proprietary vendor system calls into standardized calls.

⁵<https://github.com/LSIR/gsn>

⁶<http://www.fp7-aspire.eu/>

⁷<http://forge.ow2.org/projects/aspire>

⁸<svn://svn.forge.objectweb.org/svnroot/aspire/trunk>

This way Aspire harmonizes heterogeneous protocols and is open for existing and future protocols.

The framework builds on standards, such as EPCGlobal⁹, Java and OSGi. Components are usually distributed as OSGi bundles. This loose coupling concept allows developers to exchange components at runtime [Pedraza et al., 2010].

Aspire is designed as RFID middleware. General pervasive devices can also be handled through adapters. But only a few adapters for non-RFID sensors are available because the focus of Aspire remains on RFID tools [Donsez, 2010].

Typically, RFID sensor readings are distributed to other components for processing. Accordingly, Aspire provides connectors for communication between distributed components [Leontiadis et al., 2010]. Making use of the component based nature, parts of the middleware are available for clients. Even resource constrained machines are supported, such as RFID readers. However, connection management issues of a typical highly dynamic ubicomp environment are not addressed. For example, there are no routing features for changing node locations. Connection timeouts, e.g., due to firewalls, can emerge from the restriction to TCP and web service communication.

We consider Aspire being mature as 28 heterogeneous devices are supported¹⁰. Besides that, a comprehensive documentation available¹¹.

5.2.5 RTI Connex

RTI Connex is a commercial product line which connects distributed systems [RTI, 2012]. Its subcomponent, RTI Connex Micro software, can be installed on resource constrained machines with minimal memory, and a low-power or slow CPU. The RTI Connex Integrator component harmonizes disparate protocols using a SoA approach for letting heterogeneous systems work together.

Developers are well supported through a commercial hotline, a knowledge base and a forum. In the latter one¹², code samples can be downloaded for the most important programming languages C, C++, C# and Java. The middleware is constantly refined and as a commercial product, its maturity is expected to be very high.

The foundation for RTI Connex is the *Data-Distribution Service (DDS)* standard [Castellote, 2005]. It is a protocol for publish-subscribe messaging in distributed systems. Each component simply publishes message to a communication bus and interested components can subscribe to these types of messages [RTI, 2014]. This allows for loose coupling of components because publishers do not have to care about the receivers. Hence, no change in the publishers is needed when receivers are exchanged. At the same time, the approach manages the crucial networking issues.

⁹<http://www.gs1.org/gsm/kc>

¹⁰<http://wiki.aspire.ow2.org/xwiki/bin/view/Main/Readers>

¹¹<http://wiki.aspire.ow2.org/xwiki/bin/view/Main/Documentation>

¹²<http://community.rti.com/examples>

5.2.6 aWESoME

The Smart IHU project researches in the field of ambient intelligence [Stavropoulos et al., 2010]. The aWESoME middleware is developed as a part of it to deal with the inherent heterogeneous sensor environments. A web service abstraction layer provides universal access and data homogeneity [Stavropoulos et al., 2013]. It hides a driver layer that addresses the particular underlying technology. The resulting service oriented architecture allows for loose coupling of the components, i.e., single components can be exchanged at runtime. For this, a service broker component acts as the central registration engine for all web services.

Although the web service concepts allows the system to be distributed, arising networking problems are not focused by aWESoME. The components are intended to run separately from each other but at least in the same local network so that, e.g., routing issues do not arise.

aWESoME builds on standards, such as WSDL or Java. Its maturity is low since it has its roots in a university internal project. The free available software kit¹³ contains only basic functionality and basic documentation. Drivers exist for only 4 devices.

aWESoME's drivers integrate low resource devices, such as sensors. However, the middleware itself is not designed to run on those sensors. But [Stavropoulos et al., 2013] let the core services run on a board computer.

5.2.7 Discussion

Table 5.1 summarizes the reflections on the presented middleware solutions and aligns them with the requirements described in Section 5.1.

	HP	LC	DS	RCD	St	Ma	FA
Amigo	yes	yes	partly	partly	partly	low	yes
LinkSmart	yes	yes	yes	yes	yes	high	yes
GSN	partly	yes	yes	yes	yes	high	yes
Aspire	yes	yes	partly	yes	yes	high	yes
RTI Connex	yes	yes	yes	yes	yes	high	no
aWESoME	yes	yes	partly	partly	yes	low	yes

Table 5.1: Overview of which requirements are met by which middleware. The headlines refer to requirements as follows. HP - Heterogeneous Protocols. LC - Loose Coupling. DS - Distributed Structure. RCD - Resource Constrained Devices. St - Standards. Ma - Maturity. FA - Free Availability.

LinkSmart is the only middleware that meets all requirements outlined in Section 5.1. Some requirements are addressed in an especially optimal way. For instance, the OSGi architecture allows component exchange at runtime. Additionally, LinkSmart provides

¹³<http://rad.ihu.edu.gr/smartihu/storage/aWESoME%20Distributable.rar>

non-required but convenient features, such as cryptography, security, trust management and event management (cf. Section 5.3).

The design of GSN and Aspire focuses on integrating heterogeneous sensor networks. Due to their flexible architecture, non-sensor devices can be integrated as well. However, LinkSmart and the other middleware solutions might support general ubicomp devices better as those gadgets are explicitly targeted.

To summarize, we base our technical framework on LinkSmart because it provides optimal support for our requirements.

5.3 LinkSmart Details

In this section, we present details about the LinkSmart features and concepts that we use for our technical framework.

LinkSmart is intended to operate on hardware with limited computing or memory resources [Eisenhauer et al., 2010]. This increases the range of addressed platforms, especially in the ubiquitous computing domain where a lot of low performance devices like sensors and actuators appear. For this purpose, LinkSmart distinguishes between five different device classes.

D0 devices do not support IP communication and therefore cannot be directly LinkSmart-enabled. They need a D4 gateway device running a proxy application which translates from IP into the specific communication protocol of the D0 device. Sensors and other low level computing devices are common examples of D0 devices.

D1 devices can neither be LinkSmart-enabled because they are not powerful enough to host the LinkSmart middleware. However, since they provide IP support and can host web services, they can be contacted from the manager in charge, once they are discovered by a gateway. Typical examples of D1 devices are older mobile phones.

D2 devices are powerful enough to run middleware components but lack a communication possibility through the IP layer. They must use bridges to translate a specific communication protocol like Bluetooth or ZigBee to the IP layer. This is a very uncommon configuration but applies for some older PDAs.

D3 devices are devices like computers or smartphones which are powerful enough to host the LinkSmart middleware and communicate through IP.

Finally, D4 devices are a special case of D3 devices. These gateways host proxies which control a number of D0 and D1 devices. Hence, they need to be able to communicate to these D0 and D1 devices besides communicating with the LinkSmart network. Proxies encapsulate the communication to a D0 or D1 device in the way that other LinkSmart nodes think to communicate with a D0 or D1 device while they are in fact communicating with the proxy.

LinkSmart provides a set of managers which support different facets of networking issues of heterogeneous network nodes [Al-Akkad et al., 2009]. They relieve application developers from working on recurring, low level implementation tasks [Reiners et al., 2009].

Every communication between two distributed nodes is going through the Network Managers. A Network Manager applies a description based addressing theme on top of the IP layer and thus deals with recurring nodes due to changing locations or network connection errors. It additionally facilitates the communication beyond firewalls and NATs by applying a SOAP tunneling pattern [Milagro et al., 2009]. Hence, the Network Manager is taking the actual communication issues off the application developer's hands. It is related to the Discovery Manager, which automates and facilitates the discovery of LinkSmart-enabled devices.

The Crypto Manager provides cryptographic operations. It is used to protect messages between Network Managers by providing encryption and signatures [Hoffmann et al., 2007].

The Event Manager allows for exchanging messages between LinkSmart components in a topic based publish-subscribe mechanism [Pramudianto et al., 2013b]. Exceptionally, it is not entirely an OSGi bundle but split into a standalone core Event Manager and a client wrapper, which again is an OSGi bundle. Other LinkSmart components can communicate to the Event Manager through the wrapper like a normal LinkSmart component via OSGi calls.

The Trust Manager provides ways to subdivide nodes who know a secret into trusted or untrusted groups. Thanks to this manager, LinkSmart applications improve their robustness because malicious nodes can be prevented to cause harm [Vinkovits et al., 2012].

5.4 Applying LinkSmart to UbiVis

In this section, we describe how LinkSmart technology is applied to instantiate the UbiVis concepts. Together with the development view in Section 5.5, this builds the technical UbiVis framework. Figure 5.1 depicts the content of this section.

As we have discussed in Section 4.2.4, UbiVis's conceptual framework intends to integrate external devices, such as sensors and other event generators. These devices might provide proprietary software interfaces. Then, a developer would need to familiarize with a proprietary protocol instead of using a standard interface. Hence, the device's access and integration into the application can be difficult for her. The proxy concept of LinkSmart encapsulates the communication to such heterogeneous devices to standard communication on OSGi level. The idea is that developers provide proxies for individual device types to the community. Every LinkSmart user can integrate a proxy into her LinkSmart environment in order to communicate to the proxy when dealing with the device. In UbiVis, we use this technology for the external sensors and event generators. If a UbiVis application needs to integrate a sensor or external event generator, the developer integrates an existing proxy for the device or needs to implement a new one. The proxy is running in the same OSGi container as the main application and is performing technology-specific communication to the external device.

Visualization libraries that run external visualization devices work in the same way. The `VisualizationProxy`, which was introduced as UbiVis concept in Section 4.2.2, is

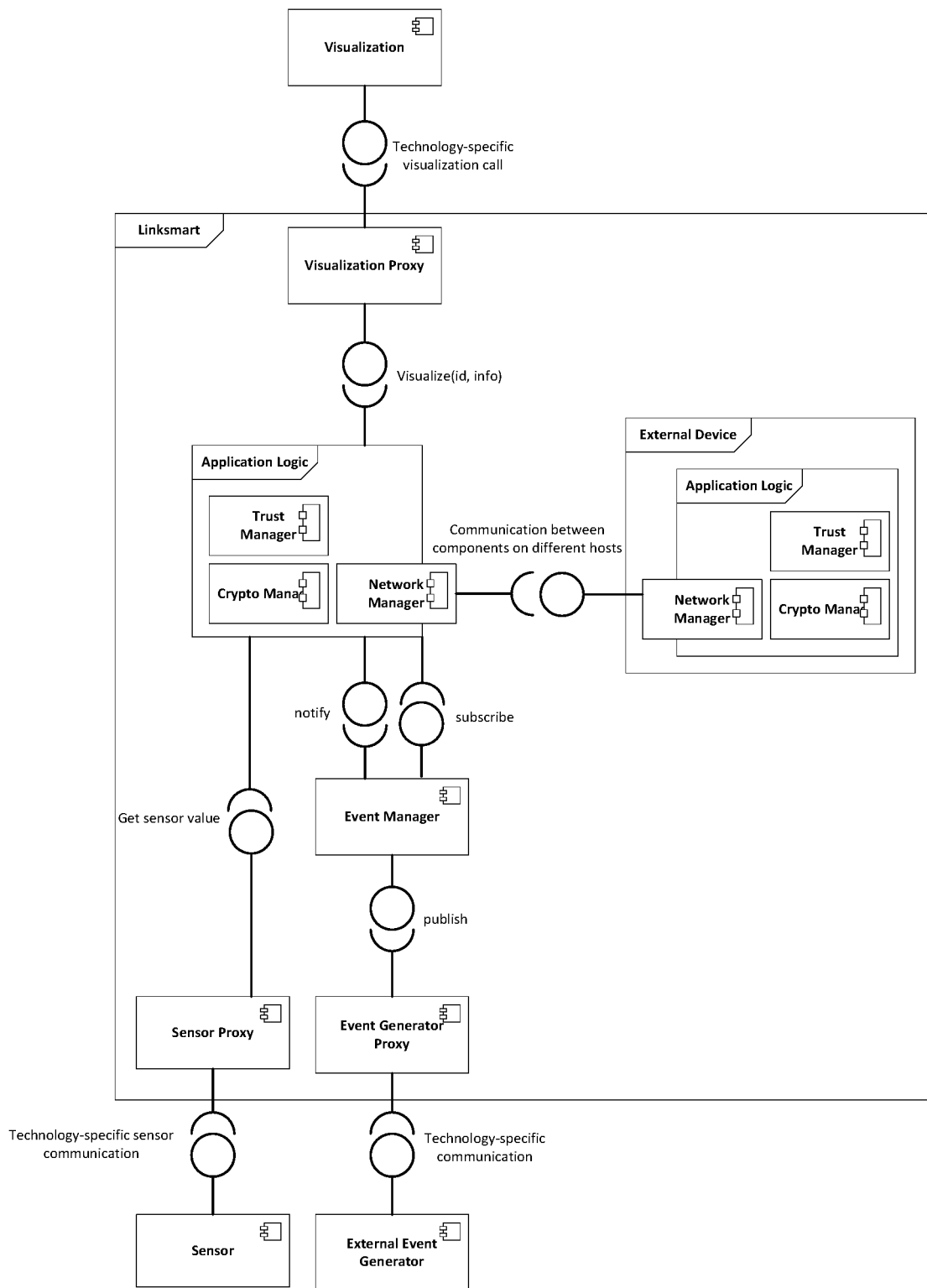


Figure 5.1: Instantiation of UbiVis concept using LinkSmart components

a LinkSmart proxy at once. The `VisualizationProxy` is the standardized interface of the visualization library for application logic. Both components run as OSGi bundle in the same container. The `VisualizationProxy` is then handling the technology-specific communication to a possible external visualization component.

UbiVis's technical framework should provide an event management mechanism for those applications which make use of event generators. Often, these can be sensors which raise an event every time the sensor value has changed to a certain degree. The application can then react on these changed conditions. The LinkSmart Event Manager provides this management functionality. An application can subscribe to certain event topics at the Event Manager. Every time an event generator publishes a new event to the Event Manager, the Event Manager notifies all subscribed components of this event.

UbiVis is designed for distributed systems. When components are located at different hosts, networking issues might arise. Developers have to handle possibly unknown network communication technologies. There is no standard way of addressing nodes. The network transmission may be interrupted by firewalls. The LinkSmart Network Manager overcomes all these problems. It provides a standard interface for network communication, applies a description based addressing scheme and allows communicating through firewalls. So, UbiVis components which run at different connected hosts communicate over Network Managers. For this, a Network Manager must run on each host. Network Manager communication is not needed between components at the same host. For this case, OSGi communication is faster.

LinkSmart's Crypto Manager and Trust Manager provide security for the technical UbiVis framework. After being started, they ensure that network communication is not intercepted during transfer or by malicious nodes. Both managers are usually started together with the Network Manager and LinkSmart core and thus, they run on each host.

5.4.1 Communication to Smartphones

LinkSmart's concepts cover the management of inter-component communication inside the LinkSmart network. However, it is likely that ubiAV applications utilize smartphones as visualization devices. At the moment, there is no stable LinkSmart version for smartphones available. So, a communication standard between LinkSmart and these common non-LinkSmart components is missing. Even if LinkSmart provides the proxy concept for integrating such components, having a default way of communication between the LinkSmart network and common smartphones would be convenient for application developers.

In order to cover both, push and pull communication, we propose an event-based communication standard for integrating smartphones into UbiVis. Event-based communication is natively push communication. A sender component raises an event, which is distributed to a receiver component. Hence, the sender is pushing information to the receiver. But it is also possible to perform pull communication via events. For this, the receiver must push a request event to the sender. The sender reacts by pushing back a reply event. Overall, information is pulled because the receiver initiates this transmission

by a request.

We need an event server that can run on different platforms since the UbiVis framework is not committed to a single platform. So, the eventing technology should also support different smartphone platforms. The *Really Small Message Broker (RSMB)* [Craggs, 2014] meets these requirements. The server application is available for Windows, Mac OS and different Linux distributions. It is using the *MQ Telemetry Transport (MQTT)* protocol [IBM, 2014]. MQTT is a standard for lightweight machine-to-machine communication in publish/subscribe messaging. It is "ideal for mobile applications because of its small size, low power usage, minimized data packets, and efficient distribution of information to one or many receivers". That is why many libraries for different smartphone platforms exist, which meets our requirement.

MQTT designs byte arrays as data exchange format. So, every serializable data format and object can be exchanged.

Figure 5.2 gives the overview of classes, interfaces and components that participate in the MQTT-based communication.

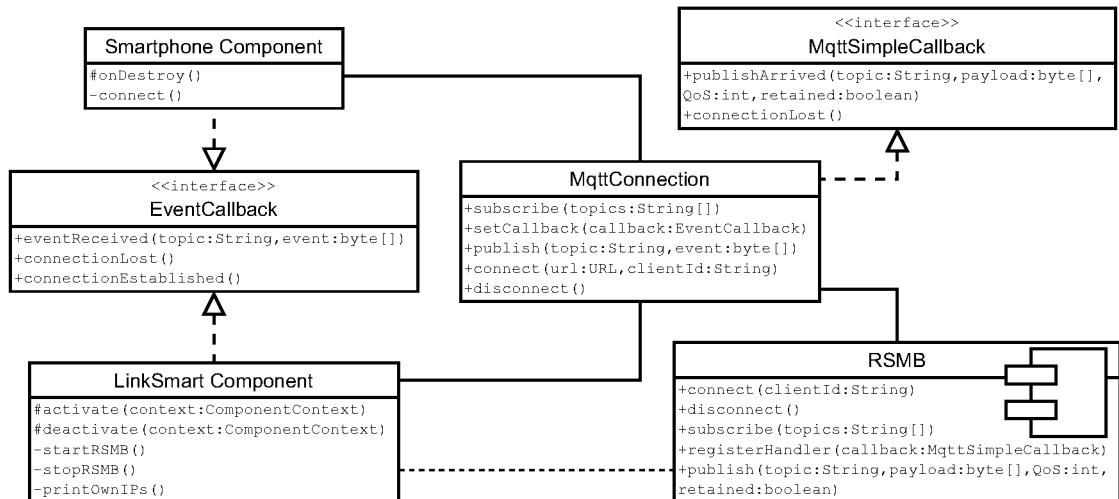


Figure 5.2: The class diagram shows involved components for the communication between LinkSmart and smartphones.

MQTT libraries require a client to implement the interface `MqttSimpleCallback`, which manages the communication between a MQTT-enabled server and its clients. This interface provides the method `connectionLost`, which is called when the connection between client and server is interrupted. The second method `publishArrived` is called when clients receive an event.

We design the `MqttConnection` class to implement the `MqttSimpleCallback` interface. This class acts as connection adapter to the RSMB server. It provides the methods `connect`, `disconnect`, `subscribe` and `publish`, which call the corresponding methods

on the server. The `MqttConnection` class is used in OSGi as well as on the smartphone.

Components which communicate to the RSMB server via `MqttConnection` must implement the `EventCallback` interface. This interface is the callback for `MqttConnection` to inform clients about an established or lost server connection or about received events. For this, components which implement `EventCallback` register themselves as callback at `MqttConnection` by calling `setCallback`.

RSMB is available for different platforms and is started as normal application. For better convenience, we recommend to include the start of this application in the start of the OSGi framework as we describe in the next paragraph. This connection between RSMB and its executing LinkSmart component is indicated by the dotted line in Figure 5.2.

In the following, we describe the process flows of the MQTT communication.

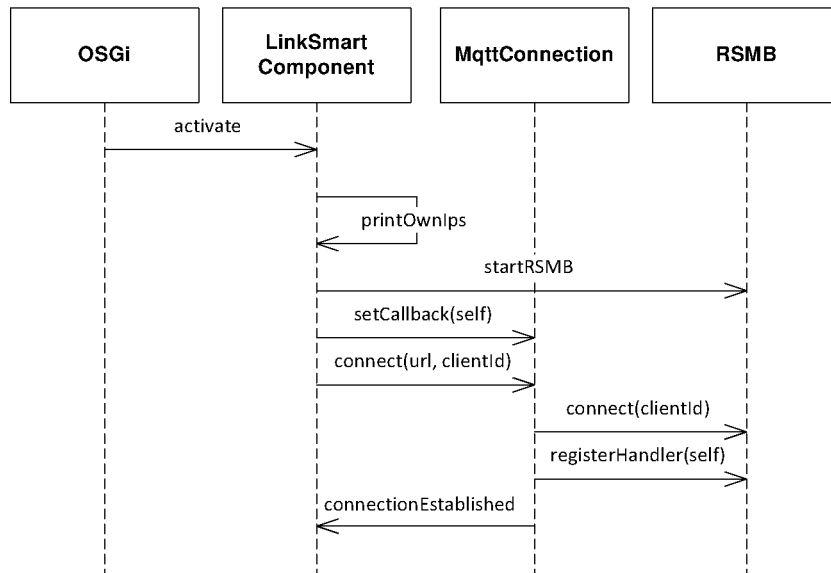


Figure 5.3: Process flow: Start RSMB and connect

Figure 5.3 summarizes how the RSMB application is started and the LinkSmart component connects to it. When a LinkSmart component is activated by OSGi, it first prints the IP addresses of the host. The IP address is later needed by the client. The user can read the host's IP address and enter it at the client side. After this step, the LinkSmart unit starts the RSMB server OS-dependently. It then registers itself at `MqttConnection` so that it can be informed about a successful connection establishment. Next, it calls `connect`, providing the RSMB's URL and an ID for registration at the server. `MqttConnection` tries to connect to the server. If there is no connection timeout, `MqttConnection` has to register itself as handler at the server. This step makes sure, that it receives notification events. Finally, `MqttConnection` informs the LinkSmart

component about a successful connection buildup by calling the method `connectionEstablished`.

Stopping RSMB works analogously to starting it. When the LinkSmart component is deactivated by OSGi, it shuts down the RSMB application. This is done OS-dependently, e.g., by killing the particular process. Figure 5.4 depicts this procedure.

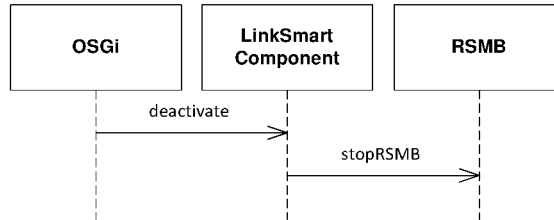


Figure 5.4: Process flow: Stop RSMB

The smartphone connects to RSMB similar to the LinkSmart part, but without having to care about starting the RSMB application. This process is shown in Figure 5.5. The connection must be initialized by a user input. The `connect` method must deliver the server URL which was requested from user input before. The other steps for connecting the smartphone component to RSMB are the same as for the LinkSmart component.

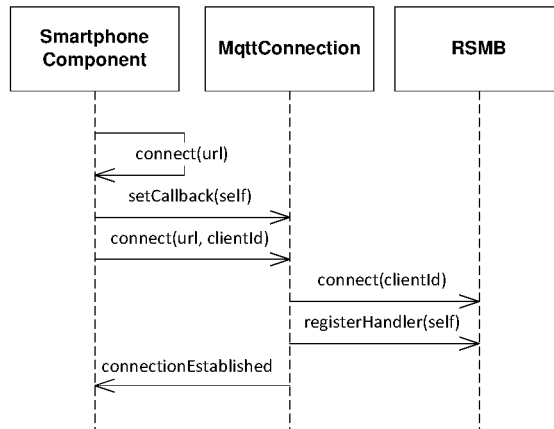


Figure 5.5: Process flow: Connect smartphone to RSMB

Figure 5.6 illustrates how smartphone clients disconnect from RSMB. When the app is exited, the smartphone component uses the `MqttConnection` adapter to disconnect from the server.

The connection to RSMB can also be interrupted by external reasons, e.g., a network breakdown. Figure 5.7 shows how this is handled. The RSMB library informs the registered `MqttSimpleCallback` unit of the interruption. This component then no-

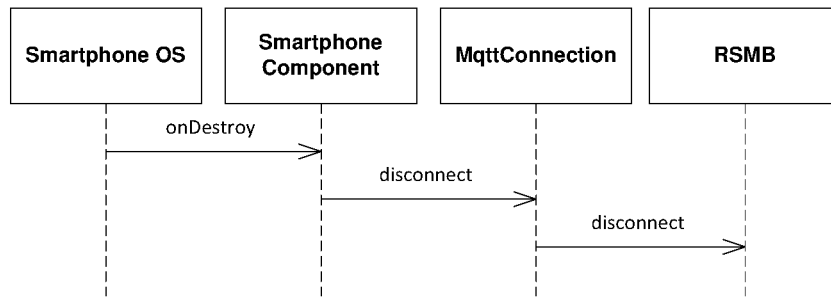


Figure 5.6: Process flow: Disconnect smartphone from RSMB

tifies all event callbacks by invoking `connectionLost`. The process flow is the same for LinkSmart and smartphone callbacks as both implement the same interfaces.

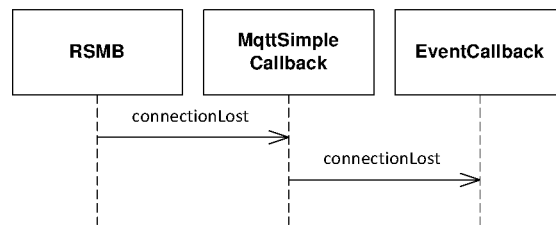


Figure 5.7: Process flow: Connection to RSMB is lost

Figure 5.8 illustrates how a LinkSmart component publishes an event, which is delivered to a smartphone component that has subscribed to the event topic before. `Subscribe`, `publish` and notification are again translated by `MqttConnection`. `publishArrived` and `eventReceived` take over the notification task. `eventReceived` can be called at the smartphone part because it has registered itself as callback before (cf. Figure 5.5). Publishing from the smartphone to LinkSmart works analogously.

5.5 Development View

In this section, we provide a set of conventions and development rules, which shall answer frequent questions during the implementation process. This assures a robust development stage. At the same time, it finalizes the specification procedure of [Rozanski and Woods, 2011] (cf. Section 4.2.1).

The following listing describes our decisions for the Development view. All decisions are based on the principle to make development as easy as possible for an application developer. In this way, we place emphasis on standard software and processes expecting them to be easy to apply and to be known by many application developers. System outputs shall be human-readable and the general language is English in order to address an international audience. We also define some structures as simple as possible.

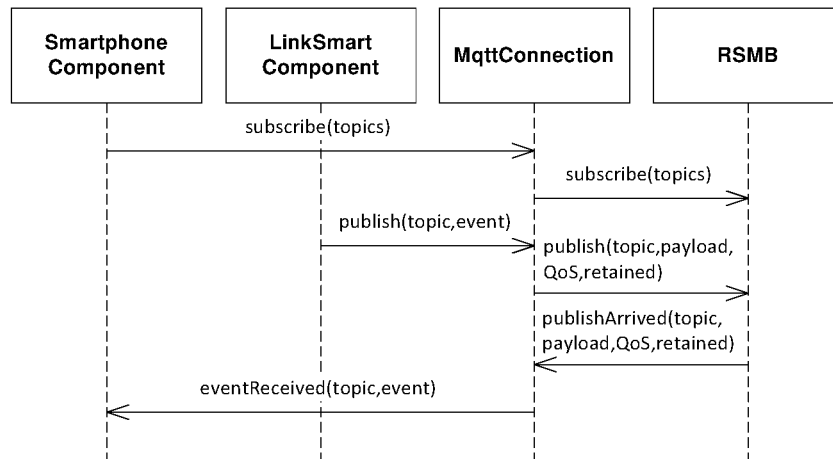


Figure 5.8: Process flow: Subscribe, publish, notify

- Standard Software Components

1. Connection

- LinkSmart Network Manager for connecting components on different hosts
- LinkSmart proxies for integrating external devices
- OSGi declarative services for connecting components on same host
- RSMB for messaging between visualization proxy and Android-based visualizations

2. Logging

- Log4j is used in a standard way as described in <http://logging.apache.org/log4j/1.2/>

3. Configuration

- Configuration file uses XML structure
- XStream library for reading and converting XML files

4. Repository

- SVN is used in a standard way as described in <http://subversion.apache.org/>

5. Events

- LinkSmart Event Manager as event broker

6. Security

- LinkSmart Crypto Manager for encrypting and decrypting communication

- LinkSmart Trust Manager for managing trust levels of nodes
- Standard Design
 1. Connection
 - The server software components are implemented according to the OSGi specification
 - Include RSMB topics in model
 2. Logging
 - Used log levels are fatal, error, warn, info and debug
 3. Configuration
 - Configuration file is part of a visualization proxy bundle
 - Configuration filename and path is <bundle>/configuration/config.xml
 - Additionally needed files are stored in subfolders of <bundle>/configuration/
 4. Repository
 - The model is kept in the UbiAV folder under /
 - A library gets two own folders under /libraries/ . The names of these folders are <library>Proxy and <library>
 5. Events
 - Model events as classes
 6. Compatibility
 - Images are stored as byte arrays
- Common Processing Required
 1. Connection
 - Each software component is implemented as OSGi bundle
 - Each model component is implemented as OSGi bundle
 - Start RSMB from visualization proxy
 - Stop broker process from visualization proxy
 2. Logging
 - All components should log
 - Logged messages are human-readable
 3. Configuration
 - Use XStream alias for replacing classnames in config.xml by human-readable attributes
 4. Repository
 - Make sure that the trunk is always compilable

- Stick to Apache’s subversion best practices: <http://svn.apache.org/repos/asf/subversion/trunk/doc/user/svn-best-practices.html>

5. Events

- Use the publish-subscribe pattern
- Subscribe to events by topic composed of package name + event class name

6. Language

- English is used for variable names and all other implementation text
- English is used for textual output
- Textual input is expected in English
- US English is used for locale-sensitive information like dates, times or currency strings

5.6 Conclusion

The technical framework prepares UbiVis for practical application by providing a technical development environment which covers open implementation issues. It provides a technique for specifying standard interfaces so that loosely coupled components can communicate with each other. The event based message exchange is a good practice for such a loose coupling, as we have shown in Section 2.2. The technical framework covers discovery, connection setup and communication for nodes in a distributed system. Software components may be distributed over different devices.

As we analyzed in Section 3.1, ubiquitous annotation visualization can be implemented through a broad range of heterogeneous technology. The main benefit of basing the technical framework on LinkSmart is to have a standard interface concept which unifies these heterogeneous technologies. This makes UbiVis applicable for a comprehensive set of ubiAV technologies. LinkSmart especially supports limited devices, a recurring factor in ubiquitous annotation visualization. At the same time, it supports the rapid prototyping approach through the unified interface, which also covers heterogeneous network interface technology.

LinkSmart presents an additional range of useful features. It handles changing locations of nodes and communication through firewalls. Security is provided through encrypted messaging. Since there is no stable smartphone version of LinkSmart we provide a standard approach for the communication of LinkSmart and smartphone components. Thus, UbiVis’s applicability to this common ubiAV device is ensured.

A development view provides the necessary constraints and standards which help application developers to implement applications with the support of the UbiVis framework. The rules are easy to apply and address an international audience.

By specifying the concrete technology that can be used for instantiating the UbiVis concepts and by particularizing implementation rules, we created the technical environment for applying the framework. This allows us to evaluate the framework by letting

application developers implement real applications. In addition, we can create visualization libraries in a concrete technical framework.

Chapter 6

UbiVis Libraries

In the previous chapters, we developed the conceptual and technical part of the UbiVis framework, which allows developers to exchange ubiquitous annotation visualization approaches without having to change application logic. A standard procedure how to apply this technique reduces development effort and complexity of rapid prototyping.

The second major feature of the UbiVis framework for reducing development effort is to provide development support for controlling ubiquitous annotation visualization technologies. A library for every ubiquitous annotation visualization technology will take over this task. An application developer only needs to configure the library for her particular application instead of having to develop it by herself.

There are too many possibilities for ubiquitous annotation visualization that a single provider could supply libraries for all of them. Additionally, future approaches cannot be supported right now. In order to overcome this problem, every developer is able to extend the set of libraries by creating a new library as it is explained in Section 4.4. Nevertheless, we need an initial set of libraries for starting to build applications with UbiVis.

In this chapter, the initial libraries which are provided in combination with the UbiVis framework are presented. For each library, the functional principle and the way of configuration is explained. Every library supports a ubiquitous augmentation technology and is compliant to the UbiVis framework specification. As we pointed out in Section 3.5, the initial libraries should support each class of visualization approaches identified in Table 3.1. Table 6.1 shows which library supports which visualization class. UbiLens is a special case. One configuration option switches the mode for identifying physical objects. Depending on this setting, UbiLens can be location-aware or location-unaware.

6.1 UbiLens

UbiLens covers an approach which is commonly used for ubiquitous annotation visualization: Location-aware augmented reality as see-through version on a smartphone. This is, for example, applied by Wikitude World Browser [Wikitude GmbH, 2014] or

	Real World	Virtual World
Location-Aware	UbiLens UbiLight	UbiMap
Location-Unaware	UbiLens	UbiBoard

Table 6.1: The initial UbiVis libraries support each ubiquitous annotation visualization class.

Layar [Layar, 2014].

The smartphone captures video images of the environment via its integrated camera and displays it on its screen. This way, the user has the feeling of seeing through the phone. The video image can be overlaid by graphical objects. These objects adapt to the current position of the phone so that a particular graphical object is also placed above a corresponding physical object (cf. Figure 6.1). This way, it acts as annotation for the physical object. Some works refer to this technique as *Magic Lens* [Schöning et al., 2006]. The idea is that the smartphone is held like a lens over an object in order to reveal additional information, which cannot be seen without the tool. That is why we call this library UbiLens.

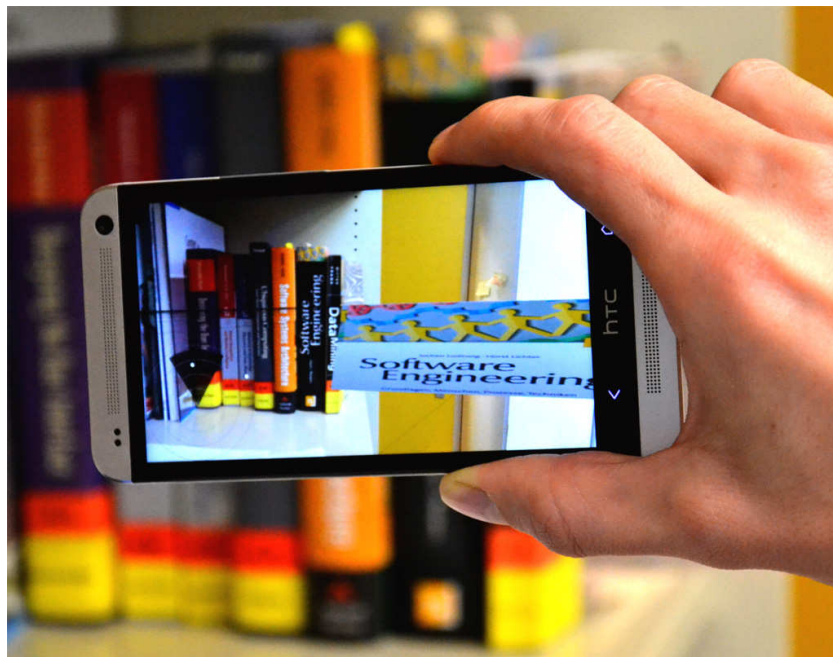


Figure 6.1: UbiLens

The approach is popular for AR because commonly available smartphones provide all necessary equipment. A camera is needed for capturing the video content. Present-

days smartphones' screens are large enough for displaying the video and virtual overlays. Calculating the location and orientation of the smartphone is often based on GPS and compass sensor measurements.

UbiLens has two operation modes, which are described in detail in Section 6.1.1. One mode requires modeling the annotated objects' locations while for the other mode, the objects' locations do not have to be modeled. The application developer who is applying UbiLens has to enable either the one mode or the other mode in the configuration (cf. Section 6.1.2). Hence, depending on this setting, UbiLens is either a location-aware or a location-unaware ubiquitous annotation visualization library.

6.1.1 Features

The UbiLens visualization component is an application for Android smartphones. It provides access to the camera's video image and displays it on the phone's screen. Additionally, UbiLens generates graphical objects of the annotations. The annotations are displayed as overlays on the camera image. For each video frame, the annotations are placed at the correct location on the screen so that an annotation is always displayed above the corresponding physical object.

UbiLens provides two alternatives for placing the annotations on the screen. Calculating the object's positions from location and rotation values or identifying the object via image recognition. Calculating the positions is a little more reliable than the image recognition. Objects are sometimes not recognized although they are in the analyzed image. However, the image recognition also works indoors while the calculation alternative is designed to work outdoors where it can make use of the smartphone's GPS sensor. Nevertheless, the calculation alternative can also be used indoors with a static location value set.

In the first alternative, UbiLens calculates the physical objects' positions on the screen from the objects' location, the smartphone's location and its rotation. The smartphone's locations can be set to a fixed value or it can be determined via GPS.

An annotation is represented by a graphical 3D object. The viewing angle of the object adapts to the angle between user and physical object. This way, the annotation is more realistically integrated in the scene because the user can walk around the object and see it from different viewpoints. Since UbiLens does not know the physical object's orientation, the graphical object is by default oriented on a north-south axis.

The concrete instantiation of each graphical object is a cube because each facet of a cube can be viewed equally well. The annotation data is placed as texture on each facet of the cube. This way, the annotation can always be read, independently from the viewing angle. If the annotation consists of an image, this image is used as texture. Textual or numeric annotations are printed as text on plain background on the facets. Background and text color can be adapted to the annotation.

UbiLens provides the user two tools for orientation support. A thin grid is displayed as continuous overlay. It represents the earth's surface, thus helping the user to hold the phone horizontally. The grid's lines also indicate north, west, south and east so that

a user has a clue about the current rotation. The second continuous overlay is a radar element at the lower left part of the screen. It shows the surroundings of the smartphone as circle. The current field of view is highlighted. Physical objects are indicated as dots in the surrounding, so that the user knows where annotated objects which are currently out of view are located. This can be especially helpful for orientation if no object is inside the current field of view. Objects which are in reality farer away from the smartphone are displayed farer away from the center of the surroundings circle than closer objects. This way, users have another way of quickly accessing objects' distances, which might assist with getting oriented.

For most of the tasks, UbiLens is using the OutdoorAR library of HIT Lab NZ [Billinghurst, 2014]. The library is freely available and provides various sub-libraries for creating augmented reality applications on Android.

In the second alternative, UbiLens tries to detect physical objects via image recognition in the camera image. Each camera frame is analyzed by an image recognition component, which tries to find a given image inside the frame. For this, it needs one or more photos of the objects that shall be detected. This pre-stored set of photos is then compared to the current video frame. The more photos of a physical object are supplied, the higher is the chance of detection.

The annotation visualization for this alternative differs from the first alternative. Since we often only have one front view picture of the annotated object, we find it more natural not to use 3D objects for this alternative. Instead, the annotation is placed on a plain 2D area overlay on the video frame.

UbiLens makes use of the image recognition algorithm that is described by [Henze et al., 2008]. It uses a FAST corner detection, stripped down SIFT descriptors and a scalable vocabulary tree. These features make it fast and robust to distortion so that objects can be detected in a different viewing angle than on the picture. Besides detecting which image is found, the algorithm returns the image's position inside the video frame through X and Y image coordinates.

The annotation visualization design of the second alternative is slightly different than in the first alternative. The annotation image, text or number is displayed on a semi-transparent overlay on a fixed position of the screen. In order to indicate to which object in the frame the annotation belongs, a green circle is placed on top of the object. Because of the semi-transparency of the annotation, the circle indicator still points to the annotated object if that object is behind the annotation.

6.1.2 Configuration Options

UbiLens requires mapping physical objects to technology-specific attributes as well as mapping annotations. Also, two global configuration options are needed. All possible configuration options are summarized in Figure 6.2.

At first, a UbiLens user has to decide on an alternative for placing the annotations. By setting the global parameter `Image Recognition` true, UbiLens uses image recognition. Setting it false makes UbiLens calculate the positions from location and rotation

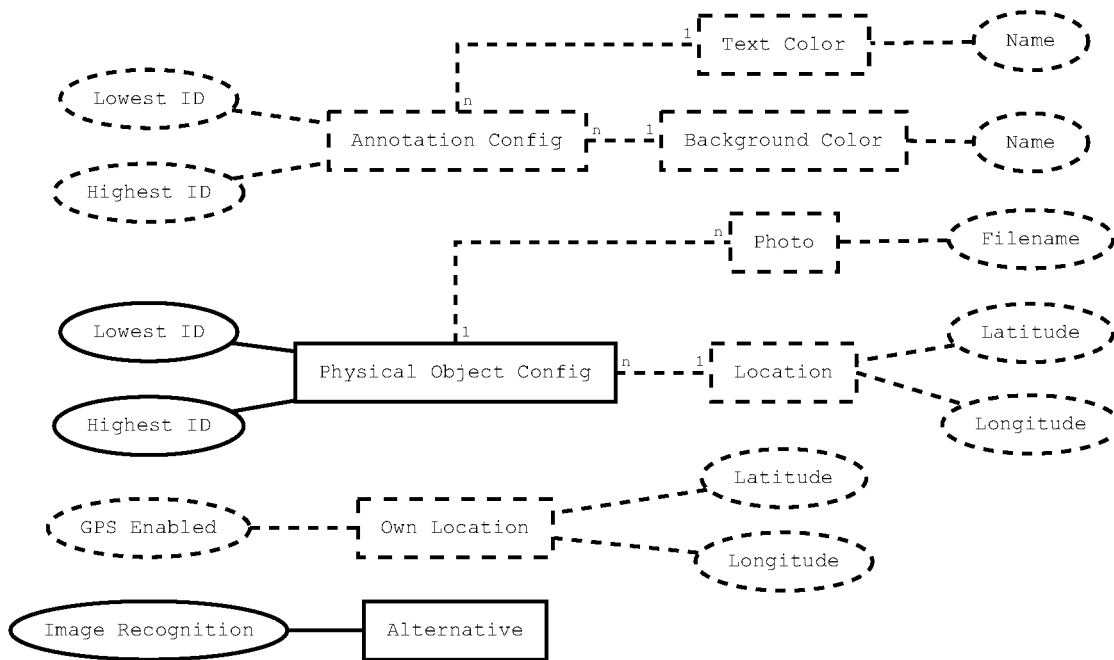


Figure 6.2: Configuration options of UbiLens

values. So, only one alternative can be enabled at a time. This setting determines which of the following options have to be filled. Those that are filled only for one alternative are marked dashed in Figure 6.2.

For the calculation alternative, the graphic's position is calculated from the physical object's real world location and the device's location and rotation value. Hence, UbiLens needs to know the real world location of each physical object. Therefore, physical objects are mapped to locations. A location consists of a latitude and longitude value in decimal WGS84 format [National Imagery and Mapping Agency, 2000].

The way how the device's location is acquired can be configured as global option. It is acquired by GPS if the parameter `GPS Enabled` is set true. Otherwise, a static location is used. This static location must be set in the configuration through latitude and longitude in the same format as the physical objects' locations.

The background color and text color of an annotation cube is customizable in UbiLens. So, annotation IDs are mapped to a background color and a text color attribute. The tone can be either indicated by a name which can be computed by Android¹. Or, in the format `"#RRGGBB"`. The `RR` has to be replaced by the 2 byte hex code of the color's red part. `GG` and `BB` must be replaced analogously for the green and blue part.

For the image recognition alternative, a photo image of a physical object must be found in the current video frame. Thus, UbiLens needs a photo of each physical object. Therefore, physical objects are mapped to photos. A photo is specified by a filename

¹[http://developer.android.com/reference/android/graphics/Color.html#parseColor\(java.lang.String\)](http://developer.android.com/reference/android/graphics/Color.html#parseColor(java.lang.String))

which points to an image file in the configuration folder. `.png` and `.jpg` files are supported. It is possible to specify more than one photo for a physical object.

6.2 UbiMap

UbiMap makes use of a classical approach for referring to close-by physical objects in a smartphone application. A map of the environment is presented on the phone. Annotated objects are made interactive at those positions on the map which refer to their real world location. Cyberguide is an example application employing this approach [Abowd et al., 1997].

A dynamic street map of the smartphone's surroundings is displayed on the screen (cf. Figure 6.3). The app can also be used indoors. For this, a building's floor plan can be overlaid over the street map (cf. Figure 6.4). Annotated objects are indicated as colored icons on the map. Clicking an icon opens a popup window in which the annotation is displayed. The annotation can consist of a text, a numeric value, an image or a combination of them.

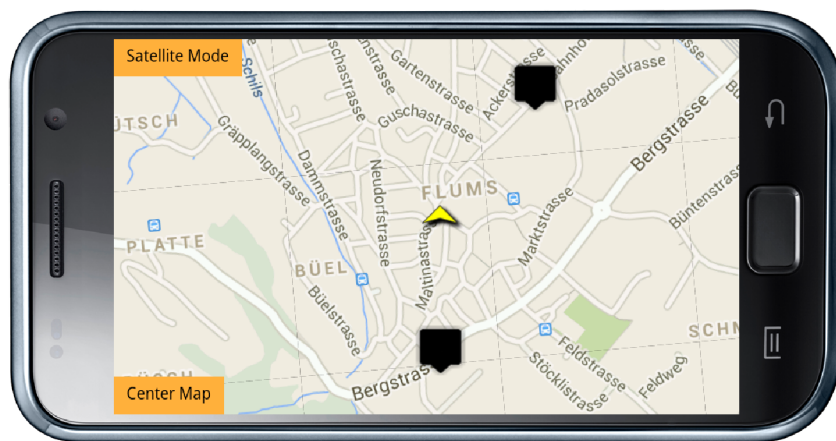


Figure 6.3: UbiMap outdoor

There are several possibilities for map interaction. Firstly, the map rotates as the phone rotates in a way that the line of view is always up. This is intended to ease the mental mapping from the map to the real world. Secondly, it is possible to pan the map as it is well known from digital mapping applications like Google Maps². A button click centers the map view in case the user has panned too far. The app uses Google Maps data. A second button allows switching between map view and satellite view.

As main advantage, this kind of application is used like a traditional augmented city map, which is enhanced by interactive elements. It is easy to handle for users because they know the concept from handling traditional paper maps.

²<http://maps.google.com>

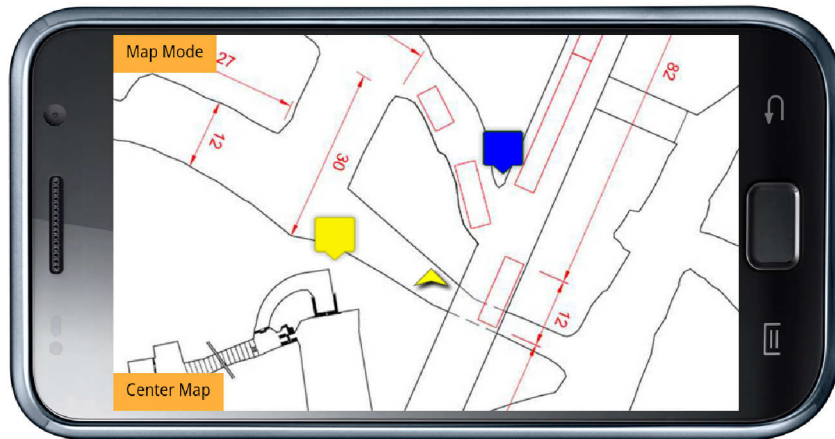


Figure 6.4: UbiMap indoor

6.2.1 Features

Equal to UbiLens, the UbiMap visualization component is an application for Android smartphones. It provides basic mapping features. Firstly, that is access and display of Google Maps data. Secondly, a button is added which determines whether satellite data or map data shall be displayed. Thirdly, basic map interaction features are provided. The map can be panned by touching and moving it with a finger. UbiMap contains a button which pans the map view back to the location of the smartphone.

UbiMap accesses the phone's location in the same way as UbiLens. Either, the smartphone's GPS location provider is requested. Or, the position can be set to a static value.

Several overlays are added to the map. Firstly, the own position is indicated by an arrow icon. Secondly, the annotation icons are added as map overlay at the locations of the annotated objects. The icon's color is adjustable. Thirdly, a building's floor plan in the form of an image can be overlaid at a customizable location.

The whole view is rotated by UbiMap. For this, UbiMap is accessing the smartphone's compass sensor. The default orientation is considered for this. This consideration prepares the application for smartphones, which usually run in portrait mode, and for tablets, which usually run in landscape mode. A map rotation is executed every time the rotation value has changed more than two degrees compared to the last map rotation. Executing more rotation events would make the view trembling and hard to read. A map rotation lets the whole canvas rotate. Then, the location and annotation icons are rotated back. This way, the icons keep facing up for better readability. UbiMap also provides the option to disable map rotation. In this case, the map is always shown with north up.

Next, UbiMap handles the clicks on the annotation icons. When the screen is clicked, UbiMap calculates whether an overlay icon was hit. For this, the map rotation is filtered out. If an icon was hit, UbiMap accesses the data of the corresponding annotation. A

popup window is created in which the annotation data is displayed. An OK button is added for closing the window again.

Finally, a usability features is provided by UbiMap. The map rotation feature only works when Android's auto rotation feature is switched off. So, UbiMap remembers the system's auto rotation setting at resolve, fixes the application to run in landscape mode and restores the original auto rotation setting at pause.

6.2.2 Configuration Options

UbiMap requires mapping physical objects and annotations to technology-specific attributes. There are also two global configuration options. The configuration options are summarized in Figure 6.5. Mandatory configuration parameters are solid, optional ones are dashed.

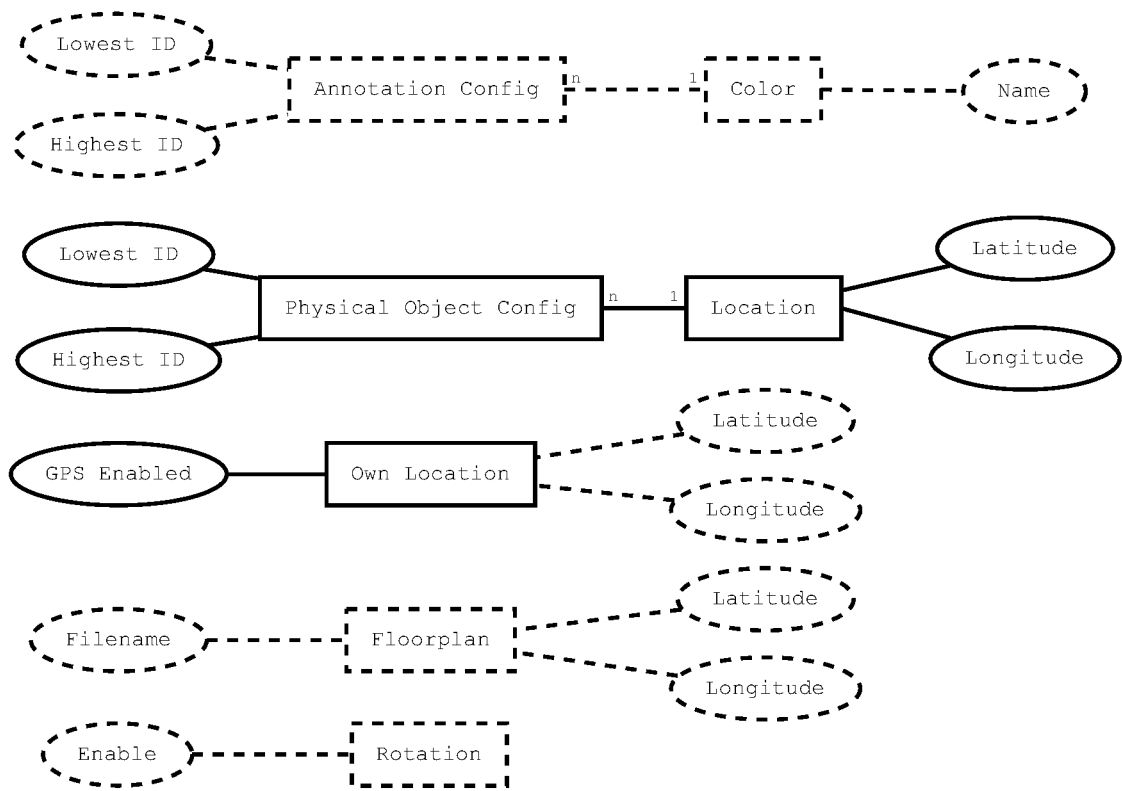


Figure 6.5: Configuration options of UbiMap

The alignment of the map and its overlays is based on real world coordinates. Thus, UbiMap needs to know the real world coordinates of every overlay item. An icon is overlaid on the map for every annotated physical object. So, physical objects must be mapped to locations. For these and all other position attributes of UbiMap, we use the same format as in UbiLens. A location consists of a latitude and longitude value in

decimal WGS84 format [National Imagery and Mapping Agency, 2000].

In order to set a hue for an icon, the annotation must be mapped to it. The tone is specified by the corresponding element where **white**, **red**, **green**, **blue**, **cyan**, **magenta** and **yellow** are supported. If no color is specified, a black icon is used.

For displaying the own location, UbiMap needs to know the location of the device. This global configuration option is equal to UbiLens. It is acquired by GPS if the parameter **GPS Enabled** is set true. Otherwise, a static position must be configured.

A building's floor plan is the last overlay which can be set in the configuration options. It consists of a filename and a location attribute. The filename points to an image file in the configuration folder. The location specifies the real world coordinate of the upper left pixel of the image. Since most buildings are not aligned to a degree of latitude, the image files must usually be rotated. The background must be set transparent so that the map can be seen around the floor plan. Therefore, it is recommended to use image file formats which support transparency. If the filename option is left empty, no indoor overlay is set.

Finally, there is an option to disable map rotation. If the boolean value for rotation is set to false, the map is always shown north up. Not specifying this option leads to the same result as setting it to true because the rotating map shall be the default behavior of UbiMap.

6.3 UbiLight

UbiLight covers an ambient display approach of ubiAV systems, in which a classical screen has disappeared. Instead, information is communicated by colored light patterns. Similar systems which transmit information by light patterns are Waterlamps [Dahley et al., 1998] and Power-Aware Cord [Gustafsson and Gyllenswård, 2005]. However, both are location-unaware because the location of the annotated object is not modeled. Comparable location-aware systems are projector-based projects like Everywhere Displays [Pinhanez, 2001] and Fluid Beam [Spasova, 2004], which also enhance objects in a room with light.

A flexible strip which consists of several LED elements is UbiLight's visualization hardware. Each element can be controlled to emit light in a different color. Physical objects are placed in front the strip. The area of the strip which is located behind a physical object is then illuminated in a homogeneous color. Different colors shall indicate different information about the physical object in front.

The setup can be made more ambient by placing the strip in a way that the colored light is reflected by an adjacent, wide object like a wall. For example, the strip can be placed along the edge of a piece of furniture or the floor. This way, the LED strip itself is hidden but the information transmission via illumination remains (cf. Figure 6.6).

UbiLight provides a rather unobtrusive method of performing ubiquitous annotation visualization. Thus, it is suitable for transmitting low priority information in the periphery of a user's awareness. Compared to the other initial libraries, it offers a different way of transmitting information. Instead of visualizing text or images, UbiLight's



Figure 6.6: UbiLight

possibilities are restricted to different colors of light.

6.3.1 Features

The UbiLight visualization component is a desktop application. It provides access to the LED strip. The strip is driven by an Arduino board. The Arduino script expects string commands from the serial port for controlling the actual strip. UbiLight provides access to that Arduino board via serial port communication.

The strip's light configuration is initialized and managed by UbiLight. There are 160 independent LED elements. Each element can be set to a color. UbiLight expects a range of elements and a color for controlling them. It builds up the particular command strings and sends them through the serial port. In addition, UbiLight manages to switch a specified range of elements off again.

6.3.2 Configuration Options

UbiLight requires mapping physical objects to technology-specific attributes as well as mapping annotations. Also one global configuration option is requested. The configuration is summarized in Figure 6.7. All of its configuration parameters are mandatory.

UbiLight illuminates a part of the LED strip in order to annotate a physical object in front of it. Hence, it needs to know where the physical object is located. So, each physical object must be mapped to a location attribute. The location is indicated by two integers. They represent the starting and ending LED element on the strip. Comprehending the strip as a one-dimensional axis of coordinate, we name these integers **Abscissa Start** and **Abscissa End**.

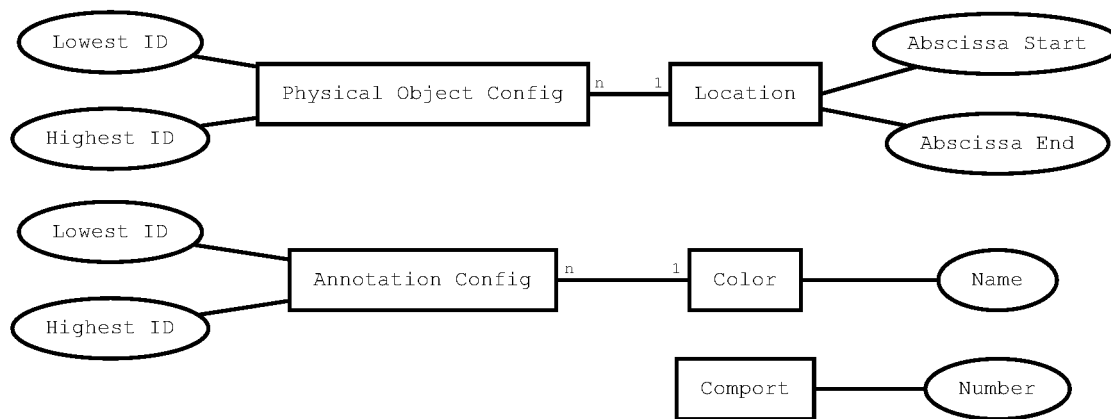


Figure 6.7: Configuration options of UbiLight

The LED strip cannot display annotation data in a traditional way. It is not capable of visualizing texts and images. Hence, the `Annotation` object must be mapped to hues which the strip can compute. So, for UbiLight, the data attributes of `Annotation` objects are neglected. Instead, the `annotationId` attribute is used for finding the corresponding color. Hence, ranges of `annotationIds` must be mapped to colors in the configuration file. The tones are specified by name. The following color names are supported: `white`, `red`, `green`, `blue`, `cyan`, `magenta` and `yellow`.

Both mappings are also used for switching lights off again. For this, an eighth "color" named `off` can be specified. This is handled in the same way as the other colors. So, when an application developer wants to switch a range of lights off again, she triggers a normal `visualize(visualizationTuple:VisualizationTuple)` call. The physical object's id of `VisualizationTuple` must then match to the range of LEDs that she wants to switch off. The `annotationId` must map to the color `off`.

Finally, the number of the COM port to which the LED strip is connected must be specified as global configuration option.

6.4 UbiBoard

UbiBoard is a system for large screens which are situated in public areas and display content related to their environment. In ubiquitous computing research, this is a common way of utilizing large screens. [O'Hara et al., 2011] present a set of these systems and call them *public and situated displays*. Examples for situated displays which show content about objects in their environment are Hermes Door Displays [Cheverst et al., 2003] or NESSIE [Prinz, 1999]. However, these systems make use of medium-scale monitors respectively projectors instead of large-scale screens.

UbiBoard utilizes a stationary public indoor display. The screen is subdivided into several virtual tiles. Each tile can hold information about a physical object in the

environment of the screen (cf. Figure 6.8). The physical object is represented by an image and its name. The annotation fills up the rest of the tile.



Figure 6.8: UbiBoard

The way this system is used reminds of classical whiteboards. A large presentation surface is placed in a public area and used for showing information that might be of interest for bystanders. For this reason, we call the system UbiBoard.

6.4.1 Features

The UbiBoard visualization component is a desktop application. It creates the graphics that are visualized on the connected large display. That includes constructing the graphics for each tile. The representation of the physical object and the annotation are arranged. Images are scaled to a default size in order to have a consistent appearance. Long text lines are wrapped to fit into the tile.

Six tiles on the screen are supported. These tiles are managed by UbiBoard. If fewer than six annotations are visualized, the remaining ones are filled with placeholders. If all tiles are already filled and a new annotation visualization is requested, UbiBoard makes sure that the oldest annotation on the board is replaced by the new annotation.

6.4.2 Configuration Options

UbiBoard requires mapping physical objects to technology-specific attributes. The configuration options are summarized in Figure 6.9. All options may be left empty, however, this would make it hard for the user to understand to which physical object a tile is referring to.

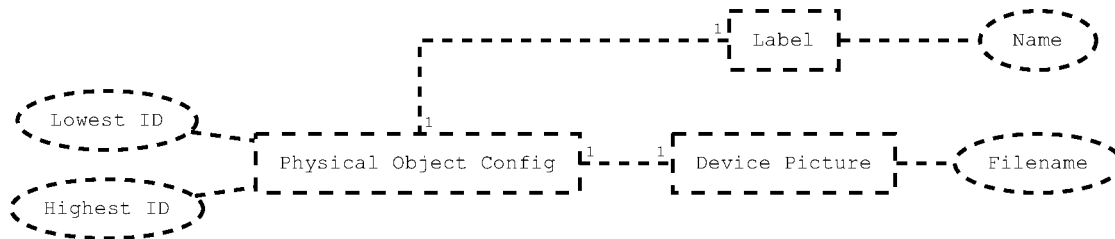


Figure 6.9: Configuration options of UbiBoard

By default, UbiBoard represents every physical object by an image and a label. Thus, each physical object must be mapped to an image and a name label in the configuration. Similar to UbiMap, the image must be indicated by a filename. The filename points to an image file in the configuration folder. It is recommended to use a compressed file format, such as `.png`, `.gif` or `.jpg`, in order to save space and to improve performance. Physical objects in the environment are easier to identify if each object is represented by a unique image file and unique label. If the image or label is not specified, UbiBoard leaves the corresponding board space blank.

6.5 Conclusion

Visualization libraries are a main feature of UbiVis. Application developers can try out different visualization approaches without technical knowledge about this visualization because the library is taking care of the implementation. Ideally, configuring provided libraries needs less efforts than developing visualizations oneself. In this chapter, we presented the functionality of the initial set of visualization libraries. For each library, we explained its features, i.e., which tasks are taken over by the library. In addition, we discussed the configuration options for each library. After explaining why a particular option is needed, we specified the option's details.

Having initial libraries available can facilitate the creation of further libraries because these can take over practices from the existing ones. The presentation of libraries in this chapter especially points out how configuration options can be designed. Moreover, UbiLight is an example how to support non-screen devices. Copying concepts to further libraries is desired because the configuration of these libraries is equal. So, an application developer can configure one library and simply copy parts or the whole configuration to the other library. Thus, applying several libraries to an application is faster if they are based on equal concepts. The initial libraries support all identified ubiquitous annotation

visualization class (cf. Section 3.3.1). Since concepts within the same class are often similar, this increases the possibility that further libraries can reuse the initial ones.

Making initial visualization libraries available was the last step needed to be able to start creating applications with UbiVis. We can now explore the space of applications that can be built with UbiVis by means of concrete examples. Furthermore, it allows us to validate and evaluate the framework on the basis of practical applications.

Chapter 7

Validation

In this chapter, we validate whether the UbiVis framework meets its main requirement: Does it give application developers the possibility to perform rapid prototyping of ubiquitous annotation visualization applications?

Validation and evaluation are linked to each other while not being the same process. Validation is about checking whether or not the requirements of a system are met [Marwedel, 2011]. Evaluation aims at finding out characteristics of the system. So, validation is about whether a system does something and evaluation is about how a system does something. Evaluation issues are described in Chapter 9.

The requirement to be validated is whether the UbiVis framework gives application developers the possibility to perform rapid prototyping of ubiquitous annotation visualization applications, which they do not have without the framework. In order to support rapid prototyping, two features need to be provided (cf. Section 2.1):

1. When exchanging the visualization, the prototype's application code must not change.
2. The visualization technology must not be developed by the user of the framework.

UbiVis provides libraries for every class of ubiquitous annotation visualization approach (cf. Chapter 6). We show that these libraries can be used for exchanging the visualization of a ubiquitous annotation visualization application without having to change the application code. The libraries control the visualization technologies so that the user of the framework does not have to implement lines of code for it. For this, an example application is created, which uses UbiMap and UbiBoard for the visualization (cf. Section 7.1).

We build our application with regard to [Edwards et al., 2003]'s criteria. [Edwards et al., 2003] investigate how to validate and evaluate software frameworks. They explain how applications must be created, which are built using a framework with the purpose to validate or evaluate that framework. For this, they identify eight lessons learned from two case studies. They conclude that for the first phase of evaluation or validation,

applications must focus only on core features of the framework. For our sample application, that means that we build a lightweight application for the validation which only focuses on the core feature of UbiVis. That core feature is the visualization of virtual information, which is associated to real-world objects.

7.1 Microphone Application

Our sample application visualizes input volumes of Bluetooth microphones. Assuming that several microphones are distributed in a noisy room, the volume of the particular microphones is varying. Different distances to sources of noise are a possible cause for that as well as technical reasons. One might be interested in the particular input volume of each microphone, e.g., because she wants to rearrange them more evenly. But microphones mostly do not reveal this information. Bluetooth microphones provide the possibility to read the input volume from a computer via Bluetooth. So, the visualization can be performed by a ubiAV application. For the instantiation, we stick to the process defined in Section 4.3.

7.1.1 Analyze Configuration of Visualizations

As first step, we need to find out which mappings and global configurations the set of libraries requires. For this, we generate a list of mappings by scanning the configuration files of UbiMap and UbiBoard. We start with the mappings to physical objects' IDs, continue with mappings of annotation IDs and finish with global configurations. The configuration options of these libraries are described in Section 6.2.2 and Section 6.4.2.

Both libraries require mapping physical objects' IDs. UbiMap needs to know the location of each annotated physical object. So, the configuration requires associating each physical object's ID with a latitude and longitude value. Latitude and longitude are expected in decimal WGS84 format [National Imagery and Mapping Agency, 2000], not using minutes and seconds. UbiBoard specifies a filename and a label for physical objects' IDs. Annotation IDs do not have to be mapped for UbiBoard. UbiMap allows mapping annotation IDs to colors if the icons shall have different colors. We refrain from this possibility because for this application it is okay to deal with black icons solely. Hence, we do not have to consider annotation mapping in the following. Finally, we use two of the three global configuration options for UbiMap. Firstly, the location of the user can be set to a fixed location or to the current GPS location. Secondly, a floor plan file including its coordinates shall be specified. Since we want to stick to UbiMap's default behavior of the rotating map, we do not specify the rotation option. UbiBoard has no global configuration options. This analysis is represented as a diagram in Figure 7.1

7.1.2 Analyze Data Space of Physical Objects and Annotations

After having found out which options must be configured for the set of libraries, we must analyze the data space of the objects that shall be annotated as well as the data space

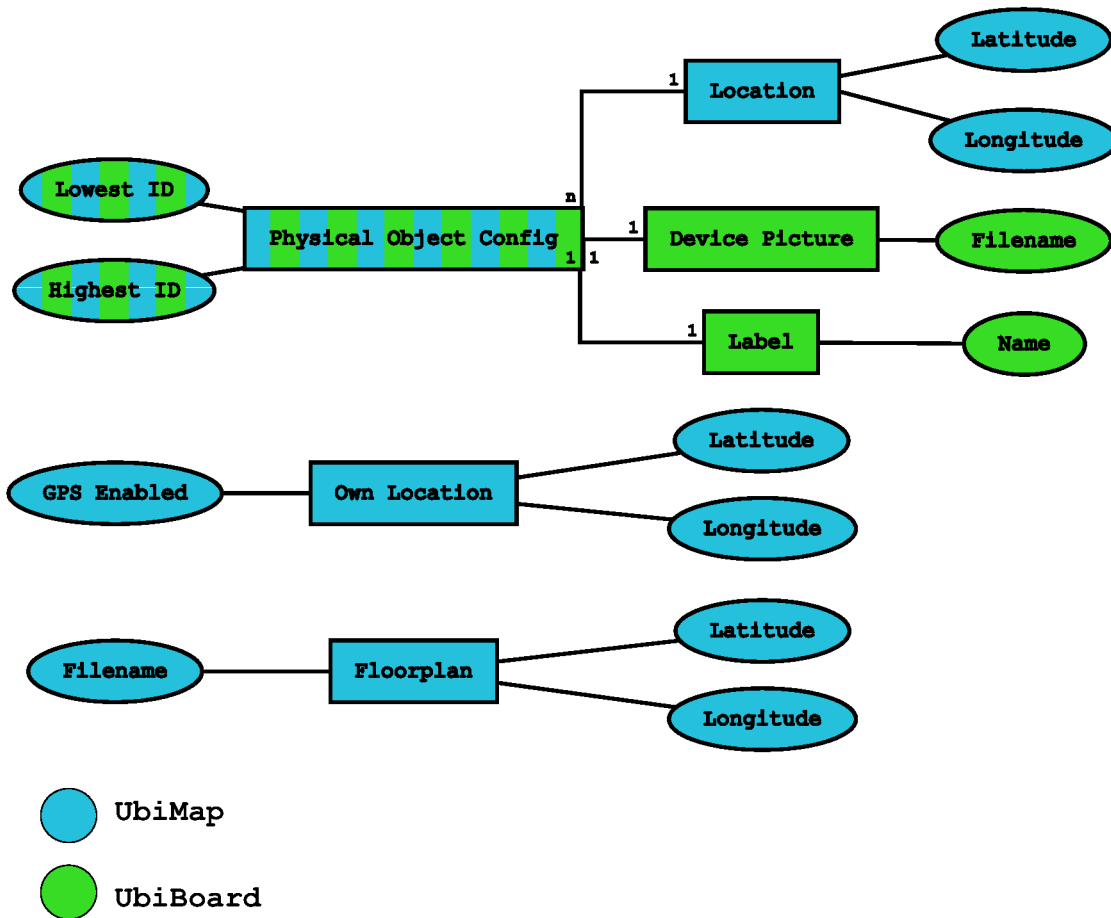


Figure 7.1: Overview of the combined configuration options of UbiMap and UbiBoard. The source library of each particular option is indicated by the colors.

of the annotation data.

Since UbiMap and UbiBoard require mapping physical objects' IDs to specific attributes, we need a method for creating microphone IDs. The IDs need to be accessed by the application developer and also by the system. The application developer must be able to enter the correct IDs in the configuration and thus must know which ID belongs to which microphone. The system must be able to gather the same ID because the application logic will create the ID attribute of the `PhysicalObject` object. The UbiMap library will use this ID attribute for searching the configuration for the corresponding location.

Since no two microphones are located at the exact same location, each individual physical object's ID is associated to a different location. Hence, grouping physical objects in order to assign them to the same location coordinates is not necessary.

We choose to use the Bluetooth MAC address of a microphone as its physical object

ID. Every Bluetooth device includes this address and since it is unique it is well suited as identifier. The MAC address can be requested by the connected technology so that the system is able to access it. Manufacturers often equip their Bluetooth devices with stickers showing the MAC address so that an application developer can also conveniently access it. If such a sticker is missing, the application developer can still request the MAC address programmatically and attach a sticker herself.

As analyzed before, annotation IDs do not necessarily have to be mapped to technology-specific attributes. But we still check whether UbiMap and UbiBoard can reasonably process the annotation data of our application. The annotation data consists of input volume values, so we are solely dealing with numeric data. Both, UbiMap and UbiBoard are capable of handling numeric, textual and image data in a way that is satisfactory for us. So, our application can just use the numeric data attribute of the `Annotation` object.

A problem appears for UbiBoard. Assuming we are using a set of equal microphone models, every microphone looks the same. Hence, UbiBoard would display the same image for each of them making it hard for the user to understand which particular microphone is meant. To overcome this problem, we take the floor plan that we use for UbiMap and make a copy for each of the individual microphones. On each copy, we mark the location of that microphone. Using these image files as device pictures lets UbiBoard display the microphone locations, which should facilitate identifying the physical objects in the real world for the user.

7.1.3 Configure Visualization

The final configuration step is to convert the reflections from the previous steps into the structure of the configuration file. A visualization library will already provide an XML file containing an initial structure. This just has to be copied and adjusted. Listing 7.1 shows the UbiMap configuration file for our sample application. The analogous listing for UbiBoard can be found in Listing 7.2.

Listing 7.1: UbiMap configuration file for the sample application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>8797854761468</lowestId>
      <highestId>8797854761468</highestId>
      <latitude>50.749612</latitude>
      <longitude>7.203817</longitude>
    </PhysicalObjectConfig>
    <PhysicalObjectConfig>
      <lowestId>8797854761387</lowestId>
      <highestId>8797854761387</highestId>
      <latitude>50.749646</latitude>
      <longitude>7.203843</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
</Configuration>
```

```

</PhysicalObjectConfig>
<PhysicalObjectConfig>
  <lowestId>8797854761402</lowestId>
  <highestId>8797854761402</highestId>
  <latitude>50.749591</latitude>
  <longitude>7.203887</longitude>
</PhysicalObjectConfig>
</physicalObjectConfigurations>
<ownLocation>
  <gpsEnabled>>false</gpsEnabled>
  <latitude>50.749619</latitude>
  <longitude>7.20387</longitude>
</ownLocation>
<floorplan>
  <floorplanFile>floormap.png</floorplanFile>
  <latitude>50.74935</latitude>
  <longitude>7.203685</longitude>
</floorplan>
</Configuration>

```

Listing 7.2: UbiBoard configuration file for the sample application

```

<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>8797854761468</lowestId>
      <highestId>8797854761468</highestId>
      <label>Microphone 1</label>
      <devicePicture>
        <filename>microphone1.png</filename>
      </devicePicture>
    </PhysicalObjectConfig>
    <PhysicalObjectConfig>
      <lowestId>8797854761387</lowestId>
      <highestId>8797854761387</highestId>
      <label>Microphone 2</label>
      <devicePicture>
        <filename>microphone2.png</filename>
      </devicePicture>
    </PhysicalObjectConfig>
    <PhysicalObjectConfig>
      <lowestId>8797854761402</lowestId>
      <highestId>8797854761402</highestId>
      <label>Microphone 3</label>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>

```

```
<devicePicture>
  <filename>microphone3.png</filename>
</devicePicture>
</PhysicalObjectConfig>
</physicalObjectConfigurations>
</Configuration>
```

The configuration options refer to the top level XML elements `physicalObjectConfigurations`, `ownLocation` and `floorplan` under the root element `Configuration`. The attributes of the two global configuration options `ownLocation` and `floorplan` are listed as child elements. Their keys are used as element names and the corresponding values as content.

For the attributes of the mapping configuration option `physicalObjectConfigurations`, one additional hierarchy level is introduced. Each single mapping is initiated by a `PhysicalObjectConfiguration` element, which is a child element of `physicalObjectConfigurations`. Then for UbiMap, the physical object's ID and the `latitude` and `longitude` which are mapped to this ID are placed as child elements of `PhysicalObjectConfiguration`. UbiBoard is analogously processed for `label` and `devicePicture`. The values of `highestId` and `lowestId` are equal for each `PhysicalObjectConfiguration` because we do not group physical objects in this application.

Since we have decided not to customize UbiMap's icon colors, `Annotation` objects do not have to be mapped. Hence, the `annotationConfigurations` is removed from the configuration file.

This configuration leads to the data structure in Figure 7.2. It is a concrete instantiation of the general data structure of the UbiVis framework, which was described in Section 4.2.3. For this concrete instantiation, the `AnnotationConfig` class is removed because `Annotation` objects are not configured. The technology-specific attributes in `PhysicalObjectConfig` and `Configuration` are set. The attributes in `Configuration`, `ownLocation` and `floorplan`, are composite data types. The corresponding classes are added to the diagram.

7.1.4 Develop Application Logic Including Calls of `VisualizationProxy`

After the library configuration is done, we can start the development of our actual application. Figure 7.3 shows what its deployment looks like. It is based on the deployment view in Section 4.2.4.

The microphone is the sensor of the application. It continuously provides volumes as input parameters. It is a D0 device because it does not support IP communication. Hence, we need a gateway computer which runs the LinkSmart proxy for the microphone. The microphone proxy shall continuously request the current input volume from the microphone and trigger an event each time the volume has changed.

So, we also need the ability to handle events and introduce the LinkSmart Event Manager for this purpose. The Event Manager can run on the same computer as the

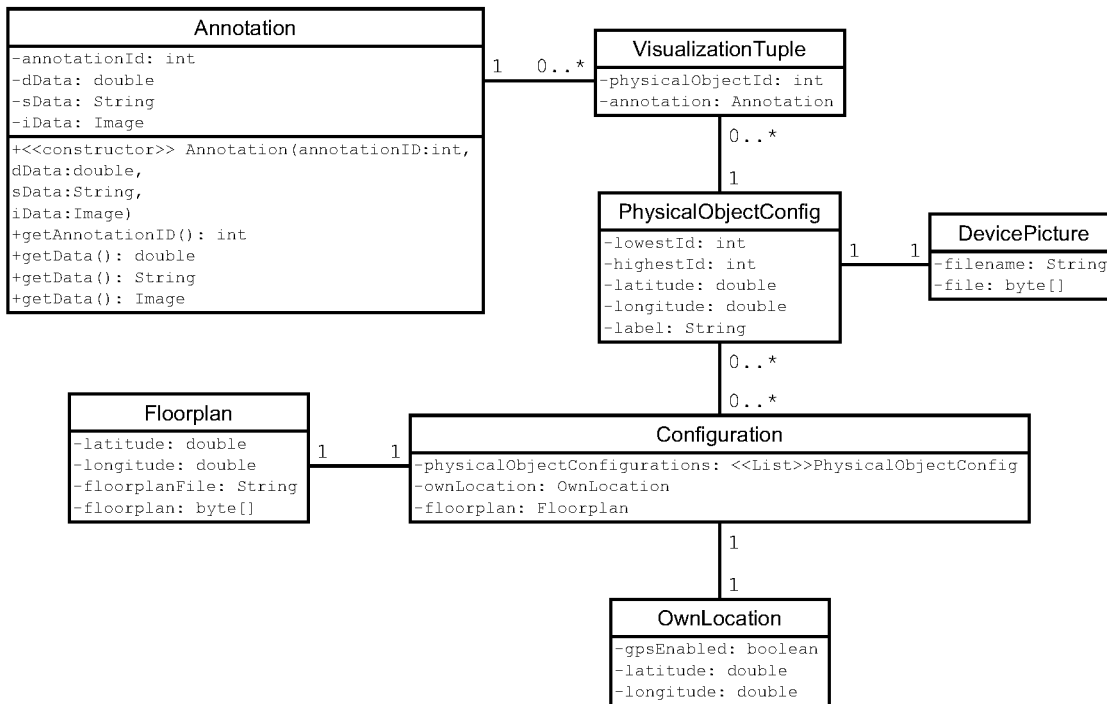


Figure 7.2: Data structure of the sample application

microphone proxy or on a different one. We choose to install it on a different computer, which is connected to the microphone gateway. Since the computers shall communicate to each other in a LinkSmart network, Network Managers are deployed to each of them. While microphone proxy and Network Manager require an OSGi environment, the Event Manager is a standalone application.

Our central control logic shall subscribe to microphone events and trigger the visualization of each new value. So, this controller logic is taking over the task of the main application, which is controlling the visualization proxies, the UbiMap proxy and UbiBoard proxy in this concrete case. We deploy controller logic, UbiBoard and both visualization proxies in an OSGi environment of a third computer in our LinkSmart network. Hence, we also need a Network Manager for this computer because the controller logic must be able to communicate to the Event Manager. Via this LinkSmart network, it is also automatically connected to the microphone gateway. A situated display is connected to this central computer acting as output device for UbiBoard. UbiMap is running on an Android smartphone, which is connected to the computer hosting the UbiMap proxy.

Figure 7.4 illustrates the interaction of the different components. It is based on the functional view in Section 4.2.2. As discussed above, the microphone provides the interface `getVolume()` through which the microphone proxy can continuously request

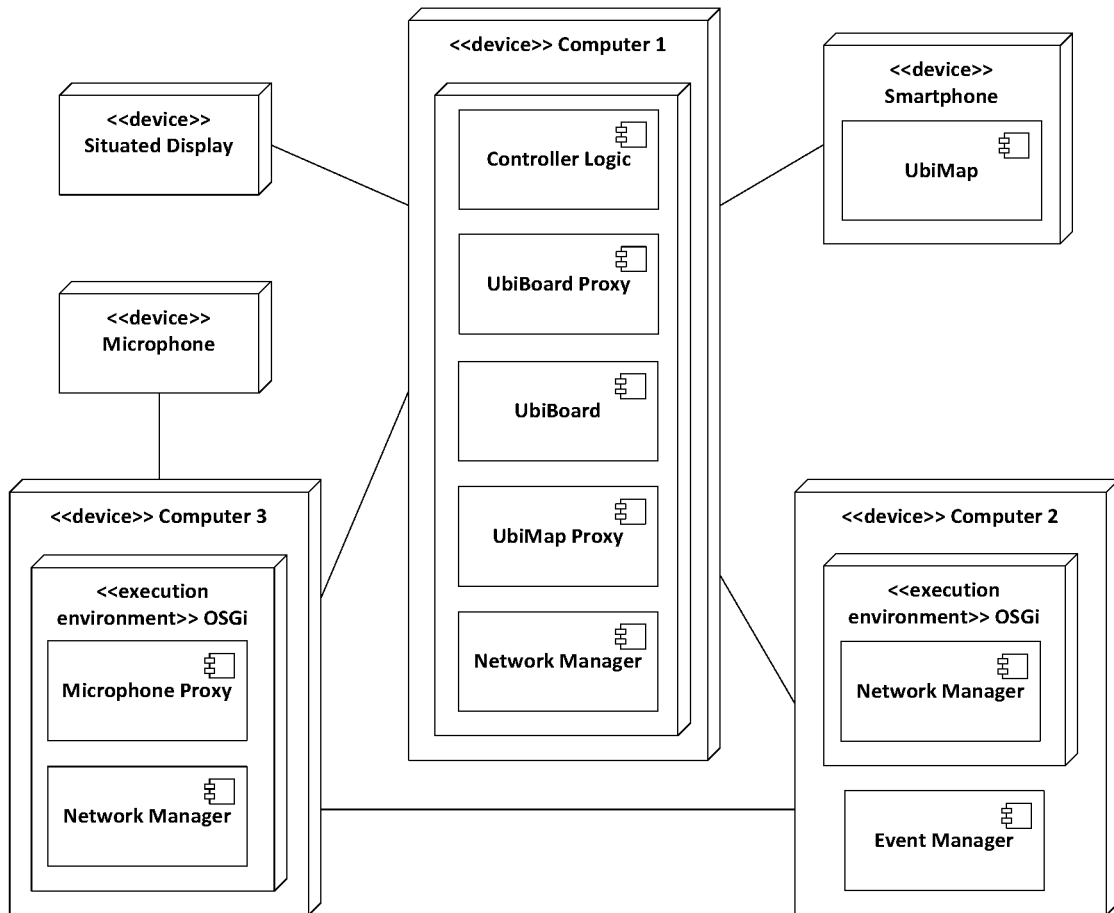


Figure 7.3: Deployment of the sample application

the current input volume.

That proxy publishes an event about a changed volume to the Event Manager. It uses the `publish(Topic, Event)` method for assigning a meaningful topic. The Event Manager forwards the event to the controller logic which has subscribed to the topic before. For this, the Event Manager calls the `notify(Event)` method and herself provides an interface called `subscribe(Topic)`. Since all the communication described in this paragraph is happening on the LinkSmart network, it is going through Network Managers on each side.

`Controller Logic` is instantiating the main application from the specification in Section 4.2.2. There are two visualization proxies and associated visualizations connected to it: `UbiMapProxy` together with `UbiMap` and `UbiBoardProxy` together with `UbiBoard`.

Figure 7.5 clarifies the inside mechanics of the sample application. This sequence diagram is illustrating the system's information flow.

7.1. Microphone Application

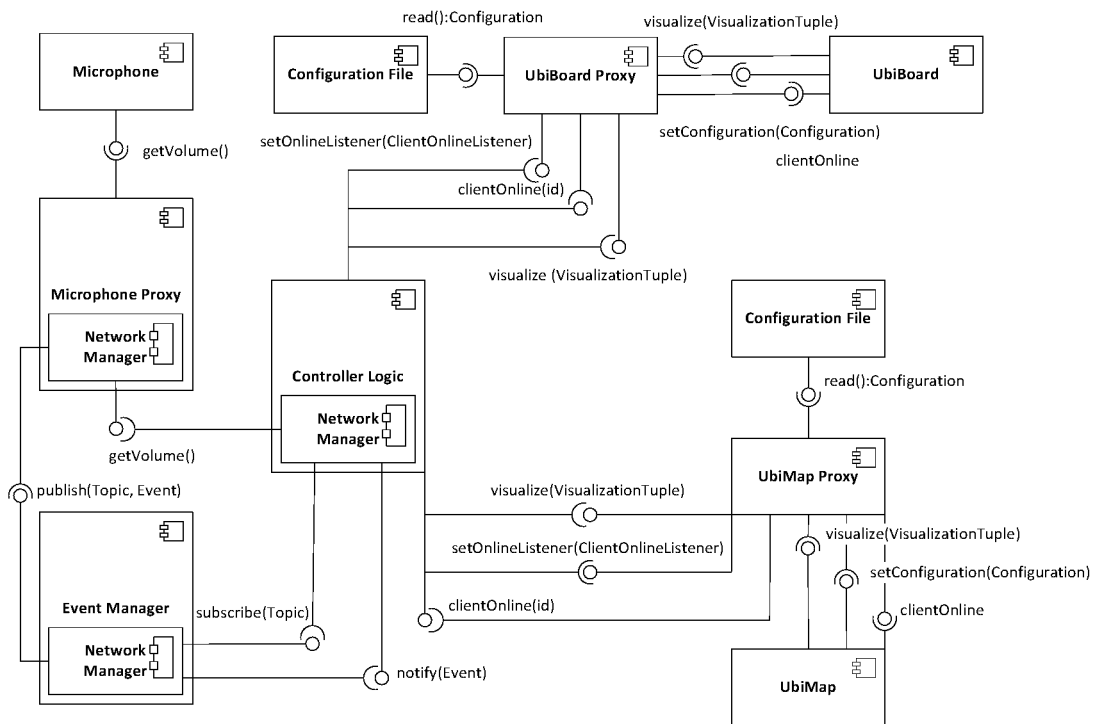


Figure 7.4: Interaction of the components of the sample application

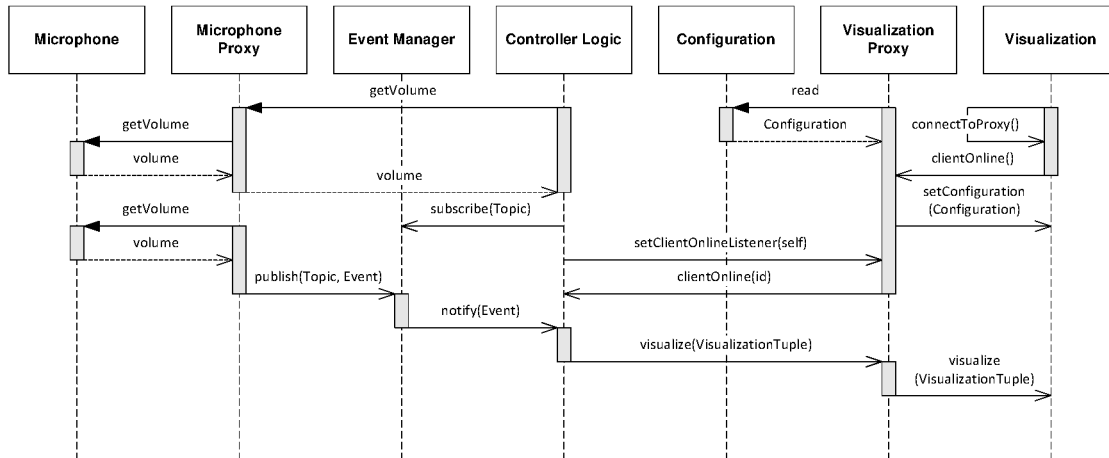


Figure 7.5: Message flow for the sample application

The controller logic can request the current input volume of a microphone by calling the `getVolume` method of the corresponding microphone proxy. In turn, that proxy calls the `getVolume` method of the actual microphone. The microphone returns the current

volume level, which is then returned from the microphone proxy to the controller logic. The microphone proxy is interposing this communication because the controller logic cannot access the microphone directly.

However, the standard behavior for the sample application is that volume levels are pushed instead of the controller logic pulling them. This subprocess starts again with the microphone proxy calling the `getVolume` method of the microphone. The proxy compares the new volume value to the last saved one and calls the `publish(Topic, Event)` upon a change. This event is forwarded to the controller logic using `notify(Event)` because the controller logic has subscribed to the topic before via `subscribe(Topic)`. Each of these events lets the controller logic trigger a call of `visualize(VisualizationTuple)` at the both visualization proxies. For the construction of the `VisualizationTuple` object, the controller logic uses the volume value and microphone ID from the `Event`. The volume value represents the annotation data and the microphone ID is used for identifying the physical object. Finally, the visualization proxy forwards the trigger to the visualization component. So, in this case, the annotated object is at once the sensor which generates the digital information.

The visualization starts after a client is online. This process begins with the visualization proxy reading the configuration from the file at component start. The configuration is translated to the object structure defined in Section 4.2.3. Afterwards, the proxy is ready for a visualization client to connect. After the visualization client has connected, it is triggering the `clientOnline()` method. This informs the proxy about the client being online and it replies by forwarding the configuration to the client using the `setConfiguration(Configuration)` method.

The call of the `clientOnline()` method also triggers another process. The visualization proxy is informing the controller logic about a client being online by calling the controller logic's `clientOnline(id)` method. This ID is a static visualization library identifier. The controller logic then knows that a particular library client is online and can decide to trigger its visualization. In order to get notified from the library about being online, the controller logic has to be registered before as `ClientOnlineListener` by calling the method `setClientOnlineListener(self)`.

Figure 7.6 shows how UbiMap visualizes the input volumes of the microphones. Each microphone's position is indicated by a black marker on a floor plan. The floor plan is overlaid over the Google Map's satellite view of the building's surrounding. Clicking a marker opens a popup window which displays the input volume of the particular microphone.

Figure 7.7 illustrates the visualization of the same application via UbiBoard. A large screen which is mounted at a room's wall displays a tile for every microphone. Each tile consists of a floor plan on which the microphones's position is indicated. The microphone's name and the input volume complete .



Figure 7.6: UbiMap visualizes the input volume of a microphone

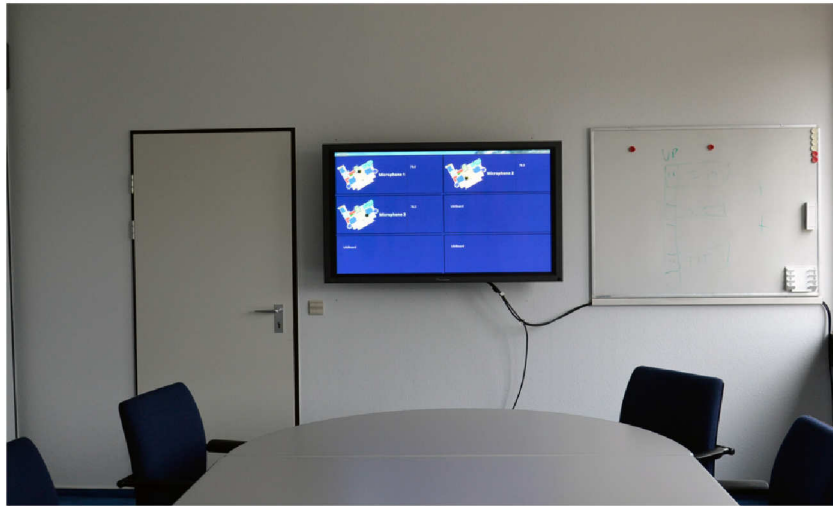


Figure 7.7: UbiBoard visualizes the input volume of the microphones

7.2 Conclusion

We were able to build an application and apply two different visualizations in the scope of the framework. This shows that the UbiVis framework supports rapid prototyping of a ubiAV system. We do not have to develop the visualization technologies ourselves and the application code remains untouched when exchanging the visualizations. So, the requirements for a framework to support rapid prototyping of ubiquitous annotation visualization systems are met (cf. Section 2.1.7).

Efficient rapid prototyping requires that the user does not have to come up with an own application process. For this reason, the conceptual framework specified such a

Chapter 7. Validation

process. In this chapter, we validated that the process is suitable for performing rapid prototyping of ubiAV.

So far, the validation is based on a single example and two concrete visualization libraries. This is still weak as it only covers a small subset of UbiVis. The validation can be substantiated by investigating more examples. Further examples should cover other libraries, different domains and more complex applications. Two more complex examples are investigated in the next chapter.

Chapter 8

Application Examples

The main motivation for creating the UbiVis framework is to facilitate the comparison of visualizations for application developers (cf. Section 1.2). In this section, we present two examples how UbiVis can be used for creating a ubiAV application, exchanging visualizations and conducting a user study that compares these visualizations. This lets us gain practical experience with the framework. Moreover, we show its practical feasibility. Finally, we aim at generalizing the validation results of Chapter 7 to more domains and all initial visualization libraries.

The first application (cf. Section 8.1) assists users in behaving more energy efficiently. The four initial visualization libraries are connected to it. In a user study presented in Section 8.2, we compare the visualizations' suitability for supporting users to save energy.

The second application (cf. Section 8.3) supports first responders in large-scale emergencies by visualizing the location and attributes of injured patients. In a study described in Section 8.4, we explore the influence of different visualizations on how users explore the data space.

For developing the application examples, we apply the standard procedure of Section 4.3 like we did for the validation. This way, we aim at substantiating the findings of Chapter 7 as its procedure is carried over to more complex applications.

8.1 Energy Efficiency Application

The energy efficiency domain has become a popular field for ubiquitous computing applications during the last years. The goal of saving energy is a current topic due to the occurrence of negative consequences of too high energy consumption. Either, these are general outcomes of global warming such as an increased number of droughts or flooding. Or, there have been single events such as the releases of radioactive materials at the Fukushima nuclear power plant or the oil spill due to the sinking of the oil drilling rig Deepwater Horizon. Finally, economic reasons such as increasing energy costs make the topic popular.

We develop an exemplary application in this domain and validate whether UbiVis supports us in rapid prototyping of the application.

8.1.1 Domain Description and Requirements

Global warming — and its disastrous environmental and economic effects — is considered one of the major challenges that mankind will face during this century. The problem is mainly attributed to CO₂ emissions that, for example, arise from the generation of electricity from fossil fuels. One way to reduce emissions of CO₂ is therefore to reduce the overall consumption of electricity in industry [Wang, 2014], public facilities [Vaccarini et al., 2013] and the private sector [Schwartz et al., 2013]; especially the latter one is what several national and international initiatives aim at.

Of course, home owners have themselves a high interest in reducing energy consumption because energy is an important cost factor. The first step for reducing the energy consumption is to be aware how much power the particular appliances in the household consume. According to [Wood and Newborough, 2003], usage awareness alone has the potential to reduce consumption by 15% in private households. However, standard electricity meters that are widely deployed in homes today, and the suppliers' analog billing systems based on yearly accounting periods, lack the feedback capabilities that are necessary to increase energy awareness and positively affect customers' behavior [Darby, 2006]. Providing more detailed data is considered useful by users because they are already willing to save energy [Jahn et al., 2011].

8.1.2 Concept

In order to make people aware of their energy consumption, our application shows the current power consumption of a set of appliances in a home scenario. Knowing about the exact power consumption values of appliances is necessary in order to find out where energy can be saved. Learning about concrete power consumption enables users to compare appliances by consumption. So, they can check out different scenarios and investigate whether these can save energy. For instance, one might compare making coffee with a coffee machine or with a water boiler.

The consumption is measured by Plugwise Circles - smart plugs which are installed in the interface of an appliance's plug and a power outlet [Plugwise B.V., 2014]. A Plugwise Circle (cf. Figure 8.1) has a ZigBee radio integrated, which allows for requesting the currently measured power consumption, e.g., by a connected computer. For this purpose, we have implemented a Java library which handles the serial port communication to the Circle. The returned consumption value is indicated in Watt.

A variant of the system developed in this section is described in [Jahn et al., 2010] and [Jentsch et al., 2011a].

8.1.3 Implementation using UbiVis Framework and Libraries

For the implementation, we stick to the process defined in Section 4.3.



Figure 8.1: A Plugwise Circle measures the current power consumption of the connected appliance [Plugwise B.V., 2014].

Analyze Configuration of Visualizations

UbiLens has a peculiarity. We have to choose between two alternatives. By setting the global parameter `imageRecognition` to `true`, we decide whether UbiLens shall use image recognition for placing the annotations or calculate them from location and rotation values. We select image recognition because it is advantageous for indoor applications.

All four visualization libraries require mapping physical objects' IDs. UbiLens maps each ID to one or more image files. UbiMap maps IDs to WGS84 location coordinate pairs. UbiLight maps them also to locations but expects two numbers representing a start and end abscissa. UbiBoard expects the filename of a device picture and the device's name for every physical object's ID. So, for all four libraries, each individual physical object is mapped to one technology-specific attribute. Hence, from this point of view, physical objects do not have to be grouped because no range of objects must be mapped to the same attribute.

UbiLens and UbiBoard do not require mapping annotation IDs. But UbiLight needs a color specified in which a particular annotation is visualized. For this, annotation IDs are mapped to color names. A range of annotation IDs can be specified for mapping all its elements to the same color. UbiMap also allows mapping annotation IDs to color names. These represent the color of the icons. We can use the same color mapping as for UbiLight.

As global configurations, UbiMap requires specifying a coordinate pair indicating a static user location. It alternatively allows for acquiring the user location by GPS. Second, UbiMap requires specifying a filename for the floor plan background including the

real world coordinates of that plan. We do not make use of UbiMap's third global option for disabling the map rotation because we want to stick to the default behavior. For UbiLight, we need to set the number of the COM port to which the LED strip is connected. UbiBoard does not expect global configuration options. The global configuration option of UbiLens was discussed before.

Figure 8.2 summarizes this analysis as a diagram.

Analyze Data Space of Physical Objects and Annotations

Step 1 has turned out that the libraries' configurations do not require clustering physical objects' IDs. Now, we check whether it makes sense to group them so that they can be mapped to the same attribute as regards content. UbiLens and UbiBoard map a physical object's ID to an appliance's photo image. If we had only a few category images which would be assigned to a set of appliances, grouping would make sense. But we decide to assign each appliance an individual picture because this may help a user finding the associated physical object. Thus, we do not need to group physical objects' IDs from this point of view. UbiMap and UbiLight map a physical object's ID to a location. The physical objects to be annotated are the appliances in our home scenario. Since two different appliances are not intended to be located at the same position, grouping does not make sense.

Each physical object must be made identifiable by the system and by the application developer through the same ID. The ID's format is arbitrary since we found out that no grouping is necessary. Each appliance is connected to a Plugwise Circle so that there is a 1:1 mapping between appliances and Circles. We choose to take the MAC address of a Circle as ID for the connected appliance because it can be easily accessed by the system and by ourselves. The MAC addresses are used for Circle identification by our Java library anyways because Plugwise already employs this identification method. So, the library allows requesting the current consumption of an appliance by specifying the connected Circle's MAC address. Coming along with this identification method, Plugwise delivers the Circles with two attached stickers, containing their MAC address. One sticker is meant to remain at the Circle, the other one can be attached to the connected appliance. For easier human access, all MAC addresses of Plugwise Circles are from a range of addresses in a way that only the last few digits change.

In Step 1, we found that UbiMap and UbiLight map annotation IDs to colors. For our energy efficiency application, we use this for introducing a traffic light metaphor. Green means zero or low energy consumption, yellow means medium consumption and red means high consumption. 100 annotations per color will probably not be exceeded, so we define green for annotation objects with IDs between 1 and 100, yellow for IDs between 101 and 200 and red for IDs between 201 and 300. For switching the light for a particular appliance off again, we specify a single `annotationId` that is mapped to the color `off`. This can be used in combination with a physical object's ID to remove the visualization for that object.

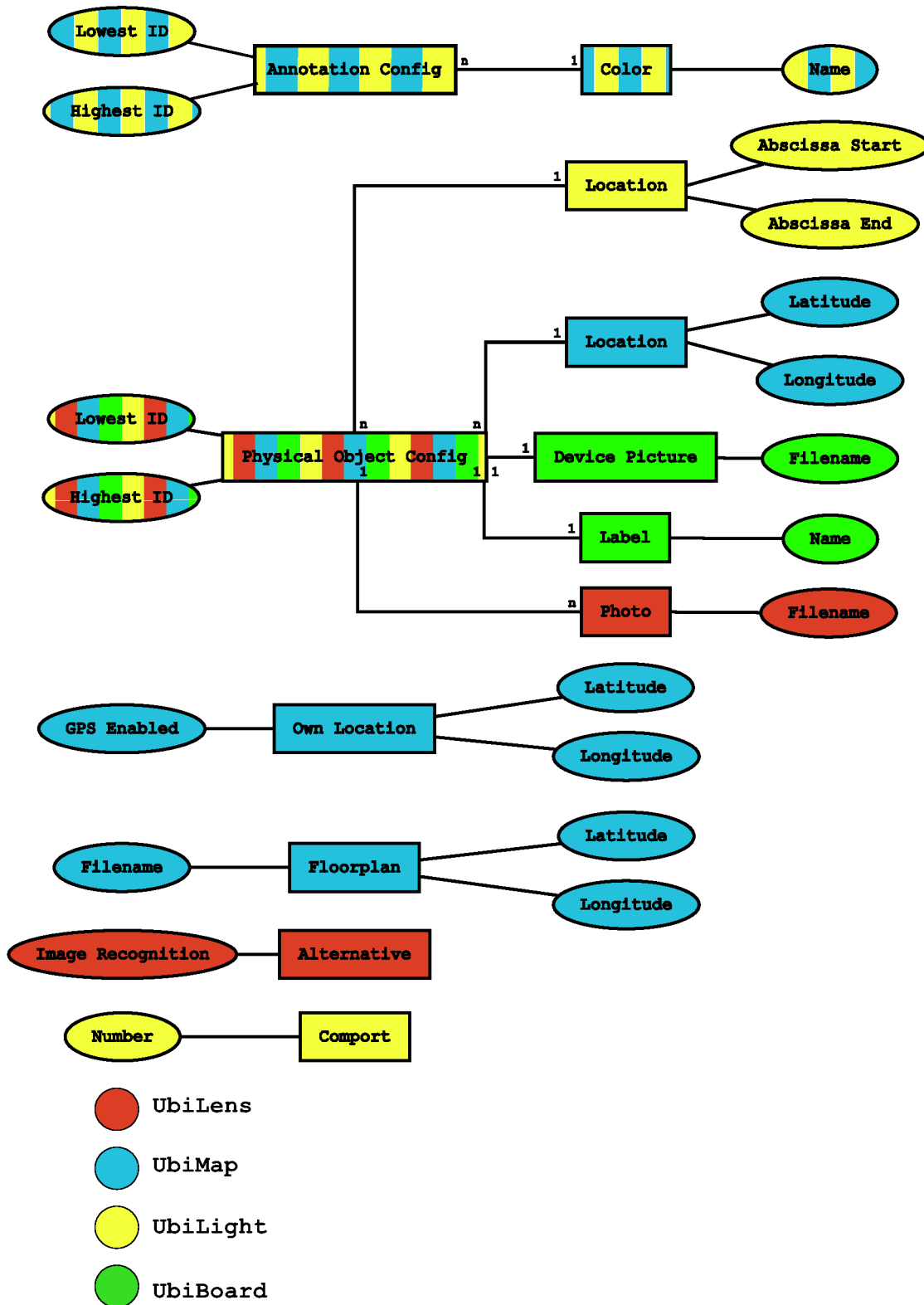


Figure 8.2: Overview of the combined configuration options of UbiLens, UbiMap, UbiLight and UbiBoard. The source library of each particular option is indicated by the colors.

Configure Visualizations

Now, the conclusions from the previous steps have to be converted into XML configuration files. In order to remain clear, we list a reduced configuration file for each library consisting of only one exemplary appliance in Listing 8.1, Listing 8.2, Listing 8.3 and Listing 8.4.

Listing 8.1: UbiLens configuration file for the energy efficiency application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>3781220497974521</lowestId>
      <highestId>3781220497974521</highestId>
      <photos>
        <Photo>
          <filename>coffee_machine1.jpg</filename>
        </Photo>
        <Photo>
          <filename>coffee_machine2.jpg</filename>
        </Photo>
      </photos>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <alternative>
    <imageRecognition>true</imageRecognition>
  </alternative>
</Configuration>
```

Listing 8.2: UbiMap configuration file for the energy efficiency application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>3781220497974521</lowestId>
      <highestId>3781220497974521</highestId>
      <latitude>52.200874</latitude>
      <longitude>8.600578</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <annotationConfigurations class="linked-list">
    <AnnotationConfig>
      <lowestId>1</lowestId>
      <highestId>100</highestId>
      <color>
        <name>green</name>
      </color>
    </AnnotationConfig>
  </annotationConfigurations>
</Configuration>
```

```

    </color>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>101</lowestId>
    <highestId>200</highestId>
    <color>
      <name>yellow</name>
    </color>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>201</lowestId>
    <highestId>300</highestId>
    <color>
      <name>red</name>
    </color>
  </AnnotationConfig>
</annotationConfigurations>
<ownLocation>
  <gpsEnabled>false</gpsEnabled>
  <latitude>52.198183</latitude>
  <longitude>8.582999</longitude>
</ownLocation>
<floorplan>
  <floorplanFile>floorplan.png</floorplanFile>
  <latitude>52.198723</latitude>
  <longitude>8.581961</longitude>
</floorplan>
</Configuration>

```

Listing 8.3: UbiLight configuration file for the energy efficiency application

```

<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>3781220497974521</lowestId>
      <highestId>3781220497974521</highestId>
      <abscissaStart>21</abscissaStart>
      <abscissaEnd>29</abscissaEnd>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <annotationConfigurations class="linked-list">
    <AnnotationConfig>
      <lowestId>1</lowestId>
      <highestId>100</highestId>
    </AnnotationConfig>
  </annotationConfigurations>
</Configuration>

```

```
<color>
  <name>green</name>
</color>
</AnnotationConfig>
<AnnotationConfig>
  <lowestId>101</lowestId>
  <highestId>200</highestId>
  <color>
    <name>yellow</name>
  </color>
</AnnotationConfig>
<AnnotationConfig>
  <lowestId>201</lowestId>
  <highestId>300</highestId>
  <color>
    <name>red</name>
  </color>
</AnnotationConfig>
<AnnotationConfig>
  <lowestId>1000</lowestId>
  <highestId>1000</highestId>
  <color>
    <name>off</name>
  </color>
</AnnotationConfig>
</annotationConfigurations>
<comportNumber>26</comportNumber>
</Configuration>
```

Listing 8.4: UbiBoard configuration file for the energy efficiency application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>3781220497974521</lowestId>
      <highestId>3781220497974521</highestId>
      <label>Coffee Machine</label>
      <devicePicture>
        <filename>coffee_machine.png</filename>
      </devicePicture>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
</Configuration>
```

The UbiLens configuration points to images of a coffee machine as well as UbiBoard's configuration. However, the images are not the same. UbiLens needs photos of the coffee machine in its real environment so that it can be detected in camera image frames. In contrast, UbiBoard displays the image for letting the user know which physical object is meant. So, in this case, the image must not necessarily show the coffee machine in its real environment. Instead, it might even show an idealized or iconized coffee machine.

Develop Application Logic Including Calls of VisualizationProxy

As final step, we start developing our actual application. Figure 8.3 shows where the components are deployed.

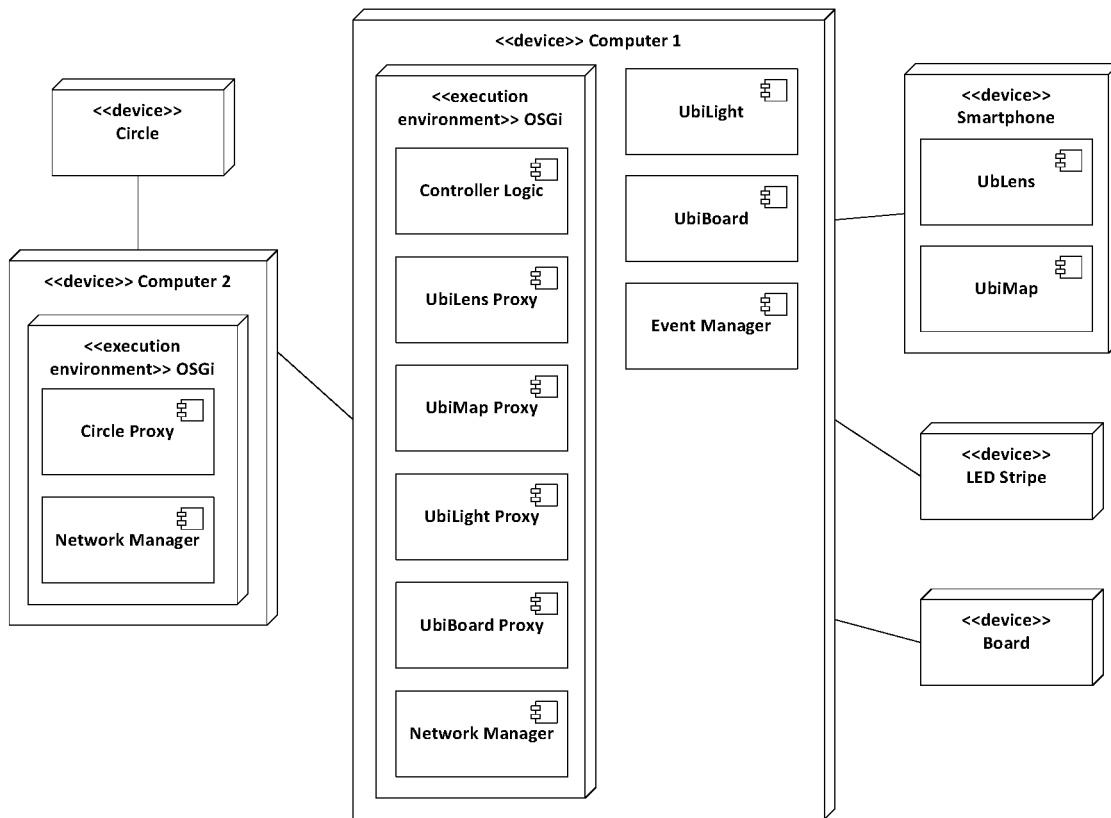


Figure 8.3: Deployment of the energy efficiency application

Plugwise Circles are D0 devices because they do not support IP communication. Hence, Computer 2 acts as gateway, hosting the Circle proxy which translates between the Circle and the main application. Each time a new power consumption value is measured, this value shall be sent to the main application logic by an event. Events are handled by the LinkSmart Event Manager on Computer 2. The Circle proxy also administers the names and images of the particular appliance which is connected to the

Circle.

The central control logic subscribes to power consumption events and triggers a visualization event for all visualization libraries at each event notification. So, the four visualization proxies are running in the same OSGi environment as the Controller Logic.

We choose to constrain the setup to these two computers and so install the UbiLight and UbiBoard visualization component also on Computer 2. Therefore, UbiLight’s LED stripe and UbiBoard’s board are also connected to Computer 2. Network Managers are introduced to both computers so that they can communicate over the LinkSmart network.

The visualization components of UbiMap and UbiLens must be installed on a smartphone that is connected to Computer 2 via Wi-Fi. If they are not used in parallel, this can be the same smartphone. Otherwise, two smartphones would be needed.

Figure 8.4 illustrates the components’ interaction.

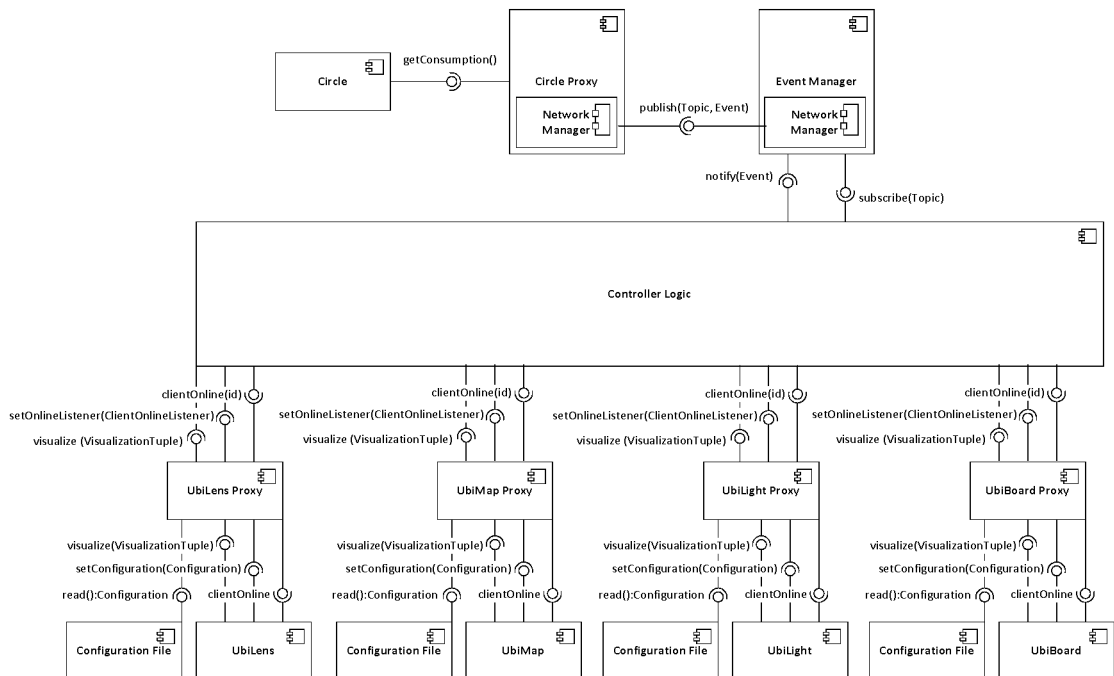


Figure 8.4: Interaction of components of the energy efficiency application

The Plugwise Java library requests the current power consumption from the Circle via the `getConsumption` method. The library is part of the Circle Proxy. Every time, a consumption value has changed, the Circle Proxy sends the new value to the Controller Logic via the Event Manager.

All four visualization proxies are connected to the Controller Logic. They are accessed in the same way. The Controller Logic has registered itself as `ClientOnlineListener` at each proxy. After a proxy has called the `clientOnline(id)` method

of the `Controller Logic` using the proxy's unique ID, the `Controller Logic` calls `visualize(VisualizationTuple)` at every consumption event. Each proxy controls its visualization component as it was described in Section 4.2.2.

The sequence diagram in Figure 8.5 illustrates the application's information flow.

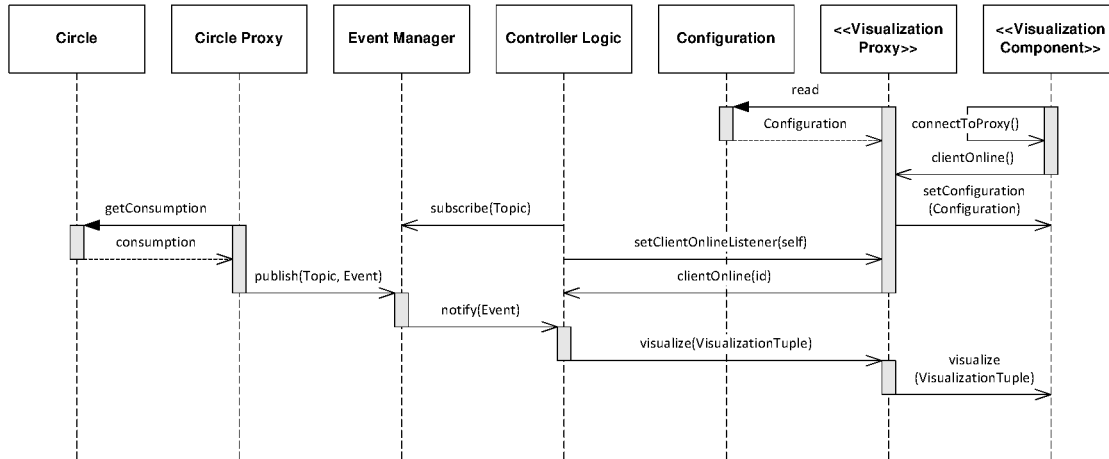


Figure 8.5: Information flow inside the energy efficiency application

The information flow is similar to the sample application described in Section 7.3. The `Circle Proxy` continuously requests the consumption of each `Circle` by calling the `getConsumption` method. The new value is compared to the stored old one. If the value has changed, it is pushed via the `Event Manager` to the `Controller Logic` using the methods `publish(Topic, Event)` and `notify(Event)`. In this step, the name and image of the appliance which is connected to that `Circle` is added to the event as well. The `Controller Logic` has subscribed to the topic using `subscribe(Topic)` before. The `Controller Logic` then triggers the visualization of the new consumption value at each online visualization proxy by calling `visualize(VisualizationProxy)`.

The online notification of each visualization component is handled in the standard way, which was defined in Section 4.2.2.

In the following, we show how the different visualization approaches are used to visualize appliances' current power consumption. With `UbiLens`, the user can scan her environment for energy consumption values. Every time an annotated physical object is inside the smartphone's screen, the object's current power consumption is displayed on an overlay (cf. Figure 8.6). A green dot indicates which physical object inside the frame is annotated.

`UbiMap` also is deployed to a tablet computer. It shows a floor plan of the room where the appliances are situated. The user location is marked by an arrow icon. Each appliance is represented by an image icon at the corresponding position in the room

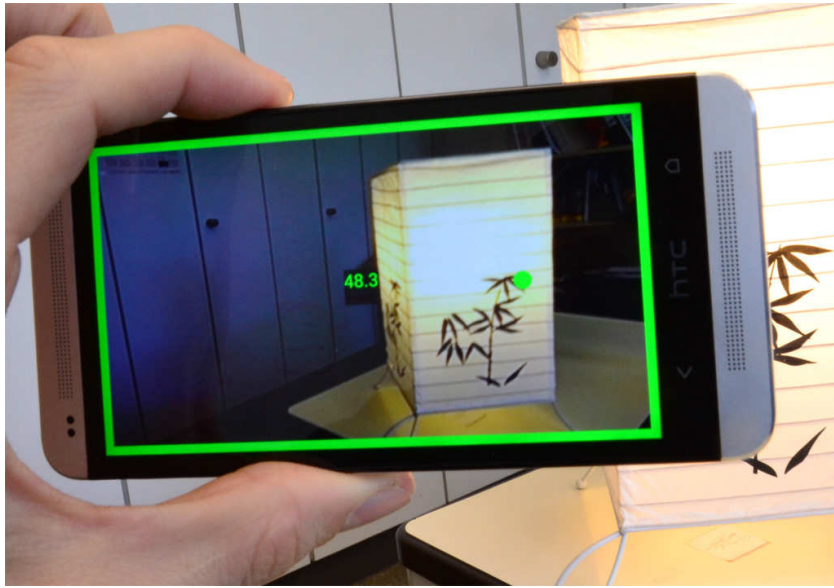


Figure 8.6: Visualizing a lamp’s power consumption via UbiLens

(cf. Figure 8.7). The icon’s color already indicates whether the appliance is consuming much, medium or little power. Clicking an icon reveals the appliance’s exact current power consumption as text in a popup window.

For UbiLight, the LED stripe is attached to the back side of a writing desk so that it illuminates the wall behind. The appliances are placed on the writing desk. By illuminating parts of the wall in red, yellow or green, UbiLight visualizes that the appliance in front of the color is currently consuming much, medium or little power (cf. Figure 8.8).

UbiBoard utilizes a public screen, which is situated in the same room as the annotated appliances. For each appliance, one part of the screen is reserved. The appliance’s name, image and current power consumption is displayed at this part (cf. Figure 8.9).

8.2 Comparing Visualization Technologies of the Energy Efficiency Application

As stated in Section 1.2, it is in general useful to test divergent ubiquitous annotation visualization approaches because the suitability for a particular use case is not obvious. In the domain of energy saving support systems, it might even be necessary to test different visualization approaches for dissimilar user groups. Demographic aspects influence attitude and motivation towards the topic of saving energy [Jentsch et al., 2011b]. So, energy saving support systems should be customized for different population groups. This also includes a customization of visual output.

8.2. Comparing Visualization Technologies of the EE Application

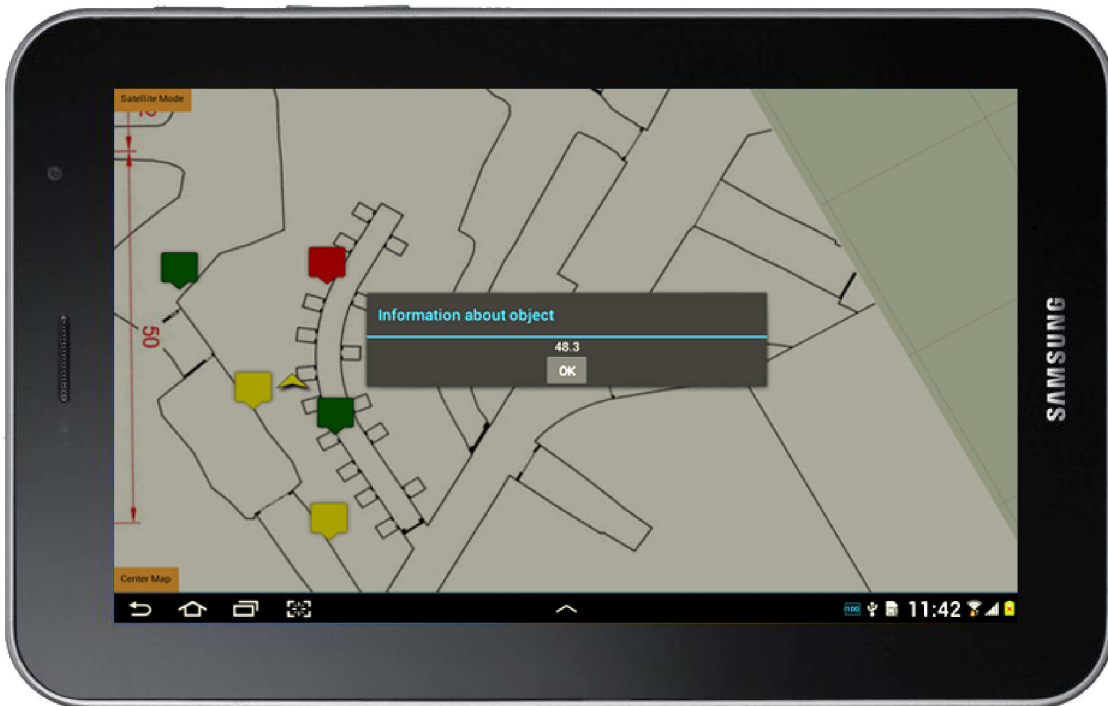


Figure 8.7: Visualizing energy saving potential of devices in the room via UbiMap

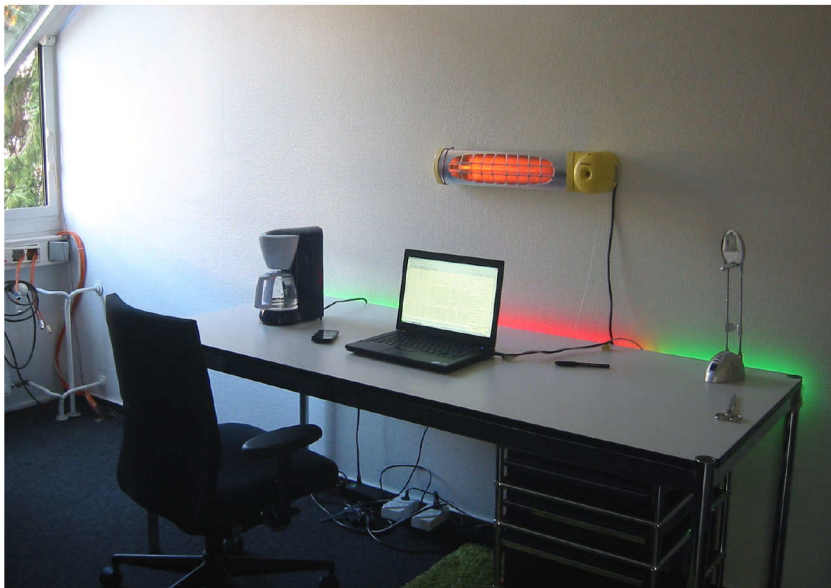


Figure 8.8: Visualizing energy saving potential of devices in the room via UbiLight



Figure 8.9: Visualizing the power consumption of nearby devices via UbiBoard

In Section 8.1, we developed an application which presents energy consumption information using divergent visualization approaches to the user, similar to [Bartram et al., 2010]. On the basis of this application, we investigate the impact of the visualization approaches to dissimilar user groups.

8.2.1 Setup

A quasi-experiment was conducted in group work. All 14 participants had used smartphones before but never used any energy-saving support system. Our aim was to find out how the participants were satisfied with the divergent visualizations and to find the most important qualitative characteristics why one is better rated than the other. At the beginning, the group was briefed that we were evaluating our software. We told that their answers were treated anonymously and they would not be judged in order to decrease potential inhibitions. We talked about the different visualizations as different

8.2. Comparing Visualization Technologies of the EE Application

”parts of the system” because we found this easier to understand for the participants. It is also irrelevant for users to know that it is actually one controller logic with different visualization methods.

For the evaluation, we set up a showcase home environment which consisted of one lamp with a light bulb, one lamp with an energy-saving lamp, a Playstation 3, a DVD player and a coffee machine. The participants were encouraged to check out the software in this showcase after we explained every function in detail. Our hope was that this showcase could reveal some surprising insights about energy consumption for the users, which were only made possible by our application. This might increase fascination for the application and improve the understanding what it could be good for. For example, the two lamps were equally bright but the light bulb consumed six times the power of the energy-saving lamp. While this relation of power consumption might have still been aware to many people, the relation of power consumption between the Playstation and the DVD player was probably unknown to most people. We showed that both devices were able to play a DVD. So, the outcome of both devices was the same. But the Playstation consumed four times the energy of the DVD player. Finally, the decision for the coffee machine was based on the fact, that the coffee machine consumes 100 times the power of the energy-saving lamp. We expected that people were aware before about the coffee machine consuming more energy than the lamps but that this relation might be surprising to many users.

After everybody got an understanding how the software parts work, we asked two questions, which were aiming at qualitative feedback and should be answered in written form (cf. Appendix A). At first, we asked the participants to ”state for each part of the system how good it would help [them] to save energy!” For each visualization, we specified a 5-point scale where 5 meant ”Absolutely” and 1 meant ”Not at all”. By this, we gathered the overall satisfaction with the different parts and their practical suitability. The second question asked to ”state one pro and one con aspect for each part of the system” in order to identify strengths and weaknesses.

8.2.2 Results

Table 8.1 lists the arithmetic mean and variance of the ratings from the 5-Point scale. Additionally, the pros and cons for each visualization are grouped by similar answers.

After answering the questions, one participant explained she was missing an important point in each subsystem. She needed ”a comparative value per device for being able to judge whether the consumption value of [her] coffee machine is low compared to other coffee machines”. This aspect was discussed in the group and supported by the others. Finally, three of the participants wrote it as an additional comment in their answer sheet.

In the following, we present a possible interpretation of the data summarized in Table 8.1, which focuses on finding out reasons for the participants’ ranking. We leave out arguments which were only mentioned once because these might not be representative.

As the UbiBoard visualization has the highest average score among all participants, its positive aspects seem to be most important. The most often named pro argument is

Chapter 8. Application Examples

	AM	VAR	Pros	Cons
Ubi-Board	4,13	1,18	Direct comparison of devices (6) Big Screen (1) Design of interface (1) Translation of watts into costs (1)	TV itself consumes energy (3) TV must be switched on the whole day (1) No reference values (1)
Ubi-Lens	3,06	1,80	Direct reference to device (10) Innovative, playful interaction style (3) Usable (1)	User must move to device (3) Cumbersome (2) Needless (1) Did not work properly (1) Easy to forget (1)
Ubi-Map	3,81	0,96	Fast access to single device (9) Mobility / remotely check consumption (7) Well-structured (1) Funny (1) Well illustrated (1)	Unclear UI (5) Needless (4) Not for iPhone (1) Too small icons (1)
Ubi-Light	3,44	1,78	Comparison (4) Understanding of consumption (3) Not distracting (1) Usable (1) Not time consuming (1)	No feeling of amount (5) Light consumes energy (4) Annoying after while (1)

Table 8.1: Participants' answers. The first two columns show arithmetic mean and variance of the overall ratings of the system, 5 being best and 1 being worst. The last two columns show pros and cons for each visualization with the number of similar answers in brackets.

the "comparison of two devices". It describes the effect that the consumption values of two devices are visualized close to each other in a structured way, which was perceived as positive for comparing two devices. This seems to be advantageous compared to the other visualizations. A con aspect which was mentioned multiple times is that the "TV itself consumes energy". This was also mentioned for UbiLight but not for UbiLens and UbiMap although the mobile phone also consumes more energy when running the applications than without. So, this is not an objective aspect but expresses how the visualization approaches are perceived differently.

Besides the UbiBoard visualization, also the UbiMap visualization was ranked as quite good. So, the main positive aspects of UbiMap seem to be important. Those are the "fast access to information about individual devices" and having a "remote, mobile interface". "Fast access to information about individual devices" refers to the fact that the participants perceived the way of acquiring knowledge with UbiMap most immediate. Clicking on an icon was felt to require less transition processes than the other visualizations. We doubt that there is really less transition for UbiMap needed than for

8.2. Comparing Visualization Technologies of the EE Application

the other visualizations. Instead, mapping map data to the real world is a process that requires huge cognitive load. So, we believe that the perception of the participants is rather due to the fact that they are more used to UbiMap style applications than to the other kind of applications. Having a map and clicking on icons for requesting more detailed information about map objects is well known from applications such as Google Maps. "Mobility / remotely check consumption" describes the advantage of UbiMap that the user can be everywhere while the other visualization approaches require the user being close to the annotated device (UbiLens, UbiLight) or close to the visualization device (UbiBoard).

UbiMap can become "unclear" when too many objects are presented because the smartphone's small display does not allow presenting a clear view on many objects at once. Unfortunately, information is missing why four participants described UbiMap as "needless".

UbiLight gets the third rank. Its multiple mentioned pro aspects are "comparison" and "understanding consumption". "Comparison" is similar to "direct comparison of devices" mentioned for UbiBoard. It means that information about several devices can be viewed at the same time, which eases comparison. However, the comparison is not on the same level of detail than for UbiBoard since for UbiLight classes of consumption are compared. This is also the main negative aspect that was mentioned for UbiLight. The participants missed a "feeling of amount" of consumption because the abstraction to three consumption classes hides more detailed consumption information. On the other hand, this abstraction also leads to some "understanding of consumption". The traffic light metaphor helps assessing whether a consumption value is high or low.

Although there are remarkably more positive than negative aspects stated for UbiLens, it gets the worst average rating. This might mean that the negative aspects "having to move to the device" and "cumbersome usage" are more important than the positive ones. The positive aspects having a "direct reference to the device" and the "innovative, playful interaction style" may only be nice to have but not very important. Most of the aspects are discussed above or are self-explaining. "Direct reference to device" refers to the effect that it is clearer for the user which object is being annotated than for UbiBoard and UbiMap. For UbiLens, the annotated object is the one which is currently seen on the displays. For UbiBoard and UbiMap, however, object information in the digital world has to be found in the real world.

UbiLens's and UbiLight's rankings have a higher variance than those of UbiBoard and UbiMap. This is because UbiLens and UbiLight tend to have either very high or very low rankings while the other two visualization approaches were ranked moderate throughout the participants. This might be due to UbiLens and UbiLight being rather uncommon compared to rather standard mobile phone apps like UbiMap or public displays applications like UbiBoard. Here, the readiness of participants towards new application approaches might have influenced the ranking.

8.3 eTriage Application

Triage (from French, meaning "sorting") is a process that is carried out in medical large-scale emergencies. Its aim is to save as many people as possible given limited medical resources. We talk of large-scale emergencies where the event is such that available medical resources are overwhelmed. For example, ten injured spectators in a stadium are not a large-scale emergency because the stadium's medical personnel can handle the situation. 100 spectators falling down from a stadium's structure will overwhelm the available medical teams and thus result in a large-scale emergency. It is in such situations, where resources are scarce, triage is necessary so that resources are allocated to those that would benefit from them most.

8.3.1 Domain Description and Requirements

First responders usually come to unfamiliar environments where they have to orientate themselves first. Information which could serve as basis to orientate themselves and take decisions is often incomplete and unreliable – major landmarks may have been destroyed, and reported information about number and status of victims may be incomplete or wrong [Holzman, 1999]. There is often time pressure and seconds can be crucial for life or death of patients. Hence, quickly understanding a situation is an important and difficult task for first responders. That is why a number of computerized support tools exist and research for new possibilities is ongoing [Al-Akkad et al., 2011].

It is also important to share information among distributed team members and between first responder teams so that information does not need to be acquired twice. Time can be saved and a shared sense of the situation can be produced [Büscher and Mogensen, 2007]. Moreover, the provenance and design of information matters: Information from team members may be more concise, 'fit for purpose' and reliable than information that bystanders can provide. In this stressful situation, the single process steps are well-known and clearly defined so that the work is performed as routine [Kyng et al., 2006].

In order to get an overview of victims and to decide the priority for treatment in case of many injured people, first responders carry out triage. Within very short time, triage personnel quickly fill out a paper triage tag containing a first, rough diagnosis of the injury, level of transportability and a category indicating the priority (cf. Figure 8.10). The exact process for determining the category and what the categories are, varies from country to country. In general, these four categories are assigned:

- Red – Immediate. Victims who need immediate specialized medical attention, otherwise they will not survive.
- Yellow – Delayed. Victims whose wounds may be serious, but will survive the next hour without medical attention.
- Green – Minor. Victims who are lightly wounded and do not face any threat to their life for the next few hours up to a day.

- Blue or Black– Deceased. Victims who are seemingly dead, so badly wounded that they are not likely to survive, no matter how much medical attention they receive.

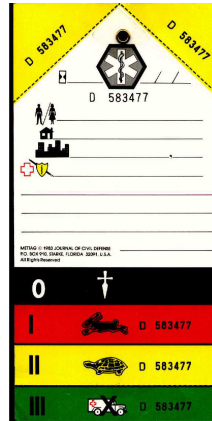


Figure 8.10: A traditional paper triage tag is attached to a patient and contains the most important information about her [Mettag.com, 2014].

The lower part of the triage tag contains the colors; the unnecessary colors are cut off to indicate the current category. Succeeding rescue personnel quickly receives medical information about the patient from this tag. They start treating red patients and continue with the yellow ones after all reds have been subserved. Last come the green patients and the blues/blacks are mainly accompanied and pain allayed.

There is room for improvement in the current triage process. For example, it is difficult for a first responder to get an overview how many patients are tagged to each category, where to find them and whom to treat next. Therefore, transporting patients to gathering places is currently performed for getting a better overview; but that is time consuming and can be optimized.

We analyzed a set of electronic triage systems and conducted several user studies with first responders and found out that the triage process is quite sensible to changes [Jentsch et al., 2013]. The introduction of technology in order to improve the triage process often has negative side effects. That is why technology has to be introduced with care. As one aspect, the consequences of different visualization technologies should be investigated.

8.3.2 Concept

The main concept of our eTriage application is to migrate the information visualization from the paper tags to electronic devices. This may improve readability because the traditional tag can become hard to read due to blood or mud. So, triage information must be digitalized in order to be able to use it on electronic devices. The visualization can be done in different ways and it is not clear if and in which way a digital visualization

Chapter 8. Application Examples

is advantageous over the format on the paper tag. Hence, a comparison of different visualization styles makes sense for this scenario.

Digital information offers other exploration possibilities than analogous data, which may have advantages for the triage process. For example, data can be filtered or two data sets can be selected and compared to each other more easily. Viewing a snapshot of the patient distribution may fasten up the rescue process because first responders can quickly decide whom to treat next.

For digitalizing triage information, the paper tag is being replaced by an electronic tag. It is integrated into a snap-on bracelet, which closes automatically when snapped against an arm or leg (cf. Figure 8.11). The bracelets are more durable than paper and easier to put on. They are available in the four triage colors. Accordingly, each tag features a preconfigured ID and a triage category. The tag incorporates a GPS unit for acquiring its location. Additionally, it features a temperature sensor equipped for measuring the temperature of its close environment. The tag can send gathered data to other devices using an integrated networking interface.



Figure 8.11: The electronic triage tag is attached to a colored bracelet, which closes automatically when snapped against an arm.

So, this advanced tag also delivers additional information to what the traditional tag can provide. A first responder is always up to date about the temperature of that environment. This can be useful, e.g., for detecting that a patient is situated too close to a fire or other dangerous heat source.

8.3.3 Implementation using UbiVis Framework and Libraries

Analogously to the examples of Section 7.1 and Section 8.1.3, we follow the UbiVis standard procedure described in Section 4.3 for implementing the application.

Analyze Configuration of Visualizations

It does not make sense to apply UbiLight for the emergency domain since it requires an instrumented environment, which is not given at emergency sites. The non-suitability of ambient light for the emergency is obvious and application developers do not need the help of UbiVis for finding that out. But for less obvious cases, one valid outcome of comparing different ubiAV technologies in a particular use case can be a detection of incompatibility. This insight can be achieved faster when using UbiVis. As conclusion for this analysis, we only need to consider the requirements of UbiLens, UbiMap and UbiBoard.

This time, we select the calculation alternative of UbiLens because we know about the patients' real world locations whereas it would be difficult to the patients them via image recognition.

All three libraries require mapping physical objects' IDs. While UbiLens and UbiMap map them to latitude and longitude pairs, UbiBoard expects a picture and label.

UbiLens requires mapping annotation IDs to background and text colors. Similarly, we map annotation IDs to icon colors for UbiMap. For UbiBoard, no further annotation ID mapping is required.

Regarding global configurations, we can allow UbiLens and UbiMap to acquire the user location by GPS since our application is preferentially used outside. However, for possible buildings on the emergency side, we introduce a floor plan overlay and the associated coordinate pair for UbiMap. With regard to the study in Section 8.4, we want to disable map rotation. UbiBoard has no global configuration options in general.

Figure 8.12 illustrates this analysis.

Analyze Data Space of Physical Objects and Annotations

We use the bracelet IDs as physical object identifiers. The IDs are unique and accessible by system and developer similar to the Circle IDs of the energy efficiency application (cf. Section 8.1.3). The actual format is arbitrary because no grouping will be necessary. Each bracelet is connected to one patient at maximum so that patients are unambiguously identified by the bracelet IDs as well.

UbiBoard actually intends to set an image for each physical object. In the eTriage setting, we do not have a picture of each patient available. Since users of the eTriage application would be interested in the patients' locations, we provide a map as image file on which that location is marked. Similarly, we do not know the patients' name or other personal attribute which could be used as label for identifying the patient. Instead, the patients' triage category may help distinguishing patients. The category indicated on

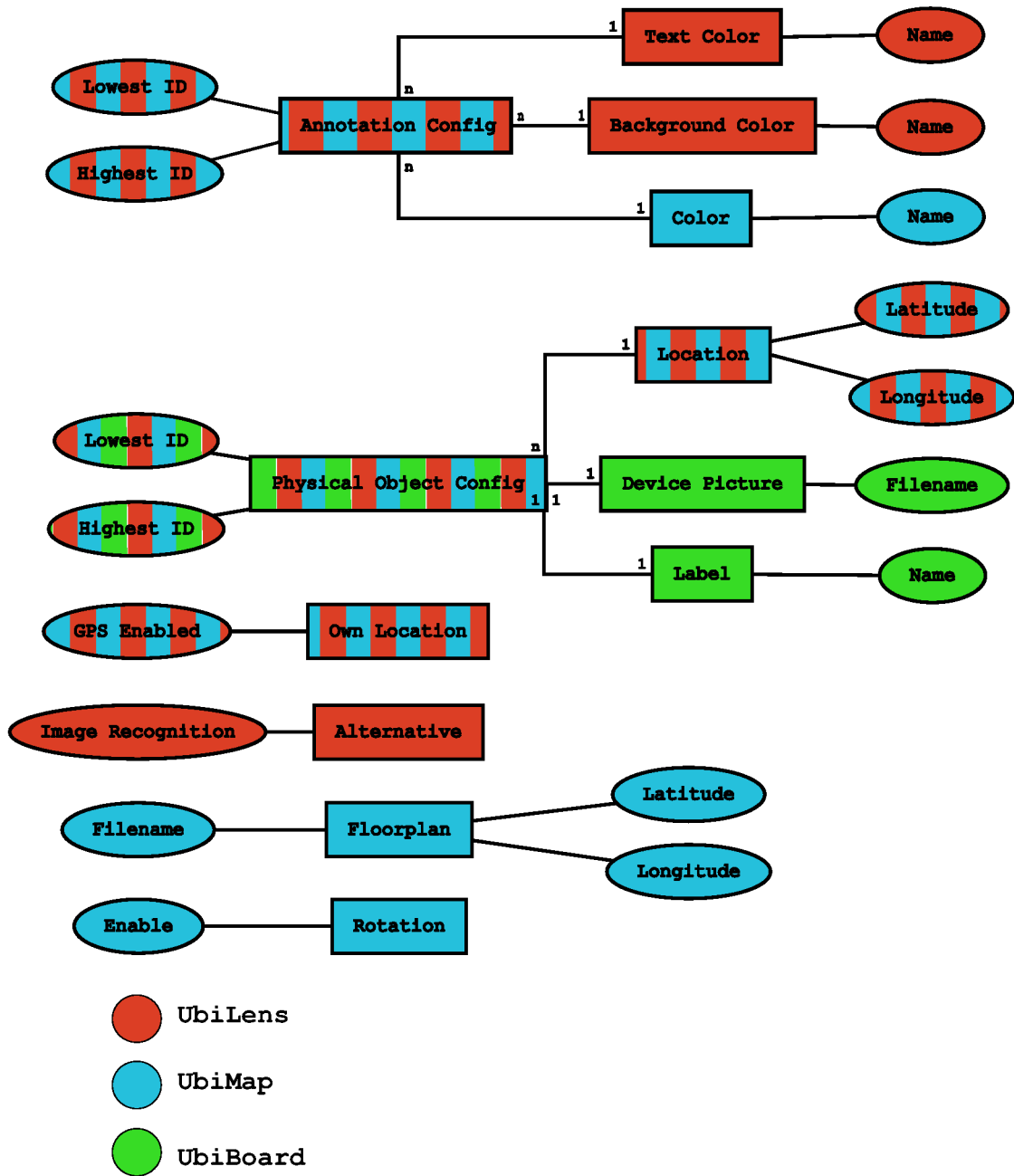


Figure 8.12: Overview of the combined configuration options of UbiLens, UbiMap and UbiBoard. The source library of each particular option is indicated by the colors.

the board can be compared to the color of the bracelet and thus eliminate mismatches. Conveniently, each bracelet has a distinct ID and triage category, which can be carried

into the configuration file.

Bracelet categories and map images cannot be meaningfully clustered. Also, each patient is located at a different location. Hence, grouping of IDs does not make sense.

The triage category can be easily used for UbiMap's and UbiLens's color configurations. The triage process associates colors with each of the categories and this method is known to the first responders. We use these triage colors as mapped background colors for UbiLens and icon colors for UbiMap. For text colors, we choose a tone that promises good readability in combination with the particular background hue. For the first run, we assume not to exceed 100 annotations per color, so we group annotation IDs as follows. IDs between 1 and 100 are mapped to red, IDs between 101 and 200 to yellow, IDs between 201 and 300 to green and IDs between 301 and 400 to blue.

The actual annotation data is numeric: a temperature value. However, we want to add the text "Temp:" prior to the value to make it self-explaining. An improvement of the eTriage application can be bracelets that send additional vital values, which would make such label text necessary. Hence, we use the String attribute of the `Annotation` object for the temperature value and its label. All libraries are capable of displaying Strings so that we do not have to consider mapping from this point of view.

Configure Visualizations

Listing 8.5, Listing 8.6 and Listing 8.7 represent a reduced configuration file for each library, consisting of only one exemplary patient (cf. Section 8.1.3).

Listing 8.5: UbiLens configuration file for the eTriage application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>244823674823</lowestId>
      <highestId>244823674823</highestId>
      <latitude>58.952373</latitude>
      <longitude>5.732283</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <annotationConfigurations class="linked-list">
    <AnnotationConfig>
      <lowestId>1</lowestId>
      <highestId>100</highestId>
      <backgroundColor>
        <name>#CC0004</name>
      </backgroundColor>
      <textColor>
        <name>white</name>
      </textColor>
    </AnnotationConfig>
  </annotationConfigurations>
</Configuration>
```

```
<AnnotationConfig>
  <lowestId>101</lowestId>
  <highestId>200</highestId>
  <backgroundColor>
    <name>#FFFA22</name>
  </backgroundColor>
  <textColor>
    <name>black</name>
  </textColor>
</AnnotationConfig>
<AnnotationConfig>
  <lowestId>201</lowestId>
  <highestId>300</highestId>
  <backgroundColor>
    <name>#037802</name>
  </backgroundColor>
  <textColor>
    <name>white</name>
  </textColor>
</AnnotationConfig>
<AnnotationConfig>
  <lowestId>301</lowestId>
  <highestId>400</highestId>
  <backgroundColor>
    <name>#0000EE</name>
  </backgroundColor>
  <textColor>
    <name>white</name>
  </textColor>
</AnnotationConfig>
</annotationConfigurations>
<ownLocation>
  <gpsEnabled>true</gpsEnabled>
</ownLocation>
<alternative>
  <imageRecognition>false</imageRecognition>
</alternative>
</Configuration>
```

Listing 8.6: UbiMap configuration file for the eTriage application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
```

```
<lowestId>244823674823</lowestId>
<highestId>244823674823</highestId>
<latitude>58.952373</latitude>
<longitude>5.732283</longitude>
</PhysicalObjectConfig>
</physicalObjectConfigurations>
<annotationConfigurations class="linked-list">
  <AnnotationConfig>
    <lowestId>1</lowestId>
    <highestId>100</highestId>
    <color>
      <name>red</name>
    </color>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>101</lowestId>
    <highestId>200</highestId>
    <color>
      <name>yellow</name>
    </color>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>201</lowestId>
    <highestId>300</highestId>
    <color>
      <name>green</name>
    </color>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>301</lowestId>
    <highestId>400</highestId>
    <color>
      <name>blue</name>
    </color>
  </AnnotationConfig>
</annotationConfigurations>
<ownLocation>
  <gpsEnabled>true</gpsEnabled>
</ownLocation>
<floorplan>
  <floorplanFile>headquarter.png</floorplanFile>
  <latitude>49.549615</latitude>
  <longitude>4.186774</longitude>
```

```
</floorplan>
<rotation>>false</rotation>
</Configuration>
```

Listing 8.7: UbiBoard configuration file for the eTriage application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>244823674823</lowestId>
      <highestId>244823674823</highestId>
      <label>Immediate</label>
      <devicePicture>
        <filename>244823674823.png</filename>
      </devicePicture>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
</Configuration>
```

Develop Application Logic Including Calls of VisualizationProxy

Figure 8.13 shows where the components are deployed. It is similar to the deployment of the energy efficiency application (cf. Section 8.1.3).

Bracelets' information is accessed by a **Bracelet Proxy** on the **Server**. For pushing this information to the **Client**, an Event Manager is deployed there. The **Client** also hosts the controller logic, all visualization proxies and the UbiBoard component. A **Smartphone** and **Board** are connected to the **Client** and used as visualization devices.

Also the component interaction (cf. Figure 8.14) works similar as in the energy efficiency application. Only the bracelet replaces the Circle as data source and UbiLight is not used.

Consequently, also the information flow (cf. Figure 8.15) is similar to the energy efficiency application. Slightly different is the way events are generated. Instead of polling the sensor, the bracelet is calling `setTemperature(temperature)` at the **Bracelet Proxy**.

The following paragraphs present the eTriage application with its different visualizations. The UbiLens visualization lets first responders scan their environment for patients. They explore the surrounding by looking through the smartphone or tablet (cf. Figure 8.16). Patients are highlighted as cubical overlays. This can be useful when being at the emergency site because patients might be hidden. Using this visualization even enables first responders to find patients behind obstacles, such as walls or trees. With UbiLens, they can also see through these obstacles.

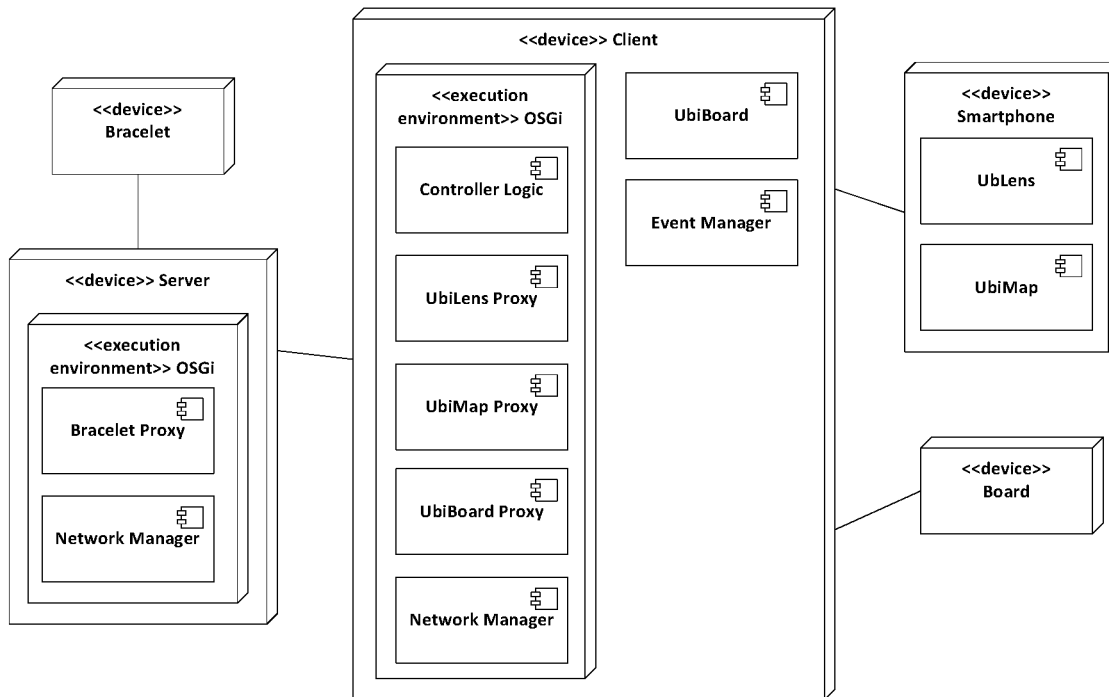


Figure 8.13: Deployment of the eTriage application

Triage data is presented twofold. The measured temperature is found as text on the cubes. The triage category is indicated by cubes' colors. This supports first responders in deciding which patient to treat next. They can compare the patients visualizations on the cubes and, e.g., move to the patient with the highest triage category or the highest temperature first.

UbiMap visualizes the patients on an outdoor map of the emergency site (cf. Figure 8.17). In addition, the inside floor plan of a building is added. Each patient is represented as a marker on the map. The marker's color represents the triage category in the same way as UbiLens. The temperature is shown in a popup after clicking a marker.

First responders can also use this visualization for getting a better overview of the patients at the emergency site. Since the smartphone can be carried with them, this eTriage visualization can be used at the emergency site. In contrast to UbiLens, the UbiMap visualization would alternatively make sense for first responders which are not at the emergency site, yet. For example, they can already decide which patient to treat first on the way to the emergency site.

However, UbiBoard can only be used remotely because it is immobile. This eTriage visualization makes sense to be used by the incident commander, a person authorized to issue directives who is located close to the emergency site. In many rescue organizations such a commander manages which first responder treats which patient. These decisions can be supported by eTriage. Often, her on-field command center provides the necessary

Chapter 8. Application Examples

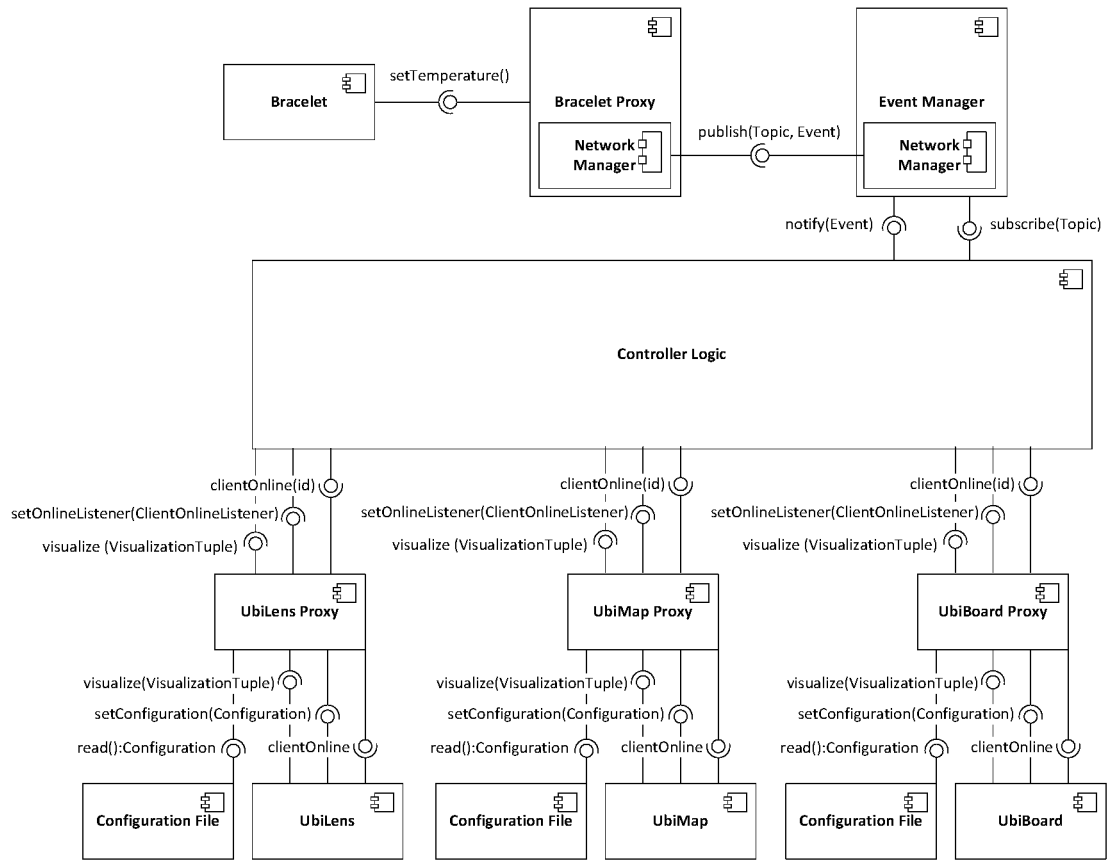


Figure 8.14: Interaction of components of the eTriage application

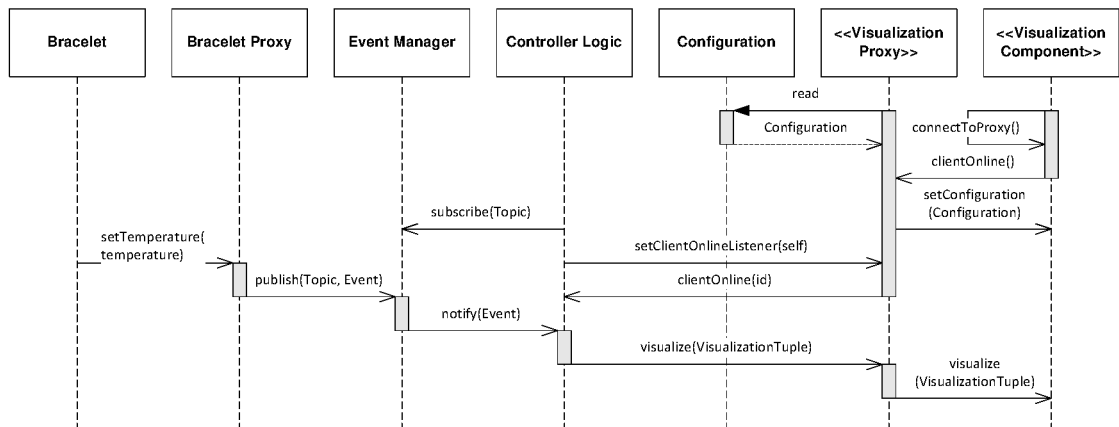


Figure 8.15: Information flow inside the eTriage application



Figure 8.16: Visualizing location and vital values of a patient via UbiLens

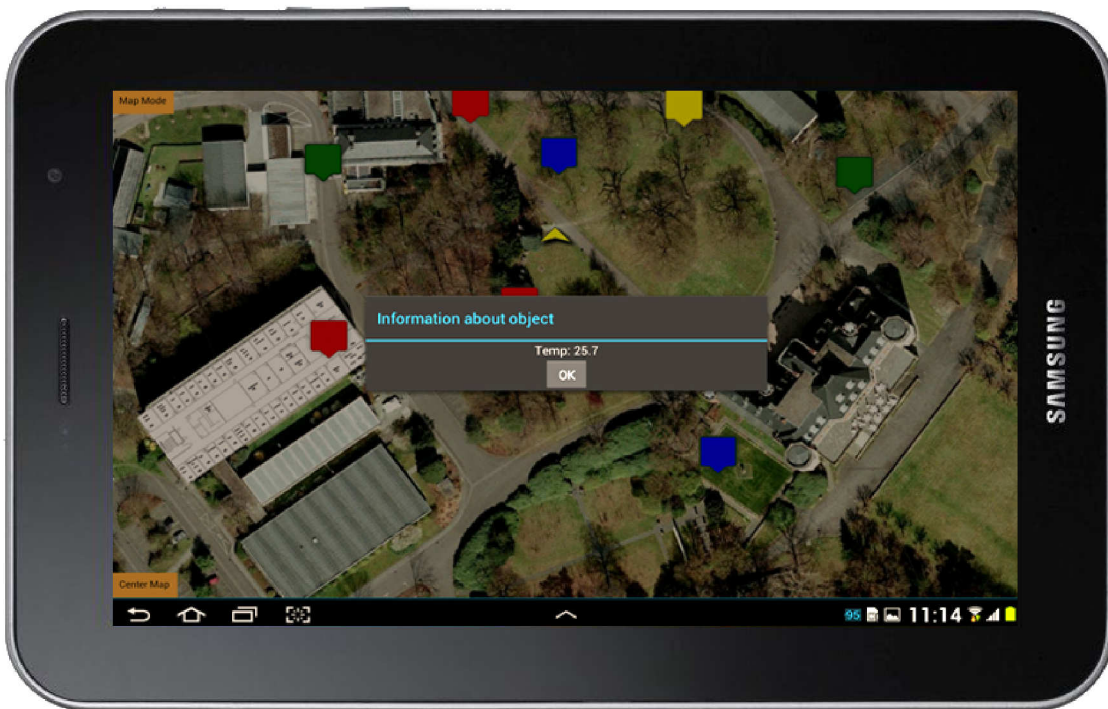


Figure 8.17: Visualizing patient information on the emergency site via UbiMap

infrastructure to run UbiBoard.

Each patient's data is visualized on a tile of UbiBoard (cf. Figure 8.18). A tile consists of three information pieces. A map shows the location of the patient. Next to it, the user finds the triage category as text. Finally, the temperature is indicated.



Figure 8.18: Visualizing patient information on the emergency site via UbiBoard

The eTriage example shows a limitation of the initial visualization libraries. Only static physical object location data can be processed. We cannot change the objects' positions at runtime according to the GPS signal and thus only visualize a snapshot of the current patient distribution. However, this is a matter of the particular libraries and not of the UbiVis framework. A visualization library developer would only have to write an extended version of the initial libraries according to Section 4.4 and make it available

to the public. The extended versions would provide the same visualization capabilities but map physical objects to dynamic location sources instead of mapping them to static locations. For instance, the extended configuration file might expect a web service URI which provides the current GPS location of the particular bracelet.

8.4 Comparing Visualization Technologies of the eTriage Application

A lot of work has been conducted for finding out how people interact with traditional paper or digital maps and how maps must be accordingly designed (e.g. [Tufté and Weise Moeller, 1997, MacEachren, 2004, Dent et al., 2008]). The knowledge how people interact with digital maps may not be applied to ubiquitous computing paradigms because these differ substantially from the old paradigm. While a map is viewed from the outside, many ubicomp approaches change the viewpoint by putting the user inside the data space. Even common mobile map applications feature the updated visualization of the user's own location, which leads to an immersion into the data space that traditional digital maps do not offer. This is even more relevant for the see-through technique where a user is exploring data in her physical 3D environment. These visualization approaches are instantiated by UbiMap and UbiLens. The aim of our study is to find out about the characteristics of these new viewpoints for exploring data spaces particularly for the triage use case.

8.4.1 Study Setup

For the study setup, eleven victims were dispersed in a 9x10m room. The victims were represented by rescue dummies. In the triage system, each patient had virtually assigned a location, diagnosis text, temperature value and triage category. This dataset was visualized via UbiMap (cf. Figure 8.19) and UbiLens (cf. Figure 8.20).

Each participant was headed to the room and asked to explore the dataset with one of the two visualizations. We alternated the visualization technology so that eleven of the 22 participants used UbiMap and the other eleven used UbiLens. For UbiMap's floor plan, we had cut off the particular part from the official floor plan of the building. Map rotation was turned off. Ten of the participants had hand-on experience in emergency response. Five of these used UbiMap; the other five used UbiLens.

We conducted a structured interview that triggered several facets of the data exploration. The following three questions had to be answered with the help of the application.

- **Question 1: Which color does this patient have? (Interviewer pointed at a patient)** With this question, we wanted to find out whether it is easier with one of the two visualization technologies to find the virtual representation of a given physical object.
- **Question 2: Which value does the one from your perspective right of the patient from Question 1 have?** With this question we wanted to find out

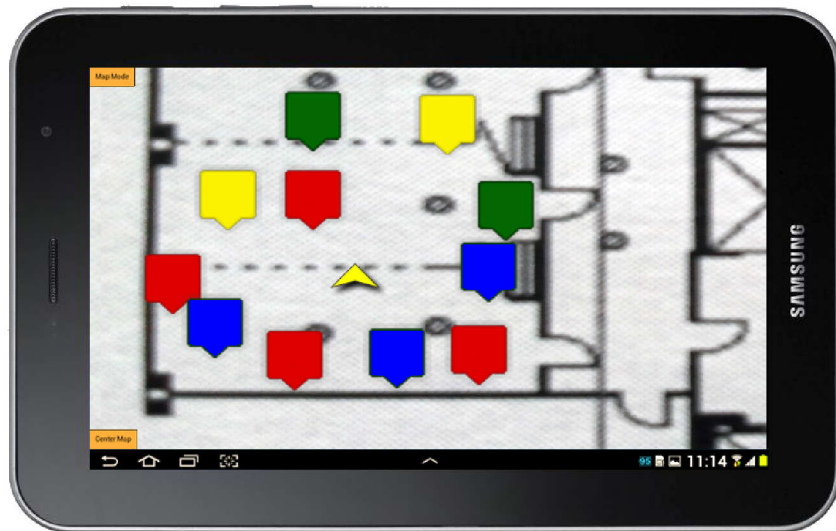


Figure 8.19: The study setup is visualized by UbiMap.



Figure 8.20: The study setup is visualized by UbiLens.

whether it is difficult for some people to match the command “right of it” to a map where the according icon is left of it. This is because the requested icons were on the bottom side of the map while the viewer’s location was in the center of the map.

- **Question 3: Point to the patient with the highest number and tell me the number!** With this question, we wanted to find out how the two visualization

8.4. Comparing Visualization Technologies of the eTriage Application

technologies support establishing an overview of a dataset and identifying single elements. Additionally, this question tested the matching from virtual object to physical object, which was the opposite direction as tested in Question 1.

The last two questions aimed at investigating how the visualization technologies support building up spatial memory about the data space. The participants were instructed as follows: "You will have to answer the last two questions without the help of the technology. Now, you may use the technology for trying to memorize the scene as good as you can. You don't have to remember numbers or text. Rather focus on locations or distribution of colors. Now, take as much time as you need for memorization and tell me when you are ready." Then, we finalized the interview with these questions.

- **Question 4: In which quadrant were the most reds?** The participants should divide the room into four equal parts (quadrants) and tell in which part the most red icons had been. We aimed at testing how well the visualization supports memorizing patterns. As can be seen in Figure 8.19, there were two red ones in the lower left quadrant, one in the upper left, one in the lower right and zero in the upper right.
- **Question 5: Describe the location of the three patients in the upper left corner of the room!** This question had to be answered from a position of the room where the participants were unable to see that part of the room because a wall restricted the view. The participants could describe the topology of the patients' locations in words (e.g. "they formed a right-angled triangle") or they could just reproduce the locations with their fingers on the table. With this question, we wanted to find out how good shapes and topology can be remembered.

8.4.2 Results

Question 1: Which color does this patient have? (Interviewer pointed at a patient)

Every UbiLens user answered the question correct whereas three UbiMap users answered wrong. The reason of this was the following. We intentionally handed over the tablet to each participant in a way that the map was rotated by 180 degrees to the participant's orientation. So, for the UbiMap participants, the first task was to find out that the map's orientation was wrong and then how the correct orientation of the map was. As Figure 8.19 shows, the correct orientation of the map could be determined by the doors' location, among other hints. Three UbiMap users did not notice that the map's orientation was wrong. This shows that the matching of a map to an unknown physical environment is a challenge, which is not present for the UbiLens view.

For the further questions, we do not consider the wrong orientation of the map as wrong answers. So, for example, when asking the participant to point to the highest number, we consider a wrong answer as correct if the map would have been oriented correct before.

Question 2: Which value does the one from your perspective right of the patient from Question 1 have?

One of the UbiMap participants stepped into our trap and answered about the icon on the right hand side of the map, which was on the left hand side in reality. Two other UbiMap users replied the value of the same person they were already referring to in Question 1. The same mistake also happened to two UbiLens users. It might be that they already forgot which icon they identified in Question 1 and took the wrong one as reference point for this question. An additional fourth UbiMap user gave a wrong answer that we were not able to understand.

To summarize, our assumption that it might be difficult to match commands such as "left of" or "right of" to a map cannot be confirmed since only one of eleven map users seemed to have that problem. On the other hand, we identify the issue that people have problems with remembering a particular virtual icon. There is no difference between the two visualization technologies regarding this fact.

Question 3: Point to the patient with the highest number and tell me the number!

Three UbiLens and two UbiMap users did not identify the correct highest number. But everybody pointed to the correct patient that was associated to the icon she identified as highest temperature. So, the matching from virtual to physical does not seem to be an issue. This confirms the finding from Question 1 in the opposite direction. Of course, the same problem with the map orientation appeared and is considered as it is described in scope of Question 1.

Question 4: In which quadrant were the most reds?

Six UbiMap users answered correct. One answered upper left, one upper right, two lower right and one could only say that it had to be one quadrant at the lower side. Only three UbiLens users answered correct. Two answered upper left, one upper right, one lower right and four said it had been equally distributed.

So, in general UbiMap seems to support memorization of clusters in areas better than UbiLens. A possible reason for this is that the map provides an additional way in which the data can be memorized. For both visualizations, it is possible to remember the real scene and memorize the virtual data connected to the real objects. The map provides an additional abstract representation of the scene in which the data is already included.

The fact that four UbiLens viewers thought the reds were equally distributed supports the finding that UbiLens is not well suited for identifying or remembering spatial patterns.

Question 5: Describe the location of the three patients in the upper left corner of the room!

Two UbiMap users answered correct. Three users described or showed the locations as triangle, but not as right-angled. One put them in a line and the two others could not answer the question. The three users which held the map upside down were difficult to assess since they described a different topology. This topology was not as clear as the intended one since it was not as remarkable as a right-angled triangle. However, the fact that the participants did not notice the difference to the locations of the real world objects lets us assume that they did not really match the map against the real world. Five UbiLens users answered correct. Two described a non-right-angled triangle and two could not answer the question. Two more arranged them in a line and one of them described four patients.

Substantially more UbiLens users answered correct. However, most UbiMap users described at least a triangle but not with the correct angle. This supports an insight from the analysis of map psychology where it is found that the memory of angles in maps is prone to distortion [Tversky, 1993]. On the other hand, remembering topologies seemed to work well with maps.

Difference Between First Responders and Users Without Such a Background

We were interested whether first responders perform better in answering the questions since they may have more experience with similar setups and they might use some content meaning for memorization.

The first substantial difference between the groups was that four of the ten first responders did not identify the correct highest temperature number while only one of twelve non-responders answered wrong. Also eight (=80%) first responders did not answer the geometrical shapes correct while only seven (=58%) of the non-responders were wrong.

So, our assumption is not confirmed. To the contrary, the non-responders performed better than the first responders. We assume the tasks were too scientific and lacked relevance for the work of first responders. Another reason might be that the non-responders were more accustomed to the use of technology. So, the challenge for the practical relevance might be to train first responders of using the technical equipment.

Further Observations

One of the users who held the map upside down rotated the map in his hand when he turned around. Thus, the possibility of rotating the map crossed his mind but he did not realize that the initial orientation was already wrong.

Two users were not immediately aware that there are dummies all over the room although some of them were obviously placed. The interviewer had to tell them. This indicates the need for highlighting physical objects.

Three participants remembered and named all colors (although not being asked for it).

We did not explain how the two technologies had to be used. Nevertheless, none of the participants had any problems in handling our technologies and all used it in the intended ways. So, the use of both visualization approaches seems to be intuitive.

8.5 Conclusion

This chapter contributed several aspects to this thesis. It demonstrated UbiVis's practical feasibility, substantiated the findings of its validation, prepared a foundation for its evaluation, deepened the understanding of the problem scope and provided practical experience with the framework's application.

We developed two example applications supported by the UbiVis framework and connected different visualization technologies to it. Both examples were taken from our own project work, thus being practice-oriented. For both applications, we conducted a comparative user study. Creating this possibility is the main driver for the creation of UbiVis. It demonstrates the practical feasibility of the framework, not only for the actual creation of systems but also for its main use case.

We applied the same procedure for developing the examples as we did for the validation in Chapter 7. Its findings are corroborated since we were again able to perform rapid prototyping of these more complex ubiquitous annotation visualization applications. Since the applications are taken from different domains we conclude that the results from Chapter 7 are not restricted to a single domain. Rapid prototyping with UbiVis is likely to work domain independently. The validation results are also extended from two of the initial visualization libraries to all four libraries in particular. To generalize, the results are extended to all four ubiAV classes (cf. Section 3.3.1) because the initial libraries represent these classes (cf. Section 6.5).

Readers' understanding of the problems of comparative user studies with ubiquitous annotation visualization technologies should be deepened through the practical examples. At the same time, we have gained a sense of the practical application of the framework. The example applications will also be used for assessing UbiVis in Chapter 9.

Chapter 9

Evaluation

In Chapter 7 and Chapter 8 we have validated that UbiVis meets feature requirements to support rapid prototyping of ubiAV applications. Visualizations can be exchanged and validated without an application code change, which avoids time-consuming and cumbersome application adaption. Developers can use provided visualization libraries so that familiarization with the particular visualization technology is not necessary. Neither needed is process knowledge how to exchange visualizations, as developers can stick to a specified process.

However, several aspects still need to be assessed in order to find out whether UbiVis is more useful than rapid prototyping without a framework. First of all, UbiVis is only valuable if it supports many visualization technologies. Therefore, it specifies a process for extending it with new libraries. Similarly as we validated the application process of the framework, we still need to confirm that the extension process works. For this, we create a new library according to the extension process and connect it to both example applications (cf. Section 9.1). This demonstrates that the support of rapid prototyping by providing visualization technology is not restricted to the initial set of libraries.

Finally, we need to assess several usability qualities of UbiVis in order to evaluate whether the validated features make sense. UbiVis avoids time-consuming and cumbersome application adaption but this is only useful if the application of the framework itself is time-saving and simple. Not having to get used to visualization technologies is only advantageous if lengthy familiarization with the framework is not necessary. There are good chances that a predefined process for visualization exchange will be used if it is easily understandable. If working with UbiVis is perceived as not being complex, it can replace the complex rapid prototyping of ubiAV applications.

The remaining studies in this chapter assess these usability qualities. Only if they are positive, UbiVis is beneficial over rapid prototyping of ubiAV without a framework. In order to find out how UbiVis is perceived by its potential users, we conduct workshops in which experienced developers implement an application using the framework. Afterwards, the following evaluation techniques are applied.

1. In Section 9.2, we calculate a key figure for the usability of UbiVis and compare this to a benchmark. The standard questionnaire *System Usability Scale* is used

for this.

2. In order to understand the way how application developers assess UbiVis's usability in more detail, we investigate its pragmatic and hedonic quality using the *AttrakDiff* questionnaire. In Section 9.3, *AttrakDiff* also reveals first qualitative attributes which participants assign to UbiVis.
3. The *User Experience Questionnaire* presented in Section 9.4 further investigates qualitative attributes. Besides, it calculates key figures for subcategories of usability for complementing *AttrakDiff*'s results and a benchmark for assessing these key figures.
4. The above described evaluation techniques also reveal some qualitative measures, based on pre-defined sets of attributes. In Section 9.5, we give insights into what we have observed during the user workshop. This lets us infer further qualitative assessments more openly.
5. The qualitative assessments through observation are complemented by a focus group discussion presented in Section 9.6. This adds qualitative measures based on the participants thoughts about UbiVis.

The main objective of these techniques is to evaluate the above mentioned usability qualities. But the techniques cover other usability and user experience qualities as well. Therefore, the studies provide a general usability and user experience assessment of UbiVis. Finally, the evaluation techniques reveal concrete issues how the framework can be improved.

9.1 UbiTorch

In this section, we develop a further UbiVis library for a new kind of technology. Then, we integrate it into the two example applications. Being able to develop a new library and to integrate it into existing UbiVis applications substantiates the extensibility of the framework.

The miniaturization of projector hardware offers new opportunities for the research field of augmented reality. Projectors provide a direct kind of augmenting reality, which was traditionally performed in an instrumented environment using a fixed projector setup. By integrating projectors into smartphones, this direct augmentation becomes mobile. As most people carry their mobile phone with them at any time, the potential for projector phone applications for everyday use is high. This technology is used in ubicomp research as one alternative for implementing augmented reality solutions [Franzen et al., 2011].

First integrated projector phones are commercially available, such as the Samsung Galaxy Beam [Samsung Electronics, 2012]. Also pico projectors, such as the MicroVision SHOWWX+ laser projector [MicroVision Inc., 2014], can be connected to smartphones

with TV out functionality. In this setup, the smartphone serves as computing unit for dynamically controlling the projector's output.

9.1.1 Features

The core of our library is an Android application which allows users to project annotations onto physical objects. It supports pico projectors that are connected to an Android based smartphone. The user must hold the projection device in her hand and point at the physical objects in her environment. As soon as she is pointing at an annotated object, the annotation is projected onto the object (cf. Figure 9.1). This way, the user is scanning her environment for annotated physical objects. We call the library UbiTorch because its interaction style reminds us of scanning a dark environment with a torch.

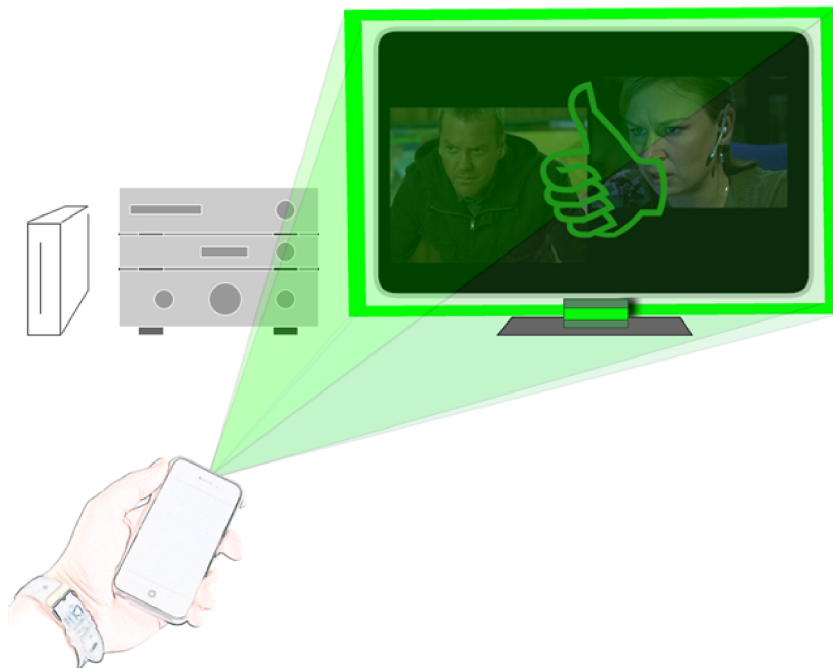


Figure 9.1: A projector reveals annotations when scanning physical objects by pointing at them.

UbiTorch detects when the user points at an annotated object. This is calculated from the object's location and the user's location and rotation. Optionally, the user's position can be determined via GPS. The UbiTorch library can also determine where exactly inside the projection area annotated objects are positioned.

UbiTorch graphically processes the digital information and places them at the correct location in the projection space so that the annotation is always projected onto the corresponding physical object. For example, if the user is pointing to the lower right

of an annotated object but the object is still inside the projection area, the graphic is displayed in the upper left part of this area. The part of the projection area where no annotation is placed is left transparent for the user. This is achieved by triggering the projector to project black color onto that area. Black is actually the absence of light, so this way the non-annotated area receives no unwanted light from the projector.

Following, we apply the procedure for developing a new library as described in Section 4.4.

Define Configuration Options

UbiTorch works in many regards similar to the calculation alternative of UbiLens. First of all, the physical object's position inside the projection area is also calculated from its real world location in connection with the device's location and compass rotation value. So, UbiTorch needs to know the real world position of each physical object. Hence, the configuration must arrange for a mapping of physical objects' IDs to latitude and longitude. The coordinate values are again expected in decimal WGS84 format [National Imagery and Mapping Agency, 2000]. Additionally, we want the application developer who is applying UbiTorch to be able to specify whether the user's location is acquired via GPS or a fixed position shall be used. So, we need two global parameters: Firstly, a boolean parameter indicating the use of automatic or fixed positioning. Secondly, a pair of fixed coordinates for the latter.

Also similar to the calculation alternative of UbiLens, we let the UbiTorch user define the background color and text color of an annotation. Hence, the configuration arranges for a mapping of annotation ID to a background color and a text color. The tone can either be indicated by name¹ or in the format "#RRGGBB" where the letters are replaced by the 2 byte hex code of the red, green and blue hue.

Figure 9.2 summarizes UbiTorch's configuration options. All parameters are mandatory.

Set up Configuration File

Listing 9.1 shows the configuration file, which is based on the definitions in Step 1. Every parameter is exemplarily added once. The contents are filled with example values.

Listing 9.1: UbiTorch's configuration file template

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>1</lowestId>
      <highestId>1</highestId>
      <latitude>50.749612</latitude>
```

¹[http://developer.android.com/reference/android/graphics/Color.html#parseColor\(java.lang.String\)](http://developer.android.com/reference/android/graphics/Color.html#parseColor(java.lang.String))

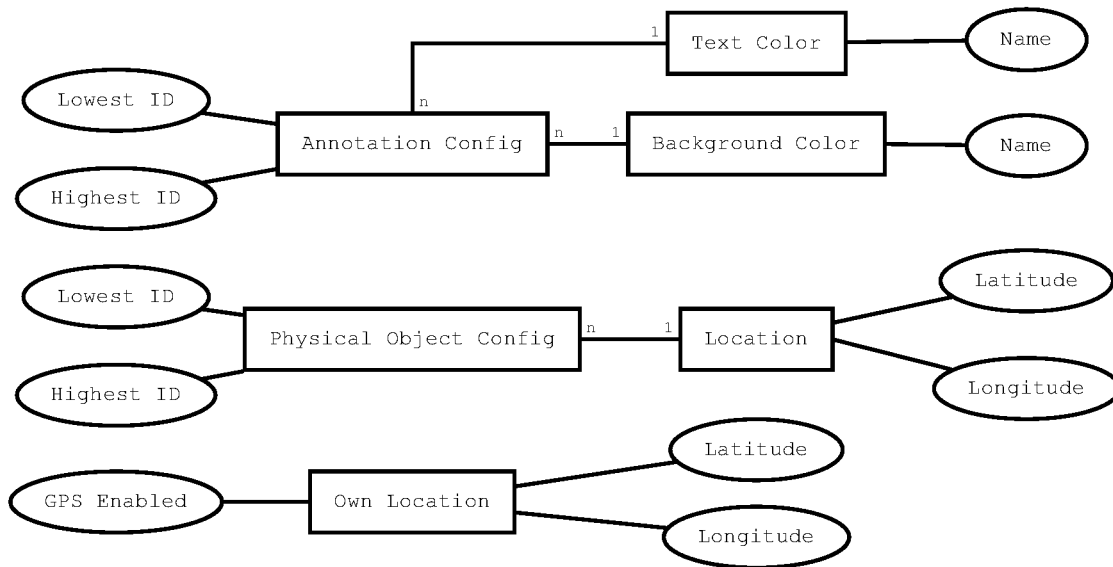


Figure 9.2: Configuration options of UbiTorch

```

    <longitude>7.203817</longitude>
  </PhysicalObjectConfig>
</physicalObjectConfigurations>
<annotationConfigurations class="linked-list">
  <AnnotationConfig>
    <lowestId>1</lowestId>
    <highestId>10</highestId>
    <backgroundColor>
      <name>#64FF30</name>
    </backgroundColor>
    <textColor>
      <name>black</name>
    </textColor>
  </AnnotationConfig>
</annotationConfigurations>
<ownLocation>
  <gpsEnabled>>false</gpsEnabled>
  <latitude>50.749619</latitude>
  <longitude>7.20387</longitude>
</ownLocation>
</Configuration>

```

Develop Visualization Component and Proxy

The UbiTorch visualization component is an Android application which generates all graphics on the smartphone's screen. Visual output by the projector is not explicitly triggered. Therefore, UbiTorch supports hardware setups which copy the normal screen visualization to the projector. These are Android smartphones, which either display the same content on the touchscreen and on the projector or disable the touchscreen when the projector is enabled. Most currently available Android smartphones with TV out feature behave this way.

The UbiTorch visualization component is oriented towards the implementation of UbiLens (cf. Section 6.1). It also makes use of the AndroidAR library for administering physical objects and the creation of their annotation's graphical presentation. For simplicity reasons, the graphical design of annotations is adopted from UbiLens.

Some graphical elements from UbiLens must be modified for UbiTorch. While the annotations can be visualized in the same way, UbiTorch does not need a compass and grid. Also, showing the video camera image does not make sense for UbiTorch. All these parameters can be customized in AndroidAR. Compass and grid can be disabled in order to not being painted. The video image can be replaced by a black background.

For UbiLens, the calculation of the annotation graphic's position on the screen makes use of all three rotation axes of the phone, i.e. pitch, roll and yaw value. This is because we can expect the user holding the smartphone in the line of sight between her eyes and the annotated object. For the UbiTorch setup, this expectation is not valid anymore. The user can hold the projector at different heights, resulting in different roll values. So, the roll value must be set to a fixed number, resulting in the graphic being always displayed on the same height inside the display. Hence, only its left or right position and rotation on the screen are still being adapted to the position of the physical object. So, our initial plan to always visualize the annotation on the physical object in the projection space cannot be completely fulfilled. However, first user tests with the setup showed that the users were intuitively correcting the missing height adaptation of the graphic by turning the projector higher or lower. The missing adaptation of the roll value even turned out to be an advantage for the handling. Some objects do not reflect projected light well so that the annotation is hard to read. In this case, the users projected the annotation slightly above or below that object, e.g., onto a wall that provides better reflection characteristics. We apply the above mentioned rotation adaptation by adding a filter to the sensor acquisition part of the AndroidAR library.

9.1.2 Applying to Energy Efficiency and eTriage

For integrating an additional library to an existing application, an application developer usually has to apply some of the rapid prototyping steps introduced in Section 4.3. The configuration file of the library must be analyzed for finding out about configuration options and mappings. In some cases, the format of physical objects' and annotations' IDs must be adopted to the new configuration options. Finally, the configuration file for the new library has to be edited.

For the eTriage application, we can speed up these configuration steps for UbiTorch because it takes over the configuration options of the calculation alternative of UbiLens. So, we can simply copy UbiLens's configuration file to UbiTorch and remove the alternative configuration (cf. Listing 9.2).

Listing 9.2: UbiTorch configuration file for the eTriage application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>244823674823</lowestId>
      <highestId>244823674823</highestId>
      <latitude>58.952373</latitude>
      <longitude>5.732283</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <annotationConfigurations class="linked-list">
    <AnnotationConfig>
      <lowestId>1</lowestId>
      <highestId>100</highestId>
      <backgroundColor>
        <name>#CC0004</name>
      </backgroundColor>
      <textColor>
        <name>white</name>
      </textColor>
    </AnnotationConfig>
    <AnnotationConfig>
      <lowestId>101</lowestId>
      <highestId>200</highestId>
      <backgroundColor>
        <name>#FFFA22</name>
      </backgroundColor>
      <textColor>
        <name>black</name>
      </textColor>
    </AnnotationConfig>
    <AnnotationConfig>
      <lowestId>201</lowestId>
      <highestId>300</highestId>
      <backgroundColor>
        <name>#037802</name>
      </backgroundColor>
      <textColor>
        <name>white</name>
      </textColor>
    </AnnotationConfig>
  </annotationConfigurations>
</Configuration>
```

```
    </textColor>
  </AnnotationConfig>
<AnnotationConfig>
  <lowestId>301</lowestId>
  <highestId>400</highestId>
  <backgroundColor>
    <name>#0000EE</name>
  </backgroundColor>
  <textColor>
    <name>white</name>
  </textColor>
</AnnotationConfig>
</annotationConfigurations>
<ownLocation>
  <gpsEnabled>true</gpsEnabled>
</ownLocation>
</Configuration>
```

For the energy efficiency application, we have to add another step. Since UbiLens was used with the image recognition option, we have to replace the photo filenames by locations of the appliances. These and the `ownLocation` setting can be copied from the energy efficiency's application's configuration of UbiMap. Analogously, we take over the color configuration of UbiLight for the `backgroundColor` configuration and set `textColor` for all annotations to black (cf. Listing 9.3).

Listing 9.3: UbiTorch configuration file for the energy efficiency application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>3781220497974521</lowestId>
      <highestId>3781220497974521</highestId>
      <latitude>52.200874</latitude>
      <longitude>8.600578</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <annotationConfigurations class="linked-list">
    <AnnotationConfig>
      <lowestId>1</lowestId>
      <highestId>100</highestId>
      <backgroundColor>
        <name>green</name>
      </backgroundColor>
      <textColor>
        <name>black</name>
      </textColor>
    </AnnotationConfig>
  </annotationConfigurations>
</Configuration>
```

```

    </textColor>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>101</lowestId>
    <highestId>200</highestId>
    <backgroundColor>
      <name>yellow</name>
    </backgroundColor>
    <textColor>
      <name>black</name>
    </textColor>
  </AnnotationConfig>
  <AnnotationConfig>
    <lowestId>201</lowestId>
    <highestId>300</highestId>
    <backgroundColor>
      <name>red</name>
    </backgroundColor>
    <textColor>
      <name>black</name>
    </textColor>
  </AnnotationConfig>
</annotationConfigurations>
<ownLocation>
  <gpsEnabled>>false</gpsEnabled>
  <latitude>52.198183</latitude>
  <longitude>8.582999</longitude>
</ownLocation>
</Configuration>

```

The final steps to do are to integrate the UbiTorch component and its visualization proxy bundle to the OSGi environment and to call the `visualize(visualizationTuple:VisualizationTuple)` method from the main application. Doing so for the energy efficiency application, UbiTorch projects the current power consumption close to an appliance when pointing the projector onto it (cf. Figure 9.3). The consumption value in Watt is projected as text. Additionally, the physical object is highlighted in red, yellow or green, which indicates whether the device consumes much, medium or little power. Applied to the eTriage application, the triage data is projected close to a patient (cf. Figure 9.4). The highlighting color refers to the triage category. The temperature is projected as text.



Figure 9.3: Visualizing an appliance's power consumption via UbiTorch



Figure 9.4: Visualizing patient information via UbiTorch

9.1.3 Summary

We developed a further UbiVis library by applying the procedure described in Section 4.4. This UbiTorch library utilizes mobile projectors for annotating physical objects in a torchlight metaphor. UbiTorch is connected to the before developed applications in a rapid prototyping manner. Hence, UbiVis’s rapid prototyping support is not restricted to the initial set of libraries. By following the described procedure, it can be enhanced by other visualization libraries. A new library can be seamlessly integrated in UbiVis applications in the same way as the initial libraries.

9.2 Quantitative Usability Assessment

UbiVis relieves application developers of writing a substantial amount of code (cf. Chapter 7). This is only useful if the application of the framework is perceived as being easy. Ease of use comprises several usability qualities. For investigating usability, subjective metrics have to be applied [Ludewig and Lichter, 2013]. So, in this and the following sections, we measure UbiVis’s usability through a set of standard questionnaires. The questionnaires were filled out by experienced ubiquitous application developers after they tested the framework.

9.2.1 System Usability Scale

For acquiring quantitative values, we use the *System Usability Scale (SUS)* of [Brooke, 1996]. The scale is a commonly used method for computing subjective usability assessment of participants to a single number. ”It has become an industry standard with references in over 600 publications” [Sauro, 2011].

After having tested a system, the participants have to answer 10 questions using a 5-point Likert scale. The answers are calculated to a value between 0 and 100. This allows comparing usability of systems which are very dissimilar. [Sauro, 2011] collected data from over 5000 users across over 500 system usability scale evaluations. He compiled them to a benchmark to which an own SUS value can be compared.

Usability assessments cover subjective opinions of effectiveness, efficiency and satisfaction of a system [International Organization for Standardization, 1998]. So, we expect important measurements for the assessment of our framework’s threshold.

The strength of SUS is that only 10 questions are needed, which makes the test very lightweight. The original SUS is intended to be applied for a software system. We slightly adjust some questions so that they fit to the evaluation of our framework. This is due to the audience of the framework, i.e., these are application developers instead of users who might need support by an experienced person instead of a technician. The adopted questions are (cf. Appendix D):

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.

3. I thought the system was easy to use.
4. I think that I would need the support of an experienced person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most application developers would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

9.2.2 Setup

15 experienced ubiquitous application developers were given a programming task to be fulfilled using the UbiVis framework. This way, the developers had the possibility to evaluate the framework based on their live experiences with it. After having completed the programming task, we collected qualitative and quantitative measures for assessing complexity, usability and understandability of the framework in order to assess its threshold. Due to scheduling reasons, the participants were distributed over two workshop dates.

The participants of the study belonged to one team of application developers working in the field of ubiquitous computing. They were allowed to use their own laptops for the programming task. The majority of them was working with LinkSmart, so their laptops were already configured with the necessary infrastructure to run UbiVis. For the others, a pre-configured virtual machine was provided. This meant especially that LinkSmart including the OSGi environment was set up but also a Java IDE and SVN client could be expected.

As programming task, the users were asked to create a system which displays the danger level for children of household articles. The particular danger levels and their representation were left to the application developers to decide. For visualization, they were asked to choose from the four initial libraries UbiLens, UbiMap, UbiLight and UbiBoard. After that, the second task was to choose another library and to add it to the application of the first task.

We conducted the study in a meeting room at the team's workplace. The room was about 30m² and therefore large enough that the visualization technologies could be employed. As household articles, we distributed a lamp, a book and a knife in the room. The necessary hardware was also already deployed to the room. For UbiLens and UbiMap, we provided Android smartphones and tablets. For UbiLight, a LED strip was

installed. Finally, a 40" LCD screen was situated at a wall for the use of UbiBoard.

Using a PowerPoint presentation (cf. Appendix B), we introduced motivation, features and mechanics of UbiVis to the participants. Then, we explained the procedure of rapid prototyping, which was described in Section 4.3. For intensifying understanding, we presented the sample application from Section 7.1. Finally, we provided the necessary administrative information. This consisted of SVN repository URLs where to find the visualization libraries, coordinates of the household articles and a given position in the room which could be used as static own location value. In addition, we provided sample images of the room's floor plan and icons for three different danger levels. All information could additionally be found in handouts (cf. Appendix C). So, the participants could have a look into the handout at any time for checking up details.

Following this introduction, we asked the participants to start the development and remained in the room for answering questions. After one hour, we asked to stop the development and delivered the SUS questionnaires.

9.2.3 Results

The calculated SUS score of UbiVis is 70. The average score of [Sauro, 2011]'s analyzed evaluations is 68. Hence, UbiVis's usability is above average. [Sauro, 2011] also provides a percentile rank, which allows associating SUS scores to letter grades. For example, a score above 80.3 is associated with grade A because this is among the top 10% of all scores. Figure 9.5 illustrates the percentile rank. According to this, UbiVis's SUS score is associated with the grade C.

9.3 Pragmatic and Hedonic Quality

SUS provides a fast but rough assessment of the overall usability of a tested system. But a system's assessment may be driven by different motivation. Ratings might be based upon how useful participants find it for a certain task. But also non-instrumental aspects might influence the rating. A system that gives pleasure without achieving a productive goal might be assigned a positive rating [Hassenzahl, 2005].

In order to find out how UbiVis's general assessment is composed in more detail, we provided another standard questionnaire. This divided general attractiveness into three questions: How successful do users achieve their goal using the framework? To what extent does it stimulate the user to move forward? To what extent does the user identify with the framework?

9.3.1 AttrakDiff

AttrakDiff is a standard questionnaire which captures users' perception of a product. In our case, this product is the UbiVis framework. Users are presented 28 pairs of opposite adjectives. They are asked to assign these to the product using a 7-point scale.

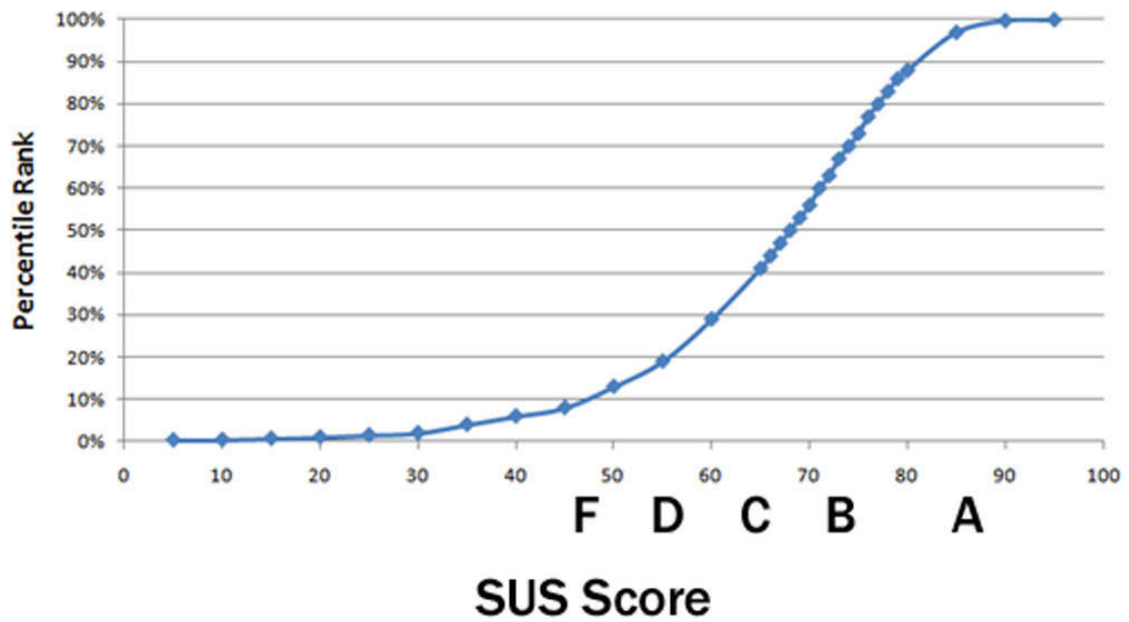


Figure 9.5: The percentile ranks of all analyzed evaluations of [Sauro, 2011] associated with SUS scores and letter grades

Several groups of adjective-pairs are evaluated to the following dimensions [Hassenzahl et al., 2003]:

- Pragmatic Quality (PQ):
Indicates how successful users are in achieving their goals using the product.
- Hedonic Quality - Stimulation (HQ-S): Mankind has an inherent need to develop and move forward. This dimension indicates to what extent the product can support those needs in terms of novel, interesting and stimulating functions, contents and interaction- and presentation-styles.
- Hedonic Quality (HQ-I):
Indicates to what extent the product allows users to identify with it.
- Attractiveness (ATT):
A global value that is based on the other dimensions. Hedonic and pragmatic qualities are independent of one another and contribute equally to the rating of attractiveness [Hassenzahl, 2001].

9.3.2 Setup

The AttrakDiff assessment was performed right after the SUS questionnaire (cf. Section 9.2). Study setup details can be found in Section 9.2.2.

The AttrakDiff questionnaire was filled out via online form (cf. Appendix E). A link with an individual access code was sent to each participant while they were completing the SUS questionnaire. Hence, they did not see the AttrakDiff questionnaire before. The online form's language could be set to German or English and the participants were asked to use the language they feel more comfortable in. Ten participants used the German version; the other five used the English one.

9.3.3 Results

The following results are interpreted with the help of the AttrakDiff online tool [User Interface Design GmbH, 2014]. The figures in this section are generated by the tool as well.

Figure 9.6 illustrates the results portfolio. Derived from the dimension values, UbiVis lies in the character-regions "self-oriented" and "desired". This results to an overall assessment of "rather desired" [Burmester et al., 2007].

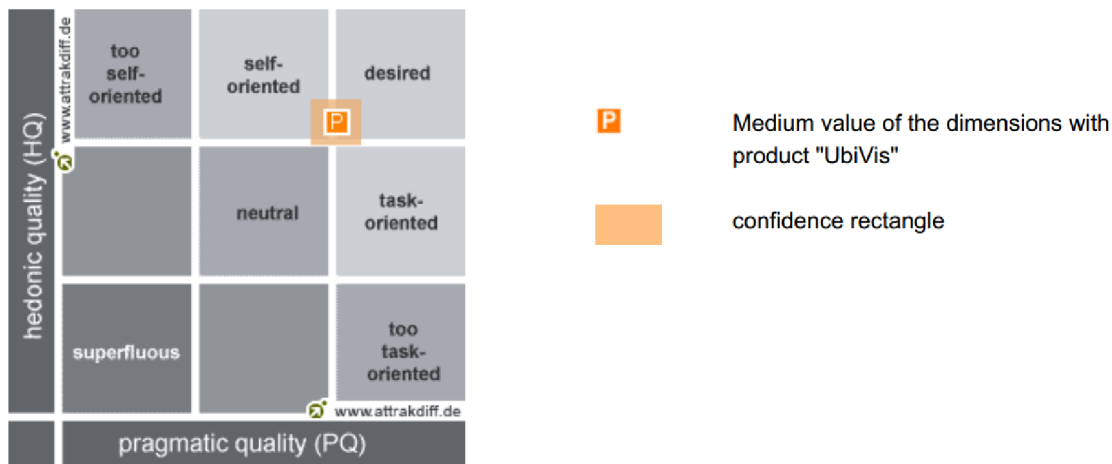


Figure 9.6: Portfolio with average values of the dimensions PQ and HQ and the confidence rectangle of UbiVis

The classification for PQ is not clearly "pragmatic" because the confidence interval overlaps into the neighboring character zone. That means the users are assisted by UbiVis; however the value of pragmatic quality only reaches the average values. Consequently there is room for improvement in terms of usability.

In terms of hedonic quality the character classification does not clearly apply because the confidence interval spills out over the character zone. So, the users are stimulated by UbiVis; however it is not certain that the hedonic value is above average. Hence, room for improvement exists in terms of hedonic quality.

The confidence rectangle states that the mean PQ and HQ value would be inside this area in a follow-up study under the same circumstances with a probability of 95%

[Hassenzahl et al., 2008]. The confidence rectangle is small. This is an advantage because it means that the investigation results are reliable and not coincidental. It shows that the users generally agree in their evaluation for both dimensions.

Figure 9.7 shows the average values of the four AttrakDiff dimensions. With regard to HQ-I, UbiVis is located in the positive region. It provides the user with identification and thus meets ordinary standards. The same applies to HQ-S. Also UbiVis’s attractiveness is located in the above-average region. This means the overall impression of it is very attractive.

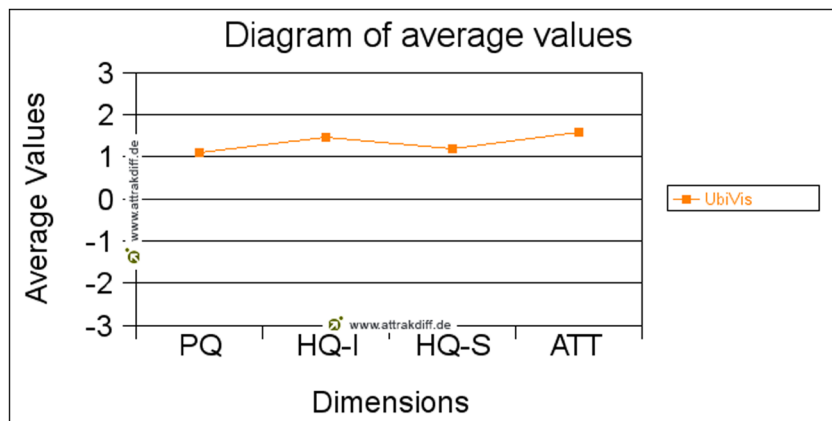


Figure 9.7: Mean values of the four AttrakDiff dimensions for UbiVis

Figure 9.8 lists the mean values of each adjective pair. Of particular interest are the extreme values as they show which characteristics are particularly critical or well-resolved.

The only adjective pair with a mean value on the negative left-hand side is "technical-human". Usually, the aim of software should be to appear as human as possible resulting in a more natural user experience. However, UbiVis is a framework for technical implementation, which was evaluated by applications developers. This explains why most participants assigned the property "technical" to it. Additionally, in this setting, "technical" might have a less negative connotation than when evaluating a software product by end users. Since this first adjective pair is used for the calculation of PQ, the actual PQ score might be better than presented.

Extreme values on the right-hand side, which have a mean value above 2, are "professional" (opposite of "unprofessional") and "good" (opposite of "bad"). "Professional" might mean that the concept is well conceived and the subparts fit to each other. "Good" is generic and may be interpreted as another summarized positive assessment of UbiVis.

There is a third point of view for identifying extreme values. Leaving out "technical-human", every adjective pair mean is on the positive side. So, the most negative categories are the ones closest to 0. These are "undemanding-challenging" and "ugly-

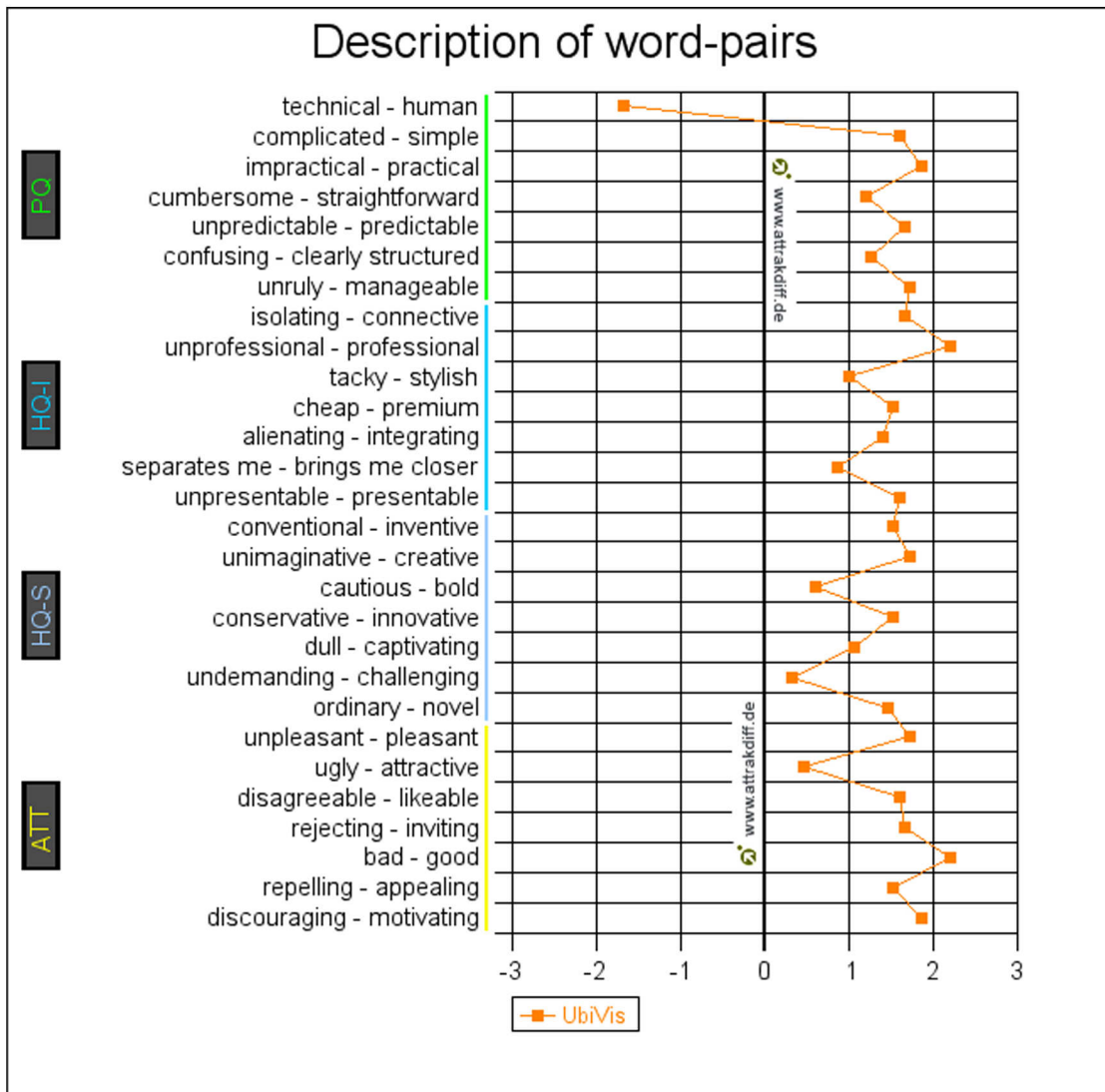


Figure 9.8: Mean values of each word pair associated with UbiVis. The colors illustrate of which word pairs each dimension is calculated from.

attractive”. We interpret the ”undemanding” assessment also as rather positive value. UbiVis is aiming to be easy to use. Hence, it shall not be challenging. The users might have rated UbiVis as not very attractive since visual elements are rarely used for the framework. As another interpretation, the provided code might be seen as not elegant. In any case, there is room for improvement regarding this aspect.

The discussion focused so far on the general assessment of UbiVis’s user experience. However, for evaluating whether the framework’s features make sense, we were especially

interested in the aspects time consumption, simplicity, ease of learning, ease of use and understandability. Positive evaluation is given especially for simplicity, ease of use and understandability through the positive assignment of the PQ adjectives.

9.4 Finding Attributes

AttrakDiff already divides the users' assessment of a product into pragmatic and hedonic quality. Pragmatic quality investigates how a product serves users in achieving a productive task. Hedonic quality is more about how people feel when using the product or what attributes they assign to the product. In order to find out more details about both qualities, we let our participants complete a third questionnaire. The particular questionnaire standard enables us to find out attributes that users assign to UbiVis. This lets us identify areas in which the framework is considered to work well and others where improvement is needed. In addition, we can explicitly handle the missing qualities of interest: time consumption and ease of learning.

9.4.1 User Experience Questionnaire

The User Experience Questionnaire (UEQ) is another standard questionnaire which captures usability and user experience perception of a product. It calculates scores for each of the following categories [Rauschenberger et al., 2013a].

- **Attractiveness:** General impression towards the product. Do users like or dislike the product? This scale is a pure valence dimension.
- **Efficiency:** Is it possible to use the product fast and efficient? Does the user interface look organized?
- **Perspiciuity:** Is it easy to understand how to use the product? Is it easy to get familiar with the product?
- **Dependability:** Does the user feel in control of the interaction? Is the interaction with the product confident and predictable?
- **Stimulation:** Is it interesting and exciting to use the product? Does the user feel motivated to further use the product?
- **Novelty:** Is the design of the product innovative and creative? Does the product grab users' attention?

Each category must be interpreted on its own; UEQ does not intend to compute an overall score from them. There is data available from 4818 persons from 163 studies [Schrepp et al., 2013]. As a benchmark, the result for the own product can be compared to the mean values for each category.

Similar to AttrakDiff (cf. Section 9.3.1), UEQ presents 26 pairs of opposite adjectives, which participants must assign to a product on a seven stage scale. Most UEQ adjective pairs are different to the ones used by AttrakDiff [Laugwitz et al., 2008].

9.4.2 Setup

The UEQ assessment was performed together with the SUS questionnaire (cf. Section 9.2) and AttrakDiff (cf. Section 9.3). The study setup details are presented in Section 9.2.2.

The UEQs were completed on sheets of paper (cf. Appendix F). The participants chose between an English and German version in the same configuration as described in Section 9.3.2.

We extended the questionnaire by six word pairs which we were especially interested in for UbiVis. The positive form of each pair is an attribute which UbiVis particularly aims to be. These adjectives pairs did not influence the calculation of the UEQ scores. Each of them was interpreted on its own. The pairs were added to the UEQ as follows.

- useless - useful
- expandable - unexpansive
- helpful - harmful
- difficult to use - easy to use
- time-consuming - time-saving
- complicating - facilitating

AttrakDiff focused on simplicity, understandability and partly ease of use, neglecting time consumption and ease of learning. So, we added the adjective pair "time-consuming - time-saving". The pair "easy to learn - difficult to learn" is already part of UEQ. The ease of use comprises several facets which are partly covered by AttrakDiff and UEQ. For corroboration, we added "useless - useful", "helpful - harmful", "complicating - facilitating" and the summarizing attribute pair "difficult to use - easy to use". Finally, we added "expandable - unexpansive" in order to find out whether users understand UbiVis's expansion concept.

9.4.3 Results

The following results and figures are based on the official UEQ analysis Excel sheet [Hinderks, 2014].

Figure 9.9 illustrates the scores of UbiVis for the UEQ categories. They are normalized to a scale ranging from -3 to 3. Values greater than 0.8 represent a positive evaluation of the corresponding dimension, which is indicated by green background. Values between -0.8 and 0.8 represent neutral evaluation (yellow) and values lower than -0.8 represent negative evaluation (red). Due to the calculation of means over a range of different persons with different answer tendencies and the potential avoidance of extreme answer categories, it is unlikely to observe values above 2 or below -2. That is why 0.8 is already considered being positive [Rauschenberger et al., 2013a]. The answer

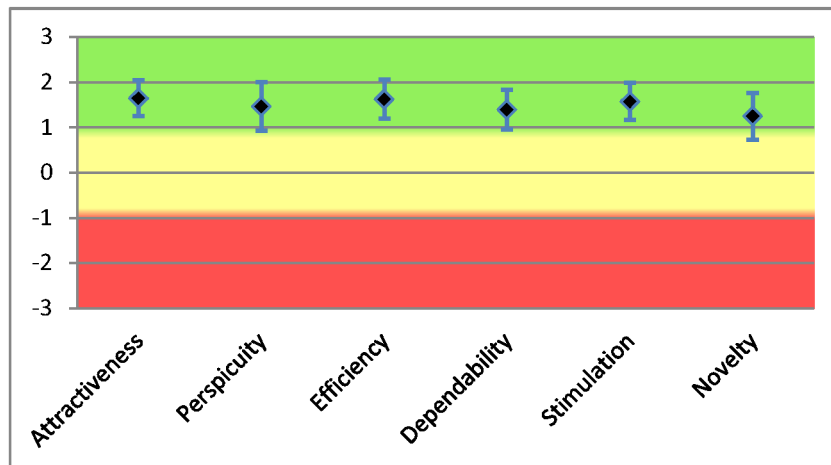


Figure 9.9: UEQ scores of UbiVis are visualized by black diamonds. Blue lines indicate standard deviations. Green is a positive evaluation, yellow a neutral evaluation and red a negative evaluation.

categories fall between 1.25 (novelty) and 1.64 (attractiveness), so we have a general, positive evaluation for UbiVis.

The dimensions' α coefficient [Cronbach, 1951] is between 0.71 (dependability) and 0.87 (attractiveness). α coefficients greater than 0.7 indicate a high correlation of items which belong to one scale [Rauschenberger et al., 2013a]. This means for UbiVis that the adjectives are understood correct and we can trust the UEQ scores.

Figure 9.10 puts the UbiVis result in relation to the results of 163 UEQ studies. Dependability is above average, which means that 25% of results are better and 50% of results are worse [Schrepp et al., 2013]. Attractiveness, perspicuity, efficiency and novelty are good, meaning that 10% of results are better whereas 75% of results are worse. Stimulation is even excellent, which means that it is in the range of the 10% best results.

From this benchmark, we can conclude that UbiVis is successful in all categories. It particularly achieves to motivate users to use it because it is regarded as interesting and exciting (cf. Section 9.4.1). Most improvement potential exists in terms of interaction confidence and predictability. Users do not optimally feel in control of interaction. However, the participants used UbiVis for the first time and only once. It can be expected, that this category would improve by using the framework more often.

Besides the UEQ scores, we also investigate the mean values of the individual adjective pairs. An overview of them can be found in Figure 9.11. The values are normalized to a scale ranging from -3 to 3. Also randomized polarity of adjectives is hidden in a way that the positive attribute is always assigned the high value on the right-hand side of the diagram. The colors indicate which category uses the particular item for calculating its

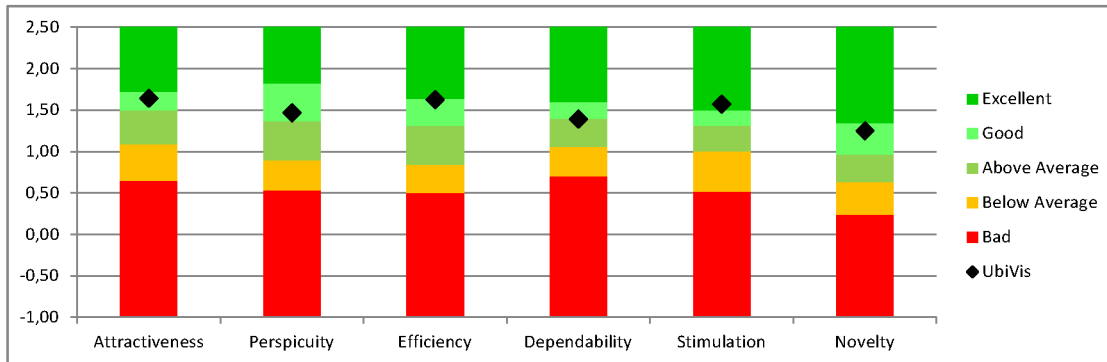


Figure 9.10: UbiVis results in relation to benchmark

score as follows. Orange: Attractiveness. Green: Perspicuity. Purple: Novelty. Yellow: Stimulation. Dark Blue: Dependability. Light Blue: Efficiency. Grey: Own added pairs, which are not used for a category.

At first, we check for outliers for each category. If a word pair showed big deviations to the other items on the same dimension, this would be a hint that it is misinterpreted by a high number of participants [Rauschenberger et al., 2013b]. This is an additional test to the α coefficients. All adjective pairs seem to be correctly understood because there are no outliers.

Mean values of single items can be interpreted in the same way as category scores: Values above 0.8 are positive, those below -0.8 are negative, the rest is neutral. We can further distinguish the positive evaluations. Because of the before mentioned reasons, a value above 2.0 is unlikely. Hence, those values are outstandingly positive [Rauschenberger et al., 2013a].

All items except two are positively evaluated. The adjective pair "fast - slow" (mean value: 0.7) clearly indicates that UbiVis users wish the framework to perform faster. However, it remains unclear whether the development process or the starting and runtime performance is meant. The other neutral adjective pair is "secure - not secure". This needs to be carefully interpreted. The study setup did not include security topics. Therefore, we did not mention LinkSmart's security features that are part of UbiVis. Hence, we understand the evaluation of this item as neutral in the sense that the participants could not say anything about it. Consequently, we cannot derive conclusions for improving UbiVis from this.

Remarkably, six items got an outstanding mean value of 2.0 or higher. These are the efficiency items "impractical - practical" (2.2) and "organized - cluttered" (2.1) as well as four of our own word pairs. In particular, these are "useless - useful" (2.0), "helpful - harmful" (2.2), "time-consuming - time-saving" (2.1) and "complicating - facilitating" (2.1). Hence, UbiVis is perceived as especially practical, organized, useful, helpful, time-saving and facilitating.

The remaining own items which got a good but not outstanding mean value are

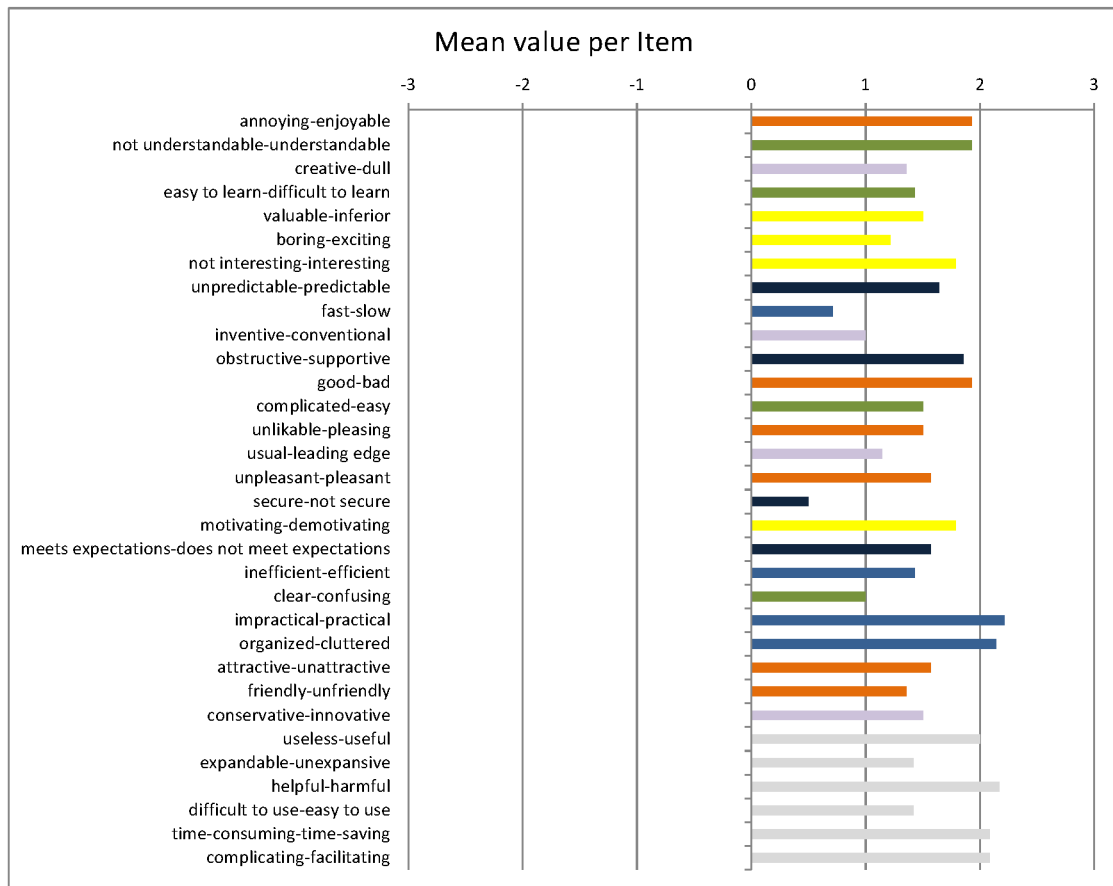


Figure 9.11: Mean values of word pairs associated with UbiVis. Colors indicate to which category a pair belongs as follows. Orange: Attractiveness. Green: Perspicuity. Purple: Novelty. Yellow: Stimulation. Dark Blue: Dependability. Light Blue: Efficiency. Grey: Own added pairs.

”expandable - unexpansive” (1.4) and ”difficult to use - easy to use” (1.4). UbiVis is expandable by new libraries. Either, this fact was not understood well enough by the participants and must be made clearer. Or, users request expandability in other regards. The ”difficult to use - easy to use” evaluation supports the fact that stimulation received the worst benchmark result among the categories. As we mentioned above, this can be due to the participants using UbiVis for the first time and only once. In any case, it means that efforts can be put in improving controllability in general and in facilitating the first steps.

9.5 Qualitative Usability Assessment Through Observation

The UEQ questionnaire revealed several attributes that users assign to UbiVis. The attributes originate in a list that is created by us. In order to find other qualitative attributes more openly, we employ two further evaluation methods. During the first one, we observe the participants while they are performing the requested tasks. This aims at corroborating the users' perception of UbiVis's usability by the practical application.

9.5.1 Observation

Observation is a method in which researchers gather data by watching users at performing a task [U.S. Department of Health & Human Services, 2008]. In overt observation the observed person knows that she is being observed. This involves the problem that people might behave different to their normal behavior when they know they are being observed [Patton, 1987]. They might feel forced to behave "as it is expected". This problem does not exist in covert observation where the participant does not know that she is observed. However, this implies ethic issues.

The researchers note interesting observations during the execution. Using a video camera for observation allows for repeated post-analysis of the study.

The advantage of the observation technique is that the researchers see what participants do. This is more direct than letting them talk about it. People would not tell what they take for granted but what would be interesting for the researcher [Mills, 2010]. Additionally, a phenomenon is investigated in its natural setting [Kumar, 1987] when using the observation technique.

Observation is useful when the researcher expects finding characteristics that are not clear to a participant so that they would not be revealed in interview-like techniques. Due to the natural setting, findings from the interplay of user, environment and system can be made.

9.5.2 Setup

The observation presented in this section was conducted during the study described in Section 9.2.2. While the application developers were working on the given task, they were overtly observed by the creator of the framework, who acted as the study's moderator. The participants were also allowed asking questions to the moderator. The moderator helped with the particular issue by giving hints instead of presenting a completed solution. In this setting, biased behavior is unlikely because there are no expectations how to behave correctly when using a framework.

Since the session was video recorded, these questions were also analyzed for qualitative assessments of the framework afterwards.

9.5.3 Results

Eleven out of 15 participants successfully completed the tasks. This is a high number considering that all used UbiVis for the first time and never heard of its concept before. Although the programming task is simple, UbiVis's concept and technology have to be understood for completing the task. Most of the participants finished it using the development guideline and asked questions to the moderator from time to time. However, two participants first read the complete handout in order to understand the UbiVis concept before starting to implement. Both participants successfully completed the tasks without any help. These observations let us conclude that concept and technology is easy to understand, even when using it for the first time. This seems contradictory to the results of UEQ (cf. Section 9.4.3) where users did not feel optimally safe in controlling UbiVis, especially for the first time. This might mean that users are confident with UbiVis but do not realize it. We might introduce more confirmation feedback and check its impact in a follow-up study. On the other hand, four participants did not successfully complete the tasks. For most of them, we could not find out the particular reasons for the failure in the short time of the workshop. For one of the participants, the reason was an activated VPN connection, which interfered with the communication between LinkSmart and the smartphone component. So, one way for improvement is to make UbiVis more stable by fixing incompatibilities with other technologies.

While most participants used the prepared virtual machine for working on the tasks (cf. Section 9.2.2), three participants used their own laptop environment. All three managed to complete the tasks. This shows that UbiVis integrates well in different environment configurations.

All except one participant worked concentrated on the task. There were no major distracting talks or other actions which would have let us conclude that the participants are not interested in the topic. So, UbiVis and its related concepts seem to be interesting and motivating. This supports our finding from UEQ (cf. Section 9.4.3) where UbiVis's stimulation properties are rated as excellent.

People had problems in remembering the IDs they assigned to a physical object in the configuration when they had to refer to them in the code. Some people wrote the allocation "ID $\hat{=}$ name of physical object" on a paper. Others entered the physical object's name as XML comment into the configuration file. The latter strategy can be adopted by the framework. The example configuration can contain an exemplary physical object' name, thus providing this as standard method.

9.6 Qualitative Usability Assessment Through Discussion

In Section 9.5, we searched for qualitative assessment of UbiVis's threshold by observing the participants. When observing participants in a natural setting, we can see what they really do. This may be different and more correct than when participants tell what they would do. However, observation does not give us insight into the users' thoughts about UbiVis. That is why we finalize our threshold assessment with a focus group discussion.

9.6.1 Focus Group

In the context of usability engineering, focus groups contain five to ten participants which discuss a software product [U.S. Department of Health & Human Services, 2014]. A moderator makes sure the discussion is kept on track. She introduces the topic and a set of goals for the discussion. In addition, she prepares not more than five open-end guideline questions. If the discussion is running out of scope, the moderator can use these questions to guide the participants back on track. However, the questions may be changed to adapt to the discussion.

The moderator's second task is to stop statements if they are misleading or to restrict one participant in dominating the discussion. At the same time, the moderator must interfere as sparse as possible to preserve the free flow of comments [Nielsen, 1993]. Reducing moderation also minimizes the researchers' influence to the discussion. The aim of a focus group is to learn about users' ideas and attitudes towards the software. A comfortable environment shall encourage the participants to speak out spontaneously and freely. Participants influence each other. Responding to statements of others can lead to more detailed ideas.

The "participants have certain characteristics in common that relate to the topic of the focus group" [Krueger and Casey, 2009]. This way, they discuss on the same level. For later data analysis, the group discussion should be taped.

9.6.2 Setup

The focus group discussion was the final session of the workshop described in Section 9.2.2.

After completing the questionnaires, the moderator initiated the group debate. He asked the group to discuss the main positive and negative aspects of the framework. The aim was to find out strengths and weaknesses of UbiVis.

We prepared the following four open-end questions as discussion guideline. Only Question 1 was finally asked.

1. For which use cases can you think of using UbiVis?
2. Which aspects were difficult to understand for you?
3. Assess UbiVis's conceptual framework!
4. Assess UbiVis's technical framework!

The session was video-recorded for later analysis.

9.6.3 Results

Most of the participants agreed that they found it difficult to fill out the questionnaires described in Section 9.2, Section 9.3 and Section 9.4. They thought this was due to having used UbiVis for the first time. Even when starting with a simple example, getting used to

a new framework is mostly overwhelming for the time being. New concepts and workflows have to be understood. Because of this, the participants assumed to have judged UbiVis worse than necessary. They believed the real rating could only be revealed after having regularly used the framework because then concepts were better known and felt less complex. So the questionnaires' results represent a usability assessment of novice UbiVis users but not of professionals. Since the questionnaires turned out positive attributes and key figures, we conclude that UbiVis is already usable when using it for the first time. Its usability for regular work still has to be found out in a long term evaluation.

One aspect was mentioned in both workshops. UbiVis claims that visualizations can be exchanged without having to change the application logic. Although this is correct, it does not mean that no coding is required when connecting a new visualization. A reference to the library must be established, the `ClientOnlineListener` must be registered and the visualization must be triggered. The participants expected a different approach. They wanted to code the application for Task 1 as they did. But they expected to complete Task 2 solely through configuration of UbiVis. They proposed to add a configuration file to the main application, similar to those for the visualization libraries. This file could take over the coding tasks mentioned above. Some participants interposed that it was a matter of personal preferences whether to perform this task via code or configuration file. A third option, GUI-based configuration, was also possible and preferred by a set of developers. The participants argued this could be compared to GUI layout design for Android applications, which could be performed code-wise, per XML file or via design GUI.

Beyond that, it was criticized that different bundles had to be touched. While most actions are required at the main application, the configuration is located at the visualization bundles. The participants considered it being more elegant and usable if everything would be located at the same place. They wanted to configure the visualizations at the main application and proposed to find some mechanism that enabled to transfer this data to the particular bundles.

Some participants suggested introducing physical object's names in the configuration file so that it is easier to remember which ID is assigned to them. This confirms our finding from the observation (cf. Section 9.5.3).

One participant complained that the framework was limited to the given visualization hardware. Thereupon, the moderator again explained the framework's extensibility by libraries which support other hardware. The participant agreed and rejected his critic. As consequence, the relatively low rating of UbiVis's expandability in Section 9.4.3 might be explained by the fact that some participants did not understand the expandability concept. On the other hand, another participant doubted enough library developers would provide libraries to the public. This needs to be investigated in a long term evaluation.

When asked for which use case UbiVis should be used, the most prominent answer was for rapid prototyping. This might be biased because it is touched in the motivation part of the framework's introducing presentation (cf. Appendix B). However, it is not explicitly mentioned. In any case, this answer confirms our motivation that UbiVis is

suited for rapid prototyping of ubiquitous computing applications. While our aim is especially to exchange divergent visualizations for a given application, one participant thought of rapid prototyping with UbiVis in a contrary way. He is dealing with diverse aspects of ubiquitous computing applications but has less expertise in visualization. So, he would apply the framework to add a graphical user interface to his apps. Thus, he can then concentrate on his actual task and let UbiVis take care of the visualization. Another participant mentioned that he would use UbiVis with a Google Glass² library. This would help him explore a new technology without having to actually learn its API.

Some opinions led to discussions among the participants. Some agreed while others disagreed so that no common conclusion could be found. Some participants found OSGi too complex and eclipse cumbersome and suggested to instantiate the good concept on "another technology". They referred to application developers that do not have OSGi experience. Others replied that other technologies would bring different problems and developers would have to become acquainted to new technology anyways.

One participant criticized the modeling of location to be left to each library. Three of the four initial libraries intend to model physical objects locations. So, the participant claimed the modeling of locations to be important enough for UbiVis to create an own method for it that every library can apply. Other participants replied that a framework probably could not specify a generic location model. Location models are too diverse, e.g., because of inconsistent dimensions or reference systems.

There were several individual opinions which were neither rejected nor confirmed nor discussed by the other participants. One found the `Annotation` object constructor with three nullable data parameters non-elegant and inflexible. Instead, he proposed the constructor to accept data as instance of the Java class `Object`. Since each other data type is derived from `Object`, this would remove the restriction of predefined data types. While this adds flexibility for application developers, visualization library developers would have the problem of having to deal with an unknown set of data types. Then, grouping through highest and lowest ID was described by one participant as efficient but difficult to understand for the first time. Next, one participant found the use of different IDs for physical object and annotation confusing. Finally, a participant stated to dislike frameworks which hide functionality because he could not understand and control every part.

9.7 Conclusion

As the final validation, we confirmed that the framework can be extended with new libraries using the specified expansion process. For this, we created the UbiTorch library and applied it to both example applications according to the framework's rules. An effective expansion concept lets the framework potentially support many visualization technologies. This is a requirement to make UbiVis useful.

We conducted a couple of studies in order to evaluate whether the UbiVis features are advantageous. Time-consumption, simplicity, ease of learning, ease of use and un-

²<http://www.google.com/glass/start/>

derstandability were focused. Besides, we were interested in a general usability and user experience assessment as well as in concrete issues for improving UbiVis.

UbiVis's UEQ score for efficiency was good, which means that it is possible to use the framework fast and efficiently. The more detailed look into the attributes revealed that especially the efficiency seems to be good while there might be room for improvement in terms of speed. The explicit attribute "time-saving" even received an outstanding rating. To summarize, UbiVis is considered rather being time-saving than time-consuming. This makes the framework advantageous over rapid prototyping of ubiAV without a framework, which is perceived as being time-consuming.

Secondly, framework-less rapid prototyping is regarded as being cumbersome. The comparison process is complex. In contrast, we evaluated that working with UbiVis is thought of as being simple and straightforward. Thus, UbiVis can also be advantageous with regard to simplicity of rapid prototyping. It can hide the complexity of the comparison process. AttrakDiff explicitly contains the attributes "simple" and "straightforward", which both received positive values. UEQ adds the attributes "organized" and, on an outstanding level, "facilitating". However, simplicity can be improved, e.g., by completely outsourcing the exchange process to configuration files or by a GUI-based approach.

UbiVis received a good perspicuity value in UEQ, which implies that it is easy to get familiar with the framework. In addition, the explicit request for the attribute "easy to learn" was positively evaluated. A high number of participants successfully completed the given tasks after the short learning period. Most attributes and key figures were rated positively although the framework was used by the participants for the first time. To summarize, we assessed UbiVis as easy to learn. Omitting to get used to unknown technologies is a key feature of the framework. This is only beneficial if lengthy familiarization with the framework is not needed.

From the good perspicuity value of UEQ we can also infer that it is easy to understand how to use UbiVis. It comprises the attributes "clear" and "understandable". Most of the participants showed their understanding of UbiVis's concept and technology by successfully completing the tasks. Since the participants tried out the specified framework process for visualization exchange and described UbiVis as understandable, we consider the process itself understandable as well. In contrast, the expandability concept seemed to be not completely understood by everyone.

As further evaluation finding, UbiVis seems to be easy to use. This underlines the framework's improvement of the difficult rapid prototyping process of ubiAV. Ease of use has many different facets. AttrakDiff assesses UbiVis as predictable, clearly structured, manageable and rather undemanding. UEQ confirms these and adds further attributes: UbiVis is supportive, helpful and easy to use. The usability can be improved, e.g., by incorporating the physical objects' names in the configuration files. In addition to the above mentioned aspects, ease of use is reflected in the general usability assessment.

Overall, all relevant usability qualities are positive. This means that UbiVis is evaluated to be preferable to rapid prototyping of ubiAV systems without a framework. All findings are substantiated since they are confirmed by different evaluation techniques.

For the general usability and user experience evaluation of UbiVis, the used techniques revealed several quantitative, but also first qualitative, subjective usability assessments. UbiVis SUS score is above average and associated with grade C. The framework's pragmatic quality is ranked between the good and optimal area, its hedonic quality slightly better. Its overall impression is stated as being very attractive. Although the users essentially agree in their evaluation, AttrakDiff's results leave some uncertainty about the assessments being optimal. Hence, room for improvements might exist for both qualities. The framework's dependability is above average, its attractiveness, perspicuity, efficiency and novelty is good and its stimulation even excellent compared to the UEQ benchmark.

All attributes pairs of AttrakDiff and UEQ have a mean value on the positive pole. Extreme values on the positive side are "professional", "practical", "organized", "useful", "helpful", "time-saving" and "facilitating". Least positive (but still positive) are "attractive", "fast" and "secure".

For an open qualitative measurement, we presented the results of observing the developers during the workshop and of the following focus group discussion. UbiVis integrates well in preconfigured environments. In addition, it is interesting and motivating for its users. During the discussion some misunderstandings and difficulties during the completion of the questionnaire came up. These explain the suboptimal rating of qualities regarding UbiVis's expandability. The framework is considered suitable for rapid prototyping. It cannot only be used for the comparison of visualizations but also for providing a visualization for the system at all while the developer can focus on other tasks.

The sessions also revealed some concrete improvements to be made. Users will have fewer problems to remember the IDs they assign in the configuration if that file contains a comment XML element for the ID. Also, the interplay with other technologies must be investigated. Besides the chosen method for adding a visualization to a given application by coding, this task can be optionally performed through a configuration file or GUI-based. This would serve the personal preferences of different developers. Finally, some users prefer to centrally configure visualization libraries.

Chapter 10

Conclusion

The ubiquitous computing paradigm creates new challenges for application developers which cannot be solved by traditional concepts. The necessity to integrate the real into the virtual world demands new development mechanisms. Due to the increased number of possibilities more user tests must be conducted during the development phase in order to understand the impact of different visualization approaches. This leads to an additional need of prototypes which can be rapidly developed and easily exchanged. This thesis contributes solutions for some of the touched problems. The introduction of a new point of view on ubiquitous computing applications improves the understanding of that field by reducing complexity through generalization. The UbiVis framework actively supports developers in the rapid prototyping process. Its concept comprises a standard structure and workflow which developers can simply follow so that they can concentrate on the actual implementation process. It is supported by the technical framework that is needed for the practical application.

10.1 Contributions

This thesis contributes to the challenges of ubiquitous application developers as follows.

10.1.1 Definition, Classification and Generalization of Ubiquitous Annotation Visualization Systems

There are different overlapping views which cluster the field of ubiquitous computing. None of them focuses on the central aspect of visualizing virtual information about physical objects. We close this gap by providing our own definition: A ubiquitous annotation visualization (ubiAV) system visualizes digital information that is meaningfully related to a physical object in the user's context. The digital information in this scope is called annotation. The definition can be adapted for other modalities. The ubiAV idea interconnects with existing ubiquitous computing concepts and meanwhile provides a novel point of view.

In order to structure the field of ubiAV systems further, we generalize important characteristics. This enables us to define a classification for the field as well as to identify similarities and common challenges. Our taxonomy extends the mixed reality definition of [Milgram and Kishino, 1994]. The ubiAV classification involves two dimensions. Firstly, the predominant world must be determined. A ubiquitous annotation visualization system is classified as real world (RW)-based if the primary world being experienced is predominantly real. It is classified as virtual world (VW)-based if the primary world being experienced is predominantly virtual. Secondly, the extent of location-awareness must be assessed. A ubiAV system is classified as location-aware if the annotated object's location is modeled. It is classified as location-unaware if the annotated object's location is not modeled or the model is not used for annotation visualization.

A challenge for all ubiAV systems is to make the user understand to which object an annotation belongs. This firstly includes the problem of integrating the element from the subordinate into the predominant world. Furthermore, the question arises how to link annotation and object. Location-aware systems can establish the link through location, e.g., by placing object and annotation close to each other or in a line of sight. Location-unaware systems must establish the link on a logical level. However, intelligent setups also allow location-unaware systems to use location for the linking without having knowledge about objects' locations. Location-based linking is dependent on the mobility of the physical objects and the visualization device. If a physical object or annotation is composed of other objects or annotations, understanding the link between them is complicated.

A survey of ubiquitous annotation visualization approaches corroborates the relevance of its definition. We are also able to sort the presented works according to our classification.

10.1.2 Specification of a Conceptual and Technical Framework for Performing Rapid Prototyping of UbiAV Approaches

The UbiVis framework provides the conceptual and technical setting for supporting application developers in rapid prototyping of ubiAV applications. Its design is driven by decisions derived from the precedent generalization of ubiquitous annotation visualization systems. The concept comprises a software architecture, an application workflow and an extension workflow.

UbiVis encapsulates visualization libraries. Providing a standard interface for them allows for the exchange of visualizations without having to change application logic. Individual issues are handled via customized configuration files for each visualization library. A library consists of the component that performs the actual visualization and a proxy, which bridges between the particular visualization technology and the standard UbiVis runtime environment. A notification mechanism informs the application logic about a visualization client being online.

Triggering visualization expects a reference to a physical object and the digital information used as annotation. A model for the annotation is provided, consisting of different internal data types. Mapping annotations to other technology-specific types

is possible. UbiVis's data structure also allows developers to map ranges of objects or annotations to the same attribute. A library's configuration file is mapped to the data structure for distribution within the application.

Due to the flexible character of ubiquitous computing applications, UbiVis can be deployed on a lot of different hardware setups. The only fixed definition is a central computer hosting the application logic and at least one visualization proxy.

A standard procedure how to implement an application with UbiVis facilitates to get started with the framework. Following the four steps hides the complexity of the ubiAV development process. When sticking to the workflow, developers make sure their ubiAV application is prepared for rapidly exchanging visualization technologies. Intensive familiarization with the framework is not necessary.

A second workflow describes another four steps for application developers to implement own visualization libraries. When such a library is provided to the public, it can be applied by every UbiVis user and to every new or existing application that was implemented in the framework's rules. Providing this workflow makes UbiVis extensible for present and future visualization technologies. The threshold for providing a new library is low as no comprehensive knowledge about UbiVis or ubiAV is required.

In addition to the conceptual settings, the technical environment of UbiVis is specified. It is based on the LinkSmart middleware is. LinkSmart's proxy concept complements UbiVis's visualization proxy idea and connects heterogeneous protocols and resource constrained devices. OSGi is used as container for running the main components, providing loose coupling of components. The LinkSmart EventManager provides publish-subscribe eventing functionality. The NetworkManager takes over networking issues for distributed, heterogeneous architectures. For communication to LinkSmart-disabled devices, a MQTT based eventing exchange is proposed. A set of conventions and development rules assure a robust implementation process.

10.1.3 Provision of Visualization Libraries and Practical Application of the Framework

The conceptual and technical framework provides the environment for rapid prototyping of ubiAV applications. In order to make UbiVis ready for application, initial visualization libraries are provided. The libraries are exemplary for each of the identified ubiAV classes. Further visualization libraries can be based on these. Not having to implement visualization libraries themselves is a major contribution of reducing the complexity for developers to rapidly prototype ubiAV applications.

UbiLens is a library that supports see-through augmented reality on smartphones. The user sees reality through the camera pictures of her smartphone while annotations are overlaid. UbiMap presents a rotatable, dynamic street map on a smartphone. Physical objects are represented by colored markers. Clicking a marker reveals the annotation. Buildings' floor plans can be overlaid over the map, making UbiMap applicable for indoor use as well. UbiLight lets the elements of a flexible LED shine in different colors. This way, physical objects in front of the strip are annotated. UbiBoard controls a situated display for visualizing annotations of physical objects in the screen's environment.

Each object representation and its annotation are combined on one of six exchangeable tiles.

Two applications are implemented with the framework in order to substantiate its practical applicability. The first application helps users in behaving more energy efficiently. Different visualization approaches can be used to show the current energy consumption of household appliances. The second application supports the work of first responders for large scale emergencies. It visualizes the location and attributes of injured patients. For both settings, the impact of the visualizations on users can be compared in user studies. Both defined processes, applicability and extension of the framework are put into action.

10.1.4 Validation and Evaluation of the Framework

On the basis of an exemplary application, we show that UbiVis meets its main requirement of facilitating rapid prototyping of ubiAV applications. In the scope of the framework, different visualization technologies can be exchanged for an application without having to change program logic code. In addition, the visualization technology does not have to be implemented by the application developer. The rapid prototyping works domain independently and for all ubiAV classes. Sticking to the defined workflow ensures a proper application of UbiVis. Likewise, we show that the framework can be extended by applying the extension workflow. An additional library can be developed and integrated into the existing applications.

We assess the framework's usability and user experience through a set of quantitative and qualitative measurements. These are based on two workshops, where experienced ubiquitous application developers gain hands-on experience with UbiVis and provide feedback in several sessions.

UbiVis is considered time-saving and simple, which makes it beneficial over the replaced time-consuming and cumbersome application adaptation without framework. It is easy to learn in contrast to lengthy familiarization with many visualization technologies. UbiVis's application workflow is understandable so that there are good chances it will be used.

An SUS questionnaire calculates an overall key figure for the framework's usability, which can be compared to a benchmark. The SUS score of 70 means that UbiVis's usability is above average. An AttrakDiff questionnaire attests UbiVis a very attractive overall impression and evaluates it to be assisting and stimulating for users. UEQ also calculates key figures for UbiVis, which can be compared to a benchmark, but more fine-grained than SUS. According to these, UbiVis is in the range of the 10% best results regarding stimulation. Regarding attractiveness, perspicuity, efficiency and novelty, 75% of the overall results are worse. Dependability is still above average since 50% of the results are worse.

AttrakDiff and UEQ also reveal first general qualitative assessments by assigning average levels of adjectives to UbiVis. All attributes are on the positive pole. The most positive adjectives are "professional", "practical", "organized", "useful", "helpful", "time-saving" and "facilitating". The least positive ones are "demanding", "attractive",

”fast” and ”secure”. The presented results are probably even a little better in reality, since the debriefing of the workshop revealed some misunderstandings among the participants that might have biased the results.

A focus group discussion and observation session reveal further qualitative assessments. The framework is considered being suited for rapid prototyping. It is easy to understand, even when using it for the first time. UbiVis integrates well in different configurations. It is interesting and motivating.

Besides the general positive feedback, the user workshop also shows up some existing problems of UbiVis and proposes solutions which can be implemented in future work.

10.2 Future Work

The conceptual and technical UbiVis framework improves development of ubiquitous annotation visualization applications. However, several issues were touched within this thesis without being able to handle them. These are opportunities for future research.

Although receiving positive results in the qualitative evaluation sessions, UbiVis did not get the highest scores in all categories. The reasons for that have to be investigated while first indications have already been revealed during the user studies.

The technical framework is currently purely text-based. Application developers must write pure source code and configure XML files. This might be one reason why it is rated as rather ugly. According to the focus group discussion, some users prefer GUI-based tools over coding. So, creating a graphical toolkit for development tasks can improve the perception and acceptance of UbiVis. But the coding alternative should retain since other developers prefer this alternative.

The workshop participants bothered about rating UbiVis after having used it initially. After using a system for the first time, it is normal to feel overwhelmed, no matter of the actual complexity. Complicated aspects often feel simple after using the system regularly. So, users might feel less confident with the system than they actually are because they do not completely penetrate it after the first use. While it is important to know that UbiVis is nevertheless perceived as easy to use from the beginning, a long term study would additionally evaluate its usefulness for the daily work.

UbiVis provides security features through the use of the LinkSmart’s Crypto Manager and Trust Manager. Security aspects were not evaluated in our workshops. Hence, evaluating how useful the current security concept is perceived by application developers is future work.

Some cumbersome or non-elegant methods have been criticized in the user studies. Often, improvements for future releases were also proposed. For instance, the problem of remembering which ID is assigned to which physical object might be solved by adding a comment field to example configurations where the object’s name is stored. As another example, having all configurations at a central place would make the architecture clearer.

The initial visualization libraries are limited. The eTriage example shows the need for dynamic location handling (cf. Section 8.3). With the initial libraries, it is possible to visualize a snapshot of the patients’ distribution. Although this is already an improve-

ment compared to the traditional approach, it would be more useful if the patients' location would be updated at runtime. First responders would always have an updated overview of the emergency site. There are possibilities to solve this by libraries. For instance, the image recognition variant of UbiLens is capable of handling dynamic location data in the given framework. UbiBoard is an example of a location-unaware visualization technology which is not dealing with location at all. In Section 8.3.3, we proposed to specify a web service URI from which updated location data can be requested by a visualization library. However, it will be less cumbersome if UbiVis provides possibilities for dynamic location handling more openly. This can be achieved by extending the modeling of physical objects. The possibility to change attribute values at runtime can be introduced.

The framework validation has been conducted for a small number of libraries and domains. More UbiVis applications in other domains that use further libraries will substantiate the findings.

The practical usefulness of UbiVis is heavily dependent on the amount of available visualization libraries. Thanks to the extensibility concept, an unlimited number of libraries is theoretically possible. However, it remains unclear whether enough library developers would provide libraries to the public. A long-term evaluation may answer this question after releasing UbiVis to the public.

Own Publications

Publications contributing to this thesis

Al-Akkad, A., Reiners, R., Jentsch, M., and Zimmermann, A. (2011). Where to draw the line? Approaching a scale to negotiate in-situ civil involvement for the inquiry of crisis information. In Bødker, S., Bouvin, N. O., Lutters, W., Wulf, V., and Ciolfi, L., editors, *Proceedings of the 12th European Conference on Computer-Supported Cooperative Work*, London, UK. Springer

Eisenhauer, M., Prause, C. R., Jahn, M., and Jentsch, M. (2010). Middleware for wireless devices and sensors-energy efficiency at device level. In *Proceedings of the 7th Annual Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, pages 1–3. IEEE

Franzen, D., Avellino, I., Mauri, F., Jentsch, M., and Zimmermann, A. (2011). Magic wako - user interaction in a projector-based augmented reality game. In *Proceedings of the 3rd International Conference on Creative Content Technologies*, pages 24–28. IARIA

Jahn, M., Jentsch, M., Prause, C., Pramudianto, F., Al-Akkad, A., and Reiners, R. (2010). The energy aware smart home. In *Proceedings of the 5th International Conference on Future Information Technology*, pages 1–8. IEEE

Jahn, M., Schwartz, T., Simon, J., and Jentsch, M. (2011). EnergyPULSE: Tracking sustainable behavior in office environments. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, pages 87–96, New York, NY, USA. ACM

Jentsch, M., Prause, C. R., and Pauli, M. J. (2009). Interaction-by-Doing in der Kommissionierung. *PPS Management*, 14(1):18–22

Jentsch, M., Jahn, M., Pramudianto, F., Simon, J., and Al-Akkad, A. (2011a). An energy-saving support system for office environments. In Salah, A. A. and Lepri, B., editors, *Human Behavior Understanding*, number 7065 in Lecture Notes in Computer Science, pages 83–92. Springer, Berlin, Heidelberg, Germany

Own Publications

Jentsch, M., Jahn, M., Reiners, R., and Kirschenmann, U. (2011b). Collecting factors for motivating energy-saving behavior. In Sasaki, H., editor, *Proceedings of the 3rd International Conferences on Advanced Service Computing*, pages 1–5. IARIA, Curran Associates

Jentsch, M., Ramirez, L., Wood, L., and Elmasllari, E. (2013). The reconfiguration of triage by introduction of technology. In *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services*, New York, NY, USA. ACM

Lorenz, A. and Jentsch, M. (2010). The ambient media player: A media application remotely operated by the use of mobile devices and gestures. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, pages 1–10, New York, NY, USA. ACM

Müller, J., Jentsch, M., Kray, C., and Krüger, A. (2008). Exploring factors that influence the combined use of mobile devices and public displays for pedestrian navigation. In *Proceedings of the 5th Nordic Conference on Human-Computer Interaction*, pages 308–317, New York, NY, USA. ACM

Prause, C., Jentsch, M., and Eisenhauer, M. (2010). MICA - A mobile support system for warehouse workers. *International Journal of Handheld Computing Research (IJHCR)*, 2(1):1–24

Reiners, R., Zimmermann, A., Jentsch, M., and Zhang, Y. (2009b). Automizing home environments and supervising patients at home with the hydra middleware: Application scenarios using the hydra middleware for embedded systems. In *Proceedings of the First International Workshop on Context-Aware Software Technology and Applications*, pages 9–12, New York, NY, USA. ACM

Further Publications

Jentsch, M. (2007). A combined public-private navigation service. In Probst, F. and Kessler, C., editors, *Proceedings of the 5th Geographic Information Days*. ifgi-Prints, Münster, Germany

Jentsch, M. (2009). Spotlight - augmenting devices by pointing at them. In Magnenat-Thalmann, N., editor, *Proceedings of the International InterMedia Summer School*, pages 109–115

Jentsch, M. (2010). Towards usable projector phone prototypes. In Magnenat-Thalmann, N., editor, *Proceedings of the International InterMedia Summer School*, pages 10–12

Lorenz, A., Jentsch, M., Concolato, C., and Rukzio, E. (2010). A formative study on the use of mobile devices and gestures to control a multimedia application from the distance. In *Proceedings of the 15th Mediterranean Electrotechnical Conference*, pages 796–801. IEEE

Patti, E., Acquaviva, A., Abate, F., Osello, A., Cocuccio, A., Jahn, M., Jentsch, M., and Macii, E. (2012). Middleware services for network interoperability in smart energy efficient buildings. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 338–339. EDA Consortium

Reiners, R. and Jentsch, M. (2009). Discovery and interaction in smart environments. In van Steendam, G., editor, *Ambient Intelligence and Human Security*, pages 135–138

Reiners, R., Jentsch, M., and Prause, C. R. (2009a). Interaction metaphors for the exploration of ubiquitous environments. In van Steendam, G., editor, *Ambient Intelligence and Human Security*, pages 53–56

Simon, J., Jentsch, M., Hiltunen, J., Pazienza, G., Vaccarini, M., Giretti, A., Jahn, A., and Esquivias, V. (2014). An intelligent energy management system for sustainable public underground spaces. In *Proceedings of the World Sustainable Buildings Conference*, Barcelona, Spain. to appear

Bibliography

- [Aberer et al., 2006] Aberer, K., Hauswirth, M., and Salehi, A. (2006). The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical Report LSIR-2006-006, École polytechnique fédérale de Lausanne, Switzerland.
- [Abowd, 2012] Abowd, G. D. (2012). What next, ubicomp? Celebrating an intellectual disappearing act. In *Proceedings of the 14th International Conference on Ubiquitous Computing*, pages 31–40, New York, NY, USA. ACM.
- [Abowd et al., 1997] Abowd, G. D., Atkeson, C., Hong, J., Long, S., Kooper, R., and Pinkerton, M. (1997). Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433.
- [Abowd and Mynatt, 2000] Abowd, G. D. and Mynatt, E. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):29–58.
- [Adobe Systems, 2014] Adobe Systems (2014). Adobe Flash Professional. <http://www.adobe.com/products/flash.html>. Accessed: 21/02/2014.
- [Al-Akkad et al., 2009] Al-Akkad, A., Pramudianto, F., Jahn, M., and Zimmermann, A. (2009). Middleware for building pervasive systems. In *International Association for Development of the Information Society: International Conference Applied Computing*, pages 291–298.
- [Al-Akkad et al., 2011] Al-Akkad, A., Reiners, R., Jentsch, M., and Zimmermann, A. (2011). Where to draw the line? Approaching a scale to negotiate in-situ civil involvement for the inquiry of crisis information. In Bødker, S., Bouvin, N. O., Lutters, W., Wulf, V., and Ciolfi, L., editors, *Proceedings of the 12th European Conference on Computer-Supported Cooperative Work*, London, UK. Springer.
- [Amigo Consortium, 2007] Amigo Consortium (2007). INMIDIO user’s guide. Technical report, Inria.
- [Antenna International, 2013] Antenna International (2013). XP-Iris. <http://www.antennainternational.com/en/audio--multimedia-devices/xp-iris>. Accessed: 04/11/2013.

Bibliography

- [Arduino, 2014] Arduino (2014). Arduino. <http://arduino.cc>. Accessed: 16/06/2014.
- [Arroyo et al., 2005] Arroyo, E., Bonanni, L., and Selker, T. (2005). Waterbot: Exploring feedback and persuasive techniques at the sink. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 631–639, New York, NY, USA. ACM.
- [Azuma, 1997] Azuma, R. (1997). A survey of augmented reality. *Presence-Teleoperators and Virtual Environments*, 6(4):355–385.
- [Bajura et al., 1992] Bajura, M., Fuchs, H., and Ohbuchi, R. (1992). Merging virtual objects with the real world: Seeing ultrasound imagery within the patient. In *SIGGRAPH Computer Graphics*, volume 26, pages 203–210. ACM.
- [Balestrini et al., 2014] Balestrini, M., Bird, J., Marshall, P., Zaro, A., and Rogers, Y. (2014). Understanding sustained community engagement: a case study in heritage preservation in rural argentina. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2675–2684, New York, NY, USA. ACM.
- [Ballagas, 2007] Ballagas, R. (2007). *Bringing Iterative Design to Ubiquitous Computing: Interaction Techniques, Toolkits, and Evaluation Methods*. PhD thesis, RWTH Aachen, Germany.
- [Ballagas et al., 2007] Ballagas, R., Memon, F., Reiners, R., and Borchers, J. (2007). iStuff mobile: Rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1107–1116, New York, NY, USA. ACM.
- [Ballagas et al., 2003] Ballagas, R., Ringel, M., Stone, M., and Borchers, J. (2003). iStuff: A physical user interface toolkit for ubiquitous computing environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 537–544, New York, NY, USA. ACM.
- [Ballagas et al., 2005] Ballagas, R., Rohs, M., and Sheridan, J. G. (2005). Sweep and point and shoot: Phonedcam-based interactions for large public displays. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1200–1203, New York, NY, USA. ACM.
- [Bandelloni et al., 2008] Bandelloni, R., Paternò, F., and Santoro, C. (2008). Reverse engineering cross-modal user interfaces for ubiquitous environments. In Paternò, F., editor, *Engineering Interactive Systems*, number 5247 in Lecture Notes in Computer Science, pages 285–302, Berlin, Heidelberg, Germany. Springer.
- [Barton and Vijayaraghavan, 2003] Barton, J. J. and Vijayaraghavan, V. (2003). UBI-WISE, a simulator for ubiquitous computing systems design. Technical Report HPL-2003-93, HP Laboratories.

- [Bartram et al., 2010] Bartram, L., Rodgers, J., and Muise, K. (2010). Chasing the negawatt: Visualization for sustainable living. *Computer Graphics and Applications, IEEE*, 30(3):8–14.
- [Bäumer et al., 1996] Bäumer, D., Bischofberger, W. R., Lichter, H., and Züllighoven, H. (1996). User interface prototyping—concepts, tools, and experience. In *Proceedings of the 18th International Conference on Software Engineering*, pages 532–541. IEEE.
- [Beardsley et al., 2005] Beardsley, P., Van Baar, J., Raskar, R., and Forlines, C. (2005). Interaction using a handheld projector. *Computer Graphics and Applications, IEEE*, 25(1):39–43.
- [Beckmann and Dey, 2003] Beckmann, C. and Dey, A. (2003). Siteview: Tangibly programming active environments with predictive visualization. In Dey, A. K., Schmidt, A., and McCarthy, J. F., editors, *Proceedings of the 5th International Conference on Ubiquitous Computing*, number 2864 in Lecture Notes in Computer Science, pages 167–168, Berlin, Heidelberg, Germany. Springer.
- [Berti et al., 2004] Berti, S., Correani, F., Mori, G., Paternò, F., and Santoro, C. (2004). TERESA: A transformation-based environment for designing and developing multi-device interfaces. In *Extended abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 793–794, New York, NY, USA. ACM.
- [Bieber et al., 2013] Bieber, G., Haescher, M., and Vahl, M. (2013). Sensor requirements for activity recognition on smart watches. In *Proceedings of the 6th International Conference on Pervasive Technologies Related to Assistive Environments*, page 67, New York, NY, USA. ACM.
- [Bier et al., 1993] Bier, E., Stone, M., Pier, K., Buxton, W., and DeRose, T. (1993). Toolglass and magic lenses: The see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 73–80, New York, NY, USA. ACM.
- [Billinghurst, 2014] Billinghurst, M. (2014). Mobile AR Framework. <http://www.hitlabnz.org/index.php/products/mobile-ar-framework>. Accessed: 21/02/2014.
- [Bombara et al., 2003] Bombara, M., Cali, D., and Santoro, C. (2003). Kore: A multi-agent system to assist museum visitors. In *Proceedings of the Workshop on Objects and Agents*.
- [Brooke, 1996] Brooke, J. (1996). SUS-A quick and dirty usability scale. *Usability Evaluation in Industry*, 189:194.
- [Brown, 1996] Brown, P. J. (1996). The stick-e document: A framework for creating context-aware applications. *Electronic Publishing*, 9(1):1–14.

Bibliography

- [Burigat et al., 2005] Burigat, S., Chittaro, L., and De Marco, L. (2005). Bringing dynamic queries to mobile devices: A visual preference-based search tool for tourist decision support. In Costabile, M. F. and Paternò, F., editors, *Proceedings of the 12th International Conference on Human-Computer Interaction*, number 3585 in Lecture Notes in Computer Science, pages 213–226. IFIP, Springer.
- [Burmester et al., 2007] Burmester, M., Hassenzahl, M., and Koller, F. (2007). Engineering attraktiver Produkte - AttrakDiff. In Ziegler, J. and Beinhauer, W., editors, *Interaktion mit komplexen Informationsräumen*, pages 127–141. Oldenbourg, Munich, Germany.
- [Büscher and Mogensen, 2007] Büscher, M. and Mogensen, P. (2007). Designing for material practices of coordinating emergency teamwork. In *Proceedings of the 4th International Conference on Information Systems for Crisis Response and Management*.
- [Butz and Krüger, 2006] Butz, A. and Krüger, A. (2006). Applying the peephole metaphor in a mixed-reality room. *Computer Graphics and Applications, IEEE*, 26(1):56–63.
- [Cao and Balakrishnan, 2006] Cao, X. and Balakrishnan, R. (2006). Interacting with dynamically defined information spaces using a handheld projector and a pen. In *Proceedings of the 19th Annual Symposium on User Interface Software and Technology*, pages 225–234, New York, NY, USA. ACM.
- [Castellote, 2005] Castellote, G. P. (2005). DDS spec outfits publish-subscribe technology for the GIG. *COTS Journal*, 4.
- [Caudell and Mizell, 1992] Caudell, T. P. and Mizell, D. W. (1992). Augmented reality: An application of heads-up display technology to manual manufacturing processes. In *Proceedings of the 25th Hawaii International Conference on System Sciences*, volume 2, pages 659–669, Los Alamitos, CA, USA. IEEE.
- [Cheverst et al., 2003] Cheverst, K., Dix, A., Fitton, D., Friday, A., and Rouncefield, M. (2003). Exploring the utility of remote messaging and situated office door displays. In *Proceedings of the 5th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 336–341, New York, NY, USA. ACM.
- [Chou, 2003] Chou, C. (2003). Interactivity and interactive functions in web-based learning systems: A technical framework for designers. *British Journal of Educational Technology*, 34(3):265–279.
- [Chua et al., 2010] Chua, C. K., Leong, K. F., and Lim, C. C. S. (2010). *Rapid prototyping: Principles and applications*. World Scientific, Singapore.
- [Consolvo et al., 2008] Consolvo, S., McDonald, D. W., Toscos, T., Chen, M. Y., Froehlich, J., Harrison, B., Klasnja, P., LaMarca, A., LeGrand, L., Libby, R., Smith, I., and Landay, J. A. (2008). Activity sensing in the wild: A field trial of UbiFit

- garden. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1797–1806, New York, NY, USA. ACM.
- [Cowan et al., 2010] Cowan, L., Griswold, W., and Hollan, J. (2010). Applications of projector phones for social computing. In *Proceedings of Ubiprojection Workshop 2010*.
- [Craggs, 2014] Craggs, I. (2014). Really small message broker. <https://www.ibm.com/developerworks/community/alphaworks/tech/rsmb>. Accessed: 21/02/2014.
- [Cronbach, 1951] Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334.
- [Dahley et al., 1998] Dahley, A., Wisneski, C., and Ishii, H. (1998). Water lamp and pinwheels: Ambient projection of digital information into architectural space. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 269–270, New York, NY, USA. ACM.
- [Darby, 2006] Darby, S. (2006). The effectiveness of feedback on energy consumption. Technical Report 486, Environmental Change Institute, University of Oxford.
- [De Carolis et al., 2009] De Carolis, B., Mazzotta, I., Novielli, N., and Silvestri, V. (2009). Using common sense in providing personalized recommendations in the tourism domain. In *Workshop on Context-Aware Recommender Systems*.
- [Delicato et al., 2013] Delicato, F. C., Pires, P. F., and Batista, T. (2013). *Middleware Solutions for the Internet of Things*. SpringerBriefs in Computer Science. Springer.
- [Dent et al., 2008] Dent, B. D., Torguson, J., and Hodler, T. (2008). *Cartography: Thematic map design*. McGraw-Hill, New York, NY, USA.
- [Dey et al., 2001] Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166.
- [Dietz and Leigh, 2001] Dietz, P. and Leigh, D. (2001). DiamondTouch: A multi-user touch technology. In *Proceedings of the 14th Annual Symposium on User Interface Software and Technology*, pages 219–226, New York, NY, USA. ACM.
- [Dix et al., 2004] Dix, A., Finlay, J., Abowd, G. D., and Beale, R., editors (2004). *Human computer interaction*. Pearson Education, Harlow, England.
- [Donsez, 2010] Donsez, D. (2010). Aspire Wiki. <http://wiki.aspire.ow2.org/xwiki/bin/view/Main/Readers>. Accessed: 11/07/2014.
- [Dopping-Hepenstal, 1981] Dopping-Hepenstal, L. (1981). Head-up displays. The integrity of flight information. *IEE Proceedings F (Communications, Radar and Signal Processing)*, 128(7):440–442.

Bibliography

- [Ducatel et al., 2001] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., and Burgelman, J.-C. (2001). Scenarios for ambient intelligence in 2010. Technical report, European Commission.
- [Edwards et al., 2003] Edwards, W., Bellotti, V., Dey, A., and Newman, M. (2003). The challenges of user-centered design and evaluation for infrastructure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 297–304, New York, NY, USA. ACM.
- [Eisenhauer et al., 2010] Eisenhauer, M., Prause, C. R., Jahn, M., and Jentsch, M. (2010). Middleware for wireless devices and sensors-energy efficiency at device level. In *Proceedings of the 7th Annual Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, pages 1–3. IEEE.
- [Eisenhauer et al., 2009] Eisenhauer, M., Rosengren, P., and Antolin, P. (2009). A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *Proceedings of the 6th Annual Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, pages 1–3. IEEE.
- [Fitzmaurice, 1993] Fitzmaurice, G. (1993). Situated information spaces and spatially aware palmtop computers. *Communications of the ACM*, 36(7):39–49.
- [Fitzmaurice et al., 1995] Fitzmaurice, G., Ishii, H., and Buxton, W. (1995). Bricks: Laying the foundations for graspable user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 442–449, New York, NY, USA. ACM.
- [Franzen et al., 2011] Franzen, D., Avellino, I., Mauri, F., Jentsch, M., and Zimmermann, A. (2011). Magic wako - user interaction in a projector-based augmented reality game. In *Proceedings of the 3rd International Conference on Creative Content Technologies*, pages 24–28. IARIA.
- [Gama et al., 2011] Gama, K., Pedraza, G., Lévêque, T., and Donsez, D. (2011). Application management plug-ins through dynamically pluggable probes. In *Proceedings of the 1st Workshop on Developing Tools as Plug-ins*, pages 32–35, New York, NY, USA. ACM.
- [Gershenfeld et al., 2004] Gershenfeld, N., Krikorian, R., and Cohen, D. (2004). The internet of things. *Scientific American*, 291(4):76–81.
- [Greaves et al., 2008] Greaves, A., Hang, A., and Rukzio, E. (2008). Picture browsing and map interaction using a projector phone. In *Proceedings of the 10th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 527–530, New York, NY, USA. ACM.
- [Greaves and Rukzio, 2008] Greaves, A. and Rukzio, E. (2008). Evaluation of picture browsing using a projector phone. In *Proceedings of the 10th International Conference*

-
- on *Human-Computer Interaction with Mobile Devices and Services*, pages 351–354, New York, NY, USA. ACM.
- [Greenberg and Fitchett, 2001] Greenberg, S. and Fitchett, C. (2001). Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual Symposium on User Interface Software and Technology*, pages 209–218, New York, NY, USA. ACM.
- [Greenberg and Rounding, 2001] Greenberg, S. and Rounding, M. (2001). The notification collage: Posting information to public and personal displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 514–521, New York, NY, USA. ACM.
- [GSN-Team, 2009] GSN-Team (2009). Global sensors networks. Technical report, École polytechnique fédérale de Lausanne, Switzerland.
- [Gustafsson and Gyllenswärd, 2005] Gustafsson, A. and Gyllenswärd, M. (2005). The power-aware cord: Energy awareness through ambient information display. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1423–1426, New York, NY, USA. ACM.
- [Hartmann et al., 2006] Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. (2006). Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual Symposium on User Interface Software and Technology*, pages 299–308, New York, NY, USA. ACM.
- [Hassenzahl, 2001] Hassenzahl, M. (2001). The effect of perceived hedonic quality on product appealingness. *International Journal of Human-Computer Interaction*, 13(4):481–499.
- [Hassenzahl, 2005] Hassenzahl, M. (2005). The thing and I: Understanding the relationship between user and product. In Blythe, M., Overbeeke, C., Monk, A. F., and Wright, P., editors, *Funology: From Usability to Enjoyment*, pages 31–42. Kluwer Academic Publishers, Dordrecht, The Netherlands.
- [Hassenzahl et al., 2003] Hassenzahl, M., Burmester, M., and Koller, F. (2003). AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität. In Ziegler, J. and Szwillus, G., editors, *Mensch & Computer 2003*, pages 187–196. Teubner, Stuttgart, Leipzig, Germany.
- [Hassenzahl et al., 2008] Hassenzahl, M., Burmester, M., and Koller, F. (2008). Der User Experience (UX) auf der Spur: Zum Einsatz von www.attrakdiff.de. In Brau, H., Diefenbach, S., Hassenzahl, M., Koller, F., Peissner, M., and Röse, K., editors, *Usability Professionals*, pages 78–82. Fraunhofer IRB, Stuttgart, Germany.

Bibliography

- [Henze et al., 2008] Henze, N., Rukzio, E., Lorenz, A., Righetti, X., and Boll, S. (2008). Physical-virtual linkage with contextual bookmarks. In *Proceedings of the 10th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 523–526, New York, NY, USA. ACM.
- [Hinckley et al., 2000] Hinckley, K., Pierce, J., Sinclair, M., and Horvitz, E. (2000). Sensing techniques for mobile interaction. In *Proceedings of the 13th Annual Symposium on User Interface Software and Technology*, pages 91–100, New York, NY, USA. ACM.
- [Hinderks, 2014] Hinderks, A. (2014). UEQ-Online. <http://www.ueq-online.org/>. Accessed: 21/02/2014.
- [Hoffmann et al., 2007] Hoffmann, M., Badii, A., Engberg, S., Nair, R., Thiemert, D., and Matthes, M. (2007). Security, trust and privacy supported by context-aware middleware. In *18th WWRP Meeting-Multimedia Goes Mobile*.
- [Holleis, 2008] Holleis, P. (2008). *Integrating Usability Models into Pervasive Application Development*. PhD thesis, LMU Munich, Germany.
- [Holmquist, 2006] Holmquist, L. (2006). Tagging the world. *Interactions*, 13(4):51ff.
- [Holstius et al., 2004] Holstius, D., Kembel, J., Hurst, A., Wan, P.-H., and Forlizzi, J. (2004). Infotropism: Living and robotic plants as interactive displays. In *Proceedings of the 5th Conference on Designing Interactive Systems*, pages 215–221, New York, NY, USA. ACM.
- [Holzman, 1999] Holzman, T. (1999). Computer-human interface solutions for emergency medical care. *Interactions*, 6(3):13–24.
- [Hong and Landay, 2001] Hong, J. I. and Landay, J. A. (2001). An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2):287–303.
- [IBM, 2014] IBM (2014). MQ Telemetry Transport. <http://mqtt.org>. Accessed: 21/02/2014.
- [Institute of Electrical and Electronics Engineers Standards Association, 2000] Institute of Electrical and Electronics Engineers Standards Association (2000). IEEE Recommended practice for architectural description of software-intensive systems.
- [International Organization for Standardization, 1998] International Organization for Standardization (1998). ISO 9241-11 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability.
- [Ishii and Ullmer, 1997] Ishii, H. and Ullmer, B. (1997). Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 234–241, New York, NY, USA. ACM.

- [Jäger et al., 2008] Jäger, L., Jarke, M., Klamma, R., and Spaniol, M. (2008). Transkriptivität: Operative medientheorien als grundlage von informationssystemen für die kulturwissenschaften. *Informatik-Spektrum*, 31(1):21–29.
- [Jahn et al., 2010] Jahn, M., Jentsch, M., Prause, C., Pramudianto, F., Al-Akkad, A., and Reiners, R. (2010). The energy aware smart home. In *Proceedings of the 5th International Conference on Future Information Technology*, pages 1–8. IEEE.
- [Jahn et al., 2009] Jahn, M., Pramudianto, F., and Al-Akkad, A. (2009). Hydra middleware for developing pervasive systems: A case study in the eHealth domain. In *1st International Workshop on Distributed Computing in Ambient Environments*, pages 13–21.
- [Jahn et al., 2011] Jahn, M., Schwartz, T., Simon, J., and Jentsch, M. (2011). Energy-PULSE: Tracking sustainable behavior in office environments. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, pages 87–96, New York, NY, USA. ACM.
- [Jentsch et al., 2011a] Jentsch, M., Jahn, M., Pramudianto, F., Simon, J., and Al-Akkad, A. (2011a). An energy-saving support system for office environments. In Salah, A. A. and Lepri, B., editors, *Human Behavior Understanding*, number 7065 in Lecture Notes in Computer Science, pages 83–92. Springer, Berlin, Heidelberg, Germany.
- [Jentsch et al., 2011b] Jentsch, M., Jahn, M., Reiners, R., and Kirschenmann, U. (2011b). Collecting factors for motivating energy-saving behavior. In Sasaki, H., editor, *Proceedings of the 3rd International Conferences on Advanced Service Computing*, pages 1–5. IARIA, Curran Associates.
- [Jentsch et al., 2009] Jentsch, M., Prause, C. R., and Pauli, M. J. (2009). Interaction-by-Doing in der Kommissionierung. *PPS Management*, 14(1):18–22.
- [Jentsch et al., 2013] Jentsch, M., Ramirez, L., Wood, L., and Elmasllari, E. (2013). The reconfiguration of triage by introduction of technology. In *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services*, New York, NY, USA. ACM.
- [Johanson et al., 2002] Johanson, B., Fox, A., and Winograd, T. (2002). The interactive workspaces project: Experiences with ubiquitous computing rooms. *Pervasive Computing*, 1(2):67–74.
- [Johnson, 1997] Johnson, R. E. (1997). Frameworks = (components + patterns). *Communications of the ACM*, 40(10):39–42.
- [Kalnikaite et al., 2011] Kalnikaite, V., Rogers, Y., Bird, J., Villar, N., Bachour, K., Payne, S., Todd, P. M., Schöning, J., Krüger, A., and Kreitmayer, S. (2011). How to

Bibliography

- nudge in situ: Designing lambent devices to deliver salient information in supermarkets. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, pages 11–20, New York, NY, USA. ACM.
- [Kane et al., 2009] Kane, S., Avrahami, D., Wobbrock, J., Harrison, B., Rea, A., Philpote, M., and LaMarca, A. (2009). Bonfire: A nomadic system for hybrid laptop-tabletop interaction. In *Proceedings of the 22nd Annual Symposium on User Interface Software and Technology*, pages 129–138, New York, NY, USA. ACM.
- [Kanev et al., 2007] Kanev, K., Kimura, S., Kobayashi, N., and Yamauchi, K. (2007). Employment of physical objects as interactive interface components. In *Proceedings of the 11th Conference on Human-Computer Interaction*, number 4662 in Lecture Notes in Computer Science, pages 10–14, New York, NY, USA. Springer.
- [Kato and Billinghurst, 1999] Kato, H. and Billinghurst, M. (1999). Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd International Workshop on Augmented Reality*, pages 85–94. IEEE and ACM.
- [Kauff and Schreer, 2002] Kauff, P. and Schreer, O. (2002). An immersive 3D video-conferencing system using shared virtual team user environments. In *Proceedings of the 4th International Conference on Collaborative Virtual Environments*, pages 105–112, New York, NY, USA. ACM.
- [Kerr et al., 2011] Kerr, S. J., Rice, M. D., Teo, Y., Wan, M., Cheong, Y. L., Ng, J., Ng-Thamrin, L., Thura-Myo, T., and Wren, D. (2011). Wearable mobile augmented reality: Evaluating outdoor user experience. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 209–216, New York, NY, USA. ACM.
- [Kindberg et al., 2002] Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., et al. (2002). People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376.
- [Klemmer et al., 2004] Klemmer, S., Li, J., Lin, J., and Landay, J. (2004). Papier-mâché: Toolkit support for tangible input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 399–406, New York, NY, USA. ACM.
- [Krueger and Casey, 2009] Krueger, R. and Casey, M. (2009). *Focus groups: A practical guide for applied research*. Sage Publications, Thousand Oaks, CA, USA, 4 edition.
- [Kumar, 1987] Kumar, K. (1987). Rapid, low-cost data collection methods for A.I.D. Technical Report 10, Agency for International Development. http://pdf.usaid.gov/pdf_docs/PNAAL100.pdf.

- [Kyng et al., 2006] Kyng, M., Nielsen, E., and Kristensen, M. (2006). Challenges in designing interactive systems for emergency response. In *Proceedings of the 6th Conference on Designing Interactive Systems*, pages 301–310, New York, NY, USA. ACM.
- [Lamp, 2014] Lamp, P. (2014). ARToolkit. <http://sourceforge.net/projects/artoolkit/>. Accessed: 21/02/2013.
- [Lange et al., 1998] Lange, B. M., Jones, M. A., and Meyers, J. L. (1998). Insight lab: An immersive team environment linking paper, displays and data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 550–557, New York, NY, USA. ACM.
- [Laseau, 2000] Laseau, P. (2000). *Graphic Thinking for Architects & Designers*. Wiley, New York, NY, USA, 3 edition.
- [Laugwitz et al., 2008] Laugwitz, B., Held, T., and Schrepp, M. (2008). Construction and evaluation of a user experience questionnaire. In Holzinger, A., editor, *HCI and Usability for Education and Work*, number 5298 in Lecture Notes in Computer Science, pages 63–76. Springer, Berlin, Heidelberg, Germany.
- [Layar, 2014] Layar (2014). Layar. <http://www.layar.com/>. Accessed: 21/02/2014.
- [Lee et al., 2011] Lee, K., Chang, M., Wong, K., and Yu, Y. (2011). A hand-held augmented reality projection system using trifocal tensors and kalman filter. In *Proceedings of the International Conference on Intelligent Computer Communication and Processing*, pages 253–260. IEEE.
- [Lee and Dey, 2014] Lee, M. L. and Dey, A. K. (2014). Real-time feedback for improving medication taking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2259–2268, New York, NY, USA. ACM.
- [Leontiadis et al., 2010] Leontiadis, N., Kefalakis, N., and Soldatos, J. (2010). Bridging RFID systems and enterprise applications through virtualized connectors. *International Journal of Automated Identification Technology (IJAIT)*, 1(2):1–12.
- [Li et al., 2004] Li, Y., Hong, J. I., and Landay, J. A. (2004). Topiary: A tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual Symposium on User Interface Software and Technology*, pages 217–226, New York, NY, USA. ACM.
- [Liao et al., 2010] Liao, C., Tang, H., Liu, Q., Chiu, P., and Chen, F. (2010). FACT: Fine-grained cross-media interaction with documents via a portable hybrid paper-laptop interface. In *Proceedings of the International Conference on Multimedia*, pages 361–370. ACM.
- [Liarokapis, 2005] Liarokapis, F. (2005). Augmented reality scenarios for guitar learning. In *3rd International Conference on Eurographics UK Theory and Practice of Computer Graphics*, pages 163–170, Canterbury, UK.

Bibliography

- [Lim et al., 1999] Lim, H.-M., Newstrom, B., Shankwitz, C., and Donath, M. (1999). A heads up display based on a DGPS and real time accessible geo-spatial database for low visibility driving. In *Proceedings of the 12th International Meeting of the Satellite Division of the Institute of Navigation*. ION.
- [Lorenz, 2009] Lorenz, A. (2009). *The Separated User Interface in Ambient Computing Environments*. PhD thesis, RWTH Aachen, Germany.
- [Lorenz and Jentsch, 2010] Lorenz, A. and Jentsch, M. (2010). The ambient media player: A media application remotely operated by the use of mobile devices and gestures. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, pages 1–10, New York, NY, USA. ACM.
- [Ludewig and Lichter, 2013] Ludewig, J. and Lichter, H. (2013). *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt-Verlag, Heidelberg, Germany, 3 edition.
- [Lukowicz et al., 2001] Lukowicz, P., Anliker, U., Troster, G., Schwartz, S. J., and DeVaul, R. W. (2001). The weararm modular, low-power computing core. *Micro, IEEE*, 21(3):16–28.
- [MacEachren, 2004] MacEachren, A. M. (2004). *How maps work: Representation, visualization, and design*. Guilford Press, New York, NY, USA.
- [Mann, 1997] Mann, S. (1997). Wearable computing: A first step toward personal imaging. *Computer*, 30(2):25–32.
- [Marrin, 2011] Marrin, C. (2011). WebGL specification. Technical Report 1, Khronos WebGL Working Group.
- [Marwedel, 2011] Marwedel, P. (2011). *Embedded system design*. Springer, Heidelberg, Germany, 2 edition.
- [McCarthy et al., 2001] McCarthy, J., Costa, T., and Liongosari, E. (2001). Unicast, outcast & groupcast: Three steps toward ubiquitous, peripheral displays. In Shafer, S., Abowd, G. D., and Brumitt, B., editors, *Proceedings of the 3rd International Conference on Ubiquitous Computing*, number 2201 in Lecture Notes in Computer Science, pages 332–345, Berlin, Heidelberg, Germany. Springer.
- [McCrickard et al., 2003] McCrickard, D. S., Bussert, D., and Wrighton, D. (2003). A toolkit for the construction of real world interfaces. In Burge, M., editor, *Proceedings of the 41st Southeast Regional Conference (ACMSE'03)*, pages 118–123, Savannah, GA, USA. ACM.
- [Messeter and Molenaar, 2012] Messeter, J. and Molenaar, D. (2012). Evaluating ambient displays in the wild: Highlighting social aspects of use in public settings. In *Proceedings of the 9th Conference on Designing Interactive Systems*, pages 478–481, New York, NY, USA. ACM.

- [Mettag.com, 2014] Mettag.com (2014). Mettag. <http://www.metttag.com/>. Accessed: 21/02/2014.
- [Microsoft, 2014a] Microsoft (2014a). Direct3D (Windows). <http://msdn.microsoft.com/en-us/library/windows/desktop/hh309466>. Accessed: 21/02/2014.
- [Microsoft, 2014b] Microsoft (2014b). Microsoft Silverlight. <http://www.microsoft.com/silverlight/>. Accessed: 21/02/2014.
- [MicroVision Inc., 2014] MicroVision Inc. (2014). MicroVision SHOWWX+. http://www.microvision.com/showwxplus_hdmi/. Accessed: 21/02/2014.
- [Milagro et al., 2009] Milagro, F., Antolin, P., Fernandes, J., Zhang, W., Hansen, K. M., and Kool, P. (2009). Deploying pervasive web services over a p2p overlay. In *18th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 240–245. IEEE.
- [Milgram and Kishino, 1994] Milgram, P. and Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329.
- [Milgram et al., 1994] Milgram, P., Takemura, H., Utsumi, A., and Kishino, F. (1994). Augmented reality: A class of displays on the reality-virtuality continuum. In Das, H., editor, *Proceedings of Telem manipulator and Telepresence Technologies*, volume 2351, pages 282–292, Boston, MA, USA. SPIE.
- [Mills, 2010] Mills, A. J. (2010). *Encyclopedia of Case Study Research*. Number 1. Sage Publications, Thousand Oaks, CA, USA.
- [Mistry and Maes, 2009] Mistry, P. and Maes, P. (2009). Sixthsense: A wearable gestural interface. In *SIGGRAPH ASIA 2009 Sketches*, page 11. ACM.
- [Mistry et al., 2009] Mistry, P., Maes, P., and Chang, L. (2009). WUW-wear Ur world: A wearable gestural interface. In *Extended Abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 4111–4116, New York, NY, USA. ACM.
- [Molyneaux and Gellersen, 2009] Molyneaux, D. and Gellersen, H. (2009). Projected interfaces: Enabling serendipitous interaction with smart tangible objects. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, pages 385–392, New York, NY, USA. ACM.
- [Molyneaux et al., 2012] Molyneaux, D., Izadi, S., Kim, D., Hilliges, O., Hodges, S., Cao, X., Butler, A., and Gellersen, H. (2012). Interactive environment-aware handheld projectors for pervasive computing spaces. In Kay, J., Lukowicz, P., Tokuda, H., Olivier, P., and Krüger, A., editors, *Adjunct Proceeding of the 10th International Conference on Pervasive Computing*, number 7319 in Lecture Notes in Computer Science, pages 197–215, Berlin, Heidelberg, Germany. Springer.

Bibliography

- [Mori et al., 2004] Mori, G., Paternò, F., and Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520.
- [Müller et al., 2012] Müller, H., Fortmann, J., Pielot, M., Hesselmann, T., Poppinga, B., Heuten, W., Henze, N., and Boll, S. (2012). Ambix: Designing ambient light information displays. In *Proceedings of the Workshop on Designing Interactive Lighting*.
- [Müller et al., 2008] Müller, J., Jentsch, M., Kray, C., and Krüger, A. (2008). Exploring factors that influence the combined use of mobile devices and public displays for pedestrian navigation. In *Proceedings of the 5th Nordic Conference on Human-Computer Interaction*, pages 308–317, New York, NY, USA. ACM.
- [Myers et al., 2000] Myers, B., Hudson, S. E., and Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28.
- [Mynatt et al., 2001] Mynatt, E., Rowan, J., Craighill, S., and Jacobs, A. (2001). Digital family portraits: Supporting peace of mind for extended family members. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 333–340, New York, NY, USA. ACM.
- [National Imagery and Mapping Agency, 2000] National Imagery and Mapping Agency (2000). World geodetic system 1984 - its definition and relationships with local geodetic systems. <http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>. Accessed: 21/02/2014.
- [Nielsen, 1993] Nielsen, J. (1993). *Usability Engineering*. Interactive technologies. Academic Press, London, UK.
- [Object Management Group, 2014] Object Management Group (2014). Unified modeling language. <http://www.uml.org/>. Accessed: 21/02/2014.
- [O’Hara et al., 2011] O’Hara, K., Perry, M., Churchill, E., and Russell, D., editors (2011). *Public and situated displays: Social and interactional aspects of shared display technologies*, volume 2. Kluwer Academic Publishers, Dordrecht, The Netherlands.
- [Oppermann and Specht, 2000] Oppermann, R. and Specht, M. (2000). A context-sensitive nomadic exhibition guide. In Thomas, P. and Gellersen, H.-W., editors, *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing*, number 1927 in Lecture Notes in Computer Science, pages 31–54. Springer.
- [OSGi Alliance, 2014] OSGi Alliance (2014). OSGi Alliance. <http://www.osgi.org/>. Accessed: 21/02/2014.
- [Overmyer, 1991] Overmyer, S. P. (1991). Revolutionary vs. evolutionary rapid prototyping: Balancing software productivity and HCI design concerns. In *Proceedings of*

-
- the 4th International Conference on Human-Computer Interaction*, Stuttgart, Germany. IFIP, Elsevier.
- [Oxford University Press, 2014] Oxford University Press (2014). Oxford dictionaries online. oxforddictionaries.com/. Accessed: 21/02/2014.
- [Park et al., 2009] Park, H., Moon, H., and Lee, J. (2009). Tangible augmented prototyping of digital handheld products. *Computers in Industry*, 60(2):114–125.
- [Patel et al., 2007] Patel, S. N., Robertson, T., Kientz, J. A., Reynolds, M. S., and Abowd, G. D. (2007). At the flick of a switch: Detecting and classifying unique electrical events on the residential power line. In Krumm, J., Abowd, G. D., Seneviratne, A., and Strang, T., editors, *Proceedings of the 9th International Conference on Ubiquitous Computing*, number 4717 in Lecture Notes in Computer Science, pages 271–288. Springer, Berlin, Heidelberg, Germany.
- [Patton, 1987] Patton, M. Q. (1987). *How to Use Qualitative Methods in Evaluation*. 2. Sage Publications, Newbury Park, CA, USA.
- [Paul et al., 2005] Paul, P., Fleig, O., and Jannin, P. (2005). Augmented virtuality based on stereoscopic reconstruction in multimodal image-guided neurosurgery: Methods and performance evaluation. *IEEE Transactions on Medical Imaging*, 24(11):1500–1511.
- [Paulos and Jenkins, 2005] Paulos, E. and Jenkins, T. (2005). Urban probes: Encountering our emerging urban atmospheres. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 341–350, New York, NY, USA. ACM.
- [Pedraza et al., 2010] Pedraza, G., García, I., and Debbabi, B. (2010). An RFID architecture based on an event-oriented component model. In *Proceedings of the 4th International Conference on Distributed Event-Based Systems*, pages 87–88, New York, NY, USA. ACM.
- [Pettersson, 2004] Pettersson, M. (2004). Watercalls: An ambient call queue for cooperation between emergency service centres. *Personal and Ubiquitous Computing*, 8(3-4):192–199.
- [Pinhanez, 2001] Pinhanez, C. (2001). The everywhere displays projector: A device to create ubiquitous graphical interfaces. In Shafer, S., Abowd, G. D., and Brumitt, B., editors, *Proceedings of the 3rd International Conference on Ubiquitous Computing*, number 2201 in Lecture Notes in Computer Science, pages 315–331, Berlin, Heidelberg, Germany. Springer.
- [Plugwise B.V., 2014] Plugwise B.V. (2014). Smart wireless solutions for energy saving, energy monitoring and switching. <http://www.plugwise.com/>. Accessed: 21/02/2014.

Bibliography

- [Ponnekanti et al., 2001] Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. (2001). ICrafter: A service framework for ubiquitous computing environments. In Shafer, S., Abowd, G. D., and Brumitt, B., editors, *Proceedings of the 3rd International Conference on Ubiquitous Computing*, number 2201 in Lecture Notes in Computer Science, pages 56–75, Berlin, Heidelberg, Germany. Springer.
- [Poslad, 2009] Poslad, S. (2009). *Ubiquitous Computing*. Wiley, Chichester, UK.
- [Pramudianto et al., 2013a] Pramudianto, F., Rusmita, I., and Jarke, M. (2013a). Model driven development for internet of things application prototyping. In *The 25th International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, USA.
- [Pramudianto et al., 2013b] Pramudianto, F., Simon, J., Eisenhauer, M., Khaleel, H., Pastrone, C., and Spirito, M. (2013b). Prototyping the internet of things technology for the future factory using service oriented architecture middleware. In *18th International Conference on Emerging Technologies and Factory Automation*. IEEE.
- [Prause et al., 2010] Prause, C., Jentsch, M., and Eisenhauer, M. (2010). MICA - A mobile support system for warehouse workers. *International Journal of Handheld Computing Research (IJHCR)*, 2(1):1–24.
- [Prinz, 1999] Prinz, W. (1999). NESSIE: An awareness environment for cooperative settings. In Bodker, S., Kyng, M., and Schmidt, K., editors, *Proceedings of the 6th European Conference on Computer Supported Cooperative Work*, pages 391–410. Kluwer Academic Publishers.
- [Raskar et al., 2004] Raskar, R., Beardsley, P., Van Baar, J., Wang, Y., Dietz, P., Lee, J., Leigh, D., and Willwacher, T. (2004). RFIG lamps: Interacting with a self-describing world via photosensing wireless tags and projectors. *Transactions on Graphics*, 23(3):406–415.
- [Raskar et al., 2003] Raskar, R., Van Baar, J., Beardsley, P., Willwacher, T., Rao, S., and Forlines, C. (2003). iLamps: Geometrically aware and self-configuring projectors. *Transactions on Graphics (TOG)*, 22(3):809–818.
- [Raskar et al., 1999] Raskar, R., Welch, G., and Chen, W.-C. (1999). Table-top spatially-augmented reality: Bringing physical models to life with projected imagery. In *Proceedings of the 2nd International Workshop on Augmented Reality*, pages 64–71. IEEE and ACM.
- [Raskar et al., 1998] Raskar, R., Welch, G., and Fuchs, H. (1998). Spatially augmented reality. In *Proceedings of the 1st International Workshop on Augmented Reality*, pages 11–20. IEEE and ACM.
- [Raspberry Pi Foundation, 2014] Raspberry Pi Foundation (2014). Raspberry Pi. <http://www.raspberrypi.org/>. Accessed: 16/06/2014.

- [Rauschenberger et al., 2013a] Rauschenberger, M., Schrepp, M., Cota, M., Olschner, S., and Thomaschewski, J. (2013a). Efficient measurement of the user experience of interactive products. How to use the user experience questionnaire (UEQ). Example: Spanish language version. *International Journal of Artificial Intelligence and Interactive Multimedia*, 2(1):39–45.
- [Rauschenberger et al., 2013b] Rauschenberger, M., Thomaschewski, J., and Schrepp, M. (2013b). User Experience mit Fragebögen messen. Durchführung und Auswertung am Beispiel des UEQ. In Brau, H., Lehmann, A., Petrovic, K., and Schröder, M., editors, *Usability Professionals*, pages 348–353. German UPA, Stuttgart, Germany.
- [Reenskaug et al., 1996] Reenskaug, T., Wold, P., and Lehne, O. (1996). *Working with objects: The OOram software engineering method*. Manning, Greenwich, CT, USA.
- [Reiners et al., 2009] Reiners, R., Zimmermann, A., Jentsch, M., and Zhang, Y. (2009). Automizing home environments and supervising patients at home with the hydra middleware: Application scenarios using the hydra middleware for embedded systems. In *Proceedings of the First International Workshop on Context-Aware Software Technology and Applications*, pages 9–12, New York, NY, USA. ACM.
- [Rekimoto, 1995] Rekimoto, J. (1995). The magnifying glass approach to augmented reality systems. In *International Conference on Artificial Reality and Tele-Existence*, volume 95, pages 123–132. ACM, Nihon Keizai Shimbun.
- [Rogers et al., 2010] Rogers, Y., Hazlewood, W., Marshall, P., Dalton, N., and Hertrich, S. (2010). Ambient influence: Can twinkly lights lure and abstract representations trigger behavioral change? In *Proceedings of the 12th International Conference on Ubiquitous Computing*, pages 261–270, New York, NY, USA. ACM.
- [Rohs and Gfeller, 2004] Rohs, M. and Gfeller, B. (2004). Using camera-equipped mobile phones for interacting with real-world objects. In *Advances in Pervasive Computing*, pages 265–271.
- [Rollandt et al., 1998] Rollandt, J. P., Parsons, J., Poizatt, D., and Hancock, D. (1998). Conformal optics for 3D visualization. In *Proceedings of SPIE-the International Society for Optical Engineering, vol 3482*, pages 760–764.
- [Roman et al., 2000] Roman, M., Beck, J., and Gefflaut, A. (2000). A device-independent representation for services. In *3rd Workshop on Mobile Computing Systems and Applications*, pages 73–82. IEEE.
- [Roman and Campbell, 2002] Roman, M. and Campbell, R. H. (2002). A user-centric, resource-aware, context-sensitive, multi-device application framework for ubiquitous computing environments. Technical Report UIUCDCS-R-2002-2284 UILU-ENG-2002-1728, University of Illinois.

Bibliography

- [Rozanski and Woods, 2011] Rozanski, N. and Woods, E. (2011). *Software systems architecture: Working with stakeholders using viewpoints and perspectives*. Addison-Wesley, Upper Saddle River, NJ, USA.
- [RTI, 2012] RTI (2012). RTI Connex. Technical report, Real-Time Innovations, Sunnyvale, CA, USA. http://www.rti.com/whitepapers/RTI_Connext_WP.pdf.
- [RTI, 2014] RTI (2014). Is DDS for you? Technical report, Real-Time Innovations, Sunnyvale, CA, USA. http://www.rti.com/whitepapers/Is_DDS_for_You.pdf.
- [Rukzio, 2006] Rukzio, E. (2006). *Physical Mobile Interactions: Mobile Devices as Pervasive Mediators for Interactions with the Real World*. PhD thesis, LMU Munich, Germany.
- [Russell et al., 2002] Russell, D., Drews, C., and Sue, A. (2002). Social aspects of using large public interactive displays for collaboration. In Borriello, G. and Holmquist, L. E., editors, *Proceedings of the 4th International Conference on Ubiquitous Computing*, number 2498 in Lecture Notes in Computer Science, pages 663–670, Berlin, Heidelberg, Germany. Springer.
- [Salehi, 2010] Salehi, A. (2010). *Design and implementation of an efficient data stream processing system*. PhD thesis, École polytechnique fédérale de Lausanne, Switzerland.
- [Samsung Electronics, 2012] Samsung Electronics (2012). Samsung GALAXY Beam. <http://www.samsung.com/global/microsite/galaxybeam/>. Accessed: 21/02/2014.
- [Sauro, 2011] Sauro, J. (2011). Measuring usability with the system usability scale. <http://www.measuringusability.com/sus.php>. Accessed: 21/02/2014.
- [Schilit et al., 1994] Schilit, B., Adams, N., and Want, R. (1994). Context-aware computing applications. In *First Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE.
- [Schmalstieg and Wagner, 2009] Schmalstieg, D. and Wagner, D. (2009). Mobile phones as a platform for augmented reality. *Connections*, 1:3.
- [Schmidt, 2000] Schmidt, A. (2000). Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 4(2):191–199.
- [Schöning et al., 2006] Schöning, J., Krüger, A., and Müller, H. J. (2006). Interaction of mobile camera devices with physical maps. In Fishkin, K. P., Schiele, B., Nixon, P., and Quigley, A., editors, *Adjunct Proceeding of the 4th International Conference on Pervasive Computing*, number 3968 in Lecture Notes in Computer Science, pages 121–124, Berlin, Heidelberg, Germany. Springer.
- [Schrepp et al., 2013] Schrepp, M., Olschner, S., and Schubert, U. (2013). User Experience Questionnaire Benchmark – Praxiserfahrungen zum Einsatz im Business-Umfeld. In Brau, H., Lehmann, A., Petrovic, K., and Schröder, M., editors, *Usability Professionals*, pages 348–353. German UPA, Stuttgart, Germany.

- [Schwartz et al., 2013] Schwartz, T., Deneff, S., Stevens, G., Ramirez, L., and Wulf, V. (2013). Cultivating energy literacy: Results from a longitudinal living lab study of a home energy management system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1193–1202, New York, NY, USA. ACM.
- [Shreiner et al., 2013] Shreiner, D., Sellers, G., Kessenich, J., and Licea-Kane, B. (2013). *OpenGL programming guide: The official guide to learning OpenGL, version 4.3*. Addison-Wesley Professional, Upper Saddle River, NJ, USA, 8 edition.
- [Šimbelis et al., 2014] Šimbelis, V., Lundström, A., Höök, K., Solsona, J., and Lewandowski, V. (2014). Metaphone: machine aesthetics meets interaction design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1–10, New York, NY, USA. ACM.
- [Smith et al., 1982] Smith, D. C., Irby, C., Kimball, R., Verplank, B., and Harslem, E. (1982). Designing the star user interface. *Byte*, 7(4):242–282.
- [Song et al., 2010] Song, H., Guimbretiere, F., Grossman, T., and Fitzmaurice, G. (2010). MouseLight: Bimanual interactions on digital paper using a pen and a spatially-aware mobile projector. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2451–2460, New York, NY, USA. ACM.
- [Sousa and Garlan, 2002] Sousa, J. and Garlan, D. (2002). Aura: An architectural framework for user mobility in ubiquitous computing environments. In Bosch, Gentleman, H. K., editor, *Software Architecture: System Design, Development, and Maintenance*, pages 29–43. Kluwer Academic Publishers.
- [Spasova, 2004] Spasova, L. (2004). Fluid beam - a steerable projector and camera unit. In *Student and Newbie Colloquium at Eighth International Symposium on Wearable Computers / Third International Symposium on Mixed and Augmented Reality*. IEEE.
- [Stavropoulos et al., 2013] Stavropoulos, T. G., Gottis, K., Vrakas, D., and Vlahavas, I. (2013). aWESoME: A web service middleware for ambient intelligence. *Expert Systems with Applications*, 40(11):4380–4392.
- [Stavropoulos et al., 2010] Stavropoulos, T. G., Tsioliariidou, A., Koutitas, G., Vrakas, D., and Vlahavas, I. (2010). System architecture for a smart university building. In Diamantaras, K., Duch, W., and Iliadis, L. S., editors, *Artificial Neural Networks-ICANN 2010*, number 6354 in Lecture Notes in Computer Science, pages 477–482. Springer, Berlin, Heidelberg, Germany.
- [Stirbu, 2010] Stirbu, V. (2010). A RESTful architecture for adaptive and multi-device application sharing. In Pautasso, C., Wilde, E., and Marinos, A., editors, *Proceedings of the 1st International Workshop on RESTful Design*, pages 62–65, Raleigh, NC, USA. ACM.

Bibliography

- [Streitz et al., 1998] Streitz, N., Geißler, J., and Holmer, T. (1998). Roomware for cooperative buildings: Integrated design of architectural spaces and information spaces. In *Cooperative Buildings: Integrating Information, Organization, and Architecture*, volume 1370, pages 4–21. Springer, Darmstadt, Germany.
- [Sutherland, 1968] Sutherland, I. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, pages 757–764, New York, NY, USA. ACM.
- [Thomas et al., 1998] Thomas, B., Demczuk, V., Piekarski, W., Hepworth, D., and Gunther, B. (1998). A wearable computer system with augmented reality to support terrestrial navigation. In *Digest of Papers. Second International Symposium on Wearable Computers*, pages 168–171, Washington, DC, USA. IEEE, IEEE Computer Society.
- [Thomson and Georgantas, 2007] Thomson, G. and Georgantas, N. (2007). Amigo overall middleware: Final prototype implementation & documentation. Deliverable Report D3.5, Inria.
- [Truong and Abowd, 2004] Truong, K. and Abowd, G. D. (2004). Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In Ferscha, A. and Mattern, F., editors, *Adjunct Proceeding of the 2nd International Conference on Pervasive Computing*, number 3001 in Lecture Notes in Computer Science, pages 140–157, Berlin, Heidelberg, Germany. Springer.
- [Tufté and Weise Moeller, 1997] Tufté, E. R. and Weise Moeller, E. (1997). *Visual explanations: Images and quantities, evidence and narrative*, volume 107. Graphics Press, Cheshire, CT, USA.
- [Tversky, 1993] Tversky, B. (1993). Cognitive maps, cognitive collages, and spatial mental models. In Frank, A. U. and Campari, I., editors, *Spatial Information Theory: A Theoretical Basis for GIS*, number 716 in Lecture Notes in Computer Science, pages 14–24. Springer, Berlin, Heidelberg, Germany.
- [Underkoffler, 1997] Underkoffler, J. (1997). A view from the luminous room. *Personal and Ubiquitous Computing*, 1(2):49–59.
- [U.S. Department of Health & Human Services, 2008] U.S. Department of Health & Human Services (2008). Data collection methods for program evaluation: Observation. <http://www.cdc.gov/healthyyouth/evaluation/pdf/brief16.pdf>. Accessed: 21/02/2014.
- [U.S. Department of Health & Human Services, 2014] U.S. Department of Health & Human Services (2014). Focus groups. <http://www.usability.gov/how-to-and-tools/methods/focus-groups.html>. Accessed: 21/02/2014.
- [User Interface Design GmbH, 2014] User Interface Design GmbH (2014). AttrakDiff. <http://attrakdiff.de/>. Accessed: 21/02/2014.

- [Vaccarini et al., 2013] Vaccarini, M., Carbonari, A., and Giretti, A. (2013). Preliminary implementation of a predictive control for ventilation systems in metro stations. In *Proceedings of the 30th International Symposium on Automation and Robotics in Construction and Mining*, pages 922–929, Montreal, Canada.
- [Vallée et al., 2005] Vallée, M., Ramparany, F., and Vercouter, L. (2005). Dynamic service composition in ambient intelligence environments : a multi-agent approach. In *Proceeding of the 1st European Young Researcher Workshop on Service-Oriented Computing*, Leicester, UK.
- [Viirre et al., 1998] Viirre, E., Pryor, H., Nagata, S., and Furness 3rd, T. A. (1998). The virtual retinal display: A new technology for virtual reality and augmented vision in medicine. In Wesiwood, J., Hoffman, H., Stredney, D., and Weghorst, S., editors, *Medicine meets Virtual Reality: Art, Science, Technology: Healthcare (R)Evolution*, volume 50, page 252. IOS Press, Amsterdam, The Netherlands.
- [Vinkovits et al., 2012] Vinkovits, M., Elmasllari, E., and Pastrone, C. (2012). Anonymous networking meets real-world business requirements. In *Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 451–457. IEEE.
- [Wagner et al., 2005] Wagner, D., Pintaric, T., Ledermann, F., and Schmalstieg, D. (2005). Towards massively multi-user augmented reality on handheld devices. In Gellersen, H.-W., Wand, R., and Schmidt, A., editors, *Adjunct Proceeding of the 3rd International Conference on Pervasive Computing*, number 3468 in Lecture Notes in Computer Science, pages 77–95, Berlin, Heidelberg, Germany. Springer.
- [Wang, 2014] Wang, L. (2014). Energy efficiency technologies for sustainable food processing. *Energy Efficiency*, 7:1–20.
- [Wei et al., 2011] Wei, J., Wang, X., Peiris, R., Choi, Y., Martinez, X., Tache, R., Koh, J., Halupka, V., and Cheok, A. (2011). CoDine: An interactive multi-sensory system for remote dining. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, pages 21–30, New York, NY, USA. ACM.
- [Weis et al., 2007] Weis, T., Knoll, M., Ulbrich, A., Muhl, G., and Brandle, A. (2007). Rapid prototyping for pervasive applications. *Pervasive Computing, IEEE*, 6(2):76–84.
- [Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.
- [Weiser and Brown, 1996] Weiser, M. and Brown, J. (1996). Designing calm technology. *PowerGrid Journal*, 1(1):1–5.
- [Wikitude GmbH, 2014] Wikitude GmbH (2014). Wikitude - World’s leading augmented reality SDK. <http://www.wikitude.com/>. Accessed: 21/02/2014.

Bibliography

- [Williams, 1984] Williams, G. (1984). The apple macintosh computer. *Byte*, 9(2):30–31.
- [Wisneski et al., 1998] Wisneski, C., Ishii, H., Dahley, A., Gorbet, M., Brave, S., Ullmer, B., and Yarin, P. (1998). Ambient displays: Turning architectural space into an interface between people and digital information. In *Cooperative Buildings: Integrating Information, Organization, and Architecture*, volume 1370, pages 22–32. Springer, Darmstadt, Germany.
- [Wood and Newborough, 2003] Wood, G. and Newborough, M. (2003). Dynamic energy-consumption indicators for domestic appliances: Environment, behaviour and design. *Energy and Buildings*, 35(8):821–841.
- [Xiao et al., 2013] Xiao, R., Harrison, C., and Hudson, S. E. (2013). WorldKit: Rapid and easy creation of ad-hoc interactive applications on everyday surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 879–888, New York, NY, USA. ACM.

Appendix A

EE Questionnaire

Saving Energy

Today, we are talking about saving energy and will ask you some questions about this. Important: You are not being tested! Your name will not be on the questionnaire. Instead, we want to know what people's motivation is to save energy. Let's find out what you think about it.

1. State for each part of the system how good it would help you saving energy!

	Absolutely				Not at all
Information on the TV	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Magic Lens on the smartphone	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Map on the smartphone	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Light clues	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2. State one pro and one con aspect for each part of the system:

(a) Information on the TV

Pro:

Con:

(b) Magic Lens on the smartphone

Pro:

Con:

Appendix A. EE Questionnaire

(c) Map on the smartphone

Pro:

Con:

(d) Light clues

Pro:

Con:

Appendix B

Presentation for Qualitative and Quantitative Usability Assessment Study

UbiVis Framework Evaluation

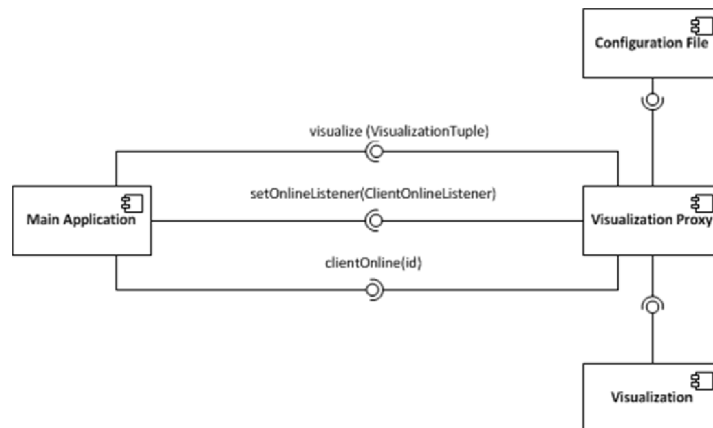
Rapid Prototyping of Ubiquitous Annotation Visualization

Marc Jentsch

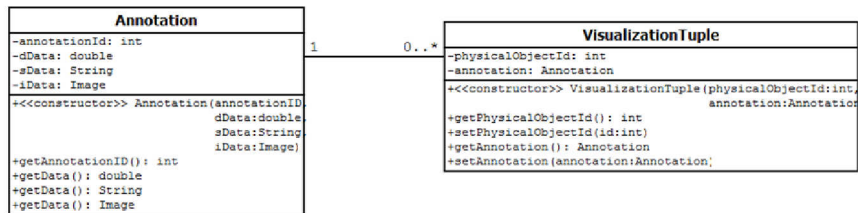
What is UbiVis?

- Framework, which helps developers to easily exchange visualizations of an application
- Linksmart-based
- Used for applications which visualize information about physical object in user's environment

What is UbiVis?

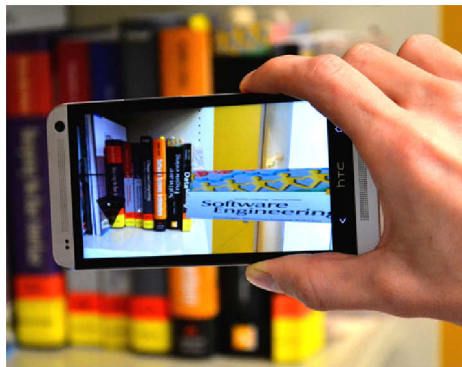


What is UbiVis?



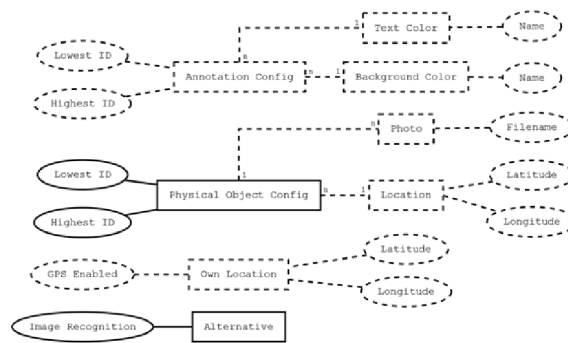
The Visualization Libraries

■ UbiLens



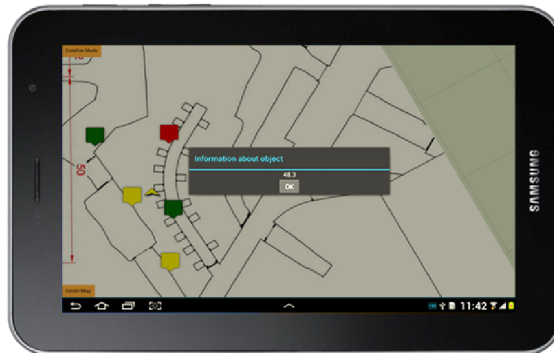
The Visualization Libraries

■ Configuration UbiLens



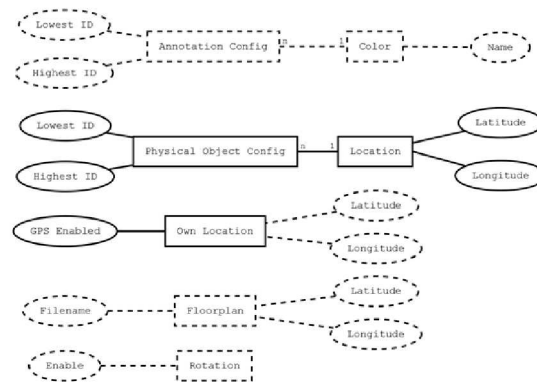
The Visualization Libraries

■ UbiMap



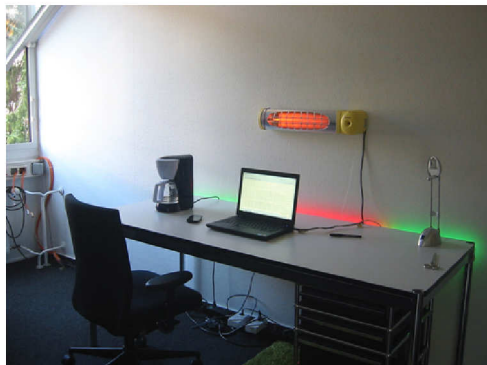
The Visualization Libraries

■ Configuration UbiMap



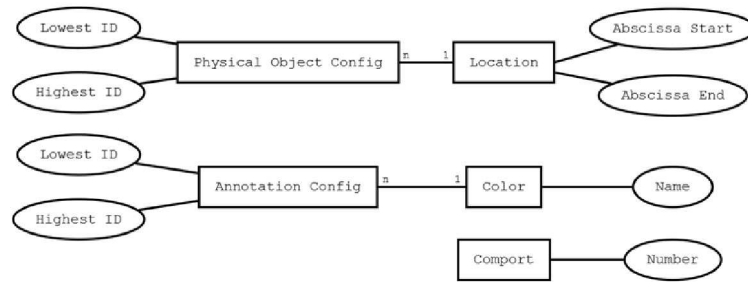
The Visualization Libraries

■ UbiLight



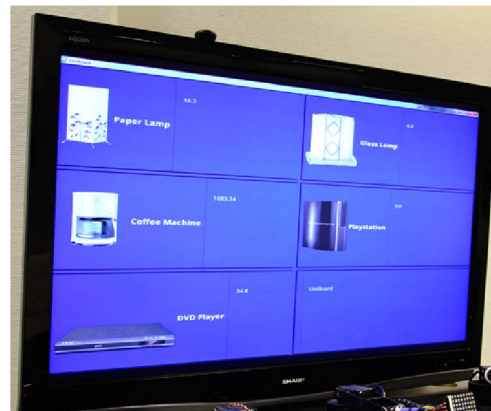
The Visualization Libraries

■ Configuration UbiLight



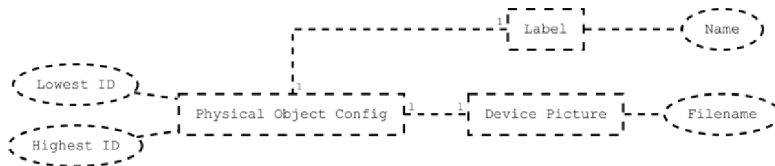
The Visualization Libraries

■ UbiBoard



The Visualization Libraries

■ Configuration UbiBoard



How do I apply UbiVis?

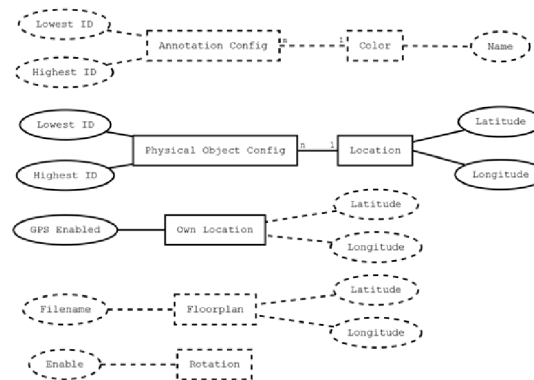
1. Analyze configuration of visualizations
2. (optional) Analyze data space of physical objects and digital information
3. (optional) Configure visualizations
4. Develop application logic including calls of VisualizationProxy

Example Walkthrough

- Task
 - Visualize input volumes of Bluetooth microphones
 - Distributed in noisy room -> varying volume
 - Read volume over Bluetooth
 - UbiMap visualization

Example Walkthrough

- 1. Analyze configuration of visualizations



Example Walkthrough

- 2. Analyze data space of physical objects and digital information
 - Bluetooth MAC as physical object ID
 - No grouping of physical objects
 - No Information mapping required

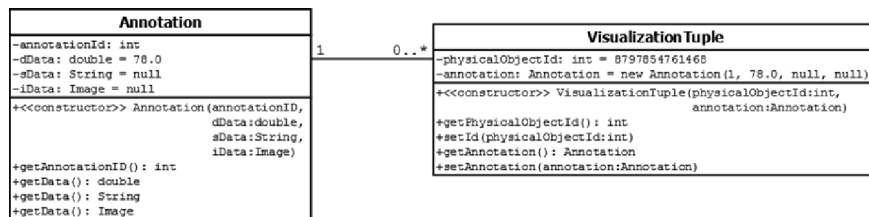
Example Walkthrough

- 3. Configure visualization

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>8797854761468</lowestId>
      <highestId>8797854761468</highestId>
      <latitude>50.749612</latitude>
      <longitude>7.203817</longitude>
    </PhysicalObjectConfig>
    <PhysicalObjectConfig>
      <lowestId>8797854761387</lowestId>
      <highestId>8797854761387</highestId>
      <latitude>50.749646</latitude>
      <longitude>7.203843</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
  <ownLocation>
    <gpsEnabled>false</gpsEnabled>
    <latitude>50.749619</latitude>
    <longitude>7.20387</longitude>
  </ownLocation>
  <floorplan>
    <floorplanFile>floormap.png</floorplanFile>
    <latitude>50.74935</latitude>
    <longitude>7.203685</longitude>
  </floorplan>
</Configuration>
```

Example Walkthrough

- 4. Develop application logic including calls of VisualizationProxy
 - Register at UbiMapProxy
 - Continuously request input volumes
 - Trigger visualization on volume change



Your Task

- Create a system which visualizes the danger level for children of household articles: book, lamp, knife
- Decide on the particular danger levels and their representation
- Task 1: There is an example application, which you checkout together with the framework and let it run. It uses UbiBoard for visualizing the danger level of the book. Add the danger levels for knife and lamp.
- Task 2: Choose another library. Configure it and connect it to your application from Task 1 without changing the code for the application logic.
- For development instructions, see handout

Appendix C

Handout for Qualitative and Quantitative Usability Assessment Study

C.1 What is UbiVis?

UbiVis is a framework which helps developers to easily exchange visualizations of an application. When exchanging visualizations, the application code does not have to be modified. For this, UbiVis provides code, processes and technical structure.

The framework is based on LinkSmart. For today's task, this means mainly that we are operating in an OSGi environment. Each component will be an OSGi bundle. Use your normal LinkSmart environment for the development and employ it as usual.

For each visualization, UbiVis provides a library. A library usually consists of two components, the actual visualization and its proxy. The visualization proxy is an OSGi bundle, which will act as interface between the actual visualization and the main application where the application logic shall reside. The actual visualization component performs the visualization. It can also be an OSGi bundle, but it is likewise possible that this component is, e.g., a smartphone application. Have a look at Figure C.1 for the architecture.

As you can see in Figure C.1, there are 3 interface methods between the components **Main Application** and **Visualization Proxy**. The method `visualize(Visualization-Tuple)` triggers a visualization. The other two methods implement a notification service so that the main application knows when the visualization component is online. For this, the main application has to implement the Java interface `ClientOnlineListener` and register itself at the visualization proxy by calling `setOnlineListener(this)`. The interface `ClientOnlineListener` also requires the main application to create the method `clientOnline(id)`. This method will be called by the visualization proxy once the visualization component is online.

Each visualization proxy consists of a **Configuration File**, which will be used for configuring the visualization component. This is an XML file in which settings can be

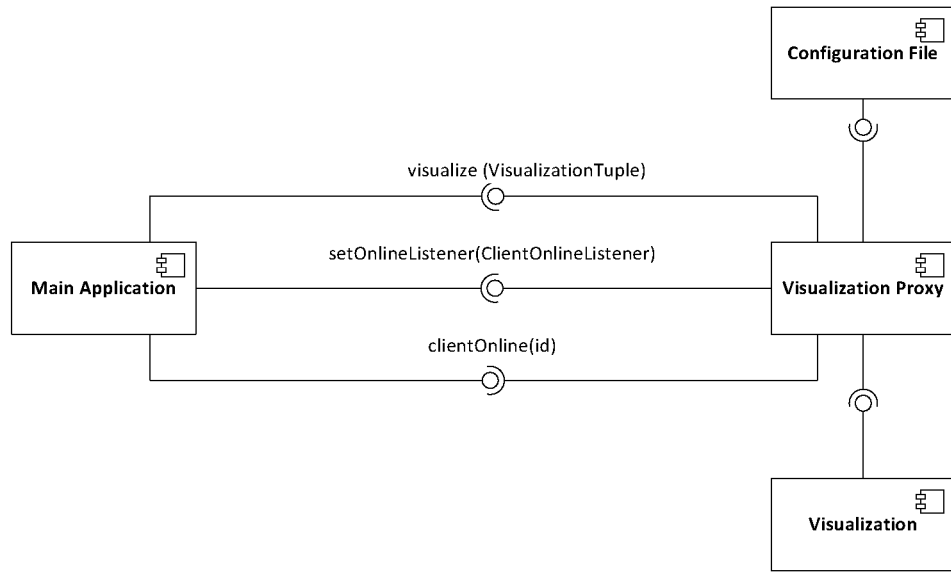


Figure C.1: Component structure and interface methods

edited or ids can be mapped.

UbiVis shall be used for applications which visualize information about physical objects in the user’s environment. We call this annotating a physical object. As explained before, a call of `visualize(VisualizationTuple)` triggers a visualization. Now, let’s have a closer look at this method. The `VisualizationTuple` class consists of two components, an `Annotation` object and an ID. These represent the information which shall be visualized and the annotated physical object. Annotation can either be a number, a text, an image or a combination of them. Annotation also has an identifier called `annotationId`. Compare also Figure C.2.

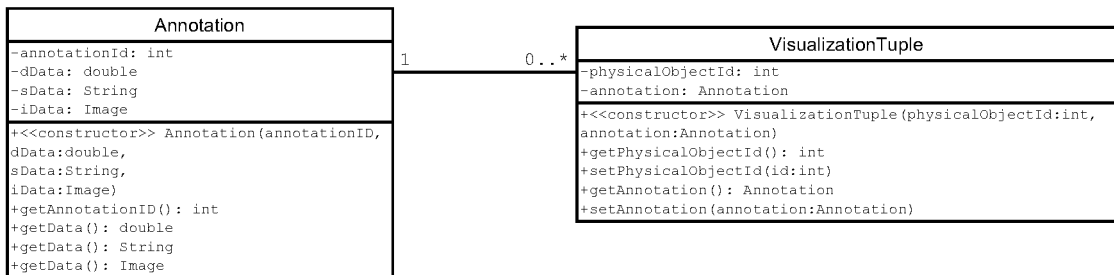


Figure C.2: The class diagram shows the static data structure of UbiVis.

C.2 The Visualization Libraries

C.2.1 UbiLens

UbiLens covers an approach which is commonly used for ubiquitous annotation visualization: Location-aware Augmented Reality as see-through version on a smartphone. This is, e.g., applied by Wikitude World Browser or Layar. The smartphone captures video images of the environment via its integrated camera and displays it on its screen. This way, the user has the feeling of seeing through the phone. The video image can be overlaid by graphical objects. These objects adapt to the current position of the phone so that a particular graphical object is also placed above a corresponding physical object (cf. Figure C.3). This way, it acts as annotation for the physical object. Some works refer to this approach as *Magic Lens*. The idea is that the smartphone is held like a lens over an object in order to reveal additional information which could not be seen without the tool.

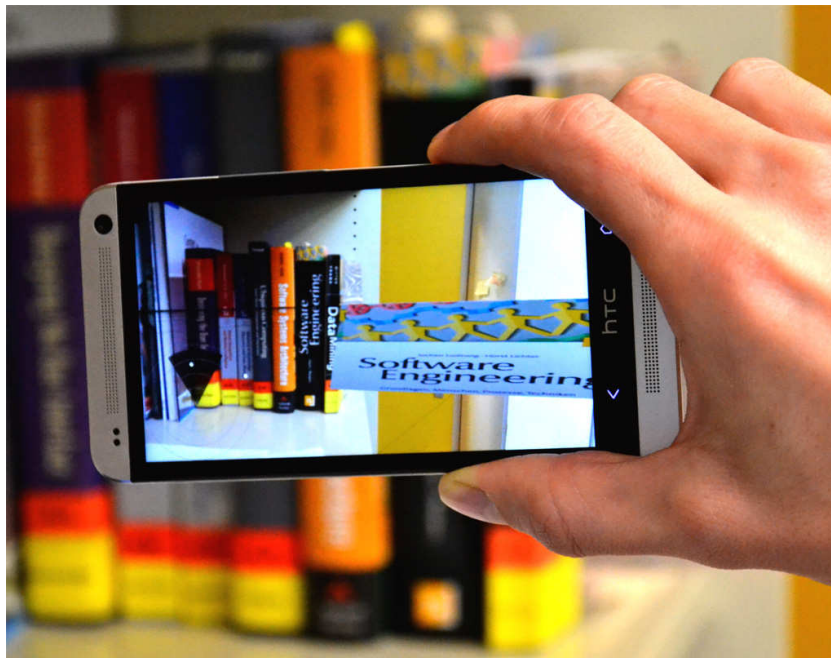


Figure C.3: UbiLens

Configuration Options

UbiLens requires mapping physical objects to technology-specific attributes as well as mapping digital information. Also, two global configuration options are needed. All possible configuration options are summarized in Figure C.4. Mandatory options are solid.

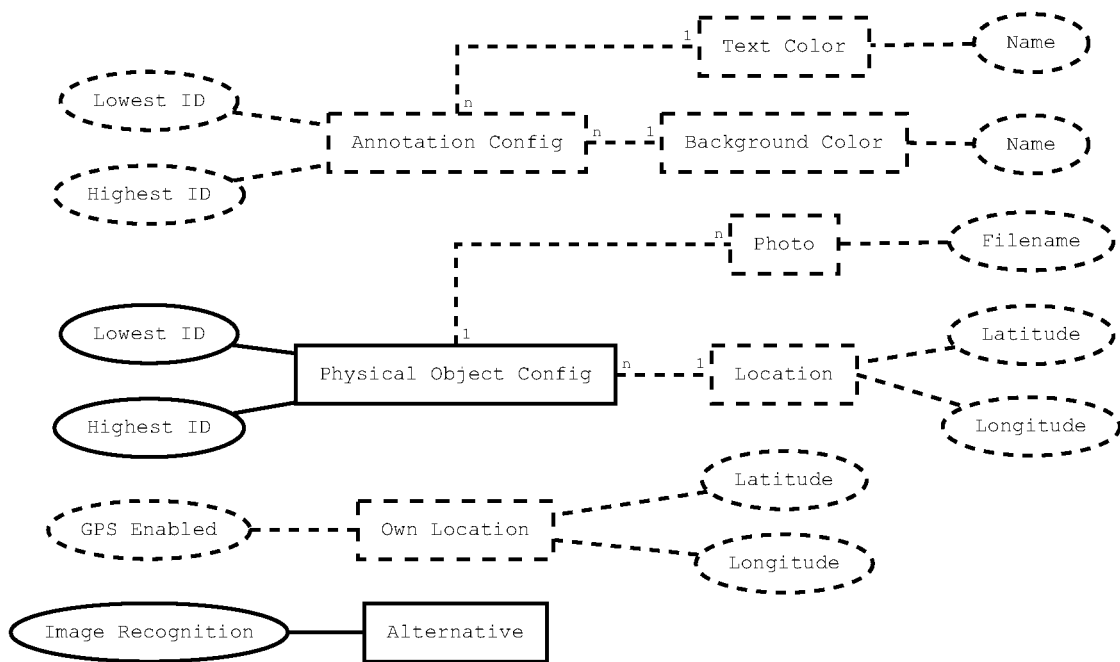


Figure C.4: Configuration options of UbiLens

First, a UbiLens user has to decide on an alternative for placing the annotations. By setting the global parameter **Image Recognition** true, UbiLens uses image recognition. Setting it false makes UbiLens calculate the positions from location and rotation values. So, only one alternative can be enabled at a time. This setting determines which of the following options have to be filled.

For the calculation alternative, the graphic’s position is calculated from the physical object’s real world location and the device’s location and rotation value. Hence, UbiLens needs to know the real world location of each physical object. Therefore, physical objects are mapped to locations. A location consists of a latitude and longitude value in decimal WGS84 format.

The way how the device’s location is acquired can be configured as global option. It is acquired by GPS if the parameter **GPS Enabled** is set true. Otherwise, a static location is used. This static location must be set in the configuration through latitude and longitude in the same format as the physical objects’ locations.

The background color and text color of an annotation cube is customizable in UbiLens. So, annotation IDs are mapped to a background color and a text color attribute. The tone can be either indicated by a name which can be computed by Android¹. Or, in the format **"#RRGGBB"**. The **RR** has to be replaced by the 2 byte hex code of the color’s red part. **GG** and **BB** must be replaced analogously for the green respectively blue part.

For the image recognition alternative, a photo image of a physical object must be

¹[http://developer.android.com/reference/android/graphics/Color.html#parseColor\(java.lang.String\)](http://developer.android.com/reference/android/graphics/Color.html#parseColor(java.lang.String))

found in the current video frame. Thus, UbiLens needs a photo of each physical object. Therefore, physical objects are mapped to photos. A photo is specified by a filename which points to an image file in the configuration folder. .png and .jpg files are supported. It is possible to specify more than one photo for a physical object.

C.2.2 UbiMap

UbiMap makes use of a classical approach for referring to close-by physical objects in a smartphone application. A map of the environment is presented on the phone. Annotated objects are made interactive at those positions on the map which refer to their real world location.

A dynamic street map of the smartphone's surroundings is displayed on the screen. The app can also be used indoors. For this, a building's floor plan can be overlaid over the street map (cf. Figure C.5). Annotated objects are indicated as icons on the map. Clicking an icon opens a popup window in which the annotation is displayed. The annotation can consist of a text, a numeric value, an image or a combination of them.

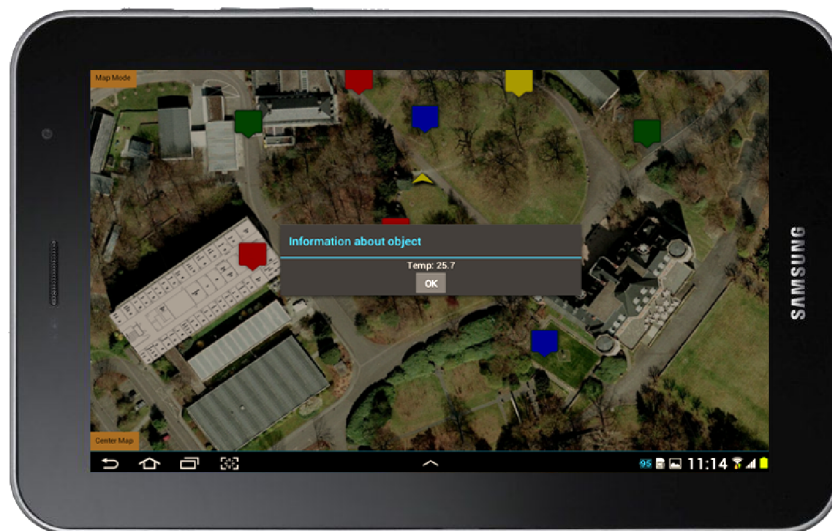


Figure C.5: UbiMap indoor

Configuration Options

UbiMap requires mapping physical objects to technology-specific attributes. There are also two global configuration options. The configuration options are summarized in Figure C.6.

The alignment of the map and its overlays is based on real world coordinates. So, UbiMap needs to know the real world coordinates of every overlay item. An icon is overlaid on the map for every annotated physical object. So, physical objects must be

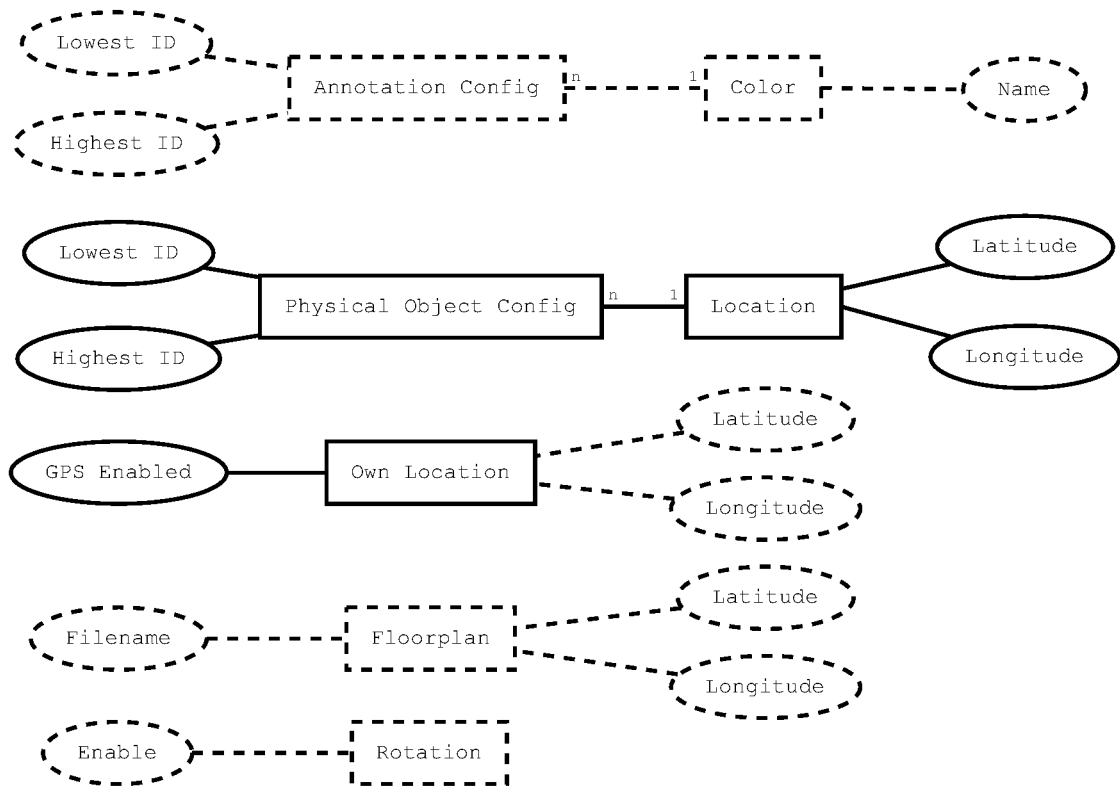


Figure C.6: Configuration options of UbiMap

mapped to locations. For these and all other location attributes of UbiMap, we use the same format as in UbiLens. A location consists of a latitude and longitude value in decimal WGS84 format.

For displaying the own location, UbiMap needs to know the location of the device. This global configuration option is equal to UbiLens. It is acquired by GPS if the parameter `GPS Enabled` is set true. Otherwise, a static location must be configured.

A building's floor plan is the final overlay which can be set in the configuration options. It consists of a filename and a location attribute. The filename points to an image file in the configuration folder. The location specifies the real world coordinate of the upper left pixel of the image. Since most buildings are not aligned to a degree of latitude, the image files must usually be rotated. The background must be set transparent so that the map can be seen around the floor plan. Therefore, it is recommended to use image file formats which support transparency. If the filename option is left empty, no indoor overlay is set.

C.2.3 UbiLight

UbiLight covers an ambient display approach of ubiAV systems, in which a classical screen has disappeared. Instead information is communicated by colored light patterns.

A flexible strip which consists of several LED elements is UbiLight's visualization hardware. Each element can be controlled to emit light in a different color. Physical objects are placed in front the strip. The area of the strip which is located behind a physical object is then illuminated in a homogeneous color. Different colors shall indicate different information about the physical object in front. The setup can be made more ambient by placing the strip in a way that the colored light is reflected by a wall. This way, the LED strip itself is hidden but the information transmission via illumination remains (cf. Figure C.7).

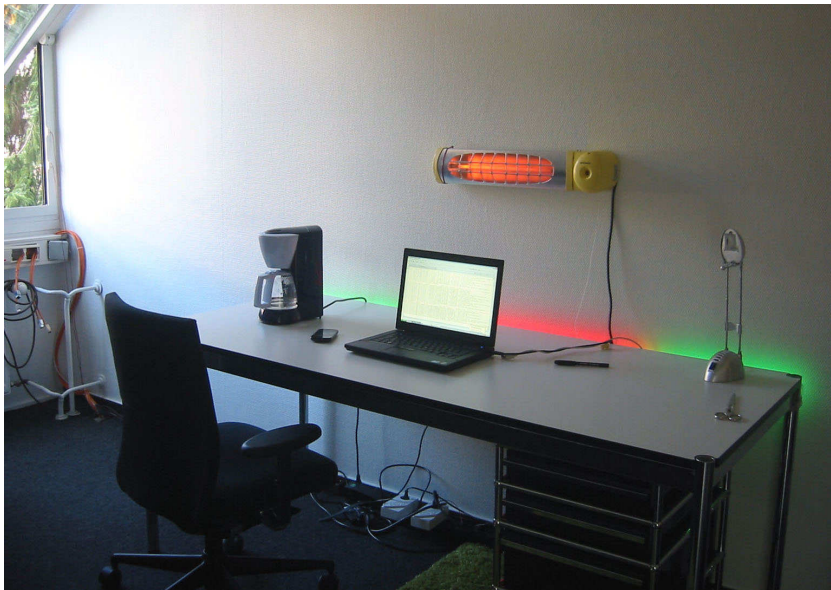


Figure C.7: UbiLight

Configuration Options

UbiLight requires mapping physical objects to technology-specific attributes as well as mapping digital information. Also one global configuration option is requested. The configuration is summarized in Figure C.8.

UbiLight illuminates a part of the LED strip in order to annotate a physical object in front of it. Hence, it needs to know where the physical object is located. So, each physical object must be mapped to a location attribute. The location is indicated by two integers. They represent the starting and ending LED element on the strip. Comprehending the strip as a one-dimensional axis of coordinate, we name these integers **Abscissa Start** and **Abscissa End**.

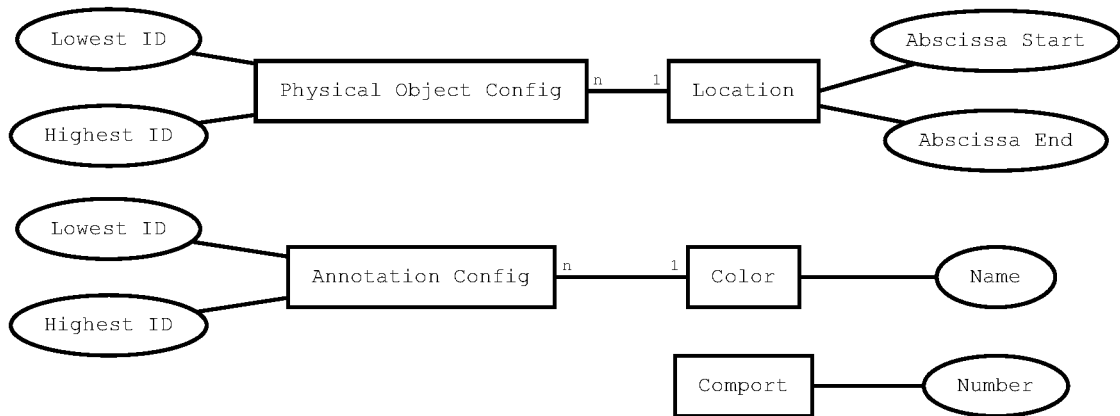


Figure C.8: Configuration options of UbiLight

The LED strip cannot display annotation information in a traditional way. It is not capable of visualizing texts and images. Hence, `Annotation` object must be mapped to colors which the strip can compute. So, for UbiLight, the data attributes of `Annotation` objects are neglected. Instead, the `annotationId` attribute is used for finding the corresponding color. Hence, ranges of `annotationIds` must be mapped to colors in the configuration file. The colors are specified by name. The following color names are supported: *white, red, green, blue, cyan, magenta, yellow*.

Both mappings are also used for switching lights off again. For this, an eighths "color" named *off* can be specified. This is handled in the same way as the other colors. So, when an application developer wants to switch a range of lights off again, she triggers a normal `visualize(visualizationTuple:VisualizationTuple)` call. The physical object's id of `VisualizationTuple` is then matched to the range of LEDs that she wants to switch off. And, the `annotationId` is mapped to the color *off*.

Finally, the number of the COM port to which the LED strip is connected must be specified as global configuration option.

C.2.4 UbiBoard

UbiBoard is a system for large screens which are situated in public areas and display content related to their environment. The screen is subdivided into several virtual tiles. Each tile can hold information about a physical object in the environment of the screen (cf. Figure C.9). The physical object is represented by an image and its name. The annotation fills up the rest of the tile.

Configuration Options

UbiBoard requires mapping physical objects to technology-specific attributes. The configuration options are summarized in Figure C.10.

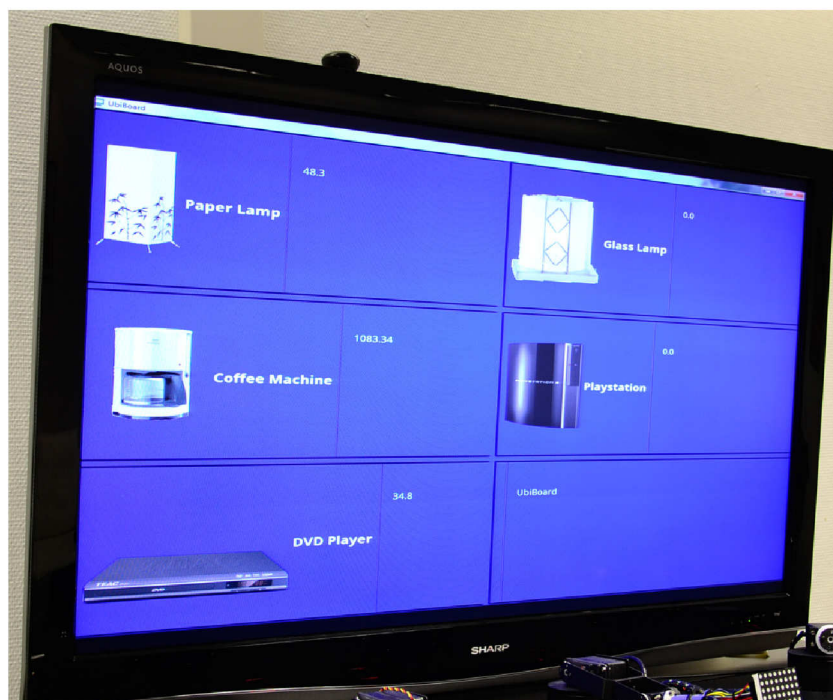


Figure C.9: UbiBoard

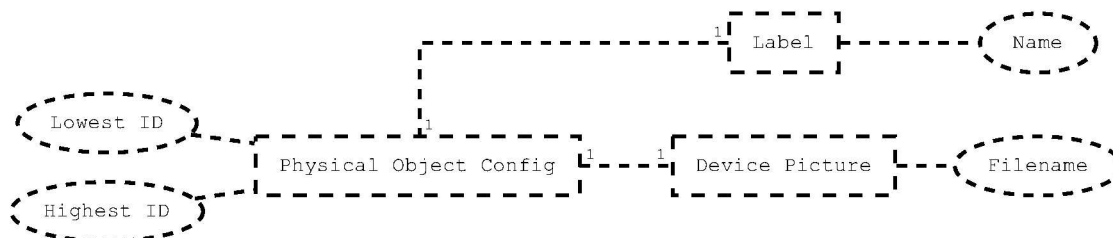


Figure C.10: Configuration options of UbiBoard

By default, UbiBoard represents every physical object by an image and a label. Thus, each physical object must be mapped to an image and a name label in the configuration. Similar to UbiMap, the image must be indicated by a filename. The filename points to an image file in the configuration folder. It is recommended to use a compressed file format, such as .png, .gif or .jpg, in order to save space and to improve performance. Physical objects in the environment are easier to identify if each object is represented by a unique image file and unique label. If the image or label is not specified, UbiBoard leaves the corresponding board space blank.

C.3 How do I apply UbiVis?

After having decided which visualization libraries shall be connected to your application, you need to follow a four step process for developing the software.

1. Analyze configuration of visualizations
2. (optional) Analyze data space of physical objects and annotations
3. (optional) Configure visualizations
4. Develop application logic including calls of VisualizationProxy

1. Analyze configuration of visualizations. Before connecting a visualization technology to the application, the technology must be configured. Each visualization library can provide a configuration file for mapping physical objects' IDs and digital information to technology-specific attributes. So, your first step is to check the configuration files of every visualization library which you want to apply. This will generate a list of what mappings are possible for the particular set of visualization libraries.

2. (optional) Analyze space of physical objects and annotations

Step 1 has revealed whether physical objects' IDs or annotations must be mapped to technology-specific attributes. If no mapping is required, Step 2 can be skipped. Otherwise, this next step is to analyze which physical objects and which annotations the application can expect. This can include several aspects. If the libraries require mapping annotation IDs to technology-specific attributes, you must determine a format for these IDs. You must be able to access the IDs in order to refer to them in the configuration. At the same time, the system must access the ID so that it can pass it to the constructor of the visualization tuple. Furthermore, the application developer might want to group or classify the IDs in some way.

Additionally, it can be necessary to set particular primitive data types of annotation objects, which would not be set by the application logic otherwise. For example, the application logic could actually only process image information but a visualization library might require to have the numeric annotation attribute set as well. Similar considerations have to be made for the space of the physical objects' IDs. Often, such an ID is mapped to specific attributes of a visualization library. Since you decide what the concrete instantiations of physical objects look like, you can also determine how these IDs are clustered. So, when a group of physical objects or annotations shall be mapped to the same attribute value, their corresponding IDs should be inside a collective range.

3. (optional) Configure visualizations

The next step is to do the actual configuration according to what was analyzed in the first two steps. The configuration file of each library might need to be edited. Besides the required mappings, it might also be possible to set some global attributes which are not

dependent on physical objects or annotations. In principle, it is possible that the visualization libraries do not have to be configured at all. In this case, Step 3 can be skipped.

4. Develop application logic including calls of VisualizationProxy

Finally, the actual development of the application logic can start. Create a new OSGi bundle and implement the application logic. Integrate visualization proxies and trigger the visualization by calling the defined `visualize(visualizationTuple:VisualizationTuple)` interface. For passing the visualization tuple parameters to the visualization proxies, the tuple objects must be created according to the analysis of Step 2.

C.4 Exemplary Walkthrough

We will walk through the process using a concrete example.

C.4.1 Example Application

The sample application visualizes input volumes of Bluetooth microphones. Assuming that several microphones are distributed in a noisy room, the volume of the particular microphones is varying. Different distances to sources of noise are a possible cause for that as well as technical reasons. We are interested in the particular input volume of each microphone because we want to rearrange them more evenly. But microphones mostly do not reveal this information. Bluetooth microphones provide the possibility to read the input volume from a computer via Bluetooth. We use the UbiMap visualization library for this sample.

1. Analyze configuration of visualizations.

UbiMap defines three configuration options. Two of them are global. First, the location of the user can be set to a fixed location or to the current GPS location. Second, a floor plan file including its coordinates shall be specified. (cf. Figure C.6).

The third configuration option is a mapping option. UbiMap needs to know the location of each annotated physical object. So, the configuration requires associating each physical object's ID with a latitude and longitude value. Latitude and longitude are expected in decimal WGS84 format, not using minutes and seconds. Annotation IDs do not have to be mapped for UbiMap.

2. Analyze data space of physical objects and annotations

Since UbiMap requires mapping physical objects' IDs to specific attributes, we need a method for creating microphone IDs. The IDs need to be accessed by us and also by the system. We must be able to enter the correct IDs in the configuration and thus must know which ID belongs to which microphone. The system must be able to gather the same ID because the application logic will create the ID attribute of the `PhysicalObject` object.

Since no two microphones are located at the exact same location, each individual physical object's ID is associated to a different location. Hence, grouping physical objects in order to assign them to the same location coordinates is not necessary.

We choose to use the Bluetooth MAC address of a microphone as its physical object ID. Every Bluetooth device includes this address and since it is unique it is well suited as identifier. The MAC address can be requested by the connected technology so that the system is able to access it. We can also easily access the Bluetooth address because it is attached on a sticker to the microphone.

As analyzed before, annotation IDs do not have to be mapped to technology-specific attributes. But we still check whether UbiMap can reasonably process the annotation data of our application. The annotation data consists of input volume values, so we are solely dealing with numeric data. UbiMap is capable of handling numeric, textual and image data in a way that is satisfactory for us. So, our application can just use the numeric data attribute of the `Annotation` object.

3. Configure Visualization

The final configuration step is to bring the reflections from the previous step into the structure of the configuration file. The UbiMap library already provides an XML file containing an initial structure. This just has to be copied and adjusted. Listing C.1 shows the configuration file for our sample application.

Listing C.1: UbiMap configuration file for the microphone application

```
<Configuration>
  <physicalObjectConfigurations class="linked-list">
    <PhysicalObjectConfig>
      <lowestId>8797854761468</lowestId>
      <highestId>8797854761468</highestId>
      <latitude>50.749612</latitude>
      <longitude>7.203817</longitude>
    </PhysicalObjectConfig>
    <PhysicalObjectConfig>
      <lowestId>8797854761387</lowestId>
      <highestId>8797854761387</highestId>
      <latitude>50.749646</latitude>
      <longitude>7.203843</longitude>
    </PhysicalObjectConfig>
    <PhysicalObjectConfig>
      <lowestId>8797854761402</lowestId>
      <highestId>8797854761402</highestId>
      <latitude>50.749591</latitude>
      <longitude>7.203887</longitude>
    </PhysicalObjectConfig>
  </physicalObjectConfigurations>
</ownLocation>
```



```

    <gpsEnabled>>false</gpsEnabled>
    <latitude>50.749619</latitude>
    <longitude>7.20387</longitude>
  </ownLocation>
  <floorplan>
    <floorplanFile>floormap.png</floorplanFile>
    <latitude>50.74935</latitude>
    <longitude>7.203685</longitude>
  </floorplan>
</Configuration>

```

4. Develop application logic including calls of VisualizationProxy

As first step, the main application registers itself at the UbiMapProxy by calling `setClientOnlineListener(this)`. Once, it is being notified about UbiMap being online through a call of `clientOnline(ClientOnlineListener.UBIMAP)` it initiates the application logic.

The application logic continuously requests the current input volumes from all microphones and triggers a visualization each time a volume has changed. For this, it constructs a new `VisualizationTuple`, using the microphones Bluetooth MAC address as id. The `Annotation` object is constructed using the updated volume as double value. The text and image of `Annotation` is left empty. In the example of Figure C.11, a new `VisualizationTuple` is created for the microphone with the `physicalObjectId=8797854761468`. The annotation object indicates a new input level of 78.0 (dB) and is assigned an arbitrary `annotationId=1`.

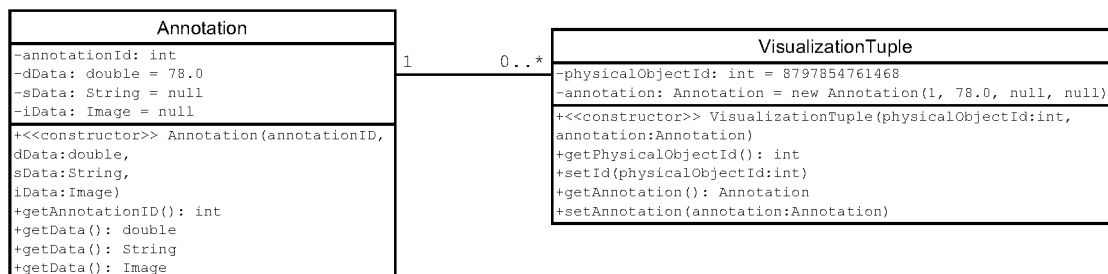


Figure C.11: Example instantiation of microphone application

C.5 The Programming Task

Create a system which displays the danger level for children of the household articles in this room. The particular danger levels and their representation are left to the application developers to decide. As household articles, please use the book, the lamp and the knife.

Task 1: There is an example application, which you checkout together with the framework and let it run. It uses UbiBoard for visualizing the danger level of the book. Add the danger levels for knife and lamp.

Task 2: Choose another library. Configure it and connect it to your application from Task 1 without changing the code for the application logic.

C.6 Development Instructions

Here is a guideline for your development steps.

Task 1

1. Connect your host to Wi-Fi. SSID: UbiVis. No password required
2. Also connect to LAN
3. Checkout the core UbiVis bundles from SVN (via subclipse):
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiAV>
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/Log4j>
4. Checkout UbiBoard components:
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiBoard>
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiBoardProxy>
5. Checkout the example application bundle:
<https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/MainApplication>
6. Start a first test run. A UbiBoard window opens and shows the danger level of the book.
Now it is time to add the knife and lamp. Remember the 4 step process
7. Open UbiBoardProxy/configuration/config.xml and add the configurations for knife and lamp
 - There are device picture files in UbiBoardProxy/configuration/img which you can use. You only have to specify the filename without path
8. Add code to the main application. If you want to use an image for your Annotation object, you can use the method `getExampleImage` for getting one.
9. Start your application and test it

Task 2

1. Download the components of your chosen library:

- UbiLens:
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiLensProxy>
- UbiMap:
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiMapProxy>
- UbiLight:
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiLight>
 - <https://subversion.fit.fraunhofer.de/moknow/papers/jentsch/UbiVisStudy/UbiLightProxy>

2. Remember the 4 step process. In your visualization proxy, open `/configuration/-config.xml` and configure it according to your needs. Here are some values you can work with:

- UbiLens:
 - Latitude knife: 50.749666
 - Longitude knife: 7.204040
 - Latitude book: 50.749661
 - Longitude book: 7.204015
 - Latitude lamp: 50.749668
 - Longitude lamp: 7.204069
 - Latitude own location: 50.749622
 - Longitude own location: 7.204048
- UbiMap:
 - Latitude and longitude values: see UbiLens
 - There is a floor plan file in `/configuration/img` which you can use. You only have to specify the filename without path
 - Latitude floor plan: 50.74935
 - Longitude floor plan: 7.203685
- UbiLight:
 - Abscissa start knife: 80
 - Abscissa end knife: 92
 - Abscissa start book: 1
 - Abscissa end book: 10
 - Abscissa start lamp: 150
 - Abscissa end lamp: 156
 - The comport can be found in the device manager once you are connected to the strip

Appendix C. Study - Handout

3. Extend the main application by triggering the new library.
 - Reference the library proxy
 - Register as ClientOnlineListener at the proxy
 - Trigger visualizations
4. For UbiLight: Connect your laptop to the strip
For UbiLens and UbiMap: The client is already deployed on the smartphones. If you are using a virtual machine, make sure that your network adapter is set to bridged
5. Start your application and test it


Appendix D

System Usability Scale

	Strongly disagree						Strongly agree
1. I think that I would like to use this system frequently	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
2. I found the system unnecessarily complex	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
3. I thought the system was easy to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
4. I think that I would need the support of an experienced person to be able to use this system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
5. I found the various functions in this system were well integrated	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
6. I thought there was too much inconsistency in this system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
7. I would imagine that most application developers would learn to use this system very quickly	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
8. I found the system very cumbersome to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
9. I felt very confident using the system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		
10. I needed to learn a lot of things before I could get going with this system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
	1	2	3	4	5		

Appendix E

AttrakDiff

■ Greeting ■ How it works ■ Your Evaluation ■ Personal Data ■ Submit[Deutsch](#)

Evaluation of the product UbiVis

We welcome your participation in the evaluation of the product UbiVis

Thank you for your time. Please read the following instructions carefully.

With your help we can evaluate the usability and appearance of the product **UbiVis** as experienced by the user. We hope to identify optimisation opportunities. These will enable us to optimise the product in such a way that it is as efficient and comprehensible as possible.

To participate in this study you will need to enter your access codes. You should already have received these via e-mail. Should your evaluation be interrupted, it is possible to submit another evaluation by 16.03.2014. In this case your previous information will not have been saved.

password: [password forgotten?](#)

Attention: [Cookies](#) must be activated to allow your data to be processed.



Evaluation of the product UbiVis

Following, are pairs of words to assist you in your evaluation. Each pair represents extreme contrasts. The possibilities between the extremes enable you to describe the intensity of the quality you choose.

An example:

disagreeable likeable

This evaluation tells us that the product is predominantly likable, but that there is marginal room for improvement.

Do not spend time thinking about the word-pairs. Try to give a spontaneous response. You may feel that some pairs of terms do not adequately describe the product. In this case please still be sure to give an answer. Keep in mind that there is no right or wrong answer. Your personal opinion is what counts!



Evaluation of the product UbiVis

With the help of the word-pairs please enter what you consider the most appropriate description for UbiVis. Please click on your choice in every line!

human	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	technical
isolating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	connective
pleasant	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unpleasant
inventive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	conventional
simple	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	complicated
professional	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unprofessional
ugly	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	attractive
practical	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	impractical
likeable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	disagreeable
cumbersome	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	straightforward

Evaluation of the product UbiVis

With the help of the word-pairs please enter what you consider the most appropriate description for **UbiVis**. Please click on your choice in every line!

stylish	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	tacky
predictable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unpredictable
cheap	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	premium
alienating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	integrating
brings me closer to people	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	separates me from people
unpresentable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	presentable
rejecting	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	inviting
unimaginative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	creative
good	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	bad

2/3

Evaluation of the product UbiVis

With the help of the word-pairs please enter what you consider the most appropriate description for **UbiVis**. Please click on your choice in every line!

confusing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	clearly structured
repelling	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	appealing
bold	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	cautious
innovative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	conservative
dull	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	captivating
undemanding	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	challenging
motivating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	discouraging
novel	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	ordinary
unruly	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	manageable

3/3

Evaluation of the product UbiVis

In the following section we ask for information about yourself and your own experience with the product.

Age:*

Gender:*

Qualifications:*

Profession:*

How long have you already been using the product UbiVis?

Product experience:*

Areas marked with * must be filled in.

Evaluation of the product UbiVis

Submitting your evaluation

The final step is to submit your evaluation data.

The data we receive from you will be anonymously analysed. A second participation in this study is not possible with the same access data.

Thank you very much for your participation. Your appraisal will be incorporated in our study.

Appendix F

User Experience Questionnaire

Please make your evaluation now.

For the assessment of the product, please fill out the following questionnaire. The questionnaire consists of pairs of contrasting attributes that may apply to the product. The circles between the attributes represent gradations between the opposites. You can express your agreement with the attributes by ticking the circle that most closely reflects your impression.

Example:

attractive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unattractive
------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	--------------

This response would mean that you rate the application as more attractive than unattractive.

Please decide spontaneously. Don't think too long about your decision to make sure that you convey your original impression.

Sometimes you may not be completely sure about your agreement with a particular attribute or you may find that the attribute does not apply completely to the particular product. Nevertheless, please tick a circle in every line.

It is your personal opinion that counts. Please remember: there is no wrong or right answer!

Appendix F. User Experience Questionnaire

Please assess the product now by ticking one circle per line.

	1	2	3	4	5	6	7		
annoying	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	enjoyable	1
not understandable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	understandable	2
creative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	dull	3
easy to learn	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	difficult to learn	4
valuable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	inferior	5
boring	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	exciting	6
not interesting	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	interesting	7
unpredictable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	predictable	8
fast	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	slow	9
inventive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	conventional	10
obstructive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	supportive	11
good	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	bad	12
complicated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	easy	13
unlikable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	pleasing	14
usual	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	leading edge	15
unpleasant	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	pleasant	16
secure	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	not secure	17
motivating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	demotivating	18
meets expectations	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	does not meet expectations	19
inefficient	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	efficient	20
clear	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	confusing	21
impractical	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	practical	22
organized	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	cluttered	23
attractive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unattractive	24
friendly	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unfriendly	25
conservative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	innovative	26
useless	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	useful	27
expandable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unexpansive	28
helpful	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	harmful	29
difficult to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	easy to use	30
time-consuming	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	time-saving	31
complicating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	facilitating	32