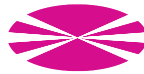


Escuela Politécnica Superior
Departamento de Computación

Doctoral Thesis

Autonomous Adaptation of User Interfaces During Application Mobility Processes in Ambient Intelligence Scenarios

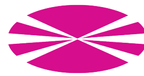
Adaptación autónoma de interfaces de usuario en procesos de movilidad de aplicaciones en escenarios de Inteligencia Ambiental

Gervasio Varela Fernández

PhD Advisors:

José Antonio Becerra Permuy
Richard J. Duro Fernández

7th April 2015

 UNIVERSIDADE DA CORUÑA

D. Richard J. Duro Fernández, Catedrático de Universidad del Departamento de Computación de la Universidade da Coruña,

D. José Antonio Becerra Permuy, Contratado Doctor del Departamento de Computación de la Universidade da Coruña,

CERTIFICAN:

Que la memoria titulada:

"Autonomous Adaptation of User Interfaces During Application Mobility Processes in Ambient Intelligence Scenarios"

"Adaptación autónoma de interfaces de usuario en procesos de movilidad de aplicaciones en escenarios de Inteligencia Ambiental"

ha sido realizada por **D. Gervasio Varela Fernández** bajo nuestra dirección en el Departamento de Computación de la Universidade da Coruña, y constituye la Tesis que presenta para optar al grado de Doctor.

Fdo. José Antonio Becerra Permuy	Fdo. Richard J. Duro Fernández
Codirector de la Tesis Doctoral	*Codirector de la Tesis Doctoral*

Abstract

Ambient Intelligence (AmI) is about systems that assist people to improve their quality of life. This work is focused on the problem of supporting the adapted interaction between those systems and their users in different usage scenarios. Unlike conventional software systems, AmI systems operate in what we have called Human Interaction Environments, which must be understood as any place where people carry out their daily life. As a consequence, the usage scenarios of an AmI system can be diverse and quite different from one another, thus making very difficult, and costly, the design of UIs capable of operating in many of them.

This work addresses the problem of supporting the adaptation of AmI UIs to the variety of scenarios through which a user moves while using an AmI system. For this purpose, this work introduces a UI abstraction framework that enhances the portability of AmI UIs. This framework elevates the level of decoupling between UI code, and the high variety of interaction resources and characteristics of each scenario. Furthermore, a complete and functional implementation of the framework is provided, enabling the development of AmI UIs capable of autonomously modifying, at run-time, their implementation to adapt it to new scenarios.

Resumen

El objetivo de la Inteligencia Ambiental (AmI) es desarrollar sistemas que mejoren la calidad de vida de las personas. Dentro de la AmI, este trabajo se centra en el problema de soportar, en diferentes escenarios, una interacción adaptada a cada usuario. A diferencia de los sistemas software convencionales, los sistemas AmI operan en entornos altamente heterogéneos que hemos denominado Entornos de Interacción Humana (HIE), y que abarcan cualquier lugar en el que las personas llevan a cabo su vida diaria. Esta diversidad hace que pueda haber notables diferencias entre los múltiples escenarios de uso de un sistema AmI, dificultando enormemente el diseño de IUs que operen en varios de ellos.

Este trabajo aborda el problema de facilitar la adaptación de las IUs de sistemas AmI a la variedad de escenarios incluidos en un HIE. Para ello, se presenta un framework de abstracción que mejora la portabilidad de las IUs, elevando el nivel de desacoplamiento entre el código y la diversidad de recursos de interacción y características de cada escenario. Además, también se presenta una implementación completa y funcional del framework, posibilitando el desarrollo de IUs capaces de modificar autónomamente, y en tiempo de ejecución, su implementación, adaptándola a nuevos escenarios.

Resumo

O obxectivo da intelixencia ambiental (AmI) é desenvolver sistemas que melloren a calidade de vida das persoas. Dentro da AmI, este traballo céntrase no problema de soportar, en diferentes escenarios, unha interacción adaptada a cada usuario. A diferenza dos sistemas de software convencionais, os sistemas AmI operan en ámbitos altamente heteroxéneos que denominamos ámbitos de interacción humana, que abranguen calquera lugar no que as persoas levan a cabo a súa vida diaria. Esta diversidade fai que poidan existir notables diferenzas entre os múltiples escenarios de uso dun sistema AmI, o que dificulta enormemente o deseño de IU que operen en varios deles.

Este traballo aborda o problema de facilitar a adaptación das IU de sistemas AmI á variedade de escenarios incluídos nun ámbito de interacción humana. Para isto, preséntase un framework de abstracción que mellora a portabilidade das IU, o que eleva o nivel de desacoplamento entre o código e a diversidade de recursos de interacción e as características de cada escenario. Ademais, tamén se presenta unha implementación completa e funcional do devandito framework, o que posibilita o desenvolvemento de IU capaces de modificar autonomamente, e en tempo de execución, a súa implementación e adaptación a novos escenarios.

Acknowledgements

Firstly I wish to thank my PhD advisors, Richard and José Antonio. Without their passion, enthusiasm, and trust, this work would have never been possible.

I would also wish to thank the support and collaboration of all the members of the Integrated Group for Engineering Research of the University of A Coruña. We have become more than colleagues, creating a great working environment in which it is a privilege to work.

Even if a doctoral thesis is a personal achievement, in practice, it is not viable without the collaboration of many people. At the risk of forgetting someone, I would like to particularly thank some people for their special contribution to this work. Alex, because we have suffered together the hard labor that have been the diverse efforts of the group in the field of Ambient Intelligence and the birth of HI3. *¡Ánimo! ¡Tu también estás a punto de acabar!.* Victor, for his invaluable collaboration in the implementation of UniDA. Santi, for his perseverance in keeping the HI3 idea alive. Juan Carlos, for the great front cover he has designed for this work. And, of course, to the "hardware sector", Andrés, Álvaro, Félix and Martín, for making many of our ideas a physical reality.

I don't want to miss the opportunity to mention my colleagues of the Louvain Interaction Laboratory, of the Université Catholique de Louvain. Thank you for your warm welcome to Belgium, and special thanks to Jean Vanderdonckt, for giving me the opportunity to collaborate with them.

I also wish to thank the different companies whose research projects has served as testbed examples for some of the technologies developed within this thesis. SCIO Innovation Technologies, for his collaboration in the development of the OMNI system, one of the main examples used in this thesis. Ghenova Ingeniería, because their EMERBUQUE project has served as inspiration for some examples shown in this thesis. And Mytech Ingenieria Aplicada, for his important collaboration in the development of the UniDA technology.

This work has been partially founded by a predoctoral research grant from

the Univerisity of A Coruña, and an Inditex-UDC predoctoral research stay grant from the University of A Coruña and Inditex S. A.

Finally, I want to take this opportunity to give especial thanks to the most important people in my life, those without which nothing would be possible.

To my parents and my sister, they instilled in me the desire to learn and explore, and they gave me the opportunity to go beyond that. *¡Gracias por todo lo que habéis hecho por mi!.*

To my future wife, Sara, her constant affection and support has been essential for this adventure to have come to fruition. *¡Gracias por estar siempre ahí!.*

To my friends, I don't know what my life would be without those "cafeses" at the terrace of the Valle Inclán. *¡Gracias por ser los mejores!.*

Publications

For the development of the present work, the following articles related with the main topic of the thesis have been published:

- Gervasio Varela, Alejandro Paz-Lopez, Jose Antonio Becerra Permuy, Richard J. Duro Fernandez, *Prototyping Distributed Physical User Interfaces in Ambient Intelligence Setups*, Proceedings of the 2nd International Conference on Distributed, Ambient and Pervasive Environments (DAPI 2014), pp 76 - 85, Heraklion, Greece, Springer, 2014.
- Varela G., Paz-Lopez A., Becerra J.A. and Duro R.J, *The Generic Interaction Protocol: Increasing portability of distributed physical user interfaces*, Romanian Journal of Human - Computer Interaction, 6 (3), pp 249 - 268, 2013.
- A. Paz-Lopez, G. Varela, J.A. Becerra, S. Vazquez-Rodriguez, R.J. Duro, *Towards ubiquity in ambient intelligence: User-guided component mobility in the HI3 architecture*, Science of Computer Programming, 78 (10), pp 1971 - 1986, Elsevier, 2013.
- Gervasio Varela, *Autonomous adaptation of user interfaces to support mobility in ambient intelligence systems*, Proceedings of the 5th ACM SIGCHI symposium on Engineering Interactive Computing Systems (EICS 2013), pp 179 - 182, London, U.K., ACM, 2013.
- G. Varela, A. Paz-Lopez, J. A. Becerra , R. J. Duro, *Decoupled Distributed User Interface Development Framework for Ambient Intelligence Systems*, Proceedings of the 3rd Workshop on Distributed User Interfaces: Models, Methods and Tools (DUI 2013), pp 23 - 26, London, U.K., 2013.
- Gervasio Varela, Alejandro Paz-Lopez, Jose A. Becerra, Richard J. Duro, *Dandelion: Decoupled Distributed User Interfaces in the HI3 Ambient Intelligence Platform*, Proceedings of the 6th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2012), pp 161 - 164, Vitoria, Spain, Springer, 2012.

- A. Paz-Lopez, G. Varela, V. Sonora, J. A. Becerra, *DAAF: a Device Abstraction and Aggregation Framework for Smart Environments*, Proceedings of the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), pp 739 - 744, Palermo, Italy, IEEE, 2012.
- G. Varela, A. Paz-Lopez, J. A. Becerra, S. Vazquez-Rodriguez, R. J. Duro, *Towards Mobility in Ambient Intelligence: Component Migration and Adaptation Strategies in the HI3 Architecture*, Proceedings of the 5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2011), Riviera Maya, Mexico, 2011.
- Gervasio Varela, Alejandro Paz-Lopez, Jose Antonio Becerra, Santiago Vazquez-Rodriguez and Richard José Duro, *UniDA: Uniform Device Access Framework for Human Interaction Environments*, Sensors, 11 (10), pp 9361 - 9392, MDPI, 2011.
- A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra and R. J. Duro, *Some Issues and Extensions of JADE to Cope with Multi-agent Operation in the Context of Ambient Inteligence*, Trends in Practical Applications of Agents and Multiagent Systems, pp 607 - 614, Springer, 2010.
- A. Paz-Lopez, G. Varela, S. Vazquez-Rodriguez, J. A. Becerra and R. J. Duro, *Integrating Ambient Intelligence Technologies Using an Architectural Approach*, Ambient Intelligence, pp 1 - 26, INTECH Open Access Publisher, 2010.
- A. Paz-Lopez, G. Varela, J.Monroy, S.Vazquez-Rodriguez, R.J. Duro, *HI3 Project: Software Architecture System for Elderly Care in a Retirement Home*, 3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008 (UCAmI 2008), pp 11 - 20, Springer Berlin Heidelberg, 2008.
- G. Varela, A. Paz-López, S. Vázquez-Rodríguez, R. J. Duro, *HI3 Project: Design and Implementation of the Lower Level Layers*, Proceedings of the 2007 IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems (VECIMS 2007), pp 36 - 41, IEEE, 2007.

Contents

Abstract **v**

Acknowledgements **xi**

Publications **xiii**

1 Introduction **1**

2 Objectives **11**

3 Related Work **13**

3.1 Introduction 13

3.2 Physical User Interfaces 15

3.2.1 Home Automation and Internet of the Things 16

3.2.2 Ambient Intelligence and Ubiquitous Computing Frameworks 18

3.2.3 Physical User Interface Frameworks 19

3.2.4 Summary 21

3.3 Plasticity in Physical User Interfaces 22

3.3.1 Model-Driven Engineering to Achieve UI Plasticity 23

3.3.2 UI Plasticity in Ambient Intelligence and Ubiquitous Computing Frameworks 27

3.3.3 UI Adaptation to the User Characteristics 29

3.3.4 Summary 31

3.4 Discussion 32

4 Analysis and Design of a Framework for Ambient Intelligence UI Development 39

4.1 Introduction 39

4.2 Analysis of the Characteristics of UIs in Ambient Intelligence Systems 40

4.2.1 OMNI Virtual Assistant 41

4.2.2 Environmental Music Player 44

4.2.3 Intelligent Ship Passenger Evacuation System 46

4.3 Supporting Ambient Intelligence UI Development 48

4.3.1 Requirements of AmI UIs 48

4.3.2 The Threefold Interaction Abstraction Framework 49

4.3.2.1 Interaction Modality Abstraction 50

4.3.2.2 Interaction Location Abstraction 53

4.3.2.3 Interaction Context Abstraction 57

4.3.2.4 The Abstract Interaction Model 60

4.3.2.5 The Generic Interaction Protocol 64

4.3.2.6 The Context Models 66

5 Supporting Portable and Distributed Physical User Interfaces 69

5.1 Introduction 69

5.2 The Dandelion Framework 71

5.3 Portable Physical User Interfaces 74

5.3.1 Abstract UI Design and Specification 76

5.3.2 UI Control Logic Implementation 80

5.3.3 From the Abstract to the Final User Interface 85

5.4 Distributed Physical User Interfaces 86

5.4.1 The Generic Interaction Protocol 88

5.4.2 Final Interaction Objects 90

5.5 Physical Device Access and Control 92

5.5.1 UniDA Conceptual Framework 96

5.5.1.1 Device Network Model 96

5.5.1.2 Uniform Device Access Paradigm 100

5.5.1.3 Distributed Operation Protocol 101

5.5.2 UniDA Framework Implementation 102
5.5.2.1 UniDA Library 103
5.5.2.2 UniDA Gateways 104
5.6 Demonstration Examples and Summary 105
5.6.1 OMNI Virtual Assistant 106
5.6.2 Environmental Music Player 116
5.6.3 Summary 121

6 Adding Real-Time, Autonomous and Dynamic Adaptation to Physical User Interfaces 125
6.1 Introduction 125
6.2 Physical UI Adaptation to Context 127
6.3 Context Models 130
6.3.1 User Profile Model 131
6.3.2 Environment Profile Model 135
6.3.3 Scene Profile Model 138
6.4 Autonomous Selection of Interaction Resources 139
6.4.1 Generating Specifications of the Ideal FIO 140
6.4.1.1 FIO Description Model 141
6.4.1.2 Ideal FIO Specification Model 144
6.4.1.3 Ideal FIO Generation 144
6.4.2 Selecting Adequate FIOs 147
6.4.2.1 FIO Adequateness Calculation using the Fuzzy Geometric Model 150
6.4.3 Building the Final User Interface 153
6.5 Demonstration Examples and Summary 155
6.5.1 Environment Adaptation: Environmental Music Player . . 156
6.5.2 User Adaptation: OMNI Virtual Assistant 165
6.5.3 Environment and User Adaptation: EvacUI 172
6.5.4 Summary 181

7 Conclusions and Future Work 183

A Resumen en castellano 193

References **201**

Chapter 1

Introduction

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."*
>
> Mark Weiser

Ambient Intelligence (AmI) is about systems that assist people to carry out their daily tasks in a transparently assisted way that improves their quality of life. This PhD thesis is focused on the interaction between people and those systems, and more specifically, on the problem of supporting the adapted interaction between the system and each user in the different scenarios in which an AmI system may be operated.

Ambient Intelligence is such a broad field that it is difficult to agree on a concise and accurate definition of the concept. However as AmI systems are going to be the central aspect of this PhD dissertation, it seems mandatory to start by establishing a definition of Ambient Intelligence. A commonly accepted definition is the one provided by J.C. Augusto and P. McCullang in their article "Ambient Intelligence: Concepts and Applications" [Augusto and McCullagh, 2007]:

> "The basic idea behind AmI is that by enriching an environment with technology (mainly sensors and devices interconnected through a network), a system can be built to take decisions to benefit the users of that environment based on real-time information gathered and historical data accumulated. AmI inherits aspects of many cognate areas of Computer Science, but should not be confused with any of those in particular. Networks, Sensors, Human Computer

> Interfaces (HCI), Pervasive Ubiquitous Computing and Artificial Intelligence (AI) are all relevant and inter-related but none of them conceptually covers the full scope of AmI. Ambient Intelligence puts together all these resources to provide flexible and intelligent services to users acting in their environments. AmI is aligned with the concept of the disappearing computer.", J.C Augusto and P. McCullagh, 2007.

This definition of AmI, even though it is quite broad, shows the two main aspects of any Ambient Intelligence system:

1. *The goal.* Improve people's life quality by making their life more comfortable and facilitating their daily tasks.
2. *The path.* By enriching physical environments with technology.

While this definition gives us a broad goal and a general path on how to achieve that goal, it does not provide any clues about the characteristics that are expected from an AmI system. By breaking up Ambient Intelligence into its individual terms we can start extracting those characteristics. On the one hand, Intelligence characterizes the response that users expect from the system, that is, proactivity, predictability and adaptability in its behaviours (functionalities offered by the system). On the other hand, the term Ambient is related to ubiquitousness and human factors, characterizing the system as a non invasive one in terms of its deployment and user interaction. The first term is directly related to the application of Artificial Intelligence techniques to achieve the expected smart behavior, while the second one is about the application of Ubiquitous Computing solutions to build systems that, paraphrasing Weisser [Weiser, 1991], "weave themselves into the fabric of everyday life until they are indistinguishable from it".

As can be extracted from this definition and specification of the characteristics of AmI systems, their scope and operating environment make them quite different from traditional software systems. It is true that they share many common similarities and even technologies with widely studied systems like distributed systems or Artificial Intelligence systems, but they have key important differences in their own operational nature that directly impact the way in which AmI systems must be conceived, designed and implemented.

Unlike traditional software systems, which usually operate in a reactive or on-demand way, AmI systems are expected to operate in a proactive way, behind the scenes, showing an autonomous behavior that allows them to anticipate user needs and provide their output using the most integrated and natural way adapted to each use scenario.

To complicate things a little bit more, usage scenarios of AmI systems are usually much more complex than those of traditional software systems. AmI systems operate in what we have called Human Interaction Environments or HIEs [Varela et al., 2011]. These environments must be understood as any place where people make their daily life, including their work, family life, leisure and social life. Therefore an HIE can include many different physical locations such as the workplace, the home, the car, and public spaces such as malls or sport centers, among others. Hence the usage scenarios of an AmI system can be diverse and quite different from one another. They can present different environmental conditions (lighting, noise, privacy, etc.), different usage constraints (is not the same to operate a system while driving than while watching a movie), different user characteristics (abilities, vision, hearing, etc.) and last, but not least, each scenario can have very different interaction resources (hardware devices and software) available to interact with the user and the environment itself.

As if the complex operating environment conditions were not enough, in order to fulfill their objective of facilitating people's daily life, AmI systems are expected to be strong performers in two key aspects [Cook et al., 2009, Abascal, 2004]: user-adapted natural interaction [Abascal et al., 2011a, Abascal et al., 2011b, Blumendorf and Albayrak, 2009, Blythe et al., 2005, Kranz et al., 2010, Blumendorf et al., 2010] and ubiquitous operation [Aizpurua et al., 2013, Blumendorf, 2009, Luyten et al., 2006, Luyten and Coninx, 2005].

Natural interaction is all about a user experience where the UI remains unobtrusive and almost invisible, blended into the physical environment [Fishkin et al., 1999, Harrison et al., 1998, Ballagas et al., 2003, Xie et al., 2008]. Natural user interfaces (NUIs) take advantage of our senses and our own knowledge about the objects, physics and the world itself in order to build user interfaces that liberate the user from having to learn new concepts to interact with computers [Ishii and Ullmer, 1997, Ullmer and Ishii, 2000, Sitdhisanguan et al., 2012]. NUIs try to make the UI invisible by relying on devices embedded in the environment and by using the most familiar interaction modality available for each environment and user.

Ubiquity pursues the idea of being available anywhere at anytime [Ranganathan et al., 2004, Ranganathan et al., 2005, Satoh, 2005]. Regarding AmI systems and HIEs, ubiquity is a feature that allows a system to provide its functionality in any of the places integrated in a specific HIE, thus supporting the mobility of users inside their own HIEs.

Ubiquity, combined with NUIs, requires AmI systems to be deployed with different configurations of the UI adapted to each scenario [Sousa and Garlan, 2002, Ranganathan et al., 2005]. When the user moves from one place to another, the execution scenario of the AmI system changes, and with it, the

available devices, the characteristics of the environment and even the users. Given the high diversity of scenarios that could exist, predicting them during the design and development stages of a system is very complex. Because of this, the majority of the UIs of AmI systems are designed and implemented for a specific scenario or set of scenarios, with a concrete set of devices and type of user. Deploying these UIs in new, not predicted scenarios, is quite complex, normally requiring modifications of the system and, therefore, hindering the portability of AmI systems, as well as the mobility of systems and users.

An adapted UI is one of the most important features of an AmI system in order to be successfully accepted by its users [Ranganathan et al., 2005, Abascal et al., 2008, Cook et al., 2009, Pavan Dadlani, 2011, Sitdhisanguan et al., 2012, Zuckerman and Gal-Oz, 2013]. For instance, a typical area of application of AmI systems is to improve the quality of life of people with disabilities, elderly people or kids, three collectives where an adapted UI makes the difference between a successful and a failed system [Abascal, 2004, Abascal et al., 2011b, Gajos et al., 2008b]. AmI systems usually rely on NUIs to build highly customized solutions for those users. In order to construct them, AmI developers employ different interaction resources (IRs) adapted to each scenario (user and environment) characteristics [Pavan Dadlani, 2011]. These devices come from different manufactures, use heterogeneous protocols and APIs, and in some scenarios, even employ custom hardware that is especially built for the system and the scenario. Furthermore they are usually embedded and distributed throughout the physical environment, configuring those customized UIs as physical distributed UIs. Supporting this variety of devices and interaction modalities is not an easy goal, and can introduce a lot of complexity in a system. Furthermore, developing specific UIs for each possible scenario increases development times and costs dramatically, and in many cases, it may be just unfeasible to predict all the possible usage scenarios at development time.

The main objective of the work carried out in this PhD thesis is to enhance the support for ubiquitous operation and user mobility in Ambient Intelligence systems. As has been seen, one of the main obstacles to achieve ubiquitous AmI systems is to support the wide range of user interaction techniques and devices required to obtain UIs adapted to the user and the environment. To tackle this problem, this PhD thesis proposes increasing the level of decoupling between system software and the interaction resources that made up the UI. The objective is to improve the portability of AmI systems and with it, their capability to support adapted interaction with different users in multiple environments.

This work poses an improvement at three different levels in the decoupling between developers, system software and interaction resources. First, at the logical level the idea is to decouple system logic developers and their code from

the specific modalities, technologies and APIs of the IRs used to build the UI. Second, at the physical level, we want to isolate the system software from the physical location of the interaction resources. And third, in order to support ubiquitous operation and facilitate the deployment of the AmI systems in multiple environments, we aspire to increase the isolation between UI developers and the concrete set of IRs used for each environment.

Better isolation between the system logic code and the final shape of the UI will facilitate the deployment of the same business logic with different realizations of the UI, moreover it would make the installation of AmI systems in HIEs that include multiple locations and users easier. An improvement of the decoupling in terms of interaction modalities and APIs of the interaction resources will allow developers to change the implementation of the IRs without affecting logic code, thus opening the possibility of deploying the same code in diverse physical scenarios with different IRs for each deployment. This PhD thesis proposes the utilization of model driven engineering (MDE) techniques to improve the logic decoupling level between AmI systems and their UIs. MDE techniques allow developers to build systems using high level models and then rely on a set of transformation steps to translate those models into real operating software. MDE approaches have been previously applied to UI development with proven results. Thevenin and Coutaz proposed the use of model-driven engineering techniques in order to support 'plasticity of user interfaces' [Thevenin and Coutaz, 1999, Coutaz, 2010], namely, the capacity of a UI to support changes in the system's physical characteristics and in the environment while preserving usability, by applying transformations of models at design time. This proposal has been very successful within the HCI community and many different authors have used it as the basis for their own approach to UI adaptation [Collignon et al., 2008, Blumendorf et al., 2010, Luyten, 2003], even in the field of Ambient Assisted Living and AmI [Abascal et al., 2008, Blumendorf and Albayrak, 2009, Blumendorf, 2009, Abascal et al., 2011b, Abascal et al., 2011a]. Nevertheless, all of those approaches have been focused on graphical, voice or gestures based UIs, with little, or no support at all, for other modalities. On the contrary, this PhD thesis will provide a model based UI development framework with integrated support for multimodal distributed physical UIs and NUIs, and UI adaptation to context at runtime.

As previously indicated in this chapter, the intrinsic nature of AmI systems usually requires building their UIs on top of a set of physical devices distributed throughout the environment [Pavan Dadlani, 2011, Luyten and Coninx, 2005, Luyten et al., 2006]. Improving the logical separation between AmI system code and the physical location of IRs will increment the portability of AmI systems, making their deployment in diverse environments with different physical

distributions of the IRs easier. The work of this PhD thesis proposes extending the previously explained MDE approach to include support for physical decoupling. The proposal is to build the model-driven UI abstraction layer on top of a distributed IR access layer that encapsulates the network technologies and protocols required by the diverse IRs available in each location. By relying on this abstraction layer, developers would be able to develop their systems without specific knowledge of those protocols and technologies, and the same AmI system logic could use the best suited devices for each scenario without modifying their code. Moreover, this solution will allow AmI UI developers to design and build distributed UIs specialized for each environment that are independent from the system logic. UI distribution using MDE approaches has also been previously explored by the research community. Two prominent approaches are: the Cameleon-rt [Balme et al., 2004] reference model for distributed, migratable and adaptive user interfaces; and the W3C conceptual framework to support multimodal UIs [W3C, 2003]. The work of this PhD thesis is inspired on the reference model proposed by Cameleon-rt, but improving it to better support distributed physical user interfaces. First, the abstract UI model is directly transformed into a final executable UI at runtime, furthermore, this transformation is done during the migration of an UI, and it uses context information to build a final UI adapted to the new context. Second, the platform layer is modeled and implemented as a distributed interaction resource abstraction layer, and it supports any kind of physical or digital IR.

By relying on the two previously explained decoupling solutions, developers would be able to design and implement AmI system logic with low or no knowledge about the modalities, protocols, network technologies and APIs of the IRs used for the interaction with the user. Furthermore UI designers and developers would be able to build their IRs using any technology and device that suit the needs of the scenario without interfering with system developers. It is at deploy time that the system logic would be connected to a set of IRs (that could be different for each scenario), thus providing the system with a distributed UI adapted to the scenario. Nevertheless, in order to support ubiquitous operation, AmI systems should be able to operate in any place included in the HIE of its users, and they should be able to change the UI as the user moves from one place to another. Each HIE would have diverse places and scenarios, which, in turn would use a variety of IRs. As previously mentioned in this chapter, designing the UIs for this variety seems unfeasible due to costs and time, therefore, this PhD proposes a third level of abstraction to isolate the AmI system and developers from the specific IRs used to implement the UI in each scenario.

The main idea behind this third level of abstraction is to postpone the

assembling of the final UI until runtime, and use models about the scenario to automate the selection of the IRs included in the final UI. This selection process will be done by the system itself, that will assemble the final UI as a collection of IRs that are adequate for each scenario's characteristics. The IRs will be selected among those available in each environment and connected to the system logic using the two previously explained abstraction layers.

The problems addressed by these three levels of abstraction, UI adaptation, reusability and migration in AmI applications, have been previously identified by different authors [Miñón and Abascal, 2012, Aizpurua et al., 2013]. In [Blumendorf, 2009, Blumendorf et al., 2010, Blumendorf and Albayrak, 2009] Blumendorf et al. introduce the problematic associated with user interaction in Ambient Assisted Living (AAL) environments, which are a subset of AmI. The paper presents a framework for UI development that also uses a MDE approach and context information to drive the adaptation at runtime. In [Abascal et al., 2008, Abascal et al., 2011b, Abascal et al., 2011a] Abascal et al. identify the necessity of adaptation to the users, because their capabilities and disabilities can greatly impact the performance of the UI of an AmI system. Nevertheless, these solutions have focused on two modalities, GUIs and voice UIs, and do not include support for physical UIs that require the use of distributed physical devices with a variety of modalities. Furthermore, only Egoki [Abascal et al., 2011b] supports adaptation to the user abilities and preferences, and none of them support UI adaptation to the physical environment characteristics, which is of great importance in AmI and UC systems, where the physical scenario and device available can change dramatically from one location to another.

An Ambient Intelligence development platform providing these three levels of abstraction between system and UI logic will effectively increase the support for ubiquitous operation in AmI systems. Even if it is only a part of a solution for ubiquitous operation, it will no only make the development of AmI systems that support their deployment in multiple scenarios (environments and users) easier, but it will also make the migration, in real-time, of AmI systems, and their users, from one environment to another without modifying the system or interaction logic code. It is important to note that autonomously built UIs will hardly be better for a specific environment than a specially designed one, nevertheless they present big advantages when a priori it is not possible to know the details of the environment and users that will operate the UI a priori, and thus, it is not possible to build an specific UI.

Summarizing, this PhD thesis addresses the problem of supporting the adaptation of AmI system UIs to the multiple scenarios through which a user moves while using an AmI system. This PhD thesis study solutions to enhance the support for ubiquitous operation of AmI systems by facilitating their

portability and mobility. It aims to provide developers with a series of technologies that allow them to decouple their systems from the wide variety of interaction resources used to interact with the users. These technologies will facilitate the development of AmI systems that are more portable and more easily deployable in new scenarios, with new user and environment characteristics. They will make the mobility of systems and users possible thanks to the ability of dynamically adapting the interaction subsystem to the characteristics of the scenario. With that objective in mind, this PhD thesis proposes a new UI abstraction technology for the development of distributed physical UIs, along with an autonomous mechanism for the selection, in real-time, of the most adequate interaction resources for each usage scenario. This two proposed solutions are implemented as an UI management system for Ambient Intelligence systems, named Dandelion, that facilitates the development and deployment of Distributed Physical User Interfaces capable of operate ubiquitously by migrating from one scenario to a different one while keeping their UIs within the constraints of natural interaction.

Structure of this thesis report

The remaining of this thesis report is structured as follows.

In chapter 2 the global objectives of this PhD thesis are established.

Chapter 3 is dedicated to review the most relevant works related to the topic of this thesis. Furthermore, a comparison framework for Plastic Distributed User Interface development solutions is introduced.

In chapter 4 we provide a thorough introduction to the field of Ambient Intelligence User Interface development, with special emphasis on exploring various different examples of AmI UIs to identify their characteristics and requirements. In chapter 4 we also introduce the Threefold Interaction Abstraction Framework (TIAF), a conceptual abstraction framework for the development of Distributed Physical User Interfaces for Ambient Intelligence.

Chapter 5 is dedicated to present Dandelion, a reference implementation of the TIAF conceptual framework previously introduced in chapter 4. Chapter 5 is specially dedicated to explore the first two levels of abstraction proposed by the TIAF. The first one, the Interaction Modality Abstraction level (IMA), in charge of decoupling developers from the modalities and technologies of the distributed interaction resources. And the second one, the Interaction Location Abstraction level (ILA), in charge of decoupling developers and their code from the physical location of the interaction resources.

Chapter 6 is devoted to explore in detail the third abstraction level proposed

by the TIAF. The Interaction Context Abstraction level (ICA), that is in charge of decoupling the UI from the particular context of use, enabling a UI to be autonomously adapted, at run-time, to the characteristics of different usage scenarios.

Finally, chapter 7 summarizes the main conclusions drawn from this work.

Chapter 2

Objectives

"It is good to have an end to journey toward; but it is the journey that matters, in the end."

Ernest Hemingway

The general goal of this PhD thesis is to improve the support for ubiquitous operation and user mobility in Ambient Intelligence (AmI) and Ubiquitous Computing (UC) systems.

Within that very broad goal, this PhD thesis aims to support ubiquitous AmI and UC systems by facilitating the development of distributed physical user interfaces (DPUIs) capable of operating in multiple scenarios with different devices, characteristics and constraints. Two fundamental goals arise from this main idea. On the one hand, to enhance the migrability of distributed physical user interfaces, so that they can be more easily deployed on a variety of scenarios and migrated between them. On the other hand, to facilitate the development of DPUIs capable of adapting their way of interaction with the user to the requirements and characteristics of different scenarios, namely, the requirements and characteristics of different users in a variety of environments.

In order to focalize and guide the work to be carried out in this PhD thesis, it seems necessary to break down those two central goals into more specific objectives that could be used as a reference for the research, design and development of the solutions proposed in this work. In what follows, we provide a list of objectives that this PhD thesis seeks to address:

- Conceive and implement mechanisms to support the design and implementation of AmI interaction and business logic with low or no knowledge

about the technologies, modalities, protocols and APIs used by the interaction resources (IRs) that made up their UI (mainly DPUIs). That is to say, facilitate the decoupling between system and user interaction logic and the IRs.

- Facilitate the development of distributed UIs capable of operating independently of the physical location of the IRs used to interact with the users.
- Facilitate the deployment of the same AmI system with different realizations of the UI. The main idea is to allow developers and installers to modify the setup of IRs used to build a DPUI without affecting the system's code.
- Provide support for the migration of AmI applications from one scenario to another without requiring modifications in the system or interaction logic code.
- Reduce the cost of developing physical distributed UIs adapted to different uses by facilitating the prototyping and testing of different physical configurations of the UI.
- Reduce the costs, in time and effort, of supporting multiple different scenarios by the same AmI application. The goal is to provide mechanisms that allow AmI applications to dynamically, and automatically, adapt their UIs (DPUIs) to the characteristics and requirements of different combinations of users, environments and devices available.

Chapter 3

Related Work

"It's useful to go out of this world and see it from the perspective of another one."

Terry Pratchet

3.1 Introduction

The main premise behind this PhD thesis is that one of the obstacles to achieve truly ubiquitous AmI systems capable of operating in a diversity of usage scenarios, while maintaining an acceptable level of user experience, is the difficulty to foresee, and provide support, for the wide range of circumstances, settings, technologies and devices (interaction resources) required to build UIs capable of providing adequate user interaction in a diversity of AmI scenarios.

This work proposes to tackle this problem by increasing the level of decoupling between the UI control logic and the particular realization of the UI for a specific usage scenario (modalities, devices, user characteristics, environment characteristics, etc.). The goal is to improve the portability of AmI systems, so that they can be more easily adaptable to different contexts of use.

The portability and adaptability of user interfaces has been an important research topic for the UI community during the last years, specially due to the proliferation of multiple hardware platforms (PC, smartphones, tablets, smart TVs, etc.) and the necessity of providing UIs adapted to people with physical or cognitive disabilities. The result of these works is the concept of plastic user interfaces [Thevenin and Coutaz, 1999].

The term plasticity of UIs refers to user interfaces that can be adapted,

either automatically or manually, to changes in the context, while preserving the utility and usability of a system at acceptable levels. Plastic UIs help in reducing the development cost of applications because they allow the reutilization of the same application and UI code with different devices and in a variety of environments.

Plastic UIs try to go beyond the code independence of the platform, provided by virtual machines, and the abstraction from graphic toolkits, provided by IDLs (Interface Definition Languages). Plastic UIs are expected to provide independence from the context of use, allowing developers to modify and adapt the mode of interaction and even change the physical shape of the UI, preferably in an autonomous way, based on the interaction resources available and the state and characteristics of the user and environment where the interaction takes place.

As we will see in this chapter, much work have been dedicated to advancing in the addition of plasticity capabilities to traditional user interfaces, like graphical and voice-based UIs. Nevertheless, we think that AmI user interfaces present a series of characteristics that make them quite different from traditional interfaces.

First, AmI UIs are usually built as distributed physical user interfaces (DPUIs) [Harrison et al., 1998, Xie et al., 2008, Kranz et al., 2010, Pavan Dadlani, 2011] using many heterogeneous devices, each one of them specifically selected to match the preferences and characteristics of one type of user or environment. Therefore, compared to traditional UIs, AmI UIs are exposed to a much broader range of technologies, APIs and interaction modalities.

Second, AmI systems are expected to be physically integrated in the environment and use the environment itself to interact with its users. Furthermore, they are expected to be ubiquitous, so they must provide their functionality in multiple physical locations. Because of that, the multiple devices that build an AmI UI tend to be physically scattered throughout one or more physical environments.

And third, due to the natural interaction constraints and the use of a combination of multiple interaction modalities, an AmI UI can be dramatically affected by context changes. A new user, a change of location, or a change in the environment characteristics can even render an UI unusable. For example, in a new location, the devices and modalities available can be completely different from the previous one. A new user may need different modalities (imagine the conflicting requirements of a deaf user compared to a blind user). And even a change in environment characteristics, like lighting or movement, can affect the functionality of the UI.

As can be seen, plasticity can be harder to support in Ambient Intelligence UIs, but, nevertheless, it is a key requirement in order to achieve ubiquitous AmI systems. In this chapter we are going to explore the most relevant works related to the introduction of plasticity characteristics in Ambient Intelligence user interfaces and, by extension, in Distributed Physical User Interfaces.

In this discussion of the related work, we are going to address very different topics because, as introduced in chapter 1, Ambient Intelligence is a multidisciplinary field, and AmI systems require the integration of multiple and varied technologies in order to build a complete and functional system.

In section 3.2, we are going to address the issue of developing physical user interfaces using a multitude of different physical devices as interaction resources (IRs), putting special focus on those technologies that allow the distribution of the UI to different remote devices. This kind of solutions commonly provide applications with protocols and APIs to manage and control physical devices. We will start by reviewing home automation solutions, which have been one of the first available technologies to control physical devices and are the foundation of many AmI and UC systems. Then, we will explore specific solutions developed for AmI projects, and finally we will look at existing frameworks for the development of physical user interfaces.

In section 3.3, we are going to address topics related to the addition of plasticity characteristics to Distributed Physical User Interfaces. Thus, we are going to explore technologies that facilitate the building of AmI and UC natural user interfaces capable of operating ubiquitously in a HIE with a variety of scenarios. The main idea behind these solutions is to provide developers with capabilities to adapt the application interaction to the changing scenarios where a user may interact with a system. In this section, we are going to explore the most prominent existing approaches to achieve UI adaptation to context, starting from the concept of UI plasticity and model-driven engineering approaches to AmI and UC focused approaches.

3.2 Physical User Interfaces

As we have seen in the introduction section, the intrinsic nature and requirements of Ambient Intelligence systems make them rely on natural user interfaces (NUIs) to interact with the users [Fishkin et al., 1999, Sitdhisanguan et al., 2012, Zuckerman and Gal-Oz, 2013]. A common way to build NUIs for AmI systems is by implementing customized distributed physical user interfaces (DPUIs) [Kranz et al., 2010, Pavan Dadlani, 2011]. Physical User Interfaces take advantage of the properties and capacities of physical objects to bridge the gap

between the users and the state of a digital system [Harrison et al., 1998, Xie et al., 2008, Antle et al., 2009]. Those objects range from everyday objects integrated in the environment to specific setups like dedicated appliances or even cockpits. These devices come from different manufacturers, use heterogeneous technologies, diverse modalities, and in some scenarios, even use custom hardware especially built for a particular system. Furthermore, if the system is going to be deployed in a variety of scenarios, with different users or environments, many different configurations of the UI may be required.

Due to their dependence on PUIs and the wide variability of device technologies associated to them, one key aspect of any AmI system is its ability to support heterogeneous distributed devices that use different technologies. Their UIs must be capable of changing their shape at run-time, relying on different devices and modalities depending on the usage scenario in order to provide natural interaction in all the scenarios of an HIE.

In this section, we are going to explore the most prominent solutions available to build distributed physical user interfaces that make use of physical devices as interaction resources (of any kind and modality) to interact with the user. As we will see, there exist many different technologies that can be used to access and control physical devices for user interaction purposes. From custom designed devices with ad-hoc technologies, up to PUI development frameworks, and even a lot of device access technologies like Home Automation technologies, uPnP or IoT technologies that, even though they have not been conceived to build PUIs, they can be, and are being, used for that purpose.

3.2.1 Home Automation and Internet of the Things

Home automation and instrumentation systems are an important aspect of many AmI systems. They allow AmI systems to transform typical objects found in homes and buildings into connected devices that can be controlled and accessed by the AmI applications. These technologies constitute a good foundation for the development of PUIs that are embedded in the environment and naturally perceived by the users.

Their components are installed in the homes during the construction of the building as a substitute or complement to traditional systems like light switches, blind open/close systems, security surveillance systems, etc. Home automation technologies often make use of dedicated and proprietary communication buses, requiring the use of proprietary devices that support them. This is the case of technologies like EIB/KNX [KNX-Association, 2010] or Lonworks [Lonmark-International, 2010]. Others, like X10 [X10-Europe, 2010], use the power lines of the home to connect devices to each other, but they also require special

supported devices.

Home automation technologies share a common pitfall. They rely on proprietary buses and protocols, requiring device manufactures to pay expensive licenses in order to sell compatible products. Furthermore, they are not interoperable at all, thus you are limited to the devices available for each technology.

An interesting approach to increase home device interoperability is to develop a home automation control gateway [Miori et al., 2006, Bonino and Corno, 2008]. This is a device that is connected to home automation buses and is in charge of providing homogeneous access to those technologies and, in some cases, some extent of high level intelligent behavior. A prominent example of these approaches is the Domotic OSGi Gateway (DOG) [Bonino et al., 2008]. They are developing a software Home Gateway that can be installed on an embedded PC and contains plug-ins to support existing home automation technologies (currently KNX and Bticino OpenWebNet [Bticino, 2015]). This PC can be connected to a home network and provide a homogeneous API to access the available home automation devices.

Related to home automation, a large number of new smart home technologies have appeared linked to the concept of the Internet of Things recently. It seems like these technologies are starting to reach the general public, and, associated to smartphones and tablets, many different companies are starting to offer new solutions. The majority of these technologies are little more than remotely controlled devices that allow users to open/close the garage door from their smartphones, or monitor different physical properties of their home, like power consumption, temperature, etc. Some examples are TWINE [Supermechanical, 2014], SmartThings [Physical-Graph-Corporation, 2014], CubeSensors [CBSR, 2014], WeMo [Belkin-International-Inc., 2014] or Philips HUE [Philips-Electronics-N.V., 2015]. These technologies are quite similar in purpose to home automation technologies, but they rely on standard network hardware like Ethernet or WiFi. However, each technology provides its own proprietary access APIs, thus hindering the development of applications that make use of devices from different manufacturers and technologies.

A recent approach to increase the interoperability of these solutions is the Apple HomeKit project [Apple-Inc., 2015]. It is a framework for iOS devices that allows the utilization, with the same API, of different distributed devices from a variety of manufacturers. HomeKit uses devices as accessories of iOS devices, allowing their control and the configuration of automated actions between them.

Home automation and IoT are important technologies for AmI and UC systems because they allow the interaction through environment embedded devices, but, even with technologies like HomeKit, they are characterized by interoperability problems, with many different hardware and software solutions

competing for the market in an incompatible way. These incompatibilities make it difficult for developers to build systems that mix devices from different technologies or even manufacturers, which in turn hinders the capability of AmI applications and UIs to rely on the best and most natural devices for each scenario.

3.2.2 Ambient Intelligence and Ubiquitous Computing Frameworks

As we have stated before, Ambient Intelligence [Cook et al., 2009] and Ubiquitous Computing are research areas that, by nature, have to address the device interoperability problem, thus they are a good place to look for solutions to decouple AmI UIs from the technologies and modalities of the devices used to build the UI.

AmI applications must use and manage a wide range of devices to do their job, and because of this, AmI and UC related projects like AMIGO, PERSONA or AmI-Space have dedicated efforts to the development of solutions in the space of physical device interoperability.

The AMIGO project [Janse et al., 2005] uses standardized solutions as much as possible, developing abstractions to hide the heterogeneity of the devices and translate their functionality to AMIGO compatible services. AMIGO proposes UPnP [UPnP-Forum, 2011] as a standard for hardware access so, when a non-UPnP compatible device needs to be used, it is necessary to develop proxy services that use ad-hoc drivers to interact with the device and translate its interface.

The AmI-Space project [Rui et al., 2007, Rui et al., 2009] proposes a similar solution to AMIGO, integrating technologies through the use of encapsulating proxies. The main difference is that they also provide a hardware solution to simplify the physical integration of technologies, as well as the development of new devices.

The PERSONA project [Lazaro-Ramos, 2010] uses SAIL [Girolami et al., 2008], a sensing abstraction layer designed to provide access to wireless sensor networks (WSN). SAIL uses a layered design in which a first layer contains the logic to directly interact with the different WSNs supported, a second layer that abstracts the functionalities of the WSNs as OSGi services, and a third layer that exports those services with interfaces that are compatible with external software, like UPnP or in the case of PERSONA, an event based system.

As can be seen, the typical approach found in the majority of AmI and UC projects is to establish one specific technology (for example UPnP) as the

Figure 3.1: Various examples of Phidgets physical devices.

default option, and then rely on ad-hoc proxies to integrate devices from other technologies when necessary. This way, their applications are able to use devices from heterogeneous technologies, but at the cost of having to build specific proxies for each new unsupported device required.

3.2.3 Physical User Interface Frameworks

Even though the previous technologies presented in this section can be used to build Physical User Interfaces, they have not been designed with that purpose in mind. Because of this, and in order to foster the development of PUIs, the research community has introduced several PUI development frameworks that are worth pointing out.

Phidgets [Greenberg et al., 2001] is one of the first and most interesting PUI development frameworks. Phidgets include a series of prebuilt physical devices that resemble typical graphical widgets, and a hardware development platform to build new physical widgets. On top of that, an API allows developers to access the diverse Phidgets functionality. Phidgets are connected to a computer via USB, so they don't support distributed UIs.

VoodooIO [Villar et al., 2006] follows a similar approach to Phidgets, but it is especially focused on gaming UIs. VoodooIO provides a kit of components (the VoodooIO Gaming Kit, VGK) that allows video-game players to build their own video game controllers using a set of atomic components and changing their layout. This custom controllers are then connected to a computer using USB and the standard driver APIs for keyboard, mouse or joysticks. With VoodooIO, players are able to build customized physical cockpits for each game,

Figure 3.2: Examples of VoodooIO devices.

thus enhancing the experience of playing games like flight or driving simulators.

iStuff [Ballagas et al., 2003, Borchers et al., 2002], which is part of the iROS and iRoom projects [Johanson et al., 1999], is a framework for the development and prototyping of post-desktop ubiquitous computing user interfaces. iStuff components are wireless physical devices that are connected to a machine running a proxy software that encapsulates the device behavior and connects it to the iROS operating environment. iStuff uses the iROS infrastructure to allow distributed access to the component proxies and provides a virtual "patch-panel" that facilitates the mapping between devices and applications, thus making it quite easy for developers to try different configurations of a PUI.

EIToolkit [Holleis, 2007, Kranz et al., 2010] follows an approach very similar to iStuff. It also connects applications to devices by using a software platform that relies on proxies to encapsulate the protocols used by each device.

Even though those frameworks are closer to UI technologies, they continue to be more designed to interconnect devices than to abstract them, thus, they continue to operate on top of device/appliance concepts like functionalities and not on top user interaction concepts, so they do not support things like UI design, migration or adaptation. Furthermore, even if all devices can be accessed using the same network protocols, each one has is own API. This makes the integration of new devices difficult because applications must be modified to use the new API. Furthermore, it complicates the development of solutions that automatically manage the connection between devices and applications,

Figure 3.3: Examples of iStuff devices.

with only iStuff providing support to avoid the modification of the application code by using a virtual patch-panel component that manually manages the mapping between applications and devices.

3.2.4 Summary

In order to build natural user interfaces for Ambient Intelligence or Ubiquitous Computing systems, it is not possible, or at least, not reasonable, to rely on a single device technology. UIs for this kind of systems require the use of many different interaction resources, and each one of them may use different modalities and technologies specifically selected to match the preferences and characteristics of a type of user or environment. While interaction resources using a particular modality or technology, or even from one particular manufacturer, may be the best option for one location, IRs with other modalities, or from other manufacturers and APIs, may be better for other location or user. This is why the main AmI and UC projects like AMIGO[Janse et al., 2005], PERSONA [Lazaro-Ramos, 2010], iRoom [Johanson et al., 1999], EIToolkit [Kranz et al., 2010], AmI-Space [Rui et al., 2007] or HomeKit [Apple-Inc., 2015] have relied, to some extent, on technology abstraction solutions that allows them to use devices and appliances from different manufacturers using different technologies.

A common drawback of these solutions is that they are more device interconnection technologies than UI development technologies, thus their APIs and protocols are built on top of device concepts and not interaction resource ones. Furthermore they do not usually provide adequate abstraction capabilities, thus hindering the integration of new devices without requiring modifications to the application code.

From this review, we conclude that there exists a need to increase the level of isolation between application developers and the devices or appliances

that made up a distributed physical UI. A common and homogeneous user interaction API to access any kind of interaction resource, using any available technology and modality, will enable developers to build their DPUIs independently of the IRs used, allowing them to change the devices without affecting the application. Furthermore, it would allow the development of autonomous or semi-autonomous solutions to manage the change of IRs at runtime.

3.3 Plasticity in Physical User Interfaces

As stated in the introduction chapter, AmI UIs commonly rely on a mix of traditional (graphical) UIs and a set of heterogeneous interaction resources (IRs) [Harrison et al., 1998, Zuckerman and Gal-Oz, 2013], which can range from physical devices embedded in the environment to custom designed devices, tangible UIs, or even voice or gesture recognition systems. As a result of this, unlike traditional software systems where, if the user moves to a new location, the interaction resources hardly change, in AmI systems, a change of context may imply a dramatic change in the IRs available (the hardware platform) to the UI. Furthermore, not only the hardware platform could be different, but the new characteristics of the environment or the users can render the previous IRs inadequate.

This characteristic of AmI systems directly impacts the needs of their mobility subsystem, so that, in the case of a change in the scenario, the adaptation of the UI to the new context is essential to keep the system within the margins of usability and transparency required by natural interaction systems. Therefore, this is one of the most important topics that must be addressed by this thesis.

In this section, we provide a review of the field of UI plasticity and adaptation to context, placing special emphasis on ubiquitous computing systems, AmI systems, and distributed natural user interfaces.

As will be seen in this review, the Model-Driven Engineering (MDE) methodology and model-based techniques are the foundation of the majority of the existing approaches to build plastic UIs. This is not a coincidence, but a consequence of the characteristics of model-based techniques. With MDE, the knowledge about the system, the users, and the environment is stored in machine readable models, allowing the system to exploit them, even at runtime, to modify and adapt its behavior according to the information provided by the models and the current context for each scene.

3.3.1 Model-Driven Engineering to Achieve UI Plasticity

Thevenin and Coutaz were the first to introduce the term plasticity of user interfaces in 1999. In [Thevenin and Coutaz, 1999], they proposed a conceptual framework to support the development of plastic UIs. It is a theoretical framework that establishes a set of guidelines for the development of techniques and tools that facilitate the implementation of plastic user interfaces.

This framework is one of the first approaches to apply MDE to UI development. It proposes the specification of the UI as a set of models that provide abstract and declarative descriptions of the interaction capabilities of the UI and the physical environment where the UI will be executed. To exploit those models, Thevenin and Coutaz propose the development of a series of tools, either automatic or semiautomatic, that transform those models through a series of decreasing levels of abstraction until the implementation is achieved. Finally, once the UI is executing, they identify a process of plastic adaptation and transformation that controls the execution and adaptation of the UI, either manually or automatically.

Thevenin and Coutaz continued improving their conceptual proposal and, in [Calvary et al., 2001b], they introduced a new model in the framework, the evolution model, that would allow developers to model possible context changes that could affect an application, thus facilitating and guiding the adaptation to them. In [Calvary et al., 2001a], in addition to supporting the vertical transformation of models, that is to say, move from abstract models to new models at lower levels of abstraction, they introduced support for horizontal transformation, allowing the translation of models to new models in the same level of abstraction but using different technologies or languages. In [Calvary et al., 2002], they introduced a revision of the framework that allows starting the development of the UI from any of the different levels of abstraction. In this revision, they also introduced support for context changes in real-time with the introduction of elements to monitor the context, detect changes in it, and react to those changes.

The works of Thevenin and Coutaz led to the creation of the Unifying Reference Framework [Thevenin et al., 2003, Calvary et al., 2003] inside de Cameleon project. This framework is designed as a reference design and guideline for the development of plastic UIs using MDE techniques.

The Cameleon framework defines six initial models to describe the context of execution of a UI and four additional models to specify the UI at different levels of abstraction. Those additional models are inferred after the initial models, either manually or automatically.

The initial models are:

- *Concept model.* Describes the different concepts managed by the end user through the UI.
- *Task model.* Specifies the tasks that the user can perform with the application.
- *Platform and environment model.* Specifies the context of use of the UI, including the execution platform and physical environment.
- *Evolution model.* Describes possible context changes and the conditions to change from one to another.
- *Interaction model.* Describes the interaction resources available to implement the UI.

Starting from those models, Cameleon proposes the inference of a new series of models, named transitory and final, where the UI is specified at different levels of abstraction. The transitory models are abstract specifications (at different levels of abstraction) of the UI, while the final model is a concrete implementation (it can be executed) of the UI.

The framework establishes four different levels of abstraction for the specification of the interfaces:

- *Task-oriented specification.* The UI is specified at the highest level of abstraction. This model only includes information about the tasks, and their related concepts, that the UI must perform.
- *Abstract interface.* This is a description of the UI in terms of its interaction capacities. The description is done at a high level, using generic interaction resources that do not have an associated modality or a execution platform.
- *Concrete interface.* It is a transformation of the abstract interface where the interaction modalities of each interaction resources are already specified.
- *Execution model.* This is the final implementation of the UI. This specification is already adapted to a context and uses the real interaction capabilities available in a platform and environment.

The main idea behind the Cameleon framework is to infer these models using the information available in the initial models and through reifications (transformations), from one level of abstraction to a more concrete one, until the final implementation of the UI is achieved.

This process of inference and transformation can be done manually by the developer and user (adaptable plasticity), autonomously by the system itself

(adaptive plasticity), or semiautonomously, with some transformations done automatically by the system and others manually by the user or developer (mixed plasticity).

Finally, in [Balme et al., 2004], with the creation of CAMELEON-RT, the Cameleon conceptual architecture was extended to support multi-modal, mobile and distributed UIs within the philosophy of ubiquitous computing.

In parallel to the development of the Unifying Reference Framework, Sendín and Lorés introduced a new development framework for plastic UIs that is very related to the proposal of Thevenin and Coutaz [Sendín et al., 2003, Sendín and Lorés, 2004]. It is also model-based and it manages the plasticity of the UI with vertical transformations of models through different levels of abstraction. The main innovation introduced by Sendín and Lorés was the division of applications into two levels. The base level, which includes the business behavior of the application and operates without knowledge of the UI, and the meta level, where the UI is transformed from an abstract view to a concrete one according to the current context state.

In [Veloso and Sendín, 2005, Sendín, 2007], Sendín and Lorés extended the concept of plasticity of UIs by dividing it into two different facets that are very similar to the division proposed by CAMELEON-RT [Balme et al., 2004]. On the one hand, explicit plasticity, that is very similar to the original concept proposed by Thevenin and Coutaz in [Thevenin and Coutaz, 1999], that is to say, the capacity of adapting a generic UI to context changes (previously identified during the design phase) that require a new specific UI, following a semi-automatic or automatic process that is activated manually. On the other hand, implicit plasticity is the capacity of the UI to adapt itself, in real-time and without human intervention, to small context changes previously identified during the design phase.

It is important to mention that those initial frameworks, the Unifying Reference Framework, CAMELEON-RT or the Sendín and Lorés framework, are reference architecture designs. They are conceptual proposals and not real frameworks that could be used to build plastic user interfaces. Therefore, after those initial works, many research groups started implementing complete frameworks based on the concepts established by the Unifying Reference Framework and Cameleon.

For example, the Transformation Environment for interactive Systems representations [Berti et al., 2004, Mori et al., 2004] (TERESA) is a model-based framework, and an associated tool and language, for the design and development of user interfaces for different platforms. With some roots in the Unifying Reference framework and Cameleon approach, TERESA is one the first implementations of a model-based transformation method. The development of UIs

starts at the abstract level with a single model specifying the tasks and different contexts supported by the application. This description is achieved using the ConcurTaskTrees notation [Paternò et al., 2000]. The second step is to transform this abstract task model into a series of specific task models for each target platform. Next, an abstract UI model for each platform is built, and, in the last step, the final UI is generated from the abstract UI. This final UI is completely platform dependent. Every model in TERESA is described using XML, and the tool is able to generate UIs using XHTML and VoiceXML.

As a continuation of the TERESA project, the Model-based lAnguage foR Interactive Applications (MARIA) [Paternò et al., 2009] introduces MARIA XML, a new set of languages for UI definition that inherit the approach of TERESA XML, with one language for the abstract description and a variety of platform-independent languages depending on the deployment platform. In comparison with TERESA, MARIA introduces new models, like the data model, and new supported platforms like digital TV, multitouch or gestures.

Another prominent approach inspired by CAMELEON is the USer Interface eXtensible Markup Language (USIXML) [Vanderdonckt et al., 2004]. It is also a model-based framework centered around an user interface definition language (UIDL), the USIXML set of languages, which support the development of UIs using a multi-directional development method that allows developers to start the development from any, and even multiple, levels of abstraction, and proceed by transforming those models towards obtaining one or many final UIs.

The multi-directional UI development framework proposed by USIXML is based on the Cameleon Reference Framework [Thevenin et al., 2003, Calvary et al., 2003, Calvary et al., 2007]. Therefore, it supports the four levels of abstraction of Cameleon and provides language support for the specification of models in each level. USIXML defines five different models: Task and concept models for the task specification level, abstract user interface (AUI) model for the abstract interaction level, concrete user interface (CUI) model for the concrete interaction level, and final user interface (FUI) model for the execution level. The framework revolves around three different kinds of transformations among models: reification, which is the process of selection of artifacts that are more concrete than the artifacts used as input to the transformation process; abstraction, which is the inverse process of reification, hence, the selection of artifacts that are more abstract than the input; and translation, which is the process of transforming a model from one context of use to a different one without changing the abstraction level. In order to support these transformation steps, USIXML introduces mappings to model and keep track of the relations between the models in different contexts or levels of abstraction.

In addition to the conceptual framework and language, USIXML provides

a set of tools that allow developers to take advantage of the framework features. ReversiXML [Bouillon et al., 2004] and VAQUITA [Bouillon et al., 2002] automatically reverse engineer XHTML UIs into a CUI model. GrafiXML [Michotte and Vanderdonckt, 2008] is an editor to allow the graphical development of CUI models. ScketchiXML [Coyette et al., 2006, Coyette and Vanderdonckt, 2010] allows the sketching of CUI models. TransformiXML [Stanciulescu et al., 2005] supports the programming of transformations between models.

Frameworks like TERESA, MARIA or UsiXML represent the most prominent approaches to UI generation using MDE approaches. They are general purpose frameworks based on their own UI definition languages and populated by many different tools for a variety of specific purposes like web UI generation, graphical UIs generation for different platforms, or voice UI generation. Furthermore, as we will see in the next section, they are also a prominent source of inspiration for many of the approaches used by AmI and UC systems regarding UI adaptation to context.

3.3.2 UI Plasticity in Ambient Intelligence and Ubiquitous Computing Frameworks

As stated in the introduction of this section, the intrinsic characteristics of AmI and UC systems require them to provide some kind of support for the development of UIs adapted to context. Because of that, there exist diverse examples of frameworks that introduced some level of support for UI plasticity in the context of AmI and UC systems.

ICrafter [Ponnekanti et al., 2001] is one of the first approaches to tackle the problem of UI development in ubiquitous computing environments. It is part of the iRoom project [Borchers et al., 2002] at Stanford university and it is focused on supporting the selection, generation, adaptation, and distribution of UIs for the different appliances and applications available in an ubiquitous computing environment. ICrafter uses a centralized component that distributes interfaces to end devices (PDAs, PCs, etc.). These interfaces are either graphical UIs or VoiceXML interfaces, and they can be either manually designed or, in some cases, automatically generated by specific generator components. Furthermore, the main characteristic of ICrafter is the aggregation of functionalities from different distributed appliances or applications into one user interface. Regarding UI adaptation, ICrafter supports adaptation to the appliance and end-user device characteristics through the automatic generation of UIs, but it does not support adaptation to the user or environment characteristics.

Another interesting proposal is the one in the Personal Universal Controller (PUC) [Nichols, 2006]. It shares some similarities with ICrafter in their goal

and final functionality, but PUC is specially focused in the generation of consistent user interfaces for remote appliances. PUC allows developers to specify the functionalities and properties of an appliance using the PUC description language and, then, automatically generate consistent UIs for different computing platforms like smartphones, desktop computers or even speech recognition UIs. Among its similarities to ICrafter, PUC also shares some of its weaknesses. While PUC is able to generate adapted UIs to the appliance capabilities and the end-user device, it ignores the characteristics of the user and the physical environment.

Model-based techniques have been also used to achieve UI plasticity in the context of ubiquitous computing. Two prominent examples are Dynamo-AID [Clerckx et al., 2004, Clerckx et al., 2006] and MASP [Blumendorf et al., 2008a, Blumendorf et al., 2008b, Blumendorf and Albayrak, 2009, Roscher et al., 2009, Blumendorf et al., 2010].

Dynamo-AID and its predecessor project, Dygimes [Vandervelpen and Coninx, 2004], started as model-based UI development frameworks for multiple devices. The core idea behind them is to rely on task models to drive the generation and adaptation of the UI to different services and devices. The task model is enhanced with abstract information about the UI and linked to context information through dynamic dialog and environment models. Then, the final UIs are generated for each device. In [Clerckx et al., 2008], Dynamo-AID was improved with better support of Ambient Intelligence and Ubiquitous Computing systems by enhancing the framework with support for UI distribution and multimodality. Dynamo-AID keeps tasks as the main concept of the framework, so the distribution and multimodal support is also related to tasks, supporting the distribution of tasks and the utilization of different modalities for each task. The UI distribution is managed by a component called "distribution controller", which selects, depending on the context, what device to use for each task. Then, it sends the task UI to the device, which is in charge of rendering it. The final UIs are generated from the task and dialog models using the UIML language [Abrams et al., 1999]. Regarding multimodality, they enhanced the task model with modality constraint information, thus letting the developer specify which modalities are better for each task.

Another prominent model-based approach is the Multi-Access Service Platform (MASP) presented by Blumendorf et al. in [Blumendorf et al., 2008a, Blumendorf et al., 2008b, Blumendorf and Albayrak, 2009, Roscher et al., 2009, Blumendorf et al., 2010]. It follows the ideas of the Cameleon framework and USIXML and defines the UI by employing a set of models at different levels of abstraction. The main contribution of MASP is that the vertical transformation of models is done at run-time, with the models evolving in memory during

the execution of the application. MASP models are called executable-models because the models are interpreted at run-time and linked with the execution state of the system. In MASP, models not only provide design information of the UI during run-time, but they also reflect the changes of the state of the system and how the UI keeps itself adapted to them. The capability to operate and reason with the models at runtime, together with the ability to enrich those models with real-time information of the context, bring the possibility of building new algorithms that exploit that information and capabilities to adapt the UI to context-changes in real-time.

As a MDE framework, MASP starts from the definition of UI with three high level models. Task model, context model, and abstract UI model. By using the task and context model information, MASP transforms, at run-time, the components of the abstract model into concrete components that use three different modalities: WIMP web applications (windows, icons, menus, pointer), voice recognition, and gesture recognition. Furthermore, the different concrete components can be rendered in a variety of distributed devices like PCs, smartphones, or tablets.

As can be seen, all of these projects, ICrafter, PUC, Dynamo-AID, MASP, and even others like [Bandelloni and Paternò, 2004] or [Berti, 2005], follow a very similar approach. They generate graphical or voice UIs for different end devices, like smartphones or tablets, starting from some kind of models describing the UI or the capabilities of remote appliances. Those projects make possible the generation of UIs adapted to each end device used for interaction, but none of them allows the adaptation to the user abilities and preferences, and, furthermore, they ignore the problem of physical user interfaces, because they rely on remotely rendered GUIs to interact with the ubiquitous system.

3.3.3 UI Adaptation to the User Characteristics

The different solutions presented in the previous sections are mainly focused on UI adaptation to end devices. They are able to generate a UI adapted (even, in some cases, at run-time) to the characteristics of the end devices used to interact with the system. In this sense, they are able, for example, to generate different UIs for a smartphone and for a desktop. However, they do not support adaptation to other characteristics of the scenario, like the physical characteristics of the environment or, more importantly, the characteristics or preferences of the user, which, for a natural interaction system, is of great importance.

In this section, we want to highlight two prominent approaches to UI adaptation to the user abilities. One of them, Egoki [Abascal et al., 2008], is a UI

framework for UC and AmI systems, while the other, SUPPLE [Gajos and Weld, 2008], is a very featured system for the generation of GUIs adapted to users with motor or psychic disabilities.

The Egoki framework [Abascal et al., 2008, Abascal et al., 2011b, Miñón and Abascal, 2012] is designed to support the adaptation of HTML web UIs for their use by different users with physical or mental disabilities. The objective of the framework is to allow the adaptation of Ambient Assisted living (AAL) UIs to the different interaction abilities and preferences of a variety of users and the devices (smartphones, tablets, PCs, etc.) they use to access the AAL system. Egoki is model-based, but follows a different approach to Cameleon. It uses only two models. On the one hand, an abstract UI model described using an UIDL like UsiXML or UIML. On the other hand, a user profile model describing the capacity of a user to interact with the system using each of the supported interaction modalities. Furthermore, during the description of the UI, developers must select the priority of each modality for each part of the UI. Finally, Egoki manages a rule database where the supported modalities are related to the interaction resources available. Using the information available in the models and the rule database, Egoki selects those IRs that better match the requirements of the user and builds a new web UI using XHTML.

SUPPLE [Gajos et al., 2004, Gajos et al., 2008a, Gajos et al., 2008b, Gajos et al., 2010] is focused on the generation of GUIs adapted to users with motor or psychic disabilities. The most prominent characteristic of SUPPLE is the fact that its UI generation system is not rule-based and does not use model transformation algorithms. Instead, it uses numerical algorithms in order to optimize the UI layout and widget selection. Some of the main factors used by the cost function of the SUPPLE optimization algorithm are: cost of navigation between any two controls, cost of using a particular control for each function, consistency between interfaces generated on different platforms, or the physical abilities of the user. As a non rule-based system, SUPPLE has the ability to explore the whole solution space in order to generate UIs, making SUPPLE more flexible but also more liable to require exponentially increasing processing power in order to build complex interfaces. Furthermore, it is difficult for a UI designer to predict and intervene in the final result of the UI, as it is not possible to provide the system with human-readable inputs as in the case of model-based and rule-based systems. While SUPPLE is very focused on adaptation to the user abilities, it also has capacities to generate UIs for different end platforms (mainly PCs and PDAs) in a similar way to ICrafter or Egoki.

While SUPPLE is one of the most complete solutions for the generation of user adapted UIs, it is focused on standalone graphical user interfaces, thus ignoring many of the requirements of AmI systems, like UI distribution and

support for physical UIs. Regarding Egoki, it follows a very similar approach to other AmI and UC solutions like ICrafter or MASP but extended with support for adaptation to user abilities. Nevertheless, as ICrafter or MAPS, it ignores the use of physical UIs.

3.3.4 Summary

MDE techniques have been adopted by the HCI research and development community as the most promising approach to achieve UI plasticity. As a result of this, during the last two decades a lot of systems have relied on model-based approaches for the generation of UIs adapted to context. There is an abundance of examples of MDE frameworks for UI generation and adaptation to context. Some prominent examples are COUSIN [Hayes et al., 1985], ITS [Wiecha et al., 1990], UIDE [Sukaviriya et al., 1993], HUMANOID [Luo et al., 1993], GENIUS [Janssen et al., 1993], TRIDENT [Bodart et al., 1995], UIML [Abrams et al., 1999], the Unifying Reference Framework [Thevenin and Coutaz, 1999, Thevenin et al., 2003] and CAMELEON-RT [Balme et al., 2004], XWeb [Olsen et al., 2000], ICrafter [Ponnekanti et al., 2001], TERESA [Berti et al., 2004] and MARIA [Paternò et al., 2009], USIXML [Vanderdonckt et al., 2004], DynaMo-AID [Clerckx et al., 2004], PUC [Nichols, 2006] and UNIFORM [Nichols et al., 2006], MASP [Blumendorf et al., 2008a, Blumendorf and Albayrak, 2009], or Egoki [Abascal et al., 2008]. In this review, we have explored the most important ones, with special emphasis on those approaches more relevant for this thesis and more related to its field of application.

The review shows how model-based techniques have been successfully applied to ubiquitous computing (UbiComp) and ambient intelligence (AmI) with approaches like ICrafter [Ponnekanti et al., 2001], PUC [Nichols, 2006], UNIFORM [Nichols et al., 2006], Dynamo-AID [Clerckx et al., 2006], MASP [Blumendorf et al., 2008a, Blumendorf and Albayrak, 2009], USIXML [Vanderdonckt et al., 2008], Egoki [Abascal et al., 2008], or MARIA [Paternò et al., 2009]. Nevertheless, from our point of view, all of these approaches share a common drawback. They have ignored the utilization of physical user interfaces (PUIs) to build natural user interfaces (NUIs) in those environments. While these UIs provide remote access to the services and appliances of an AmI or UC environment, and they can be multimodal (mainly WIMP, voice recognition or gesture recognition) [Blumendorf et al., 2008a, Vanderdonckt et al., 2008] or even distributed [Blumendorf et al., 2010], they use desktops, tablets or smartphones as hardware platforms for their interaction resources (IRs). They do not allow direct manipulation of physical devices and appliances as a means to provide natural and environment-integrated interaction with the system.

From this review, we have concluded that there is a necessity to advance in the development of solutions focused on improving the support for building distributed physical user interfaces capable of adapting themselves to the characteristics of an application, their users, and the environment where it is executed. The solution proposed in this PhD. thesis is built on top of model-based techniques inspired by approaches like USIXML [Vanderdonckt et al., 2004], CAMELEON-RT [Vanderdonckt et al., 2004], MASP [Blumendorf et al., 2008a] or PUC [Nichols, 2006]. But compared to these previous works, it will make possible the development of AmI UIs capable of using any kind of IR, based on any hardware platform, and independently of its physical location in the environment.

3.4 Discussion

If something can be extracted from the literature review presented in the previous sections, is that the field of AmI user interface development is quite broad and still open. While there are solutions proposed to facilitate the development of AmI UIs, the fact is that the majority of AmI UIs are being developed ad-hoc for each particular system and usage scenario, because, even though many of the solutions proposed shine in some particular aspects, none of them seems to cover all of the aspects required to build plastic Distributed User Interfaces for Ambient Intelligence systems.

The purpose of this subsection is twofold. A first goal is to provide a fair and direct comparison between all the solutions presented in this review in order to know how they stand against the others and, more importantly, in order to know how they stand against the ideal requirements of a solution for the development of plastic DPUIs. A secondary objective is to establish a comparison framework for solutions in the field of Plastic Distributed Physical User Interfaces, thus establishing a set of characteristics required by this kind of solutions, making possible the comparison of past and future developments in the field.

From the review of the state of the art presented in this chapter and the further study of Ambient Intelligence UIs presented in chapter 4, we have designed a comparison framework that includes ten different characteristics organized into three aspects:

- *Autonomous plasticity of the UI.* This aspect is about the capacity of a solution to adapt a UI to different contexts of use.
  - *Adaptation to the user.* Capacity to adapt the UI to different user characteristics.

  - *Adaptation to the environment.* Capacity to adapt the UI to a variety of environment characteristics.
  - *Adaptation to the devices.* Capacity to adapt the UI to different sets of interaction resources.

- *User Interface Development.* This aspect covers desirable characteristics of a UI development framework for any field of application.

  - *Distribution of the user interface.* The degree of distribution of the UI supported by a solution. This is, to what granularity can the different elements that make up an UI be distributed.
  - *UI-centric development framework.* Whether a solution provides an specific framework for the development of UIs based on user interaction concepts, or not.
  - *UI abstraction.* The degree of decoupling between the UI developers/code and the particular technologies, modalities, and APIs used to implement the user interface.
  - *Autonomous generation of the user interface.* Whether a solution supports the autonomous generation of user interfaces adapted to a particular usage scenario.
  - *Capability to customize the generated user interface.* The degree of personalization of the final user interface autonomously generated by a solution. This characteristic is only applicable to those solutions that support UI generation.

- *Physical User Interfaces.* This aspect includes important characteristics for the development of physical user interfaces.

  - *Number of different modalities supported.* The number of different types of interaction modalities supported by the solution.
  - *Number of different physical devices supported.* The number of different types of physical devices (used as interaction resources) supported by the solution.

With the knowledge obtained from the literature review presented here, we have subjected to our comparison framework each one of the solutions analyzed in this chapter. The results of this comparison are shown in table 3.1.

As can be seen in the table, these results are a clear reflection of the variety of solutions that exist in the field, as well as the diversity of degrees to which those solutions fulfill the different characteristics proposed in the comparison framework for Plastic Physical User Interfaces. It can be seen that there are

	Autonomous Plasticity			User Interface					Physical UI	
	User	Env.	Devices	Distribution	Dev.	Abstraction	Generation	Custom.	Modalities	Devices
MASP	No	No	Yes	Medium	Yes	High	Yes	Low	Low	None
Dynamo-AID	No	No	Yes	Medium	Yes	High	Yes	Low	Low	None
Egoki	Yes	No	Yes	None	Yes	High	Yes	Low	Low	None
SUPPLE	Yes	No	Yes	None	Yes	Low	Yes	Medium	Very Low	None
ICrafter	No	No	Yes	Low	No	Medium	Yes	Medium	Low	None
PUC	No	No	Yes	Low	No	Medium	Yes	Medium	Low	None
MARIA	Yes*	Yes*	Yes*	None	Yes	Medium	No	Medium	Low	None
UsiXML	Yes*	Yes*	Yes*	Low	Yes	Medium	No	Medium	Low	None
H. Automation	No	No	No	Low	No	Low	No	N/A	Very Low	Low
HomeKit	No	No	No	Low	No	Low	No	N/A	Low	Medium
Phidgets	No	No	No	None	No	Low	No	N/A	Medium	Low
VoodooIO	No	No	No	None	Yes	Low	No	Low	Low	Low
iStuff	No	No	No	Medium	Yes	Medium	No	Medium	High	Medium
EIToolkit	No	No	No	Medium	No	Medium	No	N/A	High	Medium

*Plasticity is supported by transformation processes usally performed manually by the developers

Table 3.1: Comparison of solutions for the development of Plastic Distributed Physical User Interfaces.

solutions that perform really well in one of the aspects, but very poorly in the others, and vice versa.

For example, there is a group of solutions that perform quite well in the Physical UI aspect, but they have really poor performance in Autonomous Plasticity. This is the case of technologies like HomeKit, Phidgets, EIToolkit or iStuff. While they can be used to implemented PUIs, these technologies are mainly designed to provide remote access to physical devices, thus they have good qualities regarding physical device compatibility and support for very different modalities, but they do not have any kind of support for UI abstraction, generation, or adaptation to different contexts. In fact, many of these solutions are not even UI development frameworks, but device access and control frameworks that are being used to build custom and ad-hoc user interfaces.

Looking at the table it is also easy to see another big group of solutions. The first eight technologies displayed in the table have in common a very poor support for physical user interfaces. They support very few modalities (usually graphical and voice), and they do not include any kind of support for using physical devices as interaction resources. These are technologies that come from the user interface field, thus in general they have good values in the User Interface development characteristics, like UI development framework, UI customization, or UI decoupling from implementation. However, it is possible to divide this large group into a set of smaller groups by looking at their UI distribution and Plasticity values.

Regarding distribution, on the one hand, there are some solutions that do not support any kind of UI distribution at all, like Egoki or SUPPLE. They are able to generate adapted interfaces, but these interfaces must be executed in a single device. On the other hand, there are technologies like MASP or Dynamo-AID that allow the distributed execution of some parts of the UI to different devices, and other solutions, like PUC or ICrafter, that only allow the remote distribution of the complete UI.

Regarding plasticity, all of the eight first solutions include some kind of support for UI plasticity or adaptation to context, but there are important differences between them.

UsiXML and MARIA are very big MDE frameworks designed to support the complete development lifecycle of a User Interface. They operate by transforming UI models from the abstract UI to a final UI adapted to the context, thus they are designed to support UI plasticity. Nevertheless, in practice, this transformation is performed manually or, in the best case, semi-automatically by the UI designer.

There is another small group of technologies designed to support device (or hardware platform) adaptation. These technologies are able to generate UIs customized for the particular characteristics of the end device used by the users. For example, PUC and ICrafter are able to automatically generate graphical or voice UIs adapted to the functionalities offered by remote appliances and the characteristics of the end user device (PC, smartphone, etc.).

Apart from UsiXML and MARIA, also Egoki and SUPPLE support the adaptation to the user, in this case by automatically generating the UI. SUPPLE is a very complete solution for the generation of UIs adapted to people with disabilities, nevertheless, it has very poor performance in almost all of the other categories relevant for AmI UIs, like distribution or multimodality. Egoki is designed to generate UIs adapted to users with disabilities in ubiquitous computing environments, thus it supports device adaptation too, however, it has poor support for UI distribution and Physical User Interfaces.

A result, which can be extracted from this review and the presented comparison framework, is a set of three critical aspects that a development framework for Plastic Distributed Physical User Interfaces for Ambient Intelligence or Ubiquitous Computing systems must deal with:

- Autonomous UI plasticity seems to be an essential feature for AmI and UC user interfaces. Ubiquity requires those systems to operate in different physical environments and scenarios, thus forcing the UI to provide the system's functionalities in very different contexts of use, where fundamentally different modalities and interaction resources should be used to

provide an adequate user experience.

- Physical UIs are characterized by their heterogeneity; they use different modalities, implemented with a variety of physical and digital elements, even relying on custom designed devices for particular scenarios. Therefore, another key characteristic is the ability to support heterogenous modalities and devices. A framework for DPUIs must provide a high level of decoupling between developers and the specific technologies and modalities of the interaction resources. Furthermore, it is desirable to have integrated support to use multiple different devices from different manufactures, technologies, APIs, etc.

- UI distribution is also another one of the key aspects of AmI and UC user interfaces. Those systems are eminently distributed, and so it should be their user interfaces.

In this review we have shown that, while many solutions perform well or even excel in some of those aspects, there is no solution presenting a good performance in every one of them:

- On the one hand, there exist development frameworks for distributed physical user interfaces, but they are more remote device access frameworks than UI development frameworks. They allow applications to use different distributed input/output devices, but those frameworks are too low level because their APIs are too device centric and they do not allow developers to work on top of UI concepts instead of device access concepts. Furthermore, they lack any kind of support for UI plasticity and adaptation to context.

- On the other hand, many of the model-based approaches that we talked about in section 3.3, have been extended to support multimodal and distributed user interfaces in an AmI or UC context. Nevertheless, these frameworks have been mainly focused on supporting the generation and distribution of GUI user interfaces using web technologies, with only minor support for additional modalities like voice recognition and gesture recognition. They are mainly designed to generate graphical UIs adapted to different end devices, like tablets, smartphones, PCs, or web interfaces. Because of that, those frameworks are not well suited for the development of physical user interfaces, where distributed physical and multimodal interaction resources are used to interact with users.

As will be seen throughout this document, the three critical aspects of AmI/UC user interface development previously introduced are the main build-

ing blocks of the UI abstraction framework for Distributed Physical User Interfaces prosed by this thesis. This framework, called the Threefold Interaction Abstraction Framework (TIAF), and its reference implementation, the Dandelion framework, provide a complete and functional development framework for AmI/UC user interfaces, with integrated support for heterogeneous interaction resources, distributed physical UIs, and autonomous UI plasticity.

The next chapter is devoted to providing a further analysis of the requirements and necessities of Ambient Intelligence user interfaces, and to introduce the characteristics and architecture of the Threefold Interaction Abstraction Framework. Finally, the Dandelion framework, is thoroughly presented in chapters 5 and 6.

Chapter 4

Analysis and Design of a Framework for Ambient Intelligence UI Development

"The best scientist is open to experience and begins with romance - the idea that anything is possible."

Ray Bradbury

4.1 Introduction

In the introduction provided in chapter 1, we have seen that the intrinsic characteristics of Ambient Intelligence and Ubiquitous Computing systems made their User Interfaces (UIs) quite different from the UIs of classical software systems.

First of all, AmI systems are expected to operate in a proactive and intelligent manner, providing their functionalities while staying out of the way of the users and requiring minimal interaction with them.

Second, this interaction is expected to happen in a natural way, namely, in a way adapted to the characteristics of the situation, the user preferences and abilities, the environment characteristics, and the interactive devices available. Because of that, in AmI, WIMP (Windows, Icons, Menus, Pointer) user interfaces are the exception and Natural User Interfaces (NUIs) implemented with

Physical User Interfaces (PUIs) are the norm.

And last, but not least, AmI systems are expected to be ubiquitous or, at least, to seem ubiquitous and provide their functionalities in any of the places included in the HIE of their users.

If those three characteristics are combined, we can see how different AmI UIs are compared to classical GUIs. AmI UIs must not only be proactive and transparent, but they must also be adapted to the context, and, if that were not enough, they must be able to operate in different places, with different characteristics, disparate devices and multiple users.

The objective of this doctoral thesis is to study and design techniques to facilitate the development of AmI UIs capable of complying with those three expected characteristics. To achieve that objective, we propose the utilization of model-driven engineering techniques together with the introduction of a new development framework for AmI physical UIs. This framework introduces three abstraction levels to isolate developers, and AmI application code, from many of the specific decisions and technologies required to build AmI UIs that comply with the three previously identified characteristics.

This chapter starts by providing an overview and examples of some AmI UIs in order to extract their characteristics and justify the necessity of developing new solutions to build AmI UIs. The next section continues by conceptually describing the new abstraction levels proposed by this PhD. thesis. Finally, the chapter ends by introducing a conceptual architecture and development framework that supports the proposed abstraction levels.

4.2 Analysis of the Characteristics of UIs in Ambient Intelligence Systems

Throughout the previous chapters, we have been reiterating that, because of the intrinsic and expected characteristic of AmI systems, their UIs must be quite different to classical software system UIs. However, we have not examined any example of an AmI system user interface yet. In this section, we are going to explore three different examples of real AmI systems, so that we can clearly establish the characteristics of these kind of UIs and the problems associated to them. These examples will be reused throughout the next sections and chapters to show how the different solutions contributed by this work are applied in order to improve them and to make their implementation easier and cheaper.

4.2.1 OMNI Virtual Assistant

Our research group has participated in the development of a home-care assistant system for elderly people. This project is named OMNI Virtual Assistant and its objective is to improve the self-sufficiency of elderly people in their daily life at home. The system is designed to be embedded in the user's home and provide them with three core functionalities: telepresence service through video-calls with family, friends, or tele-care centers; remote health monitoring; and daily routine management.

The system is executed in a set-top-box shaped device connected to a TV screen. The different functionalities of the system are naturally integrated in the TV workflow as new channels. One channel per family member, friend or care giver, and one channel per doctor or health monitoring application.

In order to keep the interaction simple, the system is designed to operate in a proactive way, asking the user questions when it is necessary to perform actions. This way, users interact with the system exclusively by changing channels and by answering questions using a 'Yes' or a 'No'. Figure 4.1 shows three examples of this kind of Yes/No proactive interaction between the OMNI system and the user. In the first screenshot, the user has selected the tele-care channel, so the system asks the user if she wants to call the tele-care center. In the second screenshot, the system is reminding the user that she has to perform some action, in this case, taking medication, and it asks the user to confirm whether she has taken the medication or not. And finally, in the third screenshot, the user has received a call from a friend, and the system is asking if she wants to answer the call.

As can be seen, the interaction subsystem is deliberately simple to facilitate its use by elderly people with very different levels of knowledge and experience in information technology. The TV screen is the primary output device, and a remote controller, shown in Figure 4.2, with only five buttons, is the primary input device.

While this primary interaction setup is enough for many users, elderly people represents a very heterogeneous group of people with very different abilities. There may be people with visual disabilities for whom the TV screen will be a poor output channel. There may be people with motor or psychic disabilities and problems to use the remote controller, etc. In order to accommodate this high diversity of abilities, OMNI requires a multimodal user interface. It must support different combinations of input and output devices, so that each user can use the device that better fits her requirements.

Furthermore, OMNI is required to provide many of its functionalities in different places of the home. While the video-call will be only possible in front of

Figure 4.1: Examples of Yes/No interaction with the OMNI system.

Figure 4.2: Picture of the OMNI remote controller.

the TV screen, the health and daily routine monitoring functionalities should be available in different places of the home, or even at remote places like the homes of relatives. To access those functionalities OMNI will also need a distributed user interface.

In order to better illustrate the need for multimodal distributed UIs in OMNI, lets take for example a small subsystem of OMNI and explore its UI a little bit.

For example, the OMNI notification subsystem is in charge of notifying events to the user. In OMNI, events are things like incoming calls, medication intakes, schedule sport exercises, calendar events like a visit to the doctor, or alarm events like fire alarms.

The notification subsystem requires a user interface to show the notifications to the users. In a classical PC system, with a graphical user interface, it would be a simple notification dialog window with a label for the notification message and one or two buttons to receive the response from the user. However, in OMNI, this UI can have very different shapes depending on the user:

- The default option would be to use the TV as the dialog window with a message, like in Figure 4.1, and rely on the remote controller to perform the role of the button in the classical system.

- But, for a user with finger mobility problems, instead of the remote controller, OMNI can use a gesture recognizer system or even a speech recognition system as input devices.

- For a blind user, instead of using the TV, OMNI can use a speech synthesizer to perform the output of the notification message.
- If the user is not in the living room and it can't see the TV screen, OMNI can attract the attention of deaf users by using simple devices like colored lights representing different events with colors.
- If the user is not at home, the events could be notified to different portable devices. From a smartphone with a graphical UI to read the notifications, to simple devices with vibration and color codes.

These are only some of the possible scenarios that can be quickly imagined, but there can be many more if we want OMNI to support more specific disabilities for which even custom designed devices must be used.

As can be seen, even a simple user interface like the one of the OMNI notification subsystem can become complex to implement when we want to support the characteristics of AmI systems.

4.2.2 Environmental Music Player

The Environmental Music Player (EMP) is a music player application with ubiquitous capabilities, it plays music and follows the user when she changes from one place to another inside her HIE. As a music player, it works in a similar way to radio stations in Spotify or Groveeshark; it does not allow the user to select specific songs or albums, it only allows her to select a music style or mood, and the EMP chooses, among the songs available in the user collection, artists and songs related to the selected style or mood.

Even though the final functionality is simple, the EMP user interface is fairly more complex that the OMNI user interface. If we left ubiquitous and natural interaction apart, the UI of this application for a WIMP system will probably look similar to the sketch shown in Figure 4.3. As can be seen from the sketch, the EMP application has to provide the user with many different interaction capabilities in order to control the playback:

- As inputs:
  - Music style or mood selection.
  - Music volume control.
  - Playback controls like play, pause, next, etc.
  - Customization control to notify the system whether the user has liked a song or not.

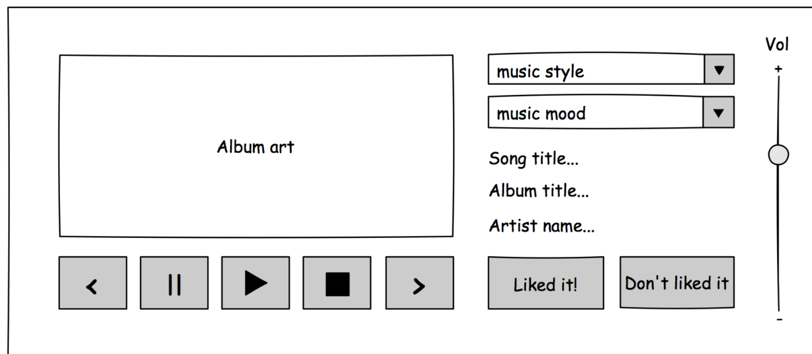


Figure 4.3: Sketched UI of the Environmental Music Player using WIMP user interfaces.

- As outputs:
  - The album art image.
  - The album and song titles, and the artist name.

In order to operate in an ubiquitous way and provide a natural user interface adapted to the environment, the EMP UI should be adapted to the particular characteristics of each physical place included in the HIE of its users. The interaction with the system will not be the same in the living room, in the car, in the garden, or even in the kitchen.

Lets take, for example, three possible scenarios and explore how the UI could be implemented in each one of them:

- The user is in the living room listening to music:
  - The EMP should use the TV screen as output channel for the album art and the metadata of the song like song tittle, album tittle, or artist name.
  - A remote control device can be used for the input of commands (play, pause, next, liked it, etc.) and selection of music style and mood.
- The user is cooking in the kitchen while listening to music:
  - The album art will not be shown as there won't be a screen, but an speech synthesizing output channel can be used for the song metadata.
  - The user will have her hands occupied while cooking, so the input of commands should be done for example with a gesture recognizing input channel.

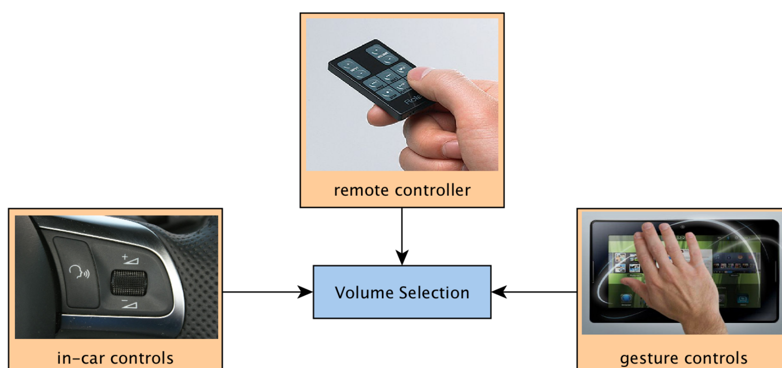


Figure 4.4: Example of various interaction possibilities for the volume selection in the Environmental Music Player application.

- The user moves to her car and starts driving while the music continues playing in the car:
  - The song metadata, and even the album art if the screen allows it, could be presented in one of the screens of the car dashboard.
  - The playback control should be done using the specific music controls integrated in the car steering wheel and dashboard.

Again, like in the case of OMNI, depending on the usage scenario of the EMP application, any of the input/output capabilities can be implemented using very different modalities and devices, as shown in the example of Figure 4.4. Furthermore, as the user moves from one place to another, in order to continue operating within the constraints of adaptation to context required by Natural User Interfaces, the UI should be modified and adapted to the characteristics of each scenario.

4.2.3 Intelligent Ship Passenger Evacuation System

Another Ambient Intelligence project where our research group has participated is an intelligent evacuation management system for passenger ships such as cruises or ferries. The objective of the system is dual. On the one hand, it has to monitor the status of the ship in order to detect emergency situations (fire, flooding, etc.) and notify this situation to the ship crew. On the other hand, once the crew orders the evacuation of the ship, the system is in charge of providing guidance to passengers to go to their nearest evacuation point.

Apart from the multiple management and crew notification user interfaces,

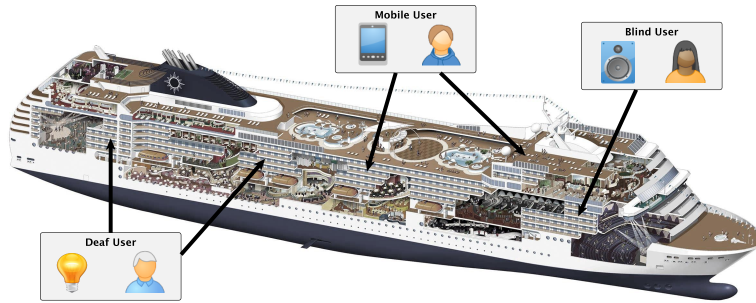


Figure 4.5: Example view of a possible EvacUI running different UIs, for a variety of users, in different locations of a ship, during an evacuation process.

which are implemented as classical WIMP UIs, this evacuation system requires a user interface to interact with passengers during the evacuation process. The objective of the Evacuation UI (EvacUI) is to provide passengers with accurate guidance of the path to follow in order to leave the ship. The conditions of the ship can change dramatically during the evolution of an emergency (limited visibility due to smoke, noise, flooded corridors , etc.), and also the users characteristics can be diverse as the passengers can have very different disabilities. Because of that, the system must operate and interact with the users in different situations with different constraints, thus requiring multiple interaction modalities operating complementarily and redundantly.

EvacUI is implemented as a combination of multiple distributed signaling devices embedded in the ship. These devices, screens, sound systems, light signals, etc., are used complementarily and redundantly in order to show directions and messages to the passengers. Therefore, depending on the context (environment conditions and user characteristics) of each place and user, the system should use, at the same time, different combinations of output devices to perform one specific action. For example:

- If the visibility is low due to smoke or a lighting failure, the system can use sound signals to guide the passengers.
- If a passenger is deaf, the system can use visual signals in her cabin to notify her of an emergency.
- A portable device, like for example the smartphone of the passenger, can be used to provide personalized directions to each user.

As can be seen, this UI is a very good example of a distributed, physical and multimodal UI. The system, which operates ubiquitously throughout the ship

(the HIE in this case), should use a large set of physical distributed devices to interact with its users, the passengers. Furthermore, depending on the context of each place included in the HIE, the devices used to perform the interaction must change, either because the environment conditions, the user characteristics, or both require it.

4.3 Supporting Ambient Intelligence UI Development

After the presentation of three representative examples of AmI applications and UIs in the pervious section, this sections is dedicated to establishing the requirements and characteristics of AmI UIs, and it continues by presenting a conceptual framework to support the development of AmI UIs capable of complying with those requirements.

4.3.1 Requirements of AmI UIs

In the previous subsections we have explored three different examples of AmI applications providing very different functionalities but with one important thing in common. They all must support a large variety of constraints in their interaction with the users.

The OMNI Virtual Assistant is designed to be used at home, so it doesn't have to support many different environments, but its users are elderly people and disabled people with very diverse abilities and requirements.

In the case of the Environmental Music Player (EMP), even though it could accommodate many different types of users, we are more interested in its natural integration in the environment. Therefore, in order to operate in an ubiquitous way, the EMP user interface should be perceived as natural in a variety of physical environments with different constraints.

And last, but not least, regarding the EvacUI user interface, it must provide evacuation directions to different users, with diverse abilities, in a variety of environments with distinct physical constraints.

As can be seen, in their effort to provide an ubiquitous and natural user interaction experience, all of these examples share the same two critical aspects previously explained in the introduction of chapter 4.1:

- They need to support a variety of environments, users, and situation characteristics; while keeping the UI natural and easy to use.

- They need to support ubiquitous interaction by using physically distributed user interfaces.

These two different problems can be solved using the same naive technique: design and implement different UIs for each type of user/environment/situation. Developers can build different UIs using different interaction modalities in order to cope with the first problem, and use multiple APIs and networking protocols to cope with the second one. Nevertheless, as can be easily imagined, this solution does not scale well and, in many cases, requires a huge amount of development effort and resources. Furthermore, any scenario that has not been predicted at design time would be unsupported.

In this PhD. work we propose to address these problems by increasing the level of decoupling between system software and the interaction resources that build the UI. The core idea is to introduce a new conceptual framework for AmI UI development that is built on top of three levels of conceptual decoupling between the business logic, the UI control logic, and the interaction resources. This framework is called the Threefold Interaction Abstraction Framework (TIAF) and we describe it it detail in the following subsections.

4.3.2 The Threefold Interaction Abstraction Framework

In the previous subsection we have established two main requirements associated to AmI UIs: Support ubiquitous operation and support natural interaction in a variety of scenarios. Unfortunately, as we have seen in chapter 3, supporting those requirements is not an easy goal, and it currently requires a lot of effort by AmI developers. The objective of this PhD. thesis is to alleviate this problem and facilitate the work of AmI developers by reducing the costs of building AmI UIs that comply with those two big requirements.

With that purpose in mind, we have designed a conceptual framework: the Threefold Interaction Abstraction Framework (TIAF), which establishes a set of conceptual layers and components to support the development of AmI UIs, significantly reducing the effort required to build AmI UIs that comply with those two requirements.

The TIAF conceptual framework uses models to store and manage the information that will drive the adaptation of the UIs to the environment and, as its name suggest, it revolves around three different and complementary levels of decoupling between the UI control logic and the interaction resources that build up a final UI.

The first level is called Interaction Modality Abstraction (IMA). The objective of this abstraction layer is to logically decouple application code and

developer knowledge from the specific technologies of the interaction resources (hardware devices or software components) used to build the final UI. The IMA allows developers to design and implement their application business logic and UI control logic without any knowledge about the final UI underlying APIs or interaction modalities. As can be seen in figure 4.6, to build a specific UI using the IMA, it would only be required to select a set of adequate interaction resources (IRs) and connect them to the IMA without any specific coding.

The second level is the Interaction Location Abstraction (ILA). This layer is in charge of isolating application and UI control code from the physical location of the IRs that physically interact with the user. The ILA allows developers to implement their application code and UI control code without bearing in mind the different networking protocols required to connect the application to the diverse IRs used. Furthermore, the ILA also allows installers to physically deploy the UI distributed and embedded in the environment, without requiring any modification to the application or UI control code.

The third level is the Interaction Context Abstraction (ICA). The objective of the ICA is to isolate developers and installers from the selection of the specific IRs used to build each particular UI implementation. While the IMA and ILA make the implementation of different UIs for the diverse usage scenarios easier and cheaper by decoupling the code from the technologies and networking protocols of the IRs, the ICA solves the scalability problem or producing multiple UIs for each scenario. The ICA liberates developers and installers from the responsibility of manually selecting the set of concrete IRs to use in each situation. Furthermore, the ICA allows systems to adapt their UI to different scenarios at run-time, supporting even unpredicted ones.

The combination of these three layers, IMA + ILA + ICA, makes the development of systems capable of true ubiquitous interaction easier. They facilitate and reduce the costs of developing UIs capable of adapting themselves to changing scenarios, using multiple modalities to interact with the user and distributed devices to provide ubiquity support.

The remainder of this chapter is devoted to exploring these three abstraction layers in more detail.

4.3.2.1 Interaction Modality Abstraction

As the name suggests, the main objective of the Interaction Modality Abstraction layer (IMA) is to isolate the application developer's knowledge from the particularities of the specific modalities used to interact with the user. The idea is to allow application developers to design and implement the business logic and the UI control logic with low, or even no knowledge at all of the modalities,

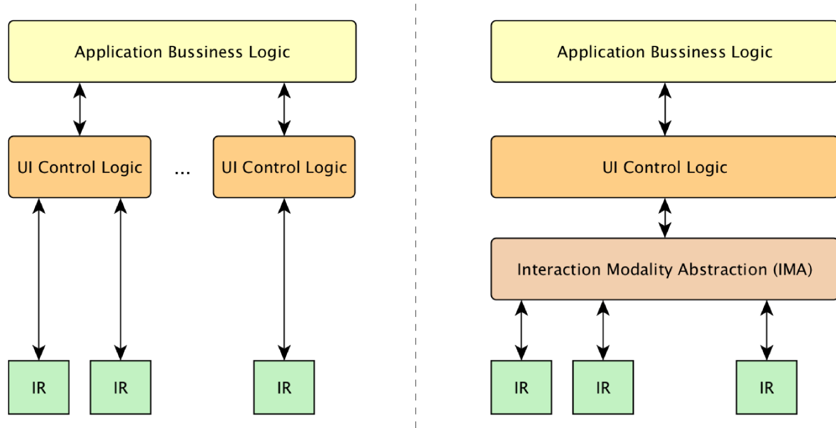


Figure 4.6: The IMA layer facilitates the implementation of different UIs by decoupling the UI control logic from the interaction resources. On the left, the application without using the IMA layer. On the right, the application when using the IMA layer.

technologies, and APIs required by the devices or software components used as interaction resources.

Figure 4.6 provides a good overview of how the IMA modifies the architecture of an application and the dependencies of their components. An AmI application without modality abstraction, but a good separation between business logic and UI control logic, would present an architecture similar to the one on the left of Figure 4.6. One implementation of the business logic would be coupled to multiple implementations of the UI control logic, one for each different interaction scenario supported by the application. On the contrary, as we can see on the right of Figure 4.6, with the IMA, only one implementation of the UI control logic is required. It is the IMA who manages the diversity of interaction scenarios and their specific interaction resources.

Figure 4.7 shows a detailed view of the Interaction Modality Abstraction layer modeled inside the TIAF conceptual framework. The IMA is the foundation of the TIAF framework. It provides the most basic and important abstraction feature, decoupling the UI control logic and the UI design from IRs specific modalities and APIs. From the developers point of view, it provides them with a common conceptual model to support the design and implementation of the UI control logic of AmI applications. This model, the Abstract Interaction Model, for which a more detailed description is provided in subsection 4.3.2.4, must provide conceptual abstractions for all the common interaction concepts, so that developers have enough freedom to support the majority of interaction scenarios. Developers design and build the UI control logic on top of those ab-

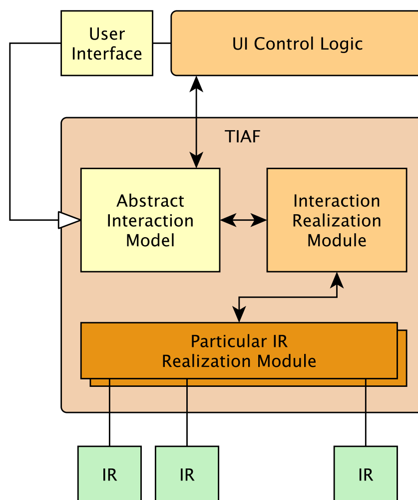


Figure 4.7: Detailed conceptual block diagram of the TIAF with support for the IMA layer. The User Interface is defined by developers using the concepts provided by the Abstract Interaction Model. Instances of those concepts are used in the UI Control Logic to implement the particular behavior of the UI and its interaction with the system's business logic.

stract concepts, instead of specific elements from concrete technologies. It is the responsibility of the IMA to provide translations between those abstracted interaction concepts and the specific APIs of the interaction resources used for each scenario. This translation is performed by the Interaction Realization Module (IRM), which relies on specific realization modules for each different IR technology utilized. This translation process can be either performed at run-time or at build-time, depending on the implementation of the framework.

To better illustrate the benefits of the IMA, in the remaining of this subsection, we are going to explore how the TIAF simplifies the development of an example application UI.

For the sake of simplicity, we have selected a small application example: the control module of a meeting room presentation system that is in charge of allowing the presenter to change from one slide to the next (or previous) one.

Lets suppose that an important requirement of this control module is that it can be used with different modalities and IRs depending on the usage scenario. Users must be able to change slides using one of various input methods, like a mouse, graphical buttons, gestures, a remote controller, or voice commands.

If we have to build it following a similar approach to the one shown on the left side of Figure 4.6, first of all, we will have to decide which IRs we want

to support. Each of them has a completely different API, and what is worse, each of them follows a completely different paradigm of interaction. In order to keep the business logic decoupled, we will be obliged to provide an abstraction of that interaction interface in the UI Control Logic module interface, and then we will have to implement one UI Control Logic module for each one of the selected devices. As can be seen, even this very small UI requires a lot of work in order to support interaction using different modalities and IRs.

Instead, if we build the UI using an implementation of the TIAF, we will be able to model the UI using a set of abstract interaction widgets with which our UI Control Logic can interact. This way, we are only required to provide one implementation of the UI Control Logic, because the TIAF, using the IMA, will be in charge of translating those abstract interactions into real interactions. As can be seen, the amount of work required has been vastly reduced, but that's not all, the use of a TIAF implementation introduces an important side-benefit. Application developers are liberated from becoming experts in a bunch of device's APIs, and furthermore, applications will be able to directly use any device that the TIAF support, thus even facilitating the adaptation of an UI to different and even unpredicted usage scenarios.

As indicated, the IMA effectively reduces the costs and complexity of AmI systems that require support for multiple and varied interaction scenarios. Developers can focus on the design and implementation of the business and interaction logic, and then, at deploy-time, or even at run-time, select the most suitable interaction resources (hardware or software) depending on the particular characteristics of the interaction scenario (physical environment, users, etc.) without affecting their system implementation. The next chapter, and in particular section 5.3, will provide a detailed description of a particular implementation of the TIAF and the IMA+ILA layers.

4.3.2.2 Interaction Location Abstraction

As explained in section 4.2, in order to support ubiquitous operation and interaction, a common situation in AmI systems is to require the utilization of a variety of devices that are physically distributed throughout the environment. In those situations, AmI systems and their UIs are required to access one or more networks of distributed devices, thus exposing developers to the knowledge of those protocols, and coupling system logic to the concrete APIs and protocols used by each IR.

As can be seen in the left part of Figure 4.8, without the ILA, the developers of AmI UIs, where distributed IR access is required, must know how to use the different networking protocols required by each IR, and modify the UI control

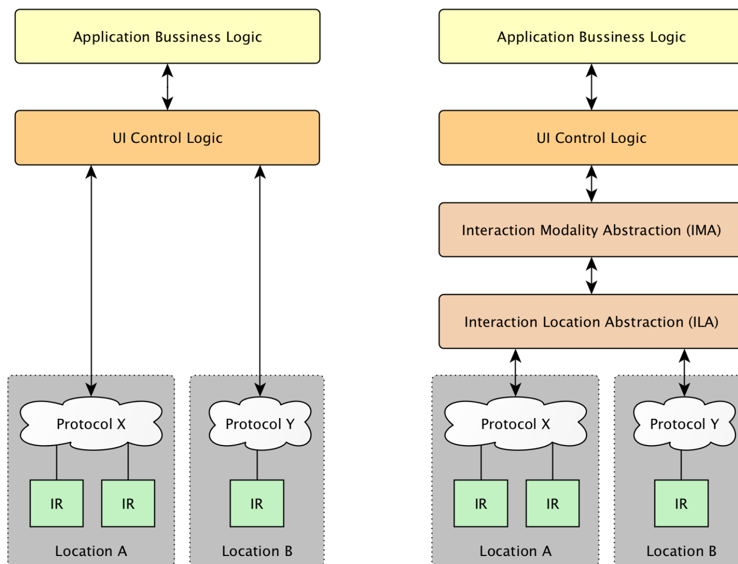


Figure 4.8: The ILA allows developers to use remote IRs without any knowledge about the required specific networking protocols. On the left, an application without the ILA has to access the IRs directly using their specific protocols. On the right, an application with the ILA is isolated from the knowledge of specific networking protocols.

logic accordingly for each IR access paradigm. However, as can be seen in the right side of Figure 4.8, if the ILA is used, the knowledge about the IRs communication protocols and APIs is transferred to the ILA, thus completely isolating application and UI developers from it.

In the TIAF framework, the ILA is conceptually introduced as an extension of the IMA. As can be seen in Figure 4.9, in order to support the ILA concepts, we have made the TIAF become a distributed framework where the specific logic required to drive each different IR technology can be physical distributed and separated from the interaction code.

We have decoupled and physically separated the Interaction Realization Module from the set of particular IR realization modules supported by the framework. This decoupling is achieved by the Generic Interaction Protocol, a distributed network protocol that must be implemented by each IR realization module. This protocol is designed to mimic the conceptual operations supported by the Abstract Interaction Model. This way, the Interaction Realization Module job is simplified, and now it will be operating like a router of interaction operations from the UI Control Logic to the specific IRs that are in charge of implementing particular interactions. In order for this to work, each IR Realization Module is in charge of providing a concrete implementation of the protocol interface for a specific device or technology.

With the introduction of the Generic Interaction Protocol, for which a more detailed description is provided in subsection 4.3.2.5, and the physical separation between the Interaction Realization Module and the IR Realization modules, the UI Control Logic is now not only decoupled from the specific technologies and APIs of each IR, but it is also decoupled from their physical location. This opens the possibility of building ubiquitous UIs that can be implemented using multiple IRs, with different modalities and deployed in different physical places.

Lets recall the example application introduced in section 4.3.2.1. Our slide controlling module must now support distributed interaction. Looking at Figure 4.8, in the first case, we will have to provide a new specific UI Control Logic implementation for each remote IR that we want to use, and we will also be in charge of establishing and managing the network connections between the application and the IRs. Again, the work is increased linearly with the number of IRs to support, and furthermore, we are required to master the networking protocols and/or remote APIs of each IR. However, by using the TIAF with IMA+ILA support, we will not be required to modify even one line of code of our application or UI Control Logic, because they interact only with abstract interaction components, which will be transparently connected by the TIAF to physically distributed IRs that will perform the required interactions.

Like in the case of the IMA, the use of an Interaction Location Abstraction

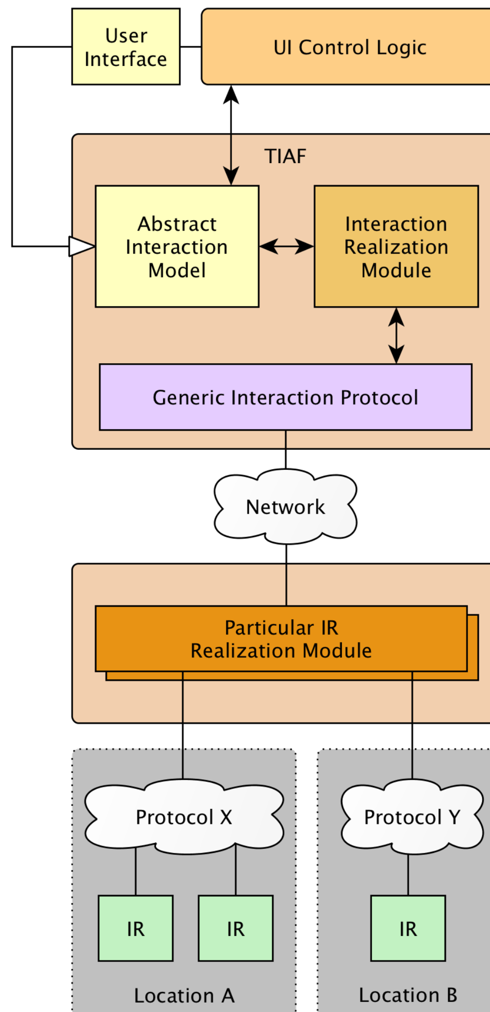


Figure 4.9: Detailed conceptual block diagram of the TIAF with support for the IMA+ILA layers.

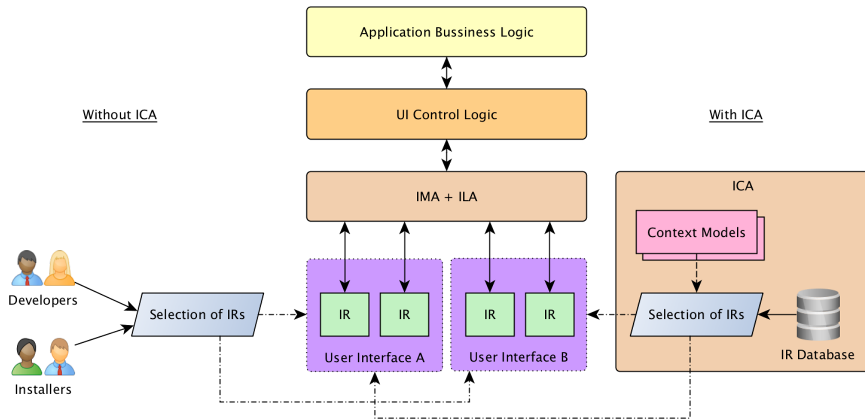


Figure 4.10: Without the ICA, developers and/or installers have the responsibility to select the adequate IRs for each scenario. With the ICA, it is the system itself, based on context models, who, autonomously, selects the most adequate IRs for each scenario.

layer allows reducing the development costs and complexity of AmI systems. First, it allows application developers to focus on AmI business logic, avoiding the need for networking expertise. Furthermore, it facilitates the use of different IRs that use diverse networking technologies or protocols, thus allowing developers and system installers to use the best IRs available without affecting the cost of supporting them.

Finally, the combination of an IMA and an ILA implementation allows developers to use any kind of interaction resources with little or no knowledge about the interaction modalities, APIs, operating systems, hardware architectures, and networking protocols used by the IRs.

4.3.2.3 Interaction Context Abstraction

While the IMA+ILA combo make the development of multiple versions of an AmI system's UI for different interaction scenarios easier and cheaper, the Interaction Context Abstraction layer (ICA) is designed to minimize the scalability problem of that solution, liberating developers and installers, to a great extent, from the need to design, implement, and deploy different versions of the UI for each scenario. The main idea behind the ICA layer is to let the system autonomously manage the selection of IRs that will be used for each particular scenario [Varela, 2013].

Figure 4.10 shows the difference between using the ICA or not. If the ICA is not used, developers must build the final UI for each scenario by selecting a set

of IRs adequate for the combination of physical environment, user and situation characteristics. However, if the ICA is used, the system itself is in charge of building the final UIs for each scenario. As can be seen, the ICA requires the utilization of the IMA+ILA, so that the system can easily accommodate any IR selected, and it uses the information provided by a set of context models (user, environment, situation, etc.) to perform the selection of the best IRs for each scenario.

Like in the case of the ILA, the ICA is integrated in the TIAF as an extension to the IMA conceptual architecture. As we have previously explained, the Interaction Realization Module is in charge of translating abstract interaction operations, defined by the Abstract Interaction Model, into real interactions with the user. With the IMA+ILA approach, this translation is reduced to a transformation of operations into messages of the Generic Interaction Protocol, and the routing of these messages to the adequate destination, the IRs that must perform the interaction. Without the ICA, the mapping between the abstract operations of the UI and the distributed IRs that perform them must be manually done by the developer or installer of the system. At deployment time, she selects, among the IRs available, which one will perform each interaction action. However, with the ICA, this selection will be done by the TIAF itself, either at deploy-time or even at run-time.

For that purpose, as can be seen in Figure 4.11, we have extended the framework with two new components. To drive the IR selection process, and therefore the UI adaptation to the context, the ICA uses contextual information that is exploited by a set of IR Selection Algorithms. This contextual information is stored in a variable set of context models that can include a user model, an environment model and a scene model. The user model provides information about the user or users, like their preferences, physical characteristics, or motor and mental abilities. The environment model contains information about the physical place, for example, constraints like noise, visibility, or space. And finally, the scene model provides information about the activity that is currently going on, like the number of users, the kind of activity, etc. A more detailed description of these models will be provided in subsection 4.3.2.6 and in chapter 6.

The IR Selection Algorithms employ computational intelligence methods to exploit the information available in the models to select, among a set of IRs, those that better fit the interaction requirements of the application, defined by the usage of the Abstract Interaction Model, and the needs of the user, environment, and interaction situation.

Lets recall again the small example presented in the previous subsections. The slide controlling module can be set up to use different IRs and modalities

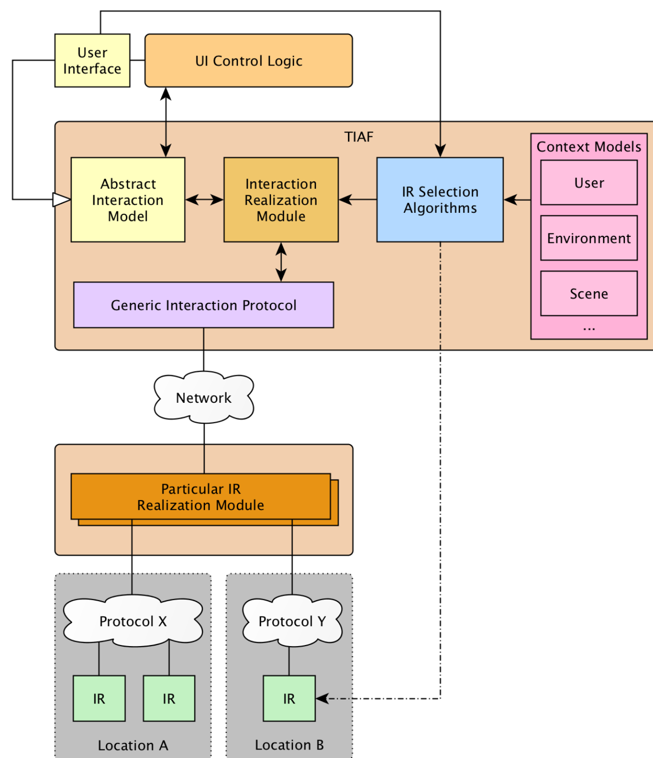


Figure 4.11: Detailed conceptual block diagram of the complete TIAF framework with support for the IMA+ILA+ICA layers.

depending on the context of use. For example, it can be set up to use a speech recognition system for a user with motor disabilities, or a gesture recognizer to operate the presentation with the hands. Nevertheless, this selection of which IR to use in each scenario must be done by the installer of the system during its setup. It is the installer the one in charge of selecting the most adequate IR (among those available) depending on the scenario characteristics (user, environment, and situation). If the scenario is changed, the installer is required to modify again the selection of the IRs in order to adapt the UI to the new characteristics.

If we want this UI, and therefore, the presentation system, to be a truly ambient intelligence system, it should be able not only to operate ubiquitously (thanks to the ILA), but it should also be able to adapt its UI to the characteristics of the context, so that the UI is kept within the margins of natural interaction requirements. This is the job of the ICA layer. It is in charge of selecting the most suitable IR for each interaction depending on the characteristics of the usage scenario, which are obtained from the models that describe the user, the environment, and the situation. Depending on the implementation of the TIAF and its ICA layer, this selection could be done either autonomously, at run-time, or semiautonomously, at deploy-time, with a tool performing the selection for different scenarios, and the installer configuring the system for each scenario.

As shown, by using the ICA it is not only possible to automate the building of multiple UIs adapted to different interaction scenarios, but it is also possible to support the dynamic change of scenarios, thus facilitating the development of ubiquitous interactive systems like AmI or UC applications. The ICA, in combination with a context monitoring system to detect context changes, can be used to build and adapt, even at run-time, the UI of an AmI application as the user moves from one physical location to another.

4.3.2.4 The Abstract Interaction Model

As we have briefly introduced in subsection 4.3.2.1, the Abstract Interaction Model is the main medium of interaction between developers, AmI UI code, and the TIAF framework. Its purpose is twofold. First, it must provide developers with a common model to describe the interaction requirements of their UIs. Second, it must provide a common set of operations to make use of the available Interaction Resources.

In order to accomplish the first goal, the Abstract Interaction Model should be able to represent very different Interaction Resources, modalities, and technologies behind a reduced set of generic concepts. But, in order to achieve the

second goal, this reduced set of concepts should provide developers with enough expressive power to describe the user interaction requirements of almost any UI.

As discussed in section 3.3, during the last years there has been a huge effort by the UI research community to apply Model-Driven Engineering (MDE) techniques to UI development. Among those developments, there have been many authors working on the design and development of models to describe UIs at different levels of abstraction, especially following the abstraction level division of Thevenin and Coutaz [Thevenin and Coutaz, 1999, Calvary et al., 2001a, Balme et al., 2004]. Three prominent approaches are the TERESA project [Berti et al., 2004, Mori et al., 2004], the MARIA project [Paternò et al., 2009] and the UsiXML project [Vanderdonckt et al., 2004]. Inspired by the ideas of the Cameleon conceptual framework, each of those projects have been building a complete model-based UI development framework. At the core of those frameworks, there are a set of models that allow developers to describe the different aspects of an UI at different levels of abstraction. To build UIs, those frameworks provide methods and tools to transform the models from one level of abstraction to a more concrete one.

With so many projects already proposing solutions to model UIs at different levels of abstraction, we decided that the best way to go would be to rely on one of those solutions for the definition of the Abstract Interaction Model of the TIAF.

By reviewing the different solutions available, as presented in section 3.3.1, we have seen that those three projects have more in common than differences. In fact, they all define almost the same abstraction levels and models, because they are inspired by the works of Thevenin and Coutaz [Thevenin and Coutaz, 1999] and by the CAMELEON framewok [Balme et al., 2004]. Regarding user interaction description, they have divided it into four different levels of abstraction:

- *Task model.* This is the highest level of abstraction. The UI is specified as the tasks, and their related concepts, that the user must perform.
- *Abstract interface model.* This is a description of the UI in terms of its interaction capacities. The description is done using generic interaction resources that do not have an associated modality or a execution platform.
- *Concrete interface model.* It is a transformation of the abstract interface where the interaction modalities of each interaction resources are already selected. Multiple CUIs may exists for each AIU, because it is modality dependent, and the same AIU can be build using different modalities.
- *Final interface model.* This is the final implementation of the UI. This

specification is already adapted to a context and uses the real interaction capabilities available in a platform and environment. Each CUI model can be also transformed into multiple FUIs, depending on the technologies and APIs used.

The most interesting models for our purposes are the Abstract interface models. In all of the cited frameworks, they serve the same purpose as in the TIAF; they provide developers with a generic set of concepts, independent of any technology or modality, to describe the interaction requirements of the UIs. With that in mind, we decided to directly use the Abstract User Interface Model of the UsiXML project as the Abstract Interaction Model of the TIAF. It has great expressive power, a lot of tools to create and manage models, and more importantly, this conceptual model has been proposed by the W3C for the definition of a standard Interface Definition Language (IDL) for abstract user interfaces.

Figure 4.3.2.1 shows a conceptual class diagram describing the Abstract Interaction Model directly inspired by the UsiXML AUI Model. The core concept of the AUI Model is the Abstract Interaction Unit (AIU). It is a generic representation of the typical widgets found in graphical user interface toolkits. Each AIU is associated to a set of different interaction facets (input, output, selection..) that represent the interaction capabilities required by the AIU. Furthermore, AIUs can be organized into hierarchies by defining AIUs that are composed of other AIUs.

While the AIU is the main organizing element of the model, the interaction facets, *InteractionSupport*, *EventSupport,* and *PresentationSupport,* are the elements that provide expressive power to it. By combining the three concepts available in the model, it is possible to describe an AIU that requires five different interaction actions:

- *Action.* The TriggerSupport indicates that the AIU requires some kind of support to trigger actions. Like a button in a GUI.
- *Input.* DataInputOutputSupport can be used to indicate that the AIU requires support to input, output, or both, some kind of data from the user.
- *Output.* The same as above.
- *Selection.* The DataSelectionSupport indicates the requirement to support a selection of an item between a collection of them.
- *Focus.* All the interaction support elements share a common interaction operation to request the attention of the user.

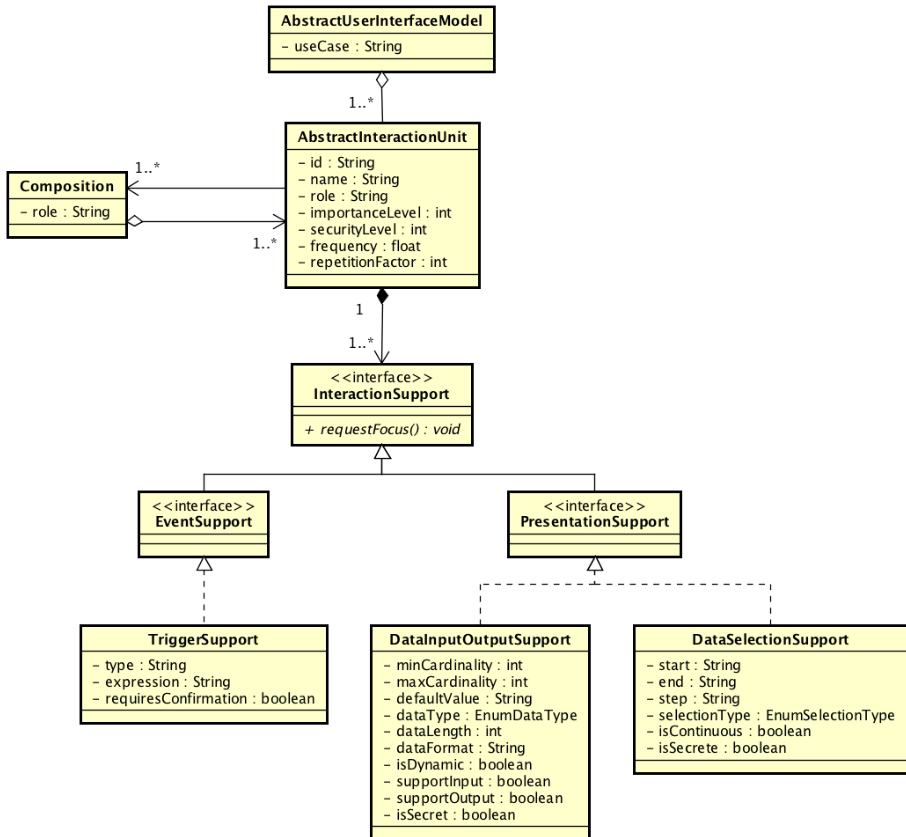


Figure 4.12: Class conceptual model of the Abstract Interaction Model directly inspired by the UsiXML Abstract User Interface Model.

In order to describe a complete user interface, developers only have to define a collection of interrelated AIUs, with each AIU describing a set of user interaction operations requirements.

As can be seen, the UsiXML AIU Model defines quite a reduced set of generic user interaction concepts, but it provides a great amount of expressive power, enough to describe even large and complex interfaces, with multiple interaction requirements [Vanderdonckt et al., 2004].

The Abstract Interaction Model is used by the TIAF not only as an IDL to describe the user interfaces of AmI systems, but also to provide an operative interface between the UI control logic and the interaction resources that finally realize the abstract interactions.

4.3.2.5 The Generic Interaction Protocol

The Generic Interaction Protocol (GIP) [Varela et al., 2013b] is conceptually conceived as a distributed communications protocol that creates a generic remote interface to any kind of interaction device. By implementing the GIP interface, any device or interaction resource can be accessed using the same set of concepts and operations, thus decoupling the application from the underlying interaction technologies and the location of the devices.

As can be seen in Figure 4.9, the GIP is right between the control logic and the interaction resources (physical devices, graphical widgets, voice recognition software, etc.), decoupling them in two ways. On the one hand, as the GIP is a distributed protocol, it provides physical decoupling between the system logic and the UI components. On the other hand, it provides logical decoupling by isolating interaction resources behind a common generic interface.

The GIP, in combination with the Interaction Realization Module, provides a way to directly map Abstract User Interface (AUI) elements to the Final User Interface (FUI) elements at runtime. Since the actual IRs available are already known at the time the mapping is defined (run-time or deploy-time), the GIP eliminates the need for the definition of different CUI and FUI models for each modality and technology, because the TIAF allows developers to build the UI, and its control logic, at the abstract level, and then connect that abstract logic, at run-time, directly to the final IRs available.

The common generic interface provided by the GIP is in charge of abstracting the behavior of concrete interaction resources. Thus, it should be generic enough to support multiple kinds of modalities and interaction devices, furthermore it must support all the expressive power of the Abstract Interaction Model. Therefore, the operations supported by the GIP are directly inspired by the Abstract Interaction Model and, because of that, by the UsiXML Abstract

User Interface Model [Vanderdonckt et al., 2004].

The GIP interface is designed to match the set of generic interaction facets described by the Abstract Interaction Model. Each IR implementing the GIP interfaces provides support for one or more interaction facets, thus facilitating the establishing of a mapping between an AIU interaction facet and an interaction resource that will perform that interaction.

Using the TIAF framework, each AIU described in the abstract UI model of an application will be associated to a collection of IRs, each one of them implementing one interaction facet. Therefore, a complete UI will be a collection of physically distributed IRs, organized in AIUs and accessed through a remote interface of generic interaction operations.

The GIP is designed as an event based distributed protocol following a hybrid publish/subscribe and one-to-one communications model. On one hand, a series of publishers, the interaction resources, publish events notifying actions that the user has performed (input/selection of data or activation of an action) and a series of subscribers (usually the system UI controller) receive those events and react accordingly. On the other hand, when the system logic has to send some event to an interaction resource, it does it by using one-to-one communications. This way, the interaction resources are more decoupled from the system logic.

In order to match the design of the Abstract Interaction Model, the GIP interface is made up of five different events: input, output, selection, action and focus. The first four events are directly inspired by the UsiXML AUI model:

- *input*. An interaction resource informs its subscribers that the user has performed a data input action.
- *output*. The system logic commands one or many interaction resources to output some data to the user.
- *selection*. This event has two different meanings depending on the sender. If it is sent by an interaction resource, it means that the user has made a selection. Otherwise, it means that the system logic requires an interaction resource to show a selection to the user.
- *action*. An interaction resource informs its subscribers that the user has triggered an action.
- *focus*. The system logic requires an interaction resource to gain focus over the user attention.

It is not mandatory for every interaction resource to support all GIP events. There can be many kinds of interaction resources with different levels of support

for user actions. Some will support input and output, while others will support only input or will not be able to reclaim the focus of the user.

Every GIP event has an associated set of data properties that indicate the data an interaction resource is able to either output to the user, or gather from it as input. These data items are represented as a string and have an associated basic type (integer, double, byte or string), so that the system can know what kind of information an interaction resource is able to represent.

In order to allow some level of customization of the user interface, GIP events are enhanced with a set of properties called Interaction Hints (IH). They are a set of fixed properties that developers can use to provide indications to the interaction resources about an interaction action (for example: priority, size or color). The support for IHs is not mandatory, and each interaction resource can interpret them as it wants.

Even if the GIP design has been inspired by the UsiXML AUI model, it is generic enough to be used in combination with other UI abstraction languages or technologies or even without previously specifying the UI at an abstract level.

4.3.2.6 The Context Models

The ICA layer described in subsection 4.3.2.3 requires the use of a set of context models in order to provide the system with a description of the context in which the UI is going to be operated. The TIAF is designed to accommodate a variable number of context models, so that different implementations of the TIAF can use a different number and type of models. It will be the nature and characteristics of the IR Selection Algorithms what will make the final selection of context models.

Nevertheless, by analyzing the use cases and user types of the majority of AmI systems, like the examples provided in section 4.2, and from our review of the literature in adaptive systems of section 3.3.1, it seems almost mandatory to include, at least, models that cover the following aspects of the context:

- *User profile information.* As previously described in section 4.3.2.3, the capabilities and characteristics of users are a key aspect for the development of natural user interfaces. Because of that, it seems important to have information about the user abilities and capacities during the adaptation process. As described by the MyUI work on Requirements for User Interface Adaptation [Edlin-White et al., 2010], this model may include information about the variability that affects a person's capacity to interact with a system, like vision ability, hearing ability, or motor and psychic capacities.

- *Environment profile information.* The environment plays an important role for ubiquitous natural user interfaces. The same UI will not be perceived as natural in different environments like a car or a sports stadium. Because of that, it also seems mandatory to provide the adaptation process with information about the status of the environment. Again, the MyUI work [Wolf et al., 2011] on models for User Interface Adaptation gives us a good starting point for modeling a personal environment. This model may include information like visibility, noise, environment movement, privacy, etc.

- *Scene or situation information.* Almost all MDE frameworks for UI development agree on the necessity of feeding the adaptation process with information about the task that the user is performing. In TIAF, we propose the use of a scene or situation model to provide information on what kind of task the user is performing and in what circumstances. It may include information like whether the task is about work or leisure, it is indoors or outdoors, etc.

When building a concrete implementation of the TIAF framework, developers should consider these models as a reference proposal. The TIAF does not impose any limitation on the number and type of models to use, thus letting developers use the models that better match the concrete field of application and the IR Selection Algorithms implemented. Chapter 6 explores a reference implementation of the ICA layer together with an implementation of the three conceptual models proposed in this section (see section 6.3).

Chapter 5

Supporting Portable and Distributed Physical User Interfaces

"If you wish to make an apple pie from scratch, you must first invent the universe."

Carl Sagan

5.1 Introduction

In the last chapter we have introduced the Threefold Interaction Abstraction Framework (TIAF), a conceptual framework designed to reduce the complexity and costs of developing applications that support natural interaction with ubiquitous operation. In this chapter, we are going to explore an implementation of the TIAF conceptual framework called Dandelion. Its main objective is to provide a reference implementation of the TIAF proposals and conceptual architecture with the aim of demonstrating the benefits of the solutions proposed.

The Dandelion Framework provides a complete implementation of the TIAF conceptual distributed architecture with support for the three proposed levels of decoupling. This chapter will be focused in the implementation and demonstration of the first two levels of decoupling, the Interaction Modality Abstraction (IMA) and the Interaction Location Abstraction (ILA), leaving the Interaction

Context Abstraction (ICA) details for the next chapter. This division is a result of the differences in nature of the three levels. The IMA and ILA are highly related to the implementation of the UI, the objective is to isolate developers and their software from the complexities of the different technologies required for an ubiquitous and natural UI. On the other hand, the ICA is related to the users, the usage environment, and their diversity.

Dandelion has been designed and implemented as a UI development framework for Ambient Intelligence systems. As a consequence, it provides an implementation of the TIAF abstraction levels as a distributed user interaction system that uses model-driven engineering techniques to make the development of distributed physical user interfaces (DPUIs) easier and cheaper.

By using Dandelion, AmI developers are decoupled from the specific modalities, technologies and even physical location of the Interaction Resources (IRs) used to implement a particular DPUI. Developers can design and describe the UIs at the abstract level using the UsiXML language, and then implement the application UI control logic on top of the abstract concepts defined in the abstract UI. Dandelion uses a distributed user interaction controller to connect those abstract elements with the physical elements that perform the interaction with the user. This connection is managed by Dandelion itself, that performs the translation from the abstract concepts to the real interaction with the user. It does so by relying on a series of distributed proxy-like components that elevate any kind of device or software component to the status of an Interaction Resource (IR). They provide a common interface of interaction operations that is remotely accessible though a networking protocol called the Generic Interaction Protocol (GIP) [Varela et al., 2013b, Varela et al., 2013a]. Finally, in order to facilitate compatibility with a wide range of physical devices, Dandelion introduces a device abstraction technology called UniDA [Varela et al., 2011], which decouples Dandelion from the specific APIs of each device manufacturer.

This chapter is devoted to provide a thorough description of the Dandelion Framework architecture and how it realizes the IMA and ILA levels of the TIAF. Sections 5.2 provides an overview of the framework, while sections 5.3 and 5.4 provide a detailed description of the Dandelion IMA and ILA, respectively, and section 5.5 provides a description of the UniDA device abstraction layer. Finally, the last section, 5.6, is dedicated to exploring the implementation of several examples of AmI systems to show how they benefit from the solutions proposed by the TIAF and implemented by Dandelion.

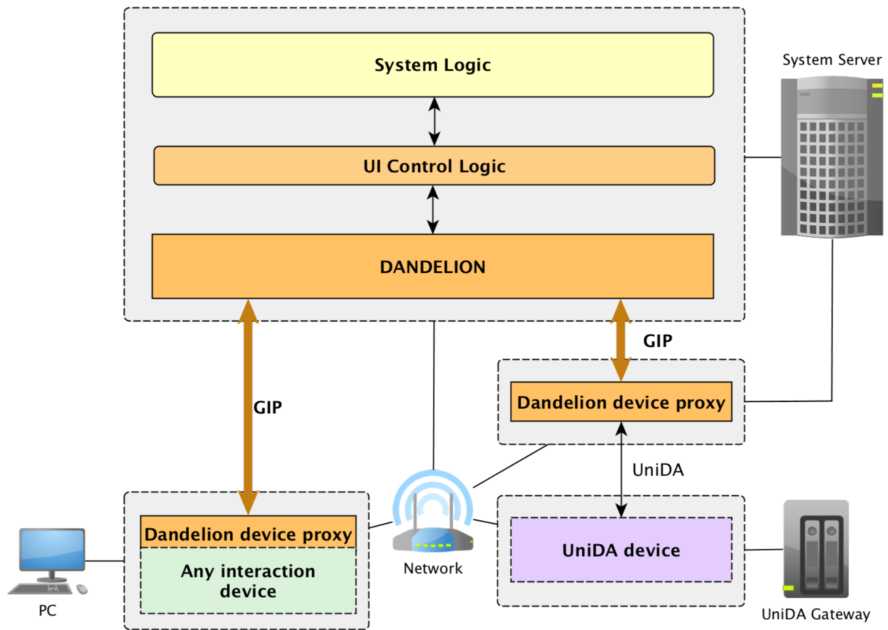


Figure 5.1: Dandelion deployment block diagram. UI Control logic is decoupled from the interaction resources by the Dandelion UI management system. Furthermore, Dandelion relies on UniDA to support a wide range of physical devices (sensors, actuators, appliances, etc.) from different manufacturers.

5.2 The Dandelion Framework

As previously introduced in section 5.1, Dandelion has been conceived as a reference implementation of the conceptual architecture and solutions introduced in section 4.3. In particular, it has been designed as a UI development framework for AmI systems, facilitating the design and implementation of distributed physical UIs, capable of adapting their shape to the usage scenario in order to preserve the system's natural interaction constraints.

Figure 5.1 shows a block deployment diagram of a system using the Dandelion framework. As can be seen, the system logic and the interaction resources operate in a distributed manner, with Dandelion in the middle decoupling them.

As an implementation of the TIAF framework, Dandelion provides decoupling mainly at three different levels. First, Dandelion provides an implementation of the IMA abstraction level, thus achieving logical decoupling between the UI control code and the interaction modalities, APIs, or specific technologies used by the devices that implement the UI. Second, by implementing the ILA level, Dandelion provides physical decoupling, so that the system logic can be

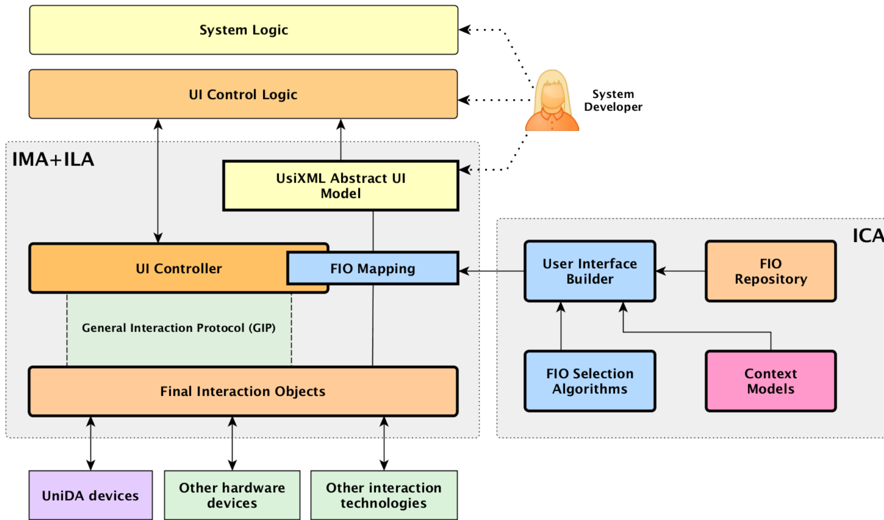


Figure 5.2: Dandelion detailed architecture block diagram.

run without knowing where the devices implementing the UI are going to be physically deployed. And third, the implementation of the ICA level allows the decoupling between the system and the specific set of IRs used to build the UI, thus allowing the UI to change its shape (the set of IRs) at run-time, without affecting the application code.

Figure 5.2 provides a detailed overview of the Dandelion architecture and how the different components of the framework support those three levels of abstraction.

The IMA level is supported by the utilization of model-driven engineering techniques to permit the design of the UI and the implementation of the UI control logic. Dandelion allows developers to design UIs at a declarative and abstract level using the UsiXML Abstract UI model, and then implement the UI control logic on top of those abstract concepts. Therefore, as can be seen in Figure 5.2, the UI designer and the UI control logic are effectively decoupled from the concrete interaction resources used. A detailed description of the Dandelion IMA implementation is provided in section 5.3.

The ILA level is supported by an implementation of the GIP distributed device abstraction layer that encapsulates the specific behavior of each device behind a generic interface of user interaction operations. This implementation of the GIP is based on publish/subscriber messaging technologies, where the set of interaction actions supported by the GIP are represented as events published and received by distributed Interaction Resources. Any device or software com-

ponent that implements this event-based interface is called a Final Interaction Object (FIO), and it is in charge of translating abstract interaction actions into real and physical interactions with a user. Every FIO, independently of their technologies or APIs, can be remotely accessed using the same set of concepts and operations, thus the GIP and the FIOs decouple the application code from the location of the interaction resources and from their underlying technologies and interaction modalities. Section 5.4 provides a complete description of the Dandelion ILA implementation.

The FIOs are the elements in charge of finally connecting Dandelion with the real devices and software components that realize the interaction to the user. To support the wide variety of IRs available, it is mandatory for Dandelion to count on a large number of different FIO implementations. In order to alleviate this problem, and reduce the cost of supporting a wide number of devices, we have introduced a device abstraction technology, called UniDA, which provides Dandelion with an homogeneous device access framework to control a network of heterogeneous devices like sensors, actuators, or appliances. UniDA allows the development of generic FIOs capable of using devices from different manufacturers, with different APIs and protocols. The UniDA framework will be described in detail in section 5.5.

The IMA+ILA combination, which allows the design and implementation of distributed physical UIs without specific knowledge of the underlying IR technologies, is supported by managing a mapping between the abstract interaction elements of the UI (from the UsiXML AUI model) and a specific set of FIOs for each usage scenario.

Finally, the ICA level is provided by a dynamic UI management system capable of modifying, at run-time, the mapping between the abstract UI of an application and the specific set of FIOs used for each scenario. In Dandelion this mapping is managed by the User Interface Builder component, which relies on a set of context models and a variety of computational intelligence (CI) algorithms to select the best suitable FIOs (IRs) for each scenario, among those available in a FIO repository.

As a final note about the Dandelion framework, it is important to mention that the implementation of the framework shown in this chapter and in chapter 6 has been released as open source software under the GNU Affero GPL v3 license, and it is accessible through a public GitHub repository available at https://github.com/GII/Dandelion. We think that it is important for software engineering thesis to provide implementations of the solutions proposed, not only to demonstrate their validity, but also to possibilitate the comparison with other solutions, and to provide the research and developer community with usable solutions for the problems addressed by the thesis.

5.3 Portable Physical User Interfaces

As previously indicated, this chapter is devoted to explaining the details of the implementation of a reference framework supporting the first two levels of abstraction proposed by the TIAF: the Interaction Modality Abstraction and the Interaction Location Abstraction. In the previous section, we have presented a small overview of that framework, Dandelion, and, in this section, we are going to explore the details of the Dandelion components and technologies that are mainly related to the support of the IMA abstraction level.

We have already shown, in section 4.3.2.1, how much physical user interface (PUI) developers can benefit from having solutions to decouple their code from the modalities, APIs, and specific technologies of the different physical interaction resources (IRs) used to build a PUI.

A common problem of systems using PUIs without a good decoupling between UI control logic and the IRs is the difficulty of deploying them in different scenarios, with diverse constraints and different IRs.

Without a good decoupling, there are two main options to deploy a PUI in different scenarios. Either use the same IRs in every scenario, which can hinder the integration of the system in the environment and its natural interaction perception, or modify the system's code to support new IRs (with different APIs, modalities, etc.) more suitable for the characteristics of the new scenario, which requires a lot of development effort.

The objective of the IMA layer is to alleviate this problem, making it easier for developers to deploy the same control logic with different sets of IRs, thus effectively improving the portability of PUIs between scenarios. The main idea is that, by implementing the UI control logic on top of the IMA conceptual model and API, developers and their code are isolated from the differences and particularities of the different IRs available. This way, the same code can be deployed on top of the most suitable devices for each scenario.

Dandelion, as a reference implementation of the TIAF, employs Model-Driven Engineering (MDE) techniques to support the IMA. Particularly, Dandelion requires developers to specify the UI interaction requirements using an abstract user interface model. This model provides developers with elements to represent user interaction actions in a generic way, thus allowing developers to specify a generic model of the interaction actions supported by their systems.

The abstract model of the UI is managed by the Dandelion User Interface Controller (DUIC), which provides developers with an API to manage and control the UI. The DUIC allows developers to directly use the abstract UI model to execute and/or receive interaction actions from the IRs that implement the final

UI. The DUIC is in charge of performing the translation between the generic abstract interaction elements and the final and real interaction resources.

The majority of frameworks that apply MDE techniques for UI development start from abstract models and perform a succession of transformations to lower level abstraction models until a final UI is achieved. However, in Dandelion, we decided to perform the transformation directly, going from abstract to final in just one transformation step.

This decision is the result of the differences in the operational nature between those frameworks and Dandelion. While the majority of frameworks based on model transformations operate at development-time, Dandelion operates at run-time.

MDE frameworks for UI development usually employ models as a supportive tool to guide developers during the development of the system. Thus, the transformation from abstract to final is performed by the developers at development-time. Consequently, the set-up of the scenario (the IRs available and the characteristics of the environment) is not known. For this reason, these frameworks start from the abstract UI and produce different versions of the UI at different levels of abstraction. First, the concrete UI level, where the interaction modalities are already selected, and then, the final UIs, which are multiple versions of each modality where the final implementation technologies or APIs are decided.

However, Dandelion performs the transformation at run-time, where the characteristics of the scenario are already known. We can avoid the intermediate transformation steps and go directly from the abstract to the final UI because we know which IRs are available, which are the characteristics of the environment and the user, etc. We don't have to start by selecting the modality and then the concrete implementation. We can already start by selecting particular IRs to implement each of the interaction actions specified in the abstract UI model.

By taking advantage of this characteristic of Dandelion, developers are only required to implement their UI control logic on top of components that represent abstract user interaction primitives. In fact, developers can work completely at the abstract level following the process illustrated in the Figure 5.3. They can design the UI using the set of concepts provided by the UsiXML Abstract UI Model, then specify the design in an XML file, and finally implement the UI control logic on top of the Dandelion UI Controller interface, which provides a very simplified interface to issue and receive user interaction actions. It is at deployment time where the transformation from abstract to final is specified by the installer of the system. This specification, essentially a mapping between abstract and final components, is used by Dandelion at run-time to translate the abstract interaction operations into real interactions with the users.

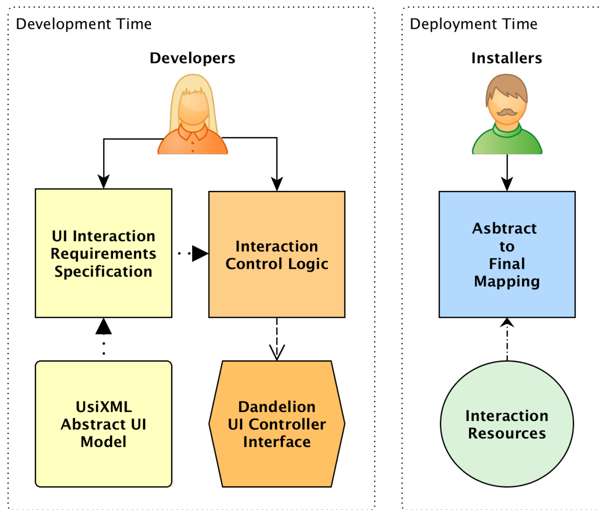


Figure 5.3: Dandelion portable UIs development process with IMA+ILA layers. In chapter 6, we will see how Dandelion implements the ICA to manage, at run-time, the mapping between abstract and final components, liberating developers/installers from that responsibility.

As previously indicated, this section is focused on the IMA+ILA layers, this is why the transformation from abstract to final, and thus, the adaptation of the UI to the context, is performed by a human installer. In chapter 6, we will see how Dandelion implements the ICA to manage, at run-time, the mapping between abstract and final components.

The next subsections explain the Dandelion development process in detail, starting from the abstract specification of the UI, then the implementation of the UI control logic, and finally the mapping from abstract to final.

5.3.1 Abstract UI Design and Specification

In Dandelion, as with almost any UI development framework, the first step for developing portable Physical UIs is to design the UI. The key characteristic of Dandelion is that this design must be performed at a very abstract and conceptual level. Developers are only required to specify the user interaction requirements of the application. Dandelion then relies on the Abstract UI model to allow developers to specify the user interaction requirements of their applications. This model, as indicated in section 4.3.2.4, provides support for a reduced set of only five different abstract user interaction primitives:

- The user can input information to the system.
- The system can output information to the user.
- The user can select information from a collection shown by the system.
- The user can request the execution of an action by system.
- The system can request the focus of the user to an specific element of the UI.

Consequently, the user interaction requirements specification of an application is reduced to a formal description of:

- How many information elements are going to be shown to the user.
- How many information elements are going to be introduced by the user.
- What type of information is going to be shown or introduced.
- How many interaction actions is the user allowed to perform.
- How the different components are organized and related to each other in the UI.

As previously indicated, this specification of the UI requirements is achieved using the Dandelion Abstract UI model, this is, as indicated in subsection 4.3.2.4, the UsiXML Abstract UI Model (AUIm) shown in Figure 4.12 of section 4.3.2.4. This model introduces the Abstract Interaction Unit concept (AIU) as an abstract representation of the widgets found in classical GUI toolkits. The AIUs work as containers for interaction primitives, thus allowing the specification of the relations between the different user interaction elements. Furthermore, AIUs can also contain other AIUs, thus permitting the specification of the UI shape at an abstract level using a hierarchical organization.

User interaction primitives are represented in UsiXML AIUm by the concept of interaction facet. They represent the different interaction features (input, output, selection, etc.) supported by an AIU, and each AIU can have many different facets (e. g. one input facet and two selection facets). There exist five different types of interaction facets to match the five different kinds of interaction primitives supported by the model:

- The DataInputOutputFacet can be used for user input, output, or both, depending on its parameters.
- The DataSelectionFacet can be used to specify user selection primitives.

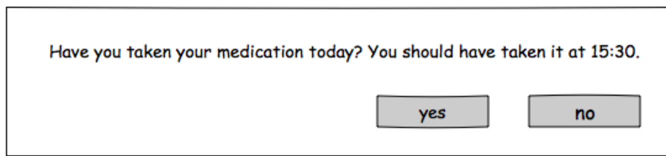


Figure 5.4: Sketched GUI version of the OMNI notification subsystem UI.

- The DataTriggerFacet can be used to specify system actions that can be issued by the user.
- The RequestFocusFacet can be used to specify that an AIU must support a way to request the focus of the user.

Both model elements, AIUs and interaction facets, can be parameterized with important requirement information like data type or data cardinality, among others.

In order to facilitate the comprehension of how the different concepts provided by the Abstract UI Model can be used to design UIs from an abstract point of view, we are going to explore some examples of abstract UI design and specification in the next part of this section.

Let's start with a small and simple example: the UI of the OMNI notification subsystem presented in section 4.2.1. It is a fairly simple UI that in a GUI system would be implemented as a dialog window with a label and one or two buttons as shown in Figure 5.4. Thinking about their specification in interaction primitives, it requires to output one piece of information to the user, the notification message, and it must allow the user to perform two different actions, either confirm the notification or reject it.

The code listing of Algorithm 5.1 shows a simplified version of the XML code required to specify the UI with the UsiXML AIUm. We use one Abstract Interaction Unit as a container for the different interaction facets, taking the roll of the dialog window. Then, we need to use three interaction facets: one DataInputOutputFacet for outputting the message, and two TriggerFacet for allowing the user to issue the 'yes' or 'no' actions.

As can be seen, a small UI can be specified with little effort just by describing the interaction primitives required, and by organizing them into AIUs in order to specify how the different interaction elements are logically related to each other.

Lets now take a look at a slightly more complex example of user interface; the one of the Environmental Music Player described in section 4.2.2. As shown in Figure 5.5, while still being a small UI, it requires many more interaction

Algorithm 5.1 XML code describing the OMNI notification UI with UsiXML.

```
<!– notification dialog –>
<aui:AbstractUIModel>
    <aui:AbstractInteractionUnit id="NotificationDialog" ... >
        <aui:DataInputOutputFacet id="NotificationMessage"
        minCardinality="1" maxCardinality="1" dataFor-
        mat="string" inputSupport="false" outputSup-
        port="true" ... >
            <aui:dataType>text</aui:dataType>
        </aui:DataInputOutputFacet>
        <aui:TriggerFacet id="YesAction">
            <aui:triggerType>operation</aui:triggerType>
        </aui:TriggerFacet>
        <aui:TriggerFacet id="NoAction">
            <aui:triggerType>operation</aui:triggerType>
        </aui:TriggerFacet>
    </aui:AbstractInteractionUnit>
</aui:AbstractUIModel>
```

primitives and specially more diverse ones.

Algorithm 5.2 shows a reduced version of the specification of the player controls part of the EMP UI. This part of the UI is in charge of providing the user with the required controls to manage the playing of music. This includes the ability to change from one song to the next, start or stop playing music, change the music style, or change the volume of the audio. These last two requirements, change the music style or change the audio volume, are the most interesting part of this example.

Regarding the volume selection, it could be designed in multiple ways. One possibility would be to model it as two separate actions, one to increase the

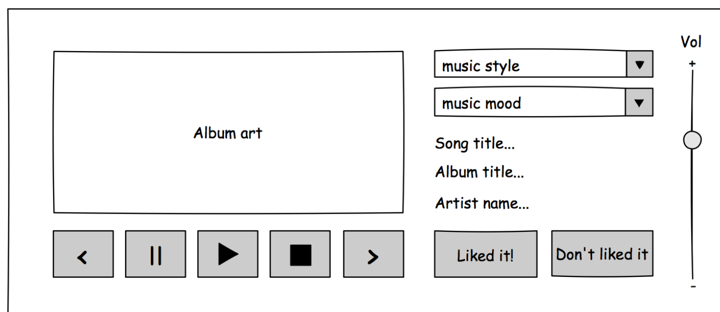


Figure 5.5: Sketched UI of the Environmental Music Player using WIMP user interfaces. Already shown in Figure 4.3.

volume and another one to decrease it, thus managing the state of the volume in the application logic. Nevertheless a better option seems to be to model it as a data selection primitive, that is, a DataSelectionFacet. This kind of primitive allows the selection of one (or more) data values from a group. Thus, in this example, it allows the selection of a number between 1 and 100, and it internally keeps control of the state of the selection.

The music style selection is a little bit more complex than the volume selection. In this case, it is not easy to model it as individual actions, because the number of music styles available can be dynamic and vary depending on the collection of music available, thus it is mandatory to model it as a selection primitive using text items. These items will be dynamically provided by the application logic at runtime.

The example of the EMP UI is also useful to show how the Abstract Interaction Unit concept can be used in different ways to organize the UI elements. In the control part of the UI, we use a main AIU as container for all the controls (imagine it as a kind of an internal panel in a GUI), and then, we use three AIUs to organize the different facets required by the UI. One AIU for music player controls, one for song selection, and one for volume selection. This organizational information is not only useful to facilitate the design by taking advantage of the famous divide and conquer strategy, but it can also be used by Dandelion as hints to select and manage the final and physical realization of the UI. For example, keeping the controls together, physically separating controls of different AIUs, etc.

It is important to note that input and output are not relegated to only basic data types, like numbers or text. As shown in section 4.3.2.4, they support a set of data types including images. For example, in the case of the EMP, it requests to output the images of the album art for each song. In the code listing of Algorithm 5.3, we display how a DataInputOutputFacet can be used to output/input images to/from the user.

In this section, we have shown how the Dandelion Abstract UI Model can be used to describe user interfaces at the abstract level, specifying their user interaction requirements as generic interaction primitives, and organizing them into containers in an hierarchical way. We have presented a couple of simple examples for illustration purposes. Nevertheless, in subsection 5.6, the reader will find a more complete version of those examples.

5.3.2 UI Control Logic Implementation

Recalling the process shown in Figure 5.3, once the UI has been designed and specified at the abstract level, the next step is to implement the UI control

Algorithm 5.2 XML code describing the player controls part of the Abstract UI model for the EMP.

```
...
<aui:AbstractInteractionUnit id="PlayerControl"">

    <aui:Composition rationale="player control panel">
        <aui:AbstractInteractionUnit id="PlayStop">
            <aui:TriggerFacet id="PlayAction">
                <aui:triggerType>operation</aui:triggerType>

            </aui:TriggerFacet>
            <aui:TriggerFacet id="StopAction">
                <aui:triggerType>operation</aui:triggerType>

            </aui:TriggerFacet>
        </aui:AbstractInteractionUnit>
        ...
        <aui:AbstractInteractionUnit id="VolumeControl">
            <aui:DataSelectionFacet id="VolumeSelector"
            start="1" end="100" isContinuous="true" se-
            lectionType="SINGLE">
                <aui:dataType>number</aui:dataType>
            </aui:DataSelectionFacet>
        </aui:AbstractInteractionUnit>
        ...
        <aui:AbstractInteractionUnit id="MusicStyleControl">
            <aui:DataSelectionFacet
            id="MusicStyleSelector" isContinuous="false"
            selectionType="SINGLE">
                <aui:dataType>text</aui:dataType>
            </aui:DataSelectionFacet>
            <aui:DataSelectionFacet
            id="MusicMoodSelector" isContinuous="false"
            selectionType="SINGLE">
                <aui:dataType>text</aui:dataType>
            </aui:DataSelectionFacet>
        </aui:AbstractInteractionUnit>
    </aui:Composition>

</aui:AbstractInteractionUnit>
...
```

Algorithm 5.3 XML code describing the audio metadata part of the Abstract UI model for the EMP.

```
...
<aui:AbstractInteractionUnit id="MusicMetadata">
    <aui:Composition rationale="player music metadata panel">
    ...
    <aui:AbstractInteractionUnit id="AlbumMetadata">
        <aui:DataInputOutputSupport id="AlbumCover"
        dataFormat="image" inputSupport="false" outputSup-
        port="true">
            <aui:dataType>image</aui:dataType>
        </aui:DataInputOutputSupport>
        <aui:DataInputOutputSupport id="AlbumTitle"
        dataFormat="string" inputSupport="false" outputSup-
        port="true">
            <aui:dataType>text</aui:dataType>
        </aui:DataInputOutputSupport>
        ...
    </aui:AbstractInteractionUnit>
    ...
    </aui:Composition>
</aui:AbstractInteractionUnit>
```

logic code. This logic is specific for each application and it is in charge of defining how it uses the different UI elements in order to exchange information between the system and the user. It specifies, for example, when a message is going to be shown to the user, what to do with the information introduced by the user, or what business logic action to execute when the user issues a specific UI action. More generally speaking, the job of the UI control logic is to fill the gap between the business logic and the UI, thus decoupling one from the other.

As an implementation of the TIAF, the main characteristic of Dandelion regarding this topic is that, given that the UI is designed at the abstract level, the UI control logic can also be implemented on top of the same abstract UI concepts. Dandelion provides developers with an external façade, the Dandelion UI Controller (DUIC) interface shown in Figure 5.6, that exposes the functionality of the framework through a reduced set of operations tightly related to the abstract concepts proposed by the Abstract UI Model. All the operations exported by the DUIC interface are executed over abstract elements (AIUs and interaction facets), which are then translated by the Dandelion User Interface Controller (an implementation of the TIAF's Interaction Realization Module described in section 4.3.2.1 and Figure 4.7) into real operations over the IRs. Thanks to that, by building the interaction control logic on top of this set of abstract interaction operations, developers are able to completely decouple that logic from the technologies, APIs, and specific hardware of the Interaction

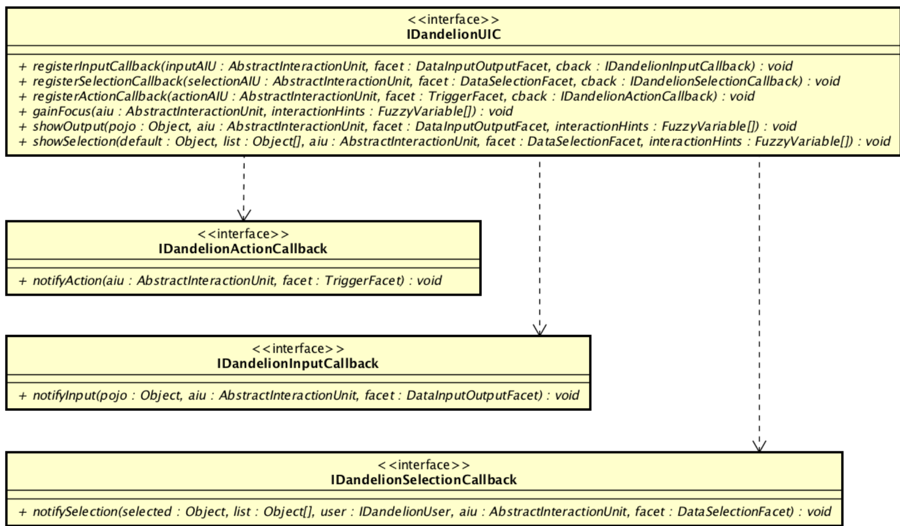


Figure 5.6: External interface of the Dandelion User Interface Controller as seen by developers and UI control logic code.

Resources (IRs).

As can be seen in Figure 5.6, this reduced set of operations supports all the five interaction primitives proposed by the Abstract UI Model. In particular, it allows the system to request Dandelion to:

- Show some information to the user.
- Show a collection of information items to the user, and let she select one of them.
- Gain the user focus over an specific element (AIU) of the UI.
- Notify a callback system object when the user inputs some information.
- Notify a callback system object when the user triggers the execution of some action.

It is noteworthy that all of these operations are abstract, and neither the UI model or the UI control logic specify how they are implemented. As we will see in more detail in section 5.3.3, it is the responsibility of the DUIC to provide a concrete implementation of those abstract operations, specifying, for example, how the UI is going to gain the focus of the user over a specific part of it, or how the UI is going to show an information message to the user.

Just to clarify how the DUIC interface can be used to implemented the UI control logic, let's examine some small code examples from the EMP UI

Algorithm 5.4 UI Control Logic code required to set the business logic actions that must be called when the user triggers the actions to start playing music or to stop it in the EMP example.

```
AbstractInteractionUnit playerControlAIU =
app.getAbstractUI().getAbstractInteractionUnitById("PlayStop");


TriggerFacet playTriggerFacet = (TriggerFacet)
playerControlAIU.getInteractionFacetById("PlayAction");

dandelionUIC.registerActionCallback(
playerControlAIU, playTriggerFacet, new StopActionCallback(musicPlayer));


TriggerFacet playTriggerFacet = (TriggerFacet)
playerControlAIU.getInteractionFacetById("StopAction");

dandelionUIC.registerActionCallback(
playerControlAIU, playTriggerFacet, new PlayActionCallback(musicPlayer));
```

example. Algorithm 5.4 shows a small JAVA code snippet from the EMP example. This code uses the DUIC façade to configure the callback objects, from the business logic, that must be called by Dandelion when the user triggers the actions “PlayAction” or “StopAction” specified in the Abstract UI.

Following with the example of the EMP, the code listed in Algorithm 5.5 shows how to use the DUIC interface to output a string message to the user, in this case the title of the song that the EMP is playing.

As can be seen in these small examples, the implementation of the UI Control Logic is completely free of any details about how the UI is going to be finally implemented. It is coded just on top of the abstract interaction elements specified in the Abstract UI Model and by using abstract interaction operations and primitives.

Finally, it is important to note that, unfortunately, the fact of designing the UI and implementing the control logic at such a high level of abstraction has an important drawback. As happens with any kind of abstraction technology, many details and particular capabilities of the IRs are hidden to the developers, thus hindering the fine grain customization of the UI and the user experience. It is more difficult for developers to take advantage of all the particular possibilities of each technology and device, and it is more difficult to implement highly customized UI behaviors. This is a common and, to some extent inevitable, issue of abstraction technologies. However, in Dandelion each abstract interaction operation supported by the DUIC can be customized by a collection of Interaction Hints (IHs). They are fuzzy variables indicating suggestions or hints

Algorithm 5.5 UI Control Logic code required to output the song title string to the user. It is executed each time the song changes.

```
//show the song title to the user
dandelionUIC.showOutput(
    currentSong.getID3v1Tag().getSongTitle(),
    songMetadataAIU, songTitleFacet, new HashSet<FuzzyVariable>(0));
```

to the DUIC about how the developers want that operation to be implemented. Some examples of Interaction Hints are the privacy level, the importance level, or the color used to show an interaction operation.

5.3.3 From the Abstract to the Final User Interface

As shown in Figure 5.3, once the UI has been designed and the UI control logic implemented, the system is ready for deployment. It is at that point where the final UI has to be assembled for each specific usage scenario. In this section, we are describing the Dandelion implementation of the IMA and ILA layers without ICA support, thus, the selection of which particular IRs are going to be used in each scenario is managed, manually, by the developer or system installer during the deployment of the system in a particular scenario.

In Dandelion, the IMA and ILA layers are highly related because, as we will see in more detail in section 5.4, the Particular IR Realization Module (see Figure 4.7 and subsection 4.3.2.1) is directly implemented using the GIP and remote proxy components that work as abstractions of the real IRs available. As the GIP continues to provide an abstract interface of interaction primitives directly inspired by the UsiXML Abstract UI model, the process of building the final UI consists exclusively on selecting, among the physical IRs available, those that better match the needs of the scenario, and then, establishing a mapping between the different interaction facets specified in the UI abstract model and the interaction capabilities of the IRs. The next section will provide much more detail on how the IRs are abstracted and connected to the UI control logic.

The selection of the IRs to build the final UI is the only point where the developers or installers are not decoupled from the context and the particular devices available. This is because they must bear in mind the characteristics of the context (environment, user, use case, etc.) and the characteristics of the IRs (modalities, physical characteristics, etc.), but, as can be seen, this happens only at deploy time, thus the system has great portability to different scenarios, possibilitating, as shown in Figure 5.7, the deployment of different UIs, depending on the scenario, without affecting the system's code. Furthermore, even if

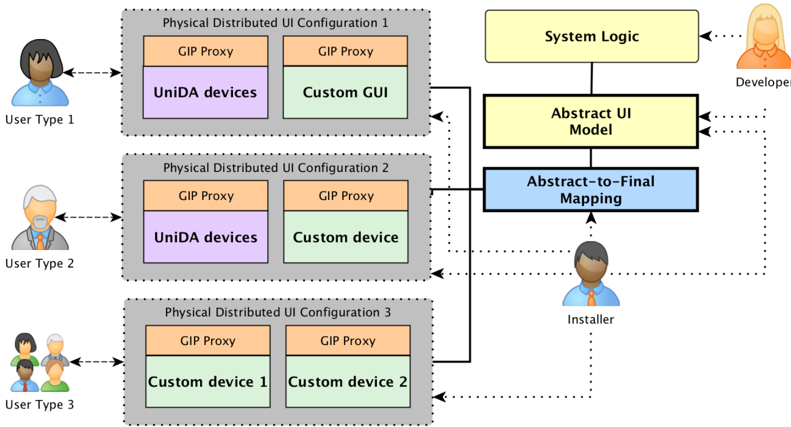


Figure 5.7: Many different UIs can be deployed for the same Abstract UI definition and the same UI Control Logic. The installer of the system only has to specify different mappings, between abstract and final UI elements, for each scenario.

developers and installers are not decoupled from the context and characteristics of IRs, they continue to be decoupled from their technologies and APIs.

In the next section, we are going to explore in detail how Dandelion implements the connection between the Abstract UI and the final IRs by implementing the ILA and the GIP.

5.4 Distributed Physical User Interfaces

In the last section, we have shown how Dandelion implements the IMA to facilitate the development of highly portable Physical User Interfaces. The IMA in Dandelion allows developers to design and implement their UIs on top of abstract user interaction components that are then transformed into final and real interaction resources at deploy-time and run-time [Varela et al., 2013a, Varela et al., 2014]. In this section, we are going to describe in detail the implementation of the ILA, which in Dandelion is highly related to the IMA, because, as we will see, it is an essential part of how Dandelion implements the transformation from abstract to final UI.

Figure 5.8 shows a detailed overview of the ILA implementation in Dandelion. As can be seen, there are three main software components:

- *The UI Controller.* It is the link between the IMA and the ILA layers.
- *The General Interaction Protocol (GIP).* It provides distributed access to

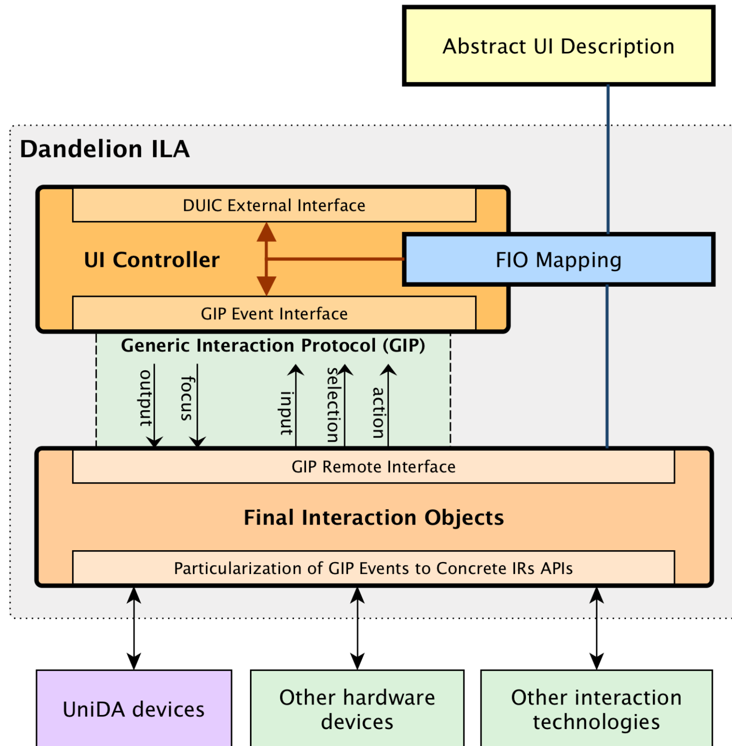


Figure 5.8: Detailed overview of the ILA implementation in Dandelion. The UI Controller operates as a router of abstract interaction operations from the UI Control Logic to the Final Interaction Objects, using the GIP as transport protocol.

the IRs using an interface of generic user interaction operations inspired by the Abstract UI model.

- *The Final Interaction Objects (FIOs).* They provide concrete implementations of the GIP interface, translating the generic abstract concepts and operations to the particular APIs of each IR supported.

The UI Controller lies between the IMA and ILA layer. It is an implementation of the Interaction Realization Module of the IMA but, thanks to the GIP, its behavior is quite simple, it operates as a router of abstract user interaction operations.

As introduced in subsection 4.3.2.5, the GIP event interface matches closely the operational interface of the DUIC. Both support the same five different user interaction primitives (input, output, selection, action and focus). Therefore, the job of the DUIC is reduced to using the mapping between abstract and final

UI (FIO Mapping), established by the installers, to decide which FIOs must receive a particular operation requested to the DUIC. For example, when the DUIC receives a request to perform an output or gain the focus of the user, it transforms the operation into a GIP event, looks at the FIO mapping table, and sends it to a set of FIOs through the network. These FIOs are in charge of translating those abstract operations into particular actions using the APIs of their IR. The process is the reverse when the interaction operation is started by the user. For an input or action operation, the FIO generates a GIP event that is received by the DUIC and, again, using the mapping, the DUIC calls the callback operation associated to the interaction facets mapped to that particular FIO.

After this brief explanation of how the Dandelion ILA works, and what is the main role of the DUIC, we are going to explore in detail the implementation of the other two main components of the Dandelion ILA, the GIP and the FIOs, in the next two subsections.

5.4.1 The Generic Interaction Protocol

The Generic Interaction Protocol [Varela et al., 2013b], already introduced in subsection 4.3.2.5, is used by Dandelion to provide the decoupling between the system and the IRs at two levels. First, it is the main element that encapsulates the behavior of heterogeneous IRs behind a common set of abstract interaction operations. Second, as a distributed network protocol, it decouples the system from the physical location of the IRs. Because of that, even if it is more related to the ILA, in Dandelion, the GIP is a key element of the IMA+ILA combination.

As the GIP is conceptually designed as an event based protocol, we decided to implement it using a messaging protocol that supports the publisher/subscriber paradigm. Each component of the system, essentially the DUIC and the FIOs, will act as a publisher or subscriber of GIP events, depending on the situation. Therefore, on the one hand, the FIOs will act as publishers of GIP events generated from actions of the user, while the DUIC will act as a subscriber of those events. On the other hand, the DUIC will act as a publisher of events generated from operations requested from the UI Control Logic and the FIOs as subscribers of those events.

As previously indicated, the FIOs are in charge of translating physical user interaction actions into GIP events representing those actions, and then send those events to every DUIC that is using the FIO. For that purpose, as shown in Figure 5.9, by taking advantage of the publish/subscribe paradigm characteristics, every FIO, on startup, creates its own topic in the messaging

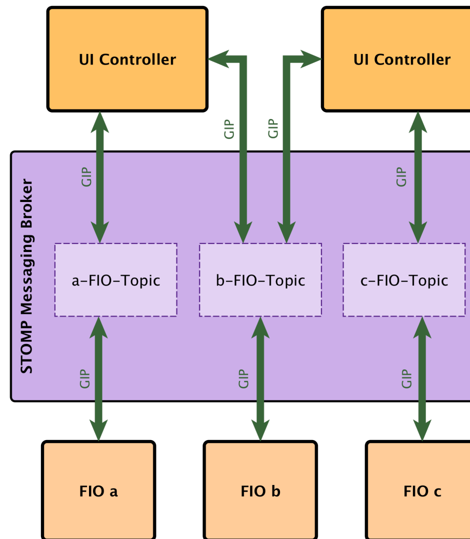


Figure 5.9: Each FIO creates its own topic in the broker and then uses it to publish its generated GIP events. DUICs of all the applications using that FIO are subscribed to that topic.

broker and uses the topic to publish its own GIP events. Then, every DUIC interested in the events of a particular FIO only needs to subscribe itself to the FIO topic. Furthermore, in order to send GIP events to the FIO, the DUIC uses that same topic to publish them.

Dandelion uses STOMP [STOMP, 2012] (Simple Text Orientated Messaging Protocol) as the transport protocol for GIP events and ActiveMQ [Apache-Foundation, 2015] as the default messaging broker. This selection of technologies allows us to acquire two important characteristic for the Dandelion GIP implementation. The use of ActiveMQ allows us to use the standard JMS API [Monson-Haefel and Chappell, 2000] for the implementation of the DUIC using Java. Furthermore, ActiveMQ abstracts its clients from the particular messaging protocol used, thus, even if STOMP is the recommended transport protocol, it could be possible to use a different one, or even mix multiple protocols, with different FIOs using different protocols depending on their implementation technologies. So, why is STOMP the recommended protocol? Because, as its name suggests, it is a very simple text protocol, very easy to implement, very light on resources, and thus, very portable across devices and software platforms. Furthermore, in order to facilitate even more the portability of the FIO layer, Dandelion encodes the data of the GIP events using JSON[JSON.org, 2015, ECMA-International, 2013], which makes them very easy to parse in any

technology.

5.4.2 Final Interaction Objects

The Final Interaction Objects are the end elements in charge of physically interacting with the user. They are software abstractions of heterogeneous interaction resources that could be either hardware (keyboards, remote controllers, appliances, sensors, etc.) or software (GUIs, voice recognition, etc.).

FIOs are implemented as software applications that implement the GIP remote interface. They can be implemented in any programming language, for any software platform, and using any API required by an IR. The only requirement to be compatible with Dandelion is to implement the GIP interface according to the JSON GIP codec of Dandelion, and to use a STOMP client to connect the FIO to the messaging broker. In fact, if the ActiveMQ broker is used, as suggested, it is even possible to use messaging protocols different than STOMP, because the broker handles the translation between protocols transparently.

As previously indicated, the FIOs assume the role of GIP publishers, abstracting the behavior of an IR as a set of events that notify the different actions the user is performing. But each particular FIO implementation is only required to support a subset of GIP events, as there can be IRs supporting only input, output, action, etc. The type and number of interaction facets supported by each FIO is defined in its description, which includes information like the type of data the FIO is able to manage, the cardinality of that data, the type of interaction supported, and the interaction modality used by the FIO. This is, each FIO describes its supported interaction facets, and it indicates the kind of data it can obtain from the user as input or selection, or show to the user as output. This data can be of any basic type: integer, float, string or boolean, and images in JPEG or PNG format. It is important to note that each FIO can provide one or many different interaction facets. For example, a FIO can specify that it is able to output a string, input an integer, and receive an action from the user.

The FIO descriptions are used during the definition of the mapping from abstract to final in order to match the abstract interaction facets to a set of adequate FIO interaction facets. For example, the output interaction facet of an AIU for a string type can be associated to an LCD display encapsulated by a FIO with an output interaction facet supporting strings. As shown in Figure 5.10, to facilitate the management of the system and to possibilitate the autonomous selection of the FIOs when the ICA layer is available, Dandelion uses a FIO repository to store the descriptions of all the FIOs available. For that

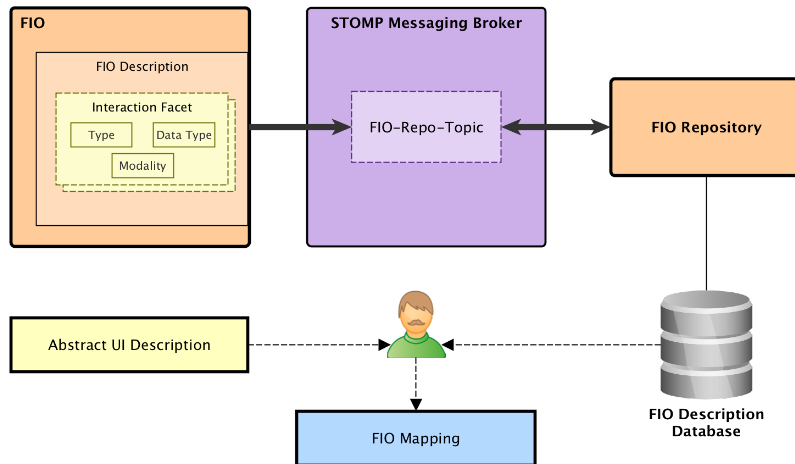


Figure 5.10: FIO's registration process in the FIO repository. FIO descriptions are stored in a repository that can be used by the installers, or the ICA, in order to set up the abstract to final mappings.

purpose, every FIO must register itself, at startup, in the FIO repository. The FIO repository is also connected to the Dandelion system using the messaging broker and a protocol using JSON and STOMP.

For every different kind of device or interaction software used by a system, a FIO must exist that abstracts it using the GIP. Obviously, a key point to facilitate the use of Dandelion is that developers should not need to develop their FIOs, or at least, not many of them. They should be provided by Dandelion itself or by the manufacturers of the interaction resources.

In order to alleviate the problem of developing FIOs for the large number of different devices and technologies available, we have designed and implemented a hardware abstraction layer called UniDA [Varela et al., 2007, Varela et al., 2011, MyTech-IA and GII, 2014], which provides a generic interface to remotely access and use any kind of hardware device. In UniDA every device is accessed using a generic model of devices and their operations, and each type of device is reduced to a set of common operations typical of that kind of devices. It is, consequently, possible to use similar devices from different manufacturers or technologies using the same exact API. This way, one FIO can be implemented on top of an API that allows it to be compatible with a large number of physical devices. The next section provides a complete description of the UnIDA abstraction technology and how it is used in Dandelion.

5.5 Physical Device Access and Control

The TIAF and Dandelion are designed to transform any kind of device or software component into an interaction resource by isolating them behind a set of abstraction layers. In the case of Ambient Intelligence user interfaces, developers usually rely on embedded devices, like sensors, actuators, or appliances, to build natural user interfaces making use of a technologically augmented environment to interact with the user. The market of this kind of devices is very fragmented and very poorly standardized, thus difficulting the provisioning of support for the wide diversity of devices than can be found in different scenarios. In Dandelion, as described in subsection 5.4.2, the way to deal with this diversity is by building different FIO implementations for each device technology/API/kind. Nevertheless, this solution can become expensive due to the large number of different technologies, APIs, and device types available, thus requiring the implementation of a large number of different FIO applications.

In order to alleviate this problem, we have designed and implemented the UniDA framework (Uniform Device Access, UniDA), which is for the world of remote device access, the same as the TIAF and Dandelion are for the world of Physical User Interfaces. It provides a distributed abstraction layer to facilitate and reduce the costs of developing applications that operate in Human Interaction Environments (HIE) and require the use of multiple heterogeneous physical devices.

It is possible to use UniDA for two different, but interrelated, purposes. As an abstraction layer, it allows the development of applications that handle hardware devices with independence of the technologies used in each device and their particular characteristics. As a complete HIE instrumentation solution, it permits building distributed device networks with support for the transparent integration of existing installations and technologies.

Figures 5.11 and 5.12 compare the vision of the hardware devices that an application has when it requires the interaction with heterogenous physical devices. The first one, Figure 5.11, shows the vision of applications that use the devices directly using their specific APIs and protocols. The second one, Figure 5.12, shows the vision of applications that use UniDA to interact with physical devices.

In the first case, the applications have a heterogeneous vision of the network of devices; they must include particular logic to interact with each specific technology and device available in the network, thus complicating the development process and making the addition of new devices more difficult, as they need to take care of all the complexities and particularities of each hardware technology deployed in the installation. In the second case, see Figure 5.12, the

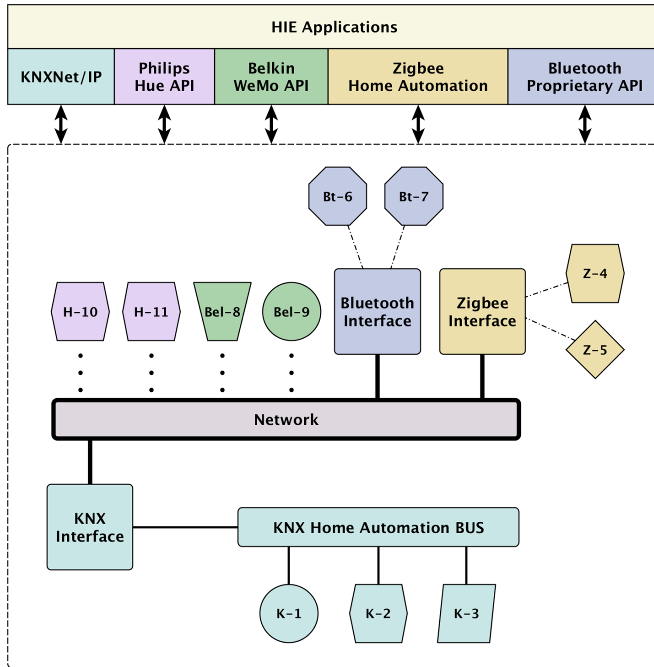


Figure 5.11: Without UniDA, an HIE application using heterogenous devices is exposed to the particular characteristics and APIs of each device technology.

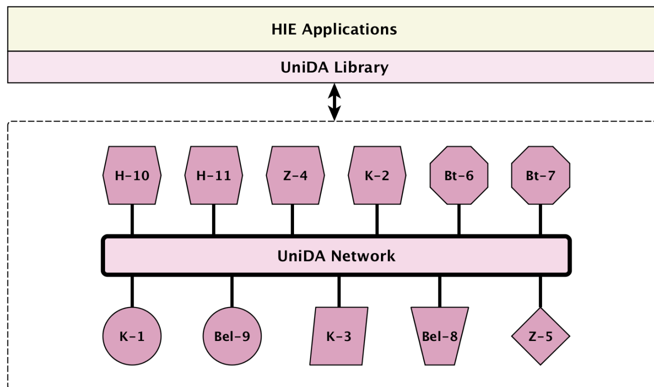


Figure 5.12: By using UniDA, HIE applications are isolated from device APIs and their particular characteristics. All the devices are accessed through the same channel and API, and all the devices of the same type (independently of their technology) are interfaced in the same way.

applications have a homogeneous vision of the network, and they are able to use the same concepts and operations to interact with every device independently of their underlying technologies and APIs. Applications do not require knowledge of specific technologies because they can use devices from different technologies homogeneously, and new devices can be easily added to the network without requiring any, or at most minimal, modifications of the application logic.

In the same way as the TIAF and Dandelion, there are two points of view to describe the UniDA framework. On the one hand there is a set of abstract conceptual components that made up a conceptual framework for the development of solutions in the HIE instrumentation field. On the other hand, there are a set of elements that implement this conceptual framework, providing usable solutions for the design, implementation, and deployment of HIE instrumentation systems.

The conceptual framework is made up of three components. A common conceptual model for the description of an instrumentation network and its devices, a uniform paradigm to model the interaction with devices, and a distributed operation protocol for the interaction between the different distributed elements of the system.

These components are realized by two main elements, complemented with some configuration tools, which allow developers to use the model to interact with the available hardware devices within their software:

- A software library (developed in JAVA) implements the common conceptual model and provides a simplified façade with the operations supported by the uniform device access paradigm. An ontology [Bonino and Corno, 2008] is used to support the model and enrich its semantics, allowing some useful inference capabilities, like device class inheritance.

- Proxy-like components, called device gateways, one for each supported instrumentation technology, are in charge of translating the abstract concepts managed by UniDA to specific concepts of a particular technology. These gateways are usually deployed on remotely accessible embedded hardware devices that are physically connected to the end devices or to other instrumentation network technologies.

The UniDA conceptual framework, together with its implementation components, builds a complete framework for interoperability of instrumentation hardware, thus alleviating the development costs of applications and installations for such heterogeneous environments as HIEs.

As shown in Figure 5.13, Dandelion, as a platform for the development of UIs for AmI systems deployed on HIEs, uses UniDA to reduce the number of

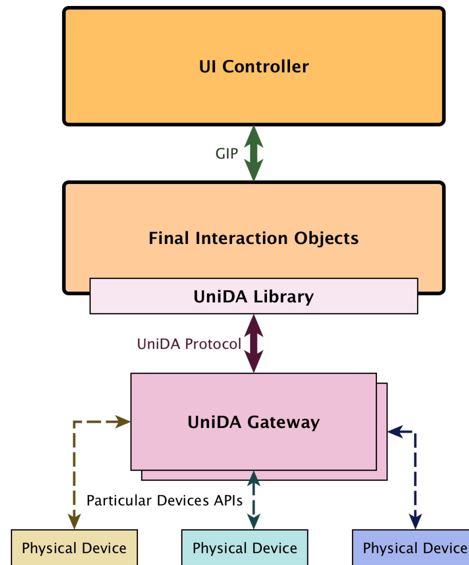


Figure 5.13: UniDA allows us building FIOs that support many devices of the same kind but with different technologies and APIs.

different FIOs necessary and to make the implementation of those FIOs easier. UniDA homogenizes the devices, thus, devices that perform the same function, but use different APIs, can be accessed using the same API and a common device model. Dandelion, as we will see in more detail in the examples of section 5.6, comes with various FIOs already implemented, some of them using UniDA. This allows Dandelion to be compatible with a larger number of devices, thus directly making AmI applications compatible with more devices. Furthermore, UniDA makes the development of such FIOs easier and cheaper.

Finally, it is important to mention that the implementation of the UniDA framework, developed within this thesis, has been released as open source software, thus making it available for others to use it, study it, and for comparison purposes with other solutions. The framework source code and documentation is available in a public GitHub repository (https://github.com/GII/UNIDA), and some binaries and further documentation is available at the project's web site (http://www.gii.udc.es/unida).

The next two subsections will provide a detailed description of the UniDA framework and its reference implementation.

5.5.1 UniDA Conceptual Framework

The UniDA framework is constructed around three conceptual components. These components make up a complete conceptual framework for the development of instrumentation systems for HIEs. Therefore, when using UniDA, these components represent the shared knowledge that developers, installers, and even manufacturers, need in order to design and implement applications, systems, and devices for HIEs. This subsection is devoted to the description of these three conceptual components.

5.5.1.1 Device Network Model

In UniDA, the homogeneous vision of a heterogeneous network of technologies is provided by a device network model and a uniform device access paradigm. This conceptual model takes the similarities between the different existing technologies and builds a new set of concepts that represent the essential characteristics of every instrumentation technology. This set of concepts abstracts the peculiarities of a heterogeneous instrumentation network, providing developers with a homogeneous language to interact with devices, backed up by a set of software and hardware components that translate it into the particular concepts and technologies required to interact with specific devices. When using UniDA, developers, and more specifically, the applications that must make use of the instrumentation network, only need to talk in the common language provided by the common conceptual model. Therefore, this conceptual model must be, on one hand, powerful enough to support all the characteristics and functionalities required by the applications, and on the other hand, simple enough to be easy to use and not transfer the hardware complexities to the applications.

Trying to hide all the particularities and complexities of every instrumentation technology is not an easy goal. Even if a set of common concepts can be easily identified, the wide variety of devices and characteristics available can pollute the model with an increasing number of low-level concepts such as the types of devices that could exist, the different states a device can have, the different operations supported, etc., complicating the usage of the model with a lot of unneeded details and information. It would be really difficult and costly to try to identify in advance every type of state or functionality that can be available, and in fact, it must be taken into account that new types of devices may appear in the future. Consequently, it was decided to decouple the model into two models, a simple abstract model to represent the high-level concepts and relationships that support the operation and management of an instrumentation network, and another model to act as a taxonomy of the specific instances of those concepts and the relationships that could exist. This way, the frame-

work can easily accommodate new types of devices by only defining them in the taxonomy using the simple abstract model, and every operation supported by the framework can be performed on top of those abstract concepts, isolated from the particularities of each device.

One of the best ways to describe such a taxonomy is to use ontology description languages [Bandara et al., 2004, Dibowski and Kabitzsch, 2011], like RDF [W3C, 2011b] or OWL [W3C, 2011a]. There are some device description ontologies available, unfortunately many of them are too related to computer devices [FIPA, 2011, W3C, 2007] and, in general, they are only conceptual ontologies, and it is difficult to find publicly accessible and usable implementations of them [Bandara et al., 2004, Dibowski and Kabitzsch, 2011]. Nevertheless, one prominent example of a device ontology is the DogOnt ontology [Bonino and Corno, 2008], a device reasoning ontology developed by the e-Lite research group of the Politecnico di Torino, in Italy.

We decided to use the DogOnt ontology as our taxonomy of devices, not only because it is very complete, well supported, and updated, but also because it matched very well the requirements of the conceptual model that we were designing. Our model is based in three main concepts: the devices, the functionality that those devices provide, and some kind of proxies or gateways that connect the devices to the instrumentation network. The DogOnt ontology offers very good support for these concepts, the device concept is directly supported and its functionalities are represented by two elements: control functionalities and notification functionalities. Furthermore, even the device proxy concept is directly supported by a gateway concept represented in the taxonomy as 'DomoticNetworkComponent'.

Therefore, many of the concepts found in the model presented here are adapted, and even some of them directly extracted, from the DogOnt ontology. Thus, the conceptual model, as shown in Figure 5.14, is made up of eight main concepts:

- *Devices.* These are the devices that a user expects in his instrumentation network. They provide end users with services and functionalities. There exist two different kinds of devices; physical devices and groups of devices.

- *Physical devices.* They are a conceptual representation of the real hardware devices.

- *Groups of devices.* Multiple devices can be grouped together and a group has the same entity as one device. Thus, a user or an application can send commands, queries, etc. to a group of devices in the same way (transparently) as to a single device.

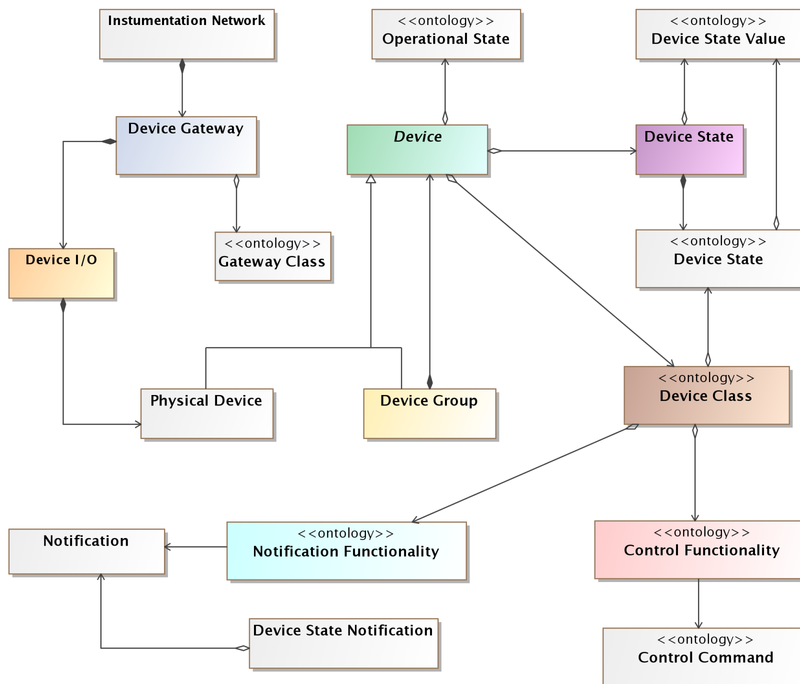


Figure 5.14: Class diagram showing the different concepts and their relations that populate the UniDA Device Network Model.

- *Device gateways.* They are the connection point between the devices and the instrumentation network. They act as a control point for the devices, controlling them and transferring to them the commands that came from the instrumentation network, and vice versa.

- *Device I/O.* Physical devices are not directly connected to the gateways, they are connected through what we call a device I/O. A gateway has a set of device I/Os to which devices can be connected (this set can be static or dynamic) and each device I/O has a list of states that restrict the type of devices that it can hold, so the I/Os of a gateway specify the number and type of devices it supports.

- *Classes of devices.* The class of a device represents its type. It supports inheritance thanks to the use of ontologies.

- *Device states.* They represent the type of states that a device has. A device can have multiple different states and they are specified by its device class. The states of a device can be read and optionally written.

- *Device commands and device notifications.* Together, they represent the functionality of a device. Commands can be received by a device and it usually reacts by changing its state and acting over the environment changing it. Notifications represent changes in the state of a device. Devices send notifications to notify the clients of the instrumentation network about a change in its state.

As the objectives of this proposal are a bit different and broader than the DogOnt objectives, even if some concepts of the model presented here directly match concepts from the DogOnt ontology, there exist important conceptual differences. One of the more prominent differences is the inclusion of the device I/O concept. Even though it is a concept that developers do not need to know, it is very important for the internal workings of UniDA and for the configuration of the system. There are gateways that are able to automatically detect when a device is connected to them and identify it. These gateways support a variable number of devices and make use of dynamic device I/Os. Thus, when a device is connected to them, they automatically create a new device I/O for the device and configure it. Furthermore, there are gateways, known as static gateways, that have a fixed number of physical interfaces to connect passive devices and, in order to be compatible with very simple or legacy devices, they are usually unable to detect the presence of the device, requiring manual configuration.

This last case is one of the main reasons for the introduction of the device I/O concept. It provides the system and installers a way to know what kind of devices can be connected to a particular gateway.

5.5.1.2 Uniform Device Access Paradigm

The device network conceptual model will not be complete without a description of how these concepts can be operated to interact with the instrumentation network. This description is the uniform device access paradigm, which provides a set of generic operations that can be used to manage and command the different elements that populate the system, allowing client applications and other elements to access the functionality supplied by the available devices.

The uniform device access paradigm is made up of a very small set of operations that, by relying on the common conceptual model, are enough to access any device functionality defined in the ontology. It has three main device access operations complemented by a set of management ones:

- *Query a state of a device.* Every state that a device supports can be queried at any moment.

- *Write a state of a device.* Optionally, a device can support the modification of its states with a state write operation.

- *Send a command for execution.* It is possible to send commands to the devices. The commands supported by a device are defined in its metadata, that is, specified in the ontology for every device class. Once a device receives a supported command, it must act in accordance to the semantics of the command.

- *Subscribe to a device state.* Devices must send notifications when one of their states changes. Therefore, clients can subscribe to those notifications in order to receive them. These three operations are complemented with a set of management operations that will allow clients of the model (developers and applications) to find the devices and gateways available or defined in a particular instrumentation network, as well as access their descriptive information and metadata.

The concepts provided by the common conceptual model and how to operate them (the uniform device access paradigm) is the only knowledge of the instrumentation network that application developers need to have. Any implementation of UniDA will provide them with an API to manage those concepts and issue operations, completely decoupling the application logic from the particular hardware technologies used to build the instrumentation network.

Finally, in the next section we describe the third component of the UniDA framework: the distributed operation protocol that enables the interaction between the different physically distributed elements of the system.

5.5.1.3 Distributed Operation Protocol

Due to the characteristics of HIEs and their intrinsic ubiquitous nature, devices need to be physically distributed throughout the environment in order to monitor and act over it. The UniDA framework is designed to support the distributed operation of some of its elements. In particular, the elements that implement the device gateway concept can be installed distributed to build a network of devices.

The objective of the distributed operation protocol is to allow the interaction between the clients of the framework and the devices deployed throughout the environment. The protocol must support all the operational, maintenance, and management tasks required for the correct operation of the instrumentation network. These include support for:

- *Control operations.* Query and write states, send commands, subscribe to state change notifications.

- *Management operations.* Access to descriptive information about devices and gateways, discovery of devices/gateways available in the network, announcements of new devices/gateways available.

- *Maintenance operations.* Detection/notification of failures, announcements of device disconnections.

This communication protocol is defined as a set of messages that can be exchanged between the components of an instrumentation network. These messages are composed of three parts, a header, that contains metadata about the message, the content of the message, and a checksum. Therefore, the protocol is defined in an abstract way and it can be implemented by using any connectionless transport protocol.

The interchange of messages is modeled as a request/response protocol, except for the gateway announce message and the notification messages. The announce message must be multicasted to all the members of the instrumentation network, so everyone can know which gateways and devices are connected to the network. The notification messages must be multicasted to all the members of the network subscribed to them.

The announce messages do not have to be answered or acknowledged, but they must be sent periodically by every gateway, in order to alleviate possible reception problems, and because there are also used by other elements of the network to monitor the state of remote gateways and devices. The period of gateway announcements is variable, defined by each gateway according to its

requirements, and it is notified to the monitoring elements using the gateway announce messages.

Notification messages can be acknowledged, but it is not mandatory, it depends on the implementation. Every other message has an associated response (or acknowledge) message, thus, it is mandatory for the implementation of the protocol to establish some kind of reception control mechanism in order to guarantee the reception of the messages, or at least, to be able to detect communications errors and act accordingly.

5.5.2 UniDA Framework Implementation

In the previous subsections we have presented UniDA as three conceptual components, a common conceptual model of an instrumentation network, a uniform device access paradigm that models the interactions between the members of the instrumentation network, and a distributed operation protocol for the remote interaction between some of the members of the network. In the present subsection, we are going to show an actual implementation of those components.

Figure 5.15 shows a diagram of the UniDA framework implementation architecture. As can be seen, it is divided into two main components: the UniDA Library and the UniDA Gateway. The UniDA Library is an implementation of the common conceptual model, the uniform device access paradigm, and the distributed operation protocol as a Java library. The library (and its dependencies) is the only software component that a client application needs in order to command and control an instrumentation network.

The UniDA Gateway is the realization of the Device Gateway concept defined in the common conceptual model. There can be multiple implementations of the UniDA Gateway, from generic implementations that will be run in common hardware, such as computers or smartphones, to embedded implementations for specifically designed gateways. It is in the gateways where the particular control logic of each device resides, and by using some components of the UniDA library, like the distributed operation protocol, they provide access to devices to clients of the UniDA Library. That is, UniDA Gateways translate the common concepts managed by UniDA to the particular ones used by each device technology.

As can be seen, Dandelion shares many similarities with UniDA, with the FIOs performing the same translation work but at a different level of abstraction.

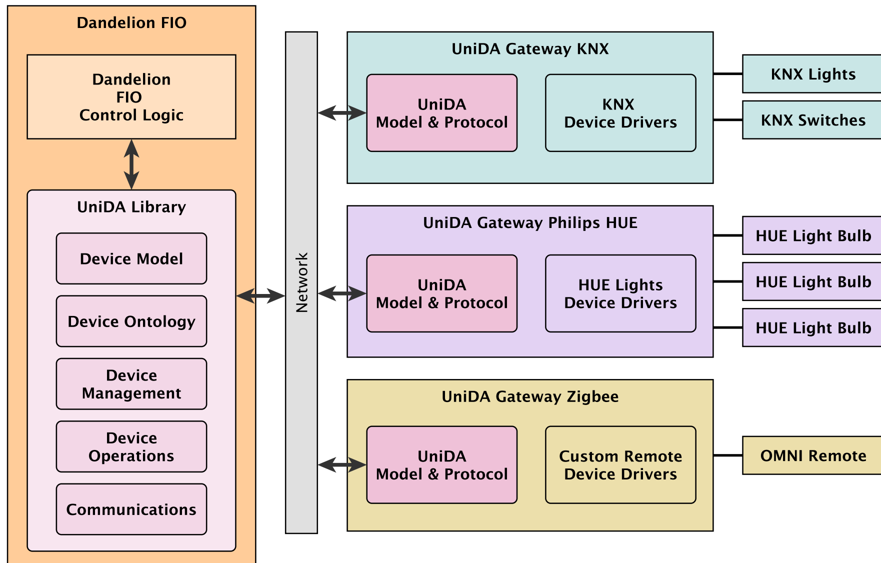


Figure 5.15: This diagram shows an architecture and deployment diagram of a system (the FIOs of Dandelion) using the UniDA Library and Gateways. The FIO logic interacts exclusively with the UniDA library through the Device Model and the Device Operations and Management. UniDA relies on a set of Gateways to translate the Device Model concepts to the particular APIs and protocols of each specific device technology.

5.5.2.1 UniDA Library

The functionality of the library is presented to external clients through a reduced set of concepts (classes of objects, as it is implemented using JAVA, an object-oriented language), directly implemented from the proposed conceptual model shown in subsection 5.5.1.1, and three façades that export the operations to manipulate those concepts in a uniform way. There exists one façade for the operational functionality of the instrumentation network, like query the state of a device or issue a command to a device, and two façades for management operations, one for device management operations and another for device gateway management operations.

As can be seen in Figure 5.15, the library is implemented as five modules that are encapsulated behind the three façades: an implementation of the common conceptual model; a management subsystem, in charge of providing access to information about the devices available in an instrumentation network; an operational subsystem, in charge of providing control capabilities over those devices; a communications subsystem, that implements the distributed oper-

ation protocol; and a device ontology module that manages the access to the DogOntontology.

The management subsystem implements two different sets of operations related to two different concepts. On one hand, it provides operations to create new devices or gateways, remove existing ones, edit their information and relationships, etc. On the other hand, it provides query functionalities to allow applications to access the information about the devices and device gateways available in an instrumentation network.

In a similar way to the management subsystem, the operational subsystem implements two different sets of functionalities. On one hand, it implements the operations required to control the devices and the instrumentation network. On the other hand, it supports the maintenance operations by implementing the functionality required to continuously monitor, in real time, the operational state of the network and update the management database through the management subsystem in order to reflect the current state of the network.

The communications subsystem implements the distributed operation protocol independently from the other subsystems, and it is divided into two components. On one hand, a message handling system, made up of a set of messages, logic to code and decode them in a binary format, and logic to manage their processing, and on the other hand, a communications channel implementing the particular logic to send and receive messages from a specific network technology. The internal division of the communications subsystem into two components allows reusing the message system with different network technologies.

Regarding the device ontology, one usual problem associated to ontologies is that their management requires lots of memory and processing power to achieve low response times. As it was of paramount importance to keep the device access library light and responsive, it was decided to use ontologies in a limited way, that is, only as a repository of metadata, instead of using them directly for storing instances of the concepts and reasoning with them. Therefore, the UniDA Library uses the DogOnt ontology exclusively as a taxonomy of the different types of devices and gateways that could exist in an installation, and as a repository of semantic metadata about those devices, their properties and their relationships.

5.5.2.2 UniDA Gateways

In UniDA, the gateways are the elements in charge of translating concepts from the common conceptual model and the uniform device access paradigm to the particular concepts and APIs required by each specific device. Their job is to control hardware devices and to interconnect them, building a kind of virtual

instrumentation network that can be accessed and managed by the clients of the UniDA Library. Therefore, gateways are highly coupled to particular devices or technologies, and they usually include specific software and hardware.

A UniDA gateway should include the particular control logic for each device connected to it, that is, the device drivers; and an implementation of the distributed operation protocol, so that it can interact with other elements of the network.

Depending on the requirements of the hardware devices, there can be many UniDA gateways deployed on very different hardware. For example, there can be gateways deployed on generic platforms, such as PCs, for devices that require complex control software, like home automation network technologies; gateways deployed on smartphones to provide access to their sensing devices; or gateways deployed on custom embedded hardware to control simple devices or build new hardware devices directly compatible with UniDA.

We have created a generic UniDA gateway software component, which can be used to build new gateways. It is implemented in Java using the J2SE API, so that it can run in any device with a J2SE virtual machine. It is a reference implementation of a UniDA gateway, with all the logic required to interact with the UniDA network and manage the processing of requests and notifications. Developers are only required to provide the particular logic to control a device, that is, the device drivers.

Using the generic UniDA gateway component, we have built gateways that allow the integration of some existing technologies into a UniDA instrumentation network. These gateways are an integral part of the UniDA framework, so they can be used by developers and installers to build their systems. By using these gateways, any application that uses the proposed framework will be directly compatible with the devices and technologies supported by these gateways, including EIB/KNX home automation networks, uPnP media renderer devices, Android smartphones, Philips HUE lights, and Belkin WeMo [Belkin-International-Inc., 2014] home automation devices.

In the next section of this chapter, we are going to show some examples of how Dandelion uses the UniDA Library and different UniDA Gateways to implement some FIOs for the OMNI and EMP examples described previously introduced in sections 4.2.1 and 4.2.2.

5.6 Demonstration Examples and Summary

In the last section of this chapter, we are going to show how two complete example systems use the Dandelion framework in order to build two distributed

physical user interfaces that are easily portable and deployable in different scenarios, with heterogeneous physical devices as IRs, and a variety of environment and user characteristics and constraints.

5.6.1 OMNI Virtual Assistant

This section provides a detailed discussion of how the OMNI Virtual Assistant system, introduced in subsection 4.2.1, has been implemented using the Dandelion Framework.

Due to the characteristics of the users of OMNI, as already indicated in subsection 4.2.1, natural user interaction is vital for the successful adoption of the system. Therefore, it is very important to use an UI adapted to each usage scenario, and in particular, in this case, adapted to the characteristics, constraints, and preferences of the user.

The OMNI system relies on the Dandelion Framework in order to reduce the costs, in time and effort, of building a Physical Distributed User Interface adaptable to the required different usage scenarios. While the main requirement of OMNI regarding the UI is adaptability, distributivity and ubiquitous interaction is a welcome addition. It allows OMNI to operate outside the users home, for example providing notifications at the homes of relatives, or even on the go.

For illustrative purposes, we have identified four representative use case scenarios that will provide a nice set of examples showing how the same abstract UI can be deployed into very different final UIs.

The remaining of this section is dedicated to describing the OMNI UI, and how it is built using Dandelion in order to support those three different interaction scenarios, without requiring any modification in the UI or system code.

Building the OMNI User Interface

Recalling the development process discussed in section 5.3, the first step to use Dandelion is to specify the system's user interface using the Abstract UI Model. As previously indicated in subsection 4.2.1, the OMNI UI is designed to be very simple and straightforward, using as few interaction primitives as possible. This interface is implemented, by default, as the remote controller device shown in Figure 4.2 and a display for outputting information. It has three main requirements:

- First, changing channels. In OMNI, the channels represent the different

Algorithm 5.6 Container skeleton of the OMNI abstract UI.

```
<aui:UserInterfaceModel>
    <aui:AbstractUIModel>
        <aui:AbstractInteractionUnit id="OmniInfo">
            <!– OMNI system information, like channel
            metadata, etc. –>
        </aui:AbstractInteractionUnit>
        <aui:AbstractInteractionUnit id="OmniControl">
            <!– OMNI controls like channel and volume
            change –>
        </aui:AbstractInteractionUnit>
        <aui:AbstractInteractionUnit id="OmniNotification">
            <!– OMNI notification dialog –>
        </aui:AbstractInteractionUnit>
    </aui:AbstractUIModel>
</aui:UserInterfaceModel>
```

functionalities of the system, so it must allow users to change from one channel to another in order to select the functionality they want.

- Second, show system information to user. For example metadata (name, etc.) of the channel selected.
- Third, answering the questions of the system. OMNI uses a proactive interaction system that prompts questions to the user. As shown in Figure 4.1, there are two types of questions, those that offer the execution of some function, and those that request some kind of confirmation of actions by the user.

The first thing when designing an Abstract UI is to organize the different interaction primitives in containers, so that we can provide the system with information about the logical relations between the different elements. For example, for the OMNI UI, we can clearly differentiate three different interaction use cases: showing system information to the user, changing channels, and answering questions. So, a good starting point would be an Abstract UI with three Abstract Interaction Units as containers, as shown in the code listing of Algorithm 5.6.

The next step is to populate these containers with the required interaction primitives. We already know from the code listing of Algorithm 5.1 in section 5.3.1, that the notification subsystem requires three interaction facets. One for outputting a message to the user, and two to receive the answer of the user, the 'yes' or 'no' actions. Regarding the control part of the system, while the most important requirement is to change channels, in fact there are two additional

Algorithm 5.7 Control part of the OMNI abstract UI. It contains the abstract interaction primitives required to power on/off the system, change the channel, and change the audio volume.

```
...
<aui:AbstractInteractionUnit id=”OmniInfo”>
    <aui:DataInputOutputFacet id=”InfoMsg”>
        <aui:dataType>text</aui:dataType>
    </aui:DataInputOutputFacet>
</aui:AbstractInteractionUnit>
<aui:AbstractInteractionUnit id="OmniControl">
    <aui:Composition rationale="Remote Control" role="groups all
    the control interactions">
        <aui:AbstractInteractionUnit id="PowerControl">
            <aui:TriggerFacet id="power">
                <aui:triggerType>operation</aui:triggerType>
            </aui:TriggerFacet>
        </aui:AbstractInteractionUnit>
        <aui:AbstractInteractionUnit id="ChannelControl">
            <aui:TriggerFacet id="next">
                <aui:triggerType>operation</aui:triggerType>
            </aui:TriggerFacet>
            <aui:TriggerFacet id="previous">
                <aui:triggerType>operation</aui:triggerType>
            </aui:TriggerFacet>
        </aui:AbstractInteractionUnit>
        <aui:AbstractInteractionUnit id="VolumeControl">
            <aui:TriggerFacet id="increase">
                <aui:triggerType>operation</aui:triggerType>
                </aui:TriggerSupport>
            <aui:TriggerFacet id="decrease">
                <aui:triggerType>operation</aui:triggerType>
            </aui:TriggerSupport>
        </aui:AbstractInteractionUnit>
    </aui:Composition>
</aui:AbstractInteractionUnit>
...
```

Algorithm 5.8 Small code snippet showing an example implementation of a user action callback (IDandelionActionCallback interface). In this case, this code corresponds to the OMNI action to change the channel to the next one available.

```
public class OmniNextChannelAction implements IDandelionActionCallback
{
    IOmniChannelManager channelManager;
    public OmniNextChannelAction(IOmniChannelManager channel-
    Manager) {
        this.channelManager = channelManager;
    }
    @Override public void notifyAction(IUser user, AbstractInteractio-
    nUnit aiu, TriggerFacet facet) {
        try {
            this.channelManager.nextChannel();
        }
        catch (ChannelErrorException ex) {
            Logging.log(Level.SEVERE, "Error while chang-
            ing channel", ex);
        }
    }
}
```

functionalities. Change the audio volume and power on/off the system. While it is possible to organize those primitives in multiple ways, we decided to logically separate them into three different AIUs, one for each block of control primitives, as shown in the code listing of Algorithm 5.7. This way, very related interaction facets, like 'previous' and 'next' channel actions, or 'increase' and 'decrease' the volume actions, are logically attached, but the groups are separated, thus indicating Dandelion to use a similar physical configuration. Furthermore, there is an additional AIU as a container of a data output facet for system information.

At this point, we have already specified the complete OMNI UI at the abstract level, so the next step would be to implement the UI control logic that will provide the behavior of the user interface by connecting the business logic to the UI.

We have three main parts to implement:

- The OMNI control part, which is basically composed of user actions and, thus, is quite easy to implement. We are only required to implement one callback function for each user action, following the example shown in the code listing of Algorithm 5.8. As can be seen, the code of each action can

Algorithm 5.9 Example of how the DUIC can be used to output information to the users. As can be seen, it is very easy to use, and it operates completely at the abstract level, without exposing the code to any particularities of the modalities and technologies of the UI.

```
...
private void showChannelInfo(IOmniChannel channel) {
    String channelInfo = channel.getChannelName()+" - "+channel.getChannelProgram();
    this.duic.showOutput(channelInfo, omniInfoAIU, infoMsgFacet, new HashSet<FuzzyVariable>(0));
}
...
private void changeChannel(int index) throws ChannelErrorException {
    ...
    if (this.currentChannel != null) this.currentChannel.close();
    this.currentChannel = nextChannel;
    this.currentChannel.open();
    this.showChannelInfo(this.currentChannel);
}
...
```

be quite simple, just redirecting the UI action to a business logic action.

- The presentation of system's information to the user, like channel names, etc. For that purpose, we have to use the DUIC interface to output a message to the user. For example, when the OMNI changes to a new TV channel, it will output a message with the channel and program name, as shown in Algorithm 5.9.

- The notification part of the UI. In this case, we have to implement two user action callbacks for the 'yes' and 'no' actions, and use the DUIC interface to output messages to the user.

As has been shown in this subsection, it is quite simple to use the DUIC to implement the connection between the abstract UI and the business logic. Furthermore, this can be completely done at the abstract level. This way, the OMNI system can be easily deployable in a variety of environments with different IRs for its final UI.

In the next subsections, we are going to describe four possible usage scenarios for the OMNI system, presenting the different FIOs required, and how the final UI can be assembled, at deploy time, for each scenario.

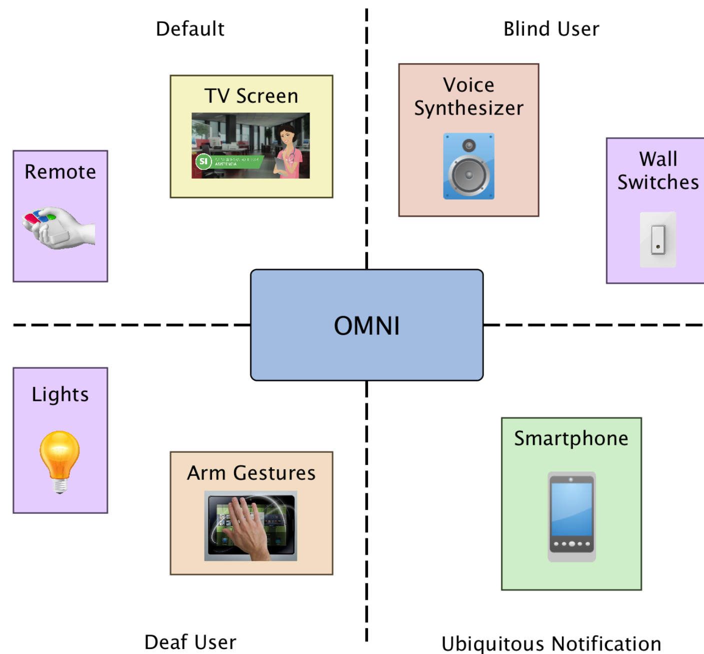


Figure 5.16: OMNI system IRs/FIOs for the different scenarios.

Default Usage Scenario

As previously indicated in subsection 4.2.1, the OMNI system uses a default user interface with a display and a remote controller. The display is used for showing system info and notifications to the user, and the remote controller is used to capture user actions.

As shown in Figure 5.16, for this particular implementation of the OMNI UI, we only need two FIOs. One of them will be in charge of presenting the messages to the user through the display, and the other will be in charge of receiving button presses from the OMNI remote controller.

The display FIO is implemented, using Java, as a software-only interaction resource. It creates a semi-translucent window and superimposes it over the OMNI channel screens during a specified amount of time. It supports only one output interaction facet with data of type string.

In order to facilitate the development of new FIOs, Dandelion provides a small development framework that only requires developers to implement a

Algorithm 5.10 Example source of the OMNI display FIO for notifying messages. It uses one implementation of the IOutputAction interface in order to support one output interaction facet.

```
//create and init the Dandelion FIO manager
IDandelionFIOManager fioManager =

        new JmsJsonDandelionFIOManager(jmsBrokerUri);

fioManager.init();

//create FIO metadata object
notificationDialog = createExtendedMetadata("Omni-Display-Dialog");

//start a new FIO with support of one OUTPUT interaction facet
gipManager = fioManager.startFIO(notificationDialog,

        new IOutputAction() {
            @Override public void doOutput(Collection<Property>
            properties, Collection<FuzzyVariable> fuzzyHints) {
                String msg = getStringProperty(properties);
                if (msg != null) {
                    display.getVideoPane().showNotificationMessage(msg,
                    true);
                }
            },
            null, null);
```

set of actions, one for each interaction facet (either output, selection or focus) supported by the FIO. For example, the code listed in Algorithm 5.10 shows the implementation of the display FIO. As it must support only one output interaction facet, it encapsulates the particular code to perform the output in an implementation of the IOutputAction, and then starts a new FIO associated to an instance of that action.

Regarding the remote controller FIO, we have used UniDA in order to make its implementation easier. Thus, a remote UniDA gateway implements the physical connection with the remote controller using Zigbee, and the FIO uses the UniDA library to interact with the gateway. This FIO does not support any interaction coming from the system to the user, it only supports the generation of actions from the user to the system, so it uses the FIO Manager in a different way than the display one. Instead of implementing any interaction facet actions, this FIO only uses the FIO Manager to publish 'action' events in the FIO topic, for which the manager provides three specific methods, one for each kind of interaction primitive (input, action and selection).

While the main idea of Dandelion is to provide as many integrated FIOs as possible, thus providing direct support for a large number of IRs, it can be seen that developing new FIOs, when necessary, can be as simple or complex as

the particular behavior requires and, moreover, the Dandelion FIO development API allows developers to focus only on one particular FIO logic.

While the default OMNI UI can be useful in many different use cases, it can be very inadequate or, at least difficult and unnatural to use, in another large number of scenarios. In the next three subsections, we are going to present three small examples of scenarios where a different UI, more adapted to the user needs, can be a better option than the default. It is important to note that, thanks to the ability of Dandelion to use multiple FIOs for the same interaction primitive, the default UI is always available to the user, the new IRs are only complementary means to interact with the system.

Blind User Scenario

Lets take, for example, a user with serious visual deficiencies. The use of the screen to output the messages is probably one of the worst options, and furthermore, the remote controller could be also a bad idea because, even if it could be easy to use, it can be difficult to find around the home.

Dandelion does not impose any restriction on how the abstract interaction facets of the AUI are associated to interaction facets from FIOs, so it is possible to associate the same abstract interaction facet to more than one FIO, thus permitting the use of IRs in a complementary and redundant way. For the case of the blind user scenario, we added two new IRs in order to complement the default UI. As shown in Figure 5.16, we added a voice synthesizer IR for outputting messages and a gesture recognizer IR to receive user actions.

The voice synthesizer FIO is implemented using the Festival Speech Synthesis System [Taylor et al., 2006], and, as the display one, it exports only one output interaction facet through its GIP interface.

Regarding the gesture recognizer FIO, it is implemented using the Open Natural Interface (OpenNI) framework for cameras with depth sensors. It exports the same GIP interface as the remote controller, supporting five action interaction facets. This FIO uses the OpenNI API to receive events of hand and arm gestures detected through the depth sensor cameras, it then translates those gestures to GIP action events, and publishes them in its FIO topic.

To facilitate the mapping of the abstract UI to the FIOs, Dandelion allows installers to specify the mapping in an XML file, so for this case, apart from the mappings of the default UI, we have to add two mappings to associate the OMNI 'InfoMsg' interaction facet and the 'NotificationMsg' to the output facet of the voice synthesizer FIO, and then seven more mappings for the different user actions required: two for the 'yes' and 'no' actions, and five for the control actions (volume selection, channel selection and power on/off). Algorithm 5.11

Algorithm 5.11 Example of mapping file for the OMNI system where the notification dialog message is associated to the OMNI display FIO, and the 'yes' and 'next channel' actions are associated to hand gestures.

<uib:AIU2FIOMapping>

<uib:AbstractInteractionUnit-ID>NotificationDialog</uib:AbstractInteractionUnit-ID> <uib:InteractionFacet-ID>NotificationMessage</uib:InteractionSupportElement-ID>
<uib:AssociatedFIO-ID>Omni-Display-Dialog</uib:AssociatedFIO-ID>
<uib:AssociatedFIO-Facet-ID>MessageOutput</uib:AssociatedFIO-ID>

</uib:AIU2FIOMapping>
<uib:AIU2FIOMapping>

<uib:AbstractInteractionUnit-ID>NotificationDialog</uib:AbstractInteractionUnit-ID>
<uib:InteractionSupportElement-ID>YesAction</uib:InteractionSupportElement-ID>
<uib:AssociatedFIO-ID>Gesture-Recognizer</uib:AssociatedFIO-ID>
<uib:AssociatedFIO-Facet-ID>Hand-Up</uib:AssociatedFIO-ID>

</uib:AIU2FIOMapping>
<uib:AIU2FIOMapping>

<uib:AbstractInteractionUnit-ID>ChannelControl</uib:AbstractInteractionUnit-ID>
<uib:InteractionSupportElement-ID>Next</uib:InteractionSupportElement-ID>
<uib:AssociatedFIO-ID>Gesture-Recognizer</uib:AssociatedFIO-ID>
<uib:AssociatedFIO-Facet-ID>Hand-Right</uib:AssociatedFIO-ID>

</uib:AIU2FIOMapping>

Algorithm 5.12 How to use interaction hints to provide some level of customization to the final UI. In this case, we are suggesting the color of the notification message.

```
HashSet<FuzzyVariable> interactionHints = new HashSet<FuzzyVariable>();
interactionHints.add(new FuzzyVariable("color", "yellow"));
this.duic.showOutput(message, omniNotificationAIU, notificationMsgFacet, interactionHints);
```

shows a small snippet of this mapping file.

Deaf User Scenario

It is worth noting that a key characteristic of the abstraction of IRs through the GIP+FIO combination is that it is possible to use, for the same interaction, a set of IRs with very different natures, modalities, and capabilities. Lets take for example the notification UI of OMNI. It is in charge of notifying agenda events to its users, so a key point is to attract their attention to the system UI when a message is shown. The speech synthesizing IR is a good option in combination with the screen, but lets imagine a user with auditive deficiencies. If she does not have the screen in view, she may miss the notification. One simple option would be to use colored lights spread through the home to signal the presence of notification messages. By using the UniDA library, we have implemented a simple FIO that uses colored lights to output messages. It exports only one output interaction facet, the same as the notification display FIO, but instead of rendering the output as a message on a screen, or by speech synthesis, it lights up a bulb to notify the presence of a message.

This FIO accepts an Interaction Hint (IH), where the developer can specify the color. As previously introduced, IHs are a mechanism to customize the generic interactions provided by the FIOs, and each FIO implementation has the right to use them or ignore them. In this case, the light bulb FIO accepts the specification of its color by using an IH as shown in the code listing of Algorithm 5.12.

Ubiquitous Notification Scenario

The colored light bulbs to notify the presence of notification messages looks like a nice way to keep the user notified, even if she is not in the living room, but what happens if the user is not at home?. One nice possibility would be to show her the notification messages using a portable device, for example a smartphone.

We have built a FIO as an Android application. It exports two output

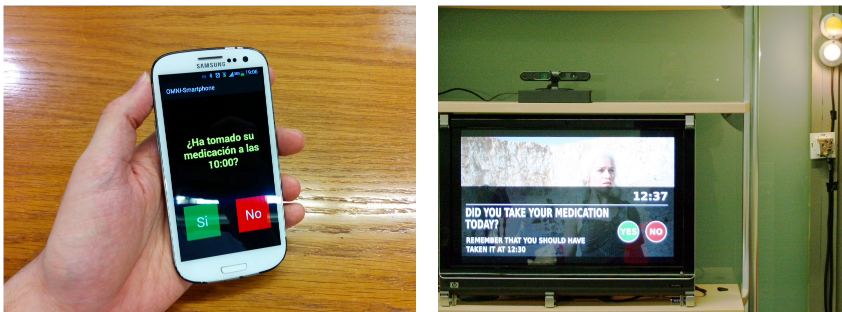


Figure 5.17: Two examples of FIOs for the OMNI notification user interface. First example: a smartphone with two output interaction facets, one using the screen, and one using voice synthesizing, and two action facets, for the yes/no actions. Second example with three FIOs:a TV display, with one output interaction facet using the screen; a voice synthesizing FIO using a Festival server; and a notification FIO that uses colored lights to notify the presence of a message.

facets through its GIP interface. One of them uses the screen to display a message, and the other one uses speech synthesis. This way, it is possible to configure the ubiquitous notification in three different ways:

- Only use the screen.
- Only use speech synthesis.
- Use both, the screen and speech synthesis.

Furthermore, this FIO accepts some IHs to customize the output, like color, volume or vibration strength, and it also exports two action interaction facets in order to allow the user to answer the notifications with a 'yes' or a 'no' using on screen buttons or the volume rocker buttons.

5.6.2 Environmental Music Player

The Environmental Music Player (EMP), described in subsection 4.2.2, is an ubiquitous music player system that follows the user, as she moves from one place to another, while playing music and providing the user with a UI to control the playing.

The EMP UI, as shown in the sketch of Figure 4.3, is a more complex UI than the OMNI one, requiring many more interaction primitives, in number and type. Therefore, for the illustration purpose of the example, and in order to keep the description short, we are going to explore a couple of particular

Algorithm 5.13 Code snippet of the EMP abstract UI dedicated to the selection of music style.

```
...
<aui:AbstractInteractionUnit id="MusicStyleControl">
    <aui:DataSelectionFacet id="MusicStyleSelector" isContinuous="false" selectionType="SINGLE">
        <aui:dataType>text</aui:dataType>
    </aui:DataSelectionFacet>
</aui:AbstractInteractionUnit>
...
```

aspects of the UI that are different from the OMNI UI. Particularly, in the next subsections, we are going to show in detail how to implement two interaction facets that are required by the EMP and not by the OMNI UI:

- *Selection.* We are going to explain how a selection of data can be implemented in the abstract UI, and also how it can be implemented in the final UI using different FIOs for diverse scenarios.
- *Output of images.* The EMP requires displaying the album art to the user. We are going to show how this interaction can be implemented from the abstract UI point of view, as well as how different FIOs support it in a variety of scenarios.

Selection of the Music Style

The Environmental Music Player does not play particular songs or albums. Instead, it allows the user to select a specific music style, and then it creates a playlist with all the songs, available in the user library, that match the selected style.

As we previously mentioned in section 5.3.1, the music styles available for selection are dynamic and dependent on the collection of the user. Because of that, the selection of music styles must be modeled using a selection interaction primitive dynamically populated from the UI control logic. The code snippet presented in Algorithm 5.13 shows the part of the EMP abstract UI dedicated to define the selection interaction facet required for music style selection.

Once the abstract UI is defined, we must implement the UI control logic required to show the list of music styles to the user, and register a callback to receive the style selected by the user when she changes the selection. The code listed in Algorithms 5.14 and 5.15 show, respectively, these two aspects of the EMP user interface control logic. First, the list of music styles is retrieved from the music collection, and then, using the DUIC, it is shown to the user.

Algorithm 5.14 Code snippet of the EMP UI control logic to show the dynamic list of music styles as a selection

```
private static void showMusicStyleList(
    ApplicationMetadata app, IUserInterfaceController dandelionUIC,
    MusicPlayer musicPlayer)
{
    AbstractInteractionUnit musicStyleControl =
    app.getAbstractUI().getAbstractInteractionUnitById("MusicStyleControl");
    DataSelectionSupport styleSelectionFacet =
        (DataSelectionSupport) musicStyleCon-
        trol.getInteractionSupportElementById("MusicStyleSelector");
    Collection<String> allAudioGenres = music-
    Player.getMusicLibrary().getAllAudioGenres();
    ArrayList<Property> audioGenresList = new Ar-
    rayList<Property>(allAudioGenres.size());
    for(String genre : allAudioGenres) {
        audioGenresList.add(new Prop-
        erty(PropertyType.stringProperty, null, genre));
    }
    Property defaultGenre = audioGenresList.get(0);
    dandelionUIC.showSelection(defaultGenre, audioGenresList, music-
    StyleControl, styleSelectionFacet, new HashSet<FuzzyVariable>());
}
```

Finally, a selection callback is implemented and registered to receive selection GIP events from the FIOs that implement the final UI.

As can be seen, like in the case of the OMNI UI, the definition of the UI and its particular behavior (UI control logic) has been performed completely at the abstract level, without any knowledge about how the selection of the music style is going to be finally implemented in the different usage scenarios.

The music style selection interaction facet can be implemented in multiple ways using different FIOs to build various final UIs adapted to diverse usage scenarios. Figure 5.18 shows three different ways of physically implementing the selection of music styles.

First, a FIO implemented as an Android application for smartphones allows the selection of the music style using the touch screen of a smartphone. It exports a simple GIP interface supporting only two interaction facets, selection and focus. This kind of selection FIO can be useful in many situations, it can be used to control de music style on-the-go, using a personal device, but it can also be useful at home.

Second, a cube-shaped device with different colors in each face. Each color represents a music style, and the color that is facing upwards represents the selected style. This FIO is implemented using an Intel Edison development

Algorithm 5.15 EMP implementation of the selection callback to change the music style.

```
public class MusicStyleSelectionAction implements IDandelionSelectionCallback
{
    ...
    @Override public void notifySelection( Property styleSelected, Collection<Property> styleList, IUser user, AbstractInteractionUnit aiu, DataSelectionSupport datass)
    {
        //is the music style selection facet
        if (this.musicStyleSelection == datass) {
            //the FIO returned a string?
            if (styleSelected.getType() == PropertyType.stringProperty) {
                //change the music style
                this.musicPlayer.setMusicStyle(styleSelected.getValue());
            }
        }
    }
}
```

board, a 3-axis accelerometer, and a battery. It is implemented in JAVA, and it exports a GIP interface supporting only a selection interaction facet. The number of available styles for selection is limited to six (the sides of the cube), so, if there are more than six, only the first six styles will be available for selection. This is an example of a restriction that a particular FIO can impose to the interaction facets it supports. This kind of FIO can be useful, for example, for kids or seniors with low ICT knowledge.

Finally, the selection can also be controlled using hand and finger gestures. For example, the number of fingers pointing can be used to select among five different music styles. This FIO is implemented using a Leap Motion controller and the Java language. It limits the maximum number of selection to five, and it exports only one selection facet.

Output of the Album Art

Dandelion not only allows the input and output of basic data types like numbers or strings, but it also allows the input and output of images in PNG format. In this subsection, we are going to present a small example of how Dandelion can be used to show the album art image of the EMP UI, while keeping the UI decoupled from its final implementation.

For the definition of the abstract UI, as with any other output interaction primitive, we need an AIU and a DataInputOutputFacet. The only difference

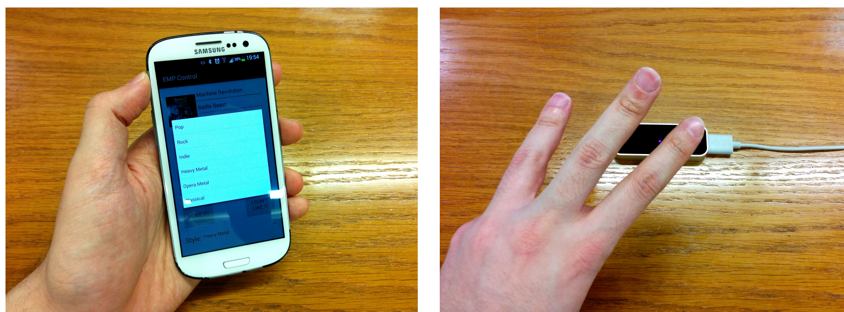


Figure 5.18: Two different possibilities for the final implementation of the music style selection interaction primitive. First, a touch UI using a smartphone, useful for mobile and outdoor environments. Second, a leap motion controller allows the selection of the style using hand and finger gestures.

Algorithm 5.16 Example of abstract UI code defining an output interaction primitive for images.

```
<aui:AbstractInteractionUnit id="AlbumMetadata">
    <aui:DataInputOutputSupport id="AlbumArt" dataFormat="image" inputSupport="false" outputSupport="true">
        <aui:dataType>image</aui:dataType>
    </aui:DataInputOutputSupport>
    ...
</aui:AbstractInteractionUnit>
```

is the data-type of the facet, that will be 'image', as shown in the code listed in Algorithm 5.16.

The UI logic required to manage images requires a little bit more effort than strings or other basic datatypes, but as shown in Algorithm 5.17, it remains quite simple. In the case of the EMP example, we retrieve the Album Art image from the music database. Next, we convert it to PNG format, because we have selected it as the standard way to interchange images using the GIP. Finally, we create an ImageProperty object by providing the RAW data of the image, that will codified in base64 to send it using STOMP and JSON. The usual Dandelion UIC API is used to perform the output interaction, as we would do in any other case. Therefore, the main difference is the necessity to convert the image to PNG, something that could be encapsulated in the Dandelion API, but we have decided to let it outside, so that the API is not coupled to particular Image APIs, that would make the JAVA implementation of Dandelion incompatible with Android or other Java platforms.

Algorithm 5.17 Example of how to use Dandelion to output an image. The image data must be first converted to PNG format. Dandelion automatically encodes the raw data of the image in base64 to send it to the FIOs using the GIP.

```
BufferedImage albumArtImage = song.getAlbumArt();
ByteArrayOutputStream bos = new ByteArrayOutputStream();
ImageIO.write(albumArtImage, "png", bos);
ImageProperty albumArtProperty =
    new ImageProperty("", bos.toByteArray());
dandelionUIC.showOutput(
    albumArtProperty, albumMetadataAUI,
    albumCoverFacet, new HashSet<FuzzyVariable>());
```

The implementation of a FIO that uses images as output or input is also quite easy. The developer only has to perform the inverse process. She will receive an ImageProperty with the raw data of the image in PNG format, and she have to convert it to an adequate data type in the receiving platform. An example is shown in Algorithm 5.18, that displays the code required to implement a FIO that shows the Album Art to the user in an Android application.

5.6.3 Summary

In this chapter we have presented a detailed description of the Dandelion framework architecture, along with a description of how it realizes the IMA+ILA abstraction levels proposed by the TIAF. Dandelion can be considered as a reference implementation of the TIAF conceptual proposals, providing developers of AmI UIs with the following key characteristics:

- Design and specify the User Interfaces of AmI systems from an abstract point of view, completely decoupled from the modalities and technologies of the set of heterogeneous IRs that will build the UI.
- Design and implement the UI control logic of their systems isolated from the APIs and particularities of the IRs.
- Assemble, at deploy-time, multiple different final UIs for the same system and UI control logic.
- Build Distributed Physical User Interfaces that are isolated from the physical location and protocols of the different IRs.

Algorithm 5.18 Example of output action of a FIO that uses an Android smartphone screen to show the album art of the EMP to the user.

```
IOutputAction outputAlbumArt = new IOutputAction()
{
    public String getId() { return "art";
}
public void doOutput(
    Collection<Property> properties,
    Collection<FuzzyVariable> fuzzyHints)
{
if (properties.size() >= 1) {
    final Property out = properties.iterator().next();
    artistTitle.post(new Runnable() { public void run() {
        if (out.getType() == PropertyType.imageProperty) {
            ImageProperty image = (ImageProperty) out;
              byte[] imgData = image.getImageData();
                Bitmap bitmap = BitmapFac-
                tory.decodeByteArray(imgData, 0,
                imgData.length);
                albumArt.setImageBitmap(bitmap);
              }
            } });
}
} };
```

Finally, in section 5.6, this capabilities have also been demonstrated with the exploration of two complete application examples that use Dandelion to build Physical Distributed UIs in Ambient Intelligence Environments.

Furthermore, in this chapter we have also presented the UniDA technology. A device abstraction technology developed within this thesis that facilitates the development of applications requiring the use of heterogeneous physical devices. This technology is used in Dandelion in order to greatly increment the compatibility of the framework with a large number of different devices, especially in the field of home automation and the Internet of Things. The implementation of the UniDA technology produced within this thesis has been released as open source software, thus provided developers and researchers with a directly usable solution for the development of applications using heterogenous devices.

Chapter 6

Adding Real-Time, Autonomous and Dynamic Adaptation to Physical User Interfaces

"Adapt or perish, now as ever, is nature's inexorable imperative."

H. G. Wells

6.1 Introduction

The previous chapter was dedicated to describing the Dandelion framework and, in particular, its implementation of the IMA and ILA abstraction layers proposed by the TIAF. In this chapter, we are going to introduce the Dandelion implementation of the ICA abstraction layer.

The goal of the Interaction Context Abstraction (ICA) layer is to decouple the system and developers from the specific characteristics of the context and, in particular, from the characteristics of the usage scenario, including the user, the environment, and the usage situation.

With the IMA and ILA, developers are able to implement a physical UI without knowledge about the modalities, APIs, and protocols of the underlying interaction resources. Therefore, they are decoupled from the IRs at

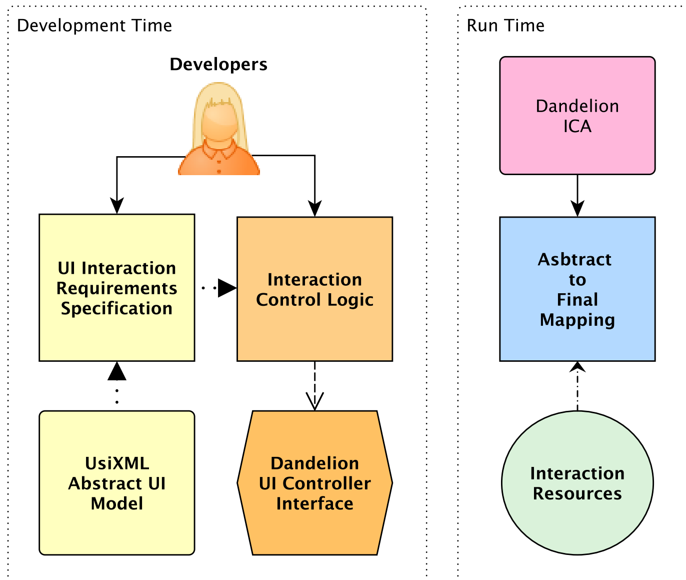


Figure 6.1: The ICA allows Dandelion to autonomously select, at run-time, which set of FIOs to use for each usage scenario. This way, Dandelion is able to react to context changes at run-time, modifying the Final UI to keep it operating within the natural interaction constraints required by Ambient Intelligence UIs.

development-time. Nevertheless, they are exposed to the IR characteristics at deployment-time, because either the developers or the installers are required to select the most appropriate IRs for each particular scenario. Furthermore, with the implementation of Dandelion introduced in chapter 5, the adaptation of UI to the context is performed manually by the installer, thus, while the UIs are more easily portable, they are not able to adapt to changes in the usage scenario.

The main idea behind the ICA layer is to change the Dandelion UI development process explained in section 5.3 and Figure 5.3, where the installer selects the FIOs to build the final UI, to the development process shown in Figure 6.1, where Dandelion itself, using the ICA, selects at run-time the most adequate FIOs for each usage scenario. This way, Dandelion will be in charge of managing, autonomously, the mapping between abstract interaction facets and FIOs.

The Dandelion implementation of the ICA is built on top of two main elements. On the one hand, a set of models describing the scenario (context). On the other hand, a set of Computational Intelligence algorithms to select, among the IRs available, those that better match the requirements of the UI,

the user, and the environment. The context models provide developers with a way to specify the characteristics of each usage scenario, while the FIO selection algorithms provide the system with a mechanism to decide which FIO is better for each particular scenario.

In this chapter, we are going to describe in detail how the ICA is implemented in the Dandelion framework. We are going to start by providing an overview of the whole physical UI adaptation to context process in section 6.2. Then, we are going to continue with two sections to explain the two main components of the ICA, the context models in section 6.3, and the autonomous FIO selection process in section 6.4. Finally, we are going to end the chapter with the presentation, in section 6.5, of three examples of how Dandelion can be used to implement distributed physical user interfaces capable of adapting to changes, at run-time, in the usage scenario.

6.2 Physical UI Adaptation to Context

The combination of the IMA and ILA abstraction layers allows developers to achieve a high level of decoupling between the system/UI logic, and the final shape of the UI. This feature facilitates the development of easily portable DPUIs, but, in addition, it is a key enabler for the implementation of a mechanism to autonomously adapt the user interface to changes in the context.

As previously shown in chapter 5, thanks to the features provided by the IMA and ILA abstraction layers, implementing a concrete Distributed Physical UI is reduced to performing a selection of which particular IRs, among those available, are going to realize each abstract interaction requirement specified by the abstract UI. Therefore, the adaptation of PUIs to context is reduced to managing the mappings between abstract interaction facets and Final Interaction Objects. This feature introduces two main characteristics that facilitate the implementation of the ICA. First, Dandelion is already designed and prepared to easily change, at run-time, the IRs used for a particular UI implementation. And second, the process of managing and modifying the shape of the final UI can be easily decoupled, as shown in 5.2, where the ICA is displayed as a separate box with only one connection to the rest of Dandelion, required to modify the FIO to AIU mapping.

As a consequence, the final goal of the Dandelion ICA implementation is to manage, at run-time and in an autonomous way, the mappings between FIOs and the Abstract UI. This means that the ICA will be in charge of autonomously managing the selection of FIOs, choosing those that provide the best possible natural user interaction experience in a particular usage scenario and that better

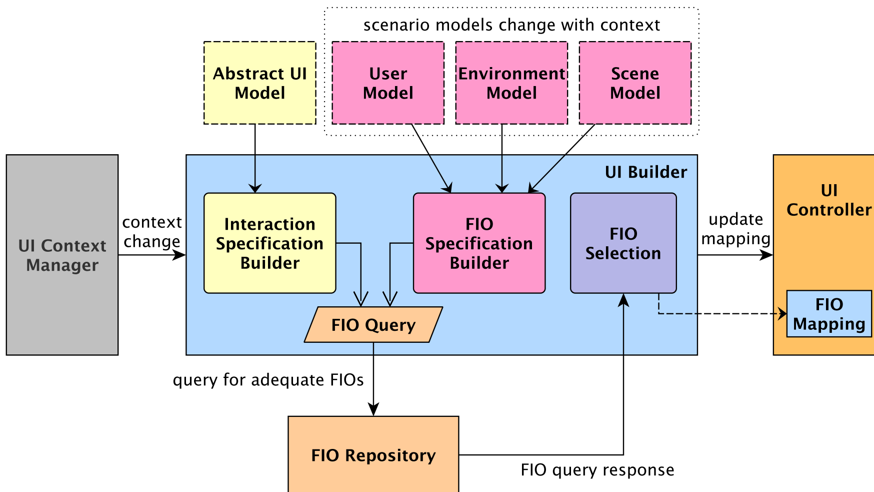


Figure 6.2: Overview of the Dandelion UI adaptation to context process. The UIB creates a query to ask the FIO repository for a list of FIOs that comply with the requirements of the usage scenario. This list is used to configure, in the UI controller, the mappings between FIOs and abstract UI elements.

match the interaction requirements of the Abstract UI.

Figure 6.2 displays a block diagram showing an overview of how Dandelion implements the autonomous FIO selection process. The main component in charge of implementing the ICA is the User Interface Builder (UIB), which receives context change events, and reacts to them by performing a new selection of FIOs and modifying the mapping in the UI Controller.

The context change events are generated by external sources to Dandelion, the UIB just creates a topic in the messaging broker and subscribes itself to the topic in order to receive context change events published by other components. This way, Dandelion is decoupled from the monitoring of the context, and it is the responsibility of the system developer to provide a way to detect and notify context changes. As we will see in more detail in section 6.5, for illustrative purposes, we have implemented a UI Context Manager application that allows the users to select, from their smartphones, in which place of the HIE they are located, thus triggering the adaptation of the system to that scenario.

Once a context change event is received by the UIB, it starts the process of adapting the UI to the context. This process is driven by a series of context models that provide the UIB with information about the scenario in which the UI is being used. As introduced in section 4.3.2.6, the TIAF does not imposes any limitation on the type and number of context models used to im-

plement the ICA, nevertheless, it suggests having, at least, context information about three different topics: user characteristics and preferences, environment characteristics, and situation characteristics. Dandelion provides reference implementations for three models covering those different topics, and, as displayed in Figure 6.2, it uses those three models and the abstract UI model to feed the FIO selection algorithms with context information. This is one possible implementation of the ICA layer proposed by the TIAF, but it would be possible to implement it in different ways using a different type and number of models and selection algorithms. In fact, given that in Dandelion the ICA is very decoupled from the rest of the framework, it would also be easy to use different implementations of the ICA (i. e. the User Interface Builder) for distinct application fields.

The UIB uses a series of Computational Intelligence (CI) algorithms to exploit the context information provided by the models in order to select the most adequate FIOs for each usage scenario. In particular, it uses fuzzy inference systems to implement many of the phases involved in the process.

First, when the FIO selection process is started, the UIB uses the information provided by the abstract UI model and the context models to generate a FIO specification for each abstract interaction facet required by the abstract UI. This specification describes the desirable characteristics of a FIO that match the interaction requirements of the AUI, and that should be adequate to the user, environment, and situation characteristics. This specification is generated by a set of fuzzy inference systems (FIS), and it is described using fuzzy variables. Therefore, it is a fuzzy description of the characteristics an IR should have to be adequate for the UI in that particular context.

Second, those FIO fuzzy specifications are used by the UIB to create queries for the FIO Repository. This queries allow the UIB to ask the FIO Repository for FIOs that are compatible with the specifications. The repository uses a series of similarity metrics to compare the fuzzy specifications provided by the UIB with the descriptions of the FIOs available in a location, and finally, it answers the query with a list of FIOs sorted by their similarity with the specification provided.

Third, the list of FIOs returned by the repository is processed by the UIB in order to select only one FIO, the one that will be notified to the UI Controller in order to establish an adequate mapping between a user interaction facet and the FIO.

In the next two sections of this chapter, we are going to describe this process in more detail, starting with a description of the different context models in section 6.3 and a thorough description of the FIO selection process and its algorithms in section 6.4.

6.3 Context Models

As previously indicated, the user interface builder relies on a set of models of the context to obtain the information needed to autonomously perform the selection of the FIOs that better match the requirements of each usage scenario. As a reference implementation of the TIAF, in Dandelion, we propose the utilization of three different models of the context, one model for each kind of context information proposed by the TIAF:

- *User Profile Model* (UPM). One of the main requirements of Ambient Intelligence UIs is to be considered as natural user interfaces. For that purpose, it is essential to bear in mind the characteristics, preferences, and abilities of the user when building the final UI. This User Profile Model provides Dandelion with information about the user's capacities, including their physical characteristics and their abilities, like vision, hearing, cognitive, or pyschomotor abitilies.

- *Environment Profile Model* (EPM). The same Physical UI will not be perceived as natural and easy to use in every environment. It is not the same to interact with a system in a car than at home in the living room. Therefore, in order to build NUIs, environmental characteristics like noise, movement, visibility, or space, must be taken into account. The Environment Profile Model provides this kind of information for each of the physical environments included in the user's HIE.

- *Scene Profile Model* (SPM). The situation or circumstances in which the interaction is taking place and the task that the user is performing can affect, in a major way, the form in which the interaction between the user and the system takes place. In a similar way to the case of the different environments, the natural perception of a UI will be different if the user is working at the office, resting at home in the living room, or practicing sports outdoors. The Scene Profile Model provides this information, including the kind of activity that she is performing, the kind of place where she is interacting, or the number of simultaneous users.

For the definition of these three models, we have relied mainly on one prominent reference, the work carried by the MyUI project in the definition of requirements for user interface adaptation [Edlin-White et al., 2010, Wolf et al., 2011, Peissner et al., 2011]. The MyUI project explores the development of user adaptive multimodal UIs for elderly people. As a consequence, their work is quite relevant for the purposes of this PhD, because elderly people is one of the main focus groups of Ambient Intelligence, and Physical User Interfaces are eminently multimodal. Therefore, the MyUI requirements definition seems a

good starting point to discover the different information that a PUI context-adaptation system requires to do its job.

While it is possible to imagine many more models supporting other types of context information, we think that those three models provide the essential information required to demonstrate the objective of this thesis, which is to allow the implementation of multimodal Physical User Interfaces for AmI systems capable of adapting to context changes at run-time. Furthermore, it is important to note that those models must be created by the installer or administrator of the system. Therefore, in order to facilitate their job, it is also essential to keep them simple and small in number.

In the next three subsections, we are going to describe in detail the three proposed models. Finally, in section 6.5, we are going to show some usage examples of how those models can be enough to provide an UI customized for the user and usage scenario.

6.3.1 User Profile Model

The job of the User Profile Model (UPM) is to provide the UI builder with information about the variability of characteristics that affect a person's capacity to interact with a system, so that Dandelion can build an UI adapted to the user characteristics and abilities.

As previously introduced, the main goal of Ambient Intelligence systems is to facilitate the life of its users, assisting and helping them in their daily life. As a consequence, one prominent field of application of AmI technologies is elderly and child care. These two groups, and in particular the elderly, are characterized by a high diversity of abilities and capacities. This variability of user characteristics hinder the design and development of natural user interfaces for those users, but, at the same time, these groups of users usually have more difficulties using information technologies, thus they tend to be who benefit most from the utilization of natural user interfaces implemented with physical user interfaces.

In order to design an UPM adequate to these groups of users, we have used the statistical study of [Brault, 2012] and the work of MyUI [Edlin-White et al., 2010, Wolf et al., 2011, Peissner et al., 2011] in user interface adaptation as our main references for the development of the UPM.

Looking at the results the statistical study of Brault [Brault, 2012], which presents the results of a report on disability prevalence across the USA, it is easy to see how disabilities affect a large number of people considered inside the group of elderly users. According to the study, in 2010, 49.8% of americans

aged 65 and older had some kind of disability and 36.6% a severe disability. But it is also important to note that 16.6% of americans between 21 and 64 have a disability and 11.4% a severe disability. Therefore, while it is clear that the disability rate increases with age and affects a large percentage of elderly people, we cannot ignore that, with a 16.6% rate in people of working age, there is a large portion of the population that is affected by some kind of disability which can hinder their capacity to interact with IT systems.

This statistical study also highlights the large diversity of disabilities that affect the elderly to varying degrees. Among the people aged 65 and older, the study found that categorizing the disabilities in three domains, communicative, physical, and mental, there is a 28.7% of people with disabilities in one domain (2.0% communicative, 26.0 physical and 0.7% mental), 16.4% with disabilities in two domains (12.3% in C+P, 0.3% in C+M and 3.9% in P+M), and 4.3% of people with disabilities in all three domains. Furthermore, even inside those three domains, the diversity of disabilities can affect the ability of the users to interact with the system in variety of ways. For example, the study found that among people aged 65 and older:

- 17.8% have a disability related to seeing, hearing, or speaking, with 13.5% having some kind of seeing difficulty (9.8% a severe one), and 10.8% having a hearing difficulty.
- 39.4% have difficulties walking or using stairs.
- 23.8% have some kind of physical disability, with 21.2% having difficulties for lifting objects, and 7.4% having difficulties grasping objects.
- 12.0% have difficulties performing activities of daily living, like getting into bed (7.8%), dressing (5.5%), or eating (2.4%).

As can be seen, there is a high diversity of disabilities affecting the elderly, with seeing, hearing, and motor disabilities among the most prevalent.

In [Edlin-White et al., 2010], the authors provide a thorough revision of standards and guidelines for the development of UIs for older people, people with disabilities, and stroke patients. Furthermore, the work also analyzes several user studies with the objective of identifying how the aging affects the capacity of people to interact with IT systems. They highlight one prominent source for the identification of user requirements in UI adaptation for the elderly, the ISO/TR 22411 "Ergonomics Data and Guidelines for the application of ISO/IEC Guide 71 to products and services to address the needs of older persons and persons with disabilities" [Bristish-Standards-Institute, 2008], and based on it, they propose in [Wolf et al., 2011] a user profile ontology with information

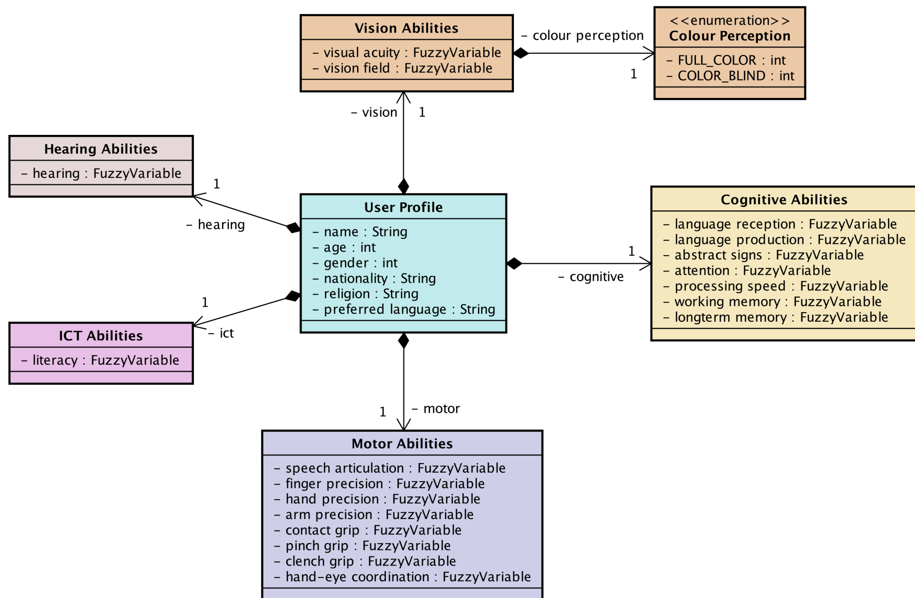


Figure 6.3: UML class diagram showing the different concepts, and their relations, in the Dandelion User Profile Model.

organized into three different areas inspired by the three disability domains proposed by Brault in a previous work [Brault, 2008]:

- *Perceptual.* Including user seeing and hearing abilities, but also environment characteristics like light conditions.
- *Cognitive.* Including language production and understanding, memory characteristics, and Information and Communication Technology (ICT) literacy.
- *Motor.* Including hand, finger or arm articulation, grip and lifting abilities, etc.

Figure 6.3 shows a conceptual class diagram of the Dandelion User Profile Model (UPM), which is directly inspired by the ISO/TR 22411 guideline [Bristish-Standards-Institute, 2008] and the user profile ontology proposed by [Wolf et al., 2011]. The main difference is that, in the UPM, we have extracted from the user model some environmental information (like ambient lighting and noise) and disaggregated the three areas into five, more specific, aspects. Furthermore, we have included additional personal data about the users, like age, religion or sex.

Regarding the reorganization of the model and the extraction of environ-

ment information, in TIAF and Dandelion we propose the use of two additional models with information about the environment and the usage situation, thus the environmental information will be covered by those models. In regards to the additional personal information of the users, as Dandelion is not only limited to elderly people, the age can be an important factor while performing the FIO selection, because, for example, it may not select the same physical device to interact with a kid than with an adult or an old man. Furthermore, cultural aspects can also affect the selection in many ways, for example, different icons are used in different countries to represent the same concepts.

The proposed User Profile Model divides the user information into five different aspects, with each aspect covering a specific set of user interaction abilities. It uses fuzzy variables to measure the difficulties the user has using a particular ability. This variables are represented by a double number where 10.0 means normal ability, and 0.0 means a severe disability.

The next list provides an explanation for each one of the different aspects and abilities covered by the model:

- *User Profile*. The main element of the model. It includes personal information of the user and links to abilities information for the different interaction abilities covered by the model.
  - *Name*. The name of the user.
  - *Age*. The age of the user.
  - *Gender*. The gender of the user.
  - *Nationality*. The country of birth of the user.
  - *Religion*. The religion followed by the user.
  - *Preferred language*. The main language of the user.
- *Vision Abilities*. Covers the seeing abilities of the user.
  - *Visual acuity*. The ability of the user to perceive the objects shown to her, including physical objects and objects displayed in screens.
  - *Visual field*. Ability to perceive without limitations in certain areas.
- *Hearing Abilities*. Covers the hearing abilities of the user.
  - *Hearing*. Ability of the user to hear sounds.
- *Cognitive Abilities*. Covers mental abilities of the user like the usage of language and memory.
  - *Language reception*. The ability to understand language, either written or spoken.

   - *Language production*. The ability to produce language, either written or spoken.
   - *Abstract signs*. Ability to comprehend icons and symbols.
   - *Attention*. Global measure of the attention abilities of the user.
   - *Processing speed*. The ability to process information in an adequate time.
   - *Working memory*. Ability of the user to use short-term memory, i. e. remembering sequences of steps during a short period of time.
   - *Longterm memory*. Ability of the user to learn and store new information.

- *Motor Abilities*. Covers psychomotor abilities of the user that can affect interaction with ICT systems, like the ability of moving hands and fingers.
   - *Speech articulation*. Ability of the user to express herself using speech.
   - *Finger precision*. Ability to move the fingers of the hand.
   - *Hand precision*. Ability to move the hands.
   - *Arm precision*. Ability to move the arms.
   - *Contact grip*. Ability to interact by touching.
   - *Pinch grip*. Ability to pick up, grasp, and manipulate objects with the fingers of one hand.
   - *Clench grip*. Ability to wrap objects with all the fingers.
   - *Hand-Eye coordination*. Ability to move hands according to visual feedback.

- *ICT Abilities*. Skills using ICT systems.
   - *Literacy*. Measure of the skills and experience using ICT systems.

The Dandelion User Profile Model must be considered as a reference implementation of a user model. It includes the essential information about the abilities of the users that can affect their interaction with a Physical User Interface.

6.3.2 Environment Profile Model

While the job of the User Profile Model is to model the variability of user abilities that can affect the interaction with the system, the job of the Environment Profile Model (EPM) is to model the variability of environment characteristics that can affect the interaction between users and a Physical User Interface.

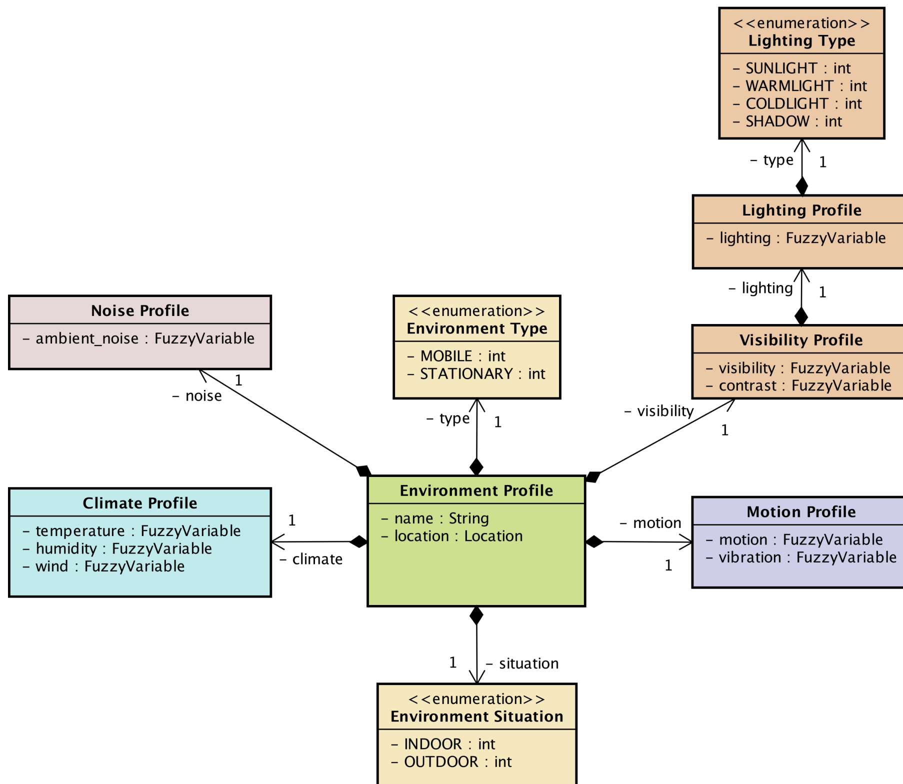


Figure 6.4: UML class diagram displaying the different concepts, and their relations, that make up the Dandelion Environment Profile Model.

The EPM is based in the analysis of environmental variability presented by Edlin-White et al. in their work on requirements analysis for UI adaptation [Edlin-White et al., 2010].

They identify five different areas of environmental variability: Lighting conditions, noise and sound, vibration and motion issues, climate issues, and space. As can be seen in Figure 6.4, the Environment Profile Model follows the same style of the UPM, a central concept associated to four different aspects representing the first four areas identified by MyUI. The space characteristics are described by the location attribute of the Environment Profile concept. Furthermore, the model includes two additional aspects that specify supplementary information about the environment, like the type (mobile or stationary) and its situation (indoors or outdoors).

The next list provide a description for every environment characteristic covered by the EPM:

- *Environment Profile.* In Dandelion, an environment represents any delimited physical location that is included in an HIE.
  - *Name*. Descriptive name of the environment.
  - *Location*. Reference of the location inside the HIE.
- *Visibility Profile.* Covers the characteristics of the environment that affect the capacity of the user to view the objects of the environment.
  - *Visibility*. An indication of the visibility quality inside the environment.
  - *Contrast*. An indication of the lighting contrast (high or low).
  - *Lighting*. A measure of the kind and quantity of light of the environment.
- *Noise Profile.* Information about the noise characteristics of the physical environment.
  - *Ambient noise*. A measure of the background noise of the environment.
- *Motion Profile.* Covers the movement characteristics of the environment.
  - *Motion*. A measure of whether the environment is in movement or not.
  - *Vibration*. An indication of the vibration level of the environment.
- *Climate Profile.* Weather information about the environment.
  - *Temperature*. The temperature of the place.
  - *Humidity*. Humidity level of the environment.
  - *Wind*. A measure of the wind force in the environment.

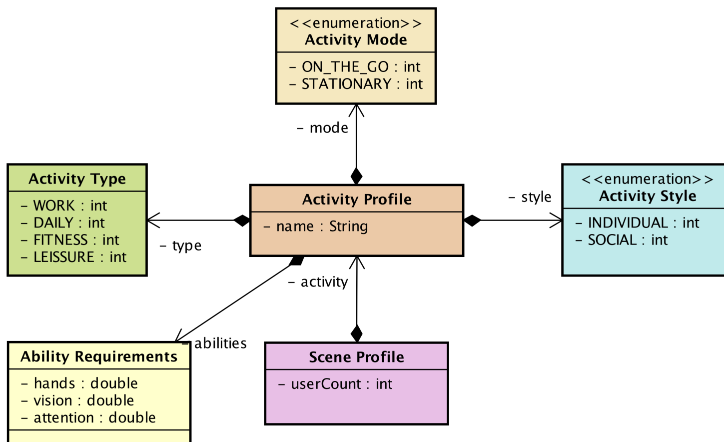


Figure 6.5: UML class diagram showing the concepts, and their relations, of the Dandelion Scene Profile Model.

- *Environment Type.* Whether it is an stationary environment (home, office, etc.) or a mobile one (car, train, etc.).
- *Environment Situation.* Physical situation of the environment (indoor, outdoor).

6.3.3 Scene Profile Model

Apart form modeling the user and environment variability, the Threefold Interaction Framework proposes the utilization of a scene or situation model to provide the adaptation system with information about the current usage scene of the UI. The Scene Profile Model (SPM) is in charge of modeling the variability of usage situations that can affect the interaction between the users and the system. As can be seen in Figure 6.5, this model covers mainly information regarding the task that the user is performing and in what kind of situations she is performing that task. This kind of information may permit the system to adjust the user interface depending on the task and objective of the user, for example, using different approaches for work tasks than leisure ones.

The next list provide a description for every characteristic covered by the SPM:

- *Scene Profile.* The main concept of the SPM. It includes the links to the different aspects describing the scene.
  - *User Count.* The number of simultaneous users of the user interface.

- *Activity Profile.* Information about the activity or task the user is performing.
- *Activity Type.* Whether the user is performing a work, daily, fitness, or leisure activity.
- *Activity Style.* Whether the activity can be considered individual or social, i. e. shared with other people.
- *Activity Mode.* Whether the task is performed while on the go or in a fixed location.
- *Ability Requirements.* An indication of to what extend are required some particular abilities of the user for an specific activity.

6.4 Autonomous Selection of Interaction Resources

As previously introduced in section 6.2 and in Figure 6.2, the main job of the ICA in the TIAF, and therefore the main job of User Interface Builder in Dandelion, is to transform an abstract UI into a final UI capable of providing natural interaction in a specific usage scenario. In Dandelion, the process of building a final UI is reduced to performing a selection of a set of FIOs and their mapping to the different elements of the abstract UI. Thus, the UIB must choose, at run-time and in an autonomous way, those IRs (among the ones available in each environment) that better match the interaction requirements of the abstract UI and the natural interaction requirements of each usage scenario (user, environment, and scene).

In this section, we are going to show how this selection process, already introduced in section 6.2, is implemented in Dandelion by using fuzzy inference systems to exploit the information available in the usage scenario models.

As displayed in Figure 6.6, the FIO selection process can be divided into four different stages or phases. In the first step, that will be described in detail in section 6.4.1, the information available in the usage scenario models (User Profile Model, Environment Profile Model, and Scene Profile Model) is used to generate an specification of how a FIO should be in order to be considered adequate for the usage scenario. This is an ideal specification that represents, to some extent, the best possible FIO for each scenario, and it is generated by exploiting the model's information using a series of fuzzy inference systems.

The second step, described in subsection 6.4.2, consists in using this ideal FIO specification to look for those IRs that comply better with the specifica-

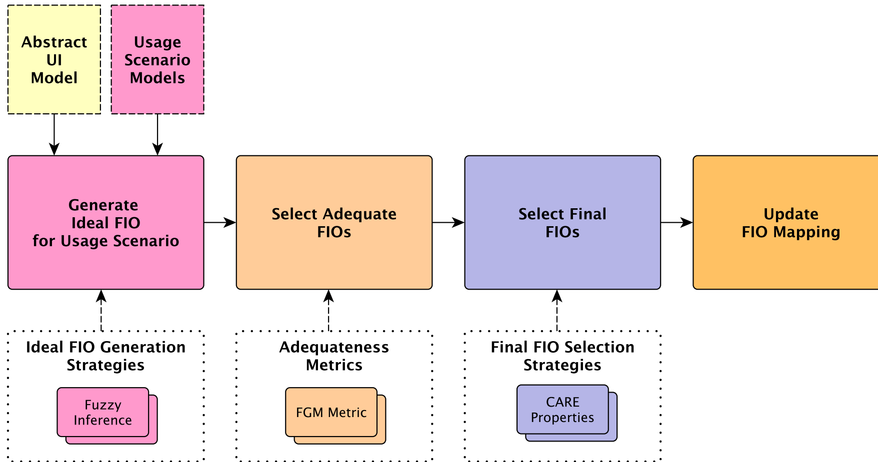


Figure 6.6: The FIO selection process is performed in four different phases. First, the models are used to create a specification of the characteristics required of a FIO to be considered adequate for the usage scenario. Second, the specification is used to select a list of FIOs from all the ones available in a particular physical environment. Third, one FIO is selected for each interaction facet. Finally, the FIO mapping is updated in the UIC.

tions. This phase is performed by the FIO Repository using an adequateness metric that measures how compliant is a particular FIO with the ideal specifications produced by the UIB. One list of candidates is generated for each abstract interaction facet of the abstract UI.

The result of the second step is a list of FIOs sorted by their adequateness for the ideal specification. This list is used by the third phase, presented in subsection 6.4.3, to select only one particular FIO that will be associated to a specific interaction facet from the abstract UI.

In the last step, also presented in subsection 6.4.3, each FIO chosen by the UIB is associated to a particular abstract interaction facet, updating the FIO mapping that is used by the User Interface Controller to route GIP events. This update is done at run-time, without affecting the AmI application, thus allowing the dynamic, and real-time, adaptation of the application to context changes.

6.4.1 Generating Specifications of the Ideal FIO

While all the four phases of the FIO selection procedure have an important role in the process, the first phase is the one that will have, by far, the greatest

impact in the shape of the final UI produced. In this first phase, the UIB must decide what kind of modalities and physical characteristic the final UI should have. For that purpose, we produce a specification of the desired modalities, physical shapes, and other characteristics, that an IR should present in order to be able to produce a natural user interaction experience in a particular usage scenario. We call this specification the *Ideal FIO*, and it must be considered as a wish list of what kind of IRs we want to use to assemble our final UI. In the second step of the process, this *Ideal FIO* will be used as the reference with which to compare all the physically available FIOs.

The *Ideal FIO* uses fuzzy variables to provide an indication of how desired each particular characteristic is, thus it is a fuzzy specification of what modalities and physical characteristics are supported (and to what extent) by a particular set of user, environment, and scene characteristics. Before describing the *Ideal FIO* specification and its generation process in detail, it is necessary to introduce the FIO Description Model. As previously indicated, the *Ideal FIO* will be used to compare to it the available FIOs, but for that, it is necessary to have a description of the FIOs in the same terms as the *Ideal FIO*. Therefore, either the IR manufactures, the developers, or the installers of the system must provide Dandelion with a description of each IR that is available for use in each environment. This description is performed using the FIO Description Model, and as will be seen, the *Ideal FIO* specification shares many concepts with it.

The next two subsections are dedicated to introducing the FIO Description Model and the *Ideal FIO* Specification Model respectively. In a third subsection after them, the *Ideal FIO* generation process is described in detail.

6.4.1.1 FIO Description Model

The goal of the FIO Description Model (FDM) is to describe the different characteristics that an interaction resource, represented by a FIO, can contribute to a final user interface. This description must include all the information required by the UIB to perform an adequate selection of FIOs for a specific abstract UI in a particular usage scenario.

Therefore, on the one hand, it must include user interaction support information (i. e. what interactions it allows the user to perform), because this information is essential to allow the FIO selection algorithms to choose IRs that match the different interaction requirements of the abstract UI. On the other hand, it should include information about the interaction modalities used by the IR and information about its physical shape. The modality information is necessary for the UIB algorithms to correctly select FIOs adequate for each user and environment characteristics, because different modalities will be better

adapted to the different abilities of each user, or even to different characteristics of the environment. For example, a speech production modality would be completely inadequate for a user with a high value of hearing difficulties, while a keyboard modality would be inadequate for a user with difficulties using her fingers and hands. Regarding the physical shape of the FIO, it can provide important information to consider in the selection process, because, depending on the user and environment characteristics, one shape or size can be considered a better option than others. For example, toy shaped devices will be, probably, more accepted by kids than by adults, an vice versa.

Figure 6.7 displays a conceptual class diagram showing the different concepts included in the FIO Description Model, and how they are related.

As can be seen, we decided to go for a quite simple and small model, specially for the description of the modalities and physical properties of the IRs. Even if an approach like the one by Obrenovic et al. in [Obrenović and Starcević, 2004, Obrenović et al., 2007] would feed the adaptation process with much more accurate information about how a human can manipulate an IR, because they propose a very complete and complex ontology that uses human functionalities and anatomical structures to describe the modalities used by interaction resources, we think that it is important to keep the model as simple as possible and as far as possible from complex definitions related to health and anatomic issues. In an ideal world, this description information should be provided by the manufacturer of the interaction resource, but, in its absence, it should be the FIO developer or even the system installer. It is obvious that neither the manufacturer, nor the developer or installers will be trained to provide a detailed anatomic and psychomotor specification as the one proposed by Obrenovic.

As shown in 6.7, the description of the interaction capabilities directly matches the style of the abstract UI model. Each IR just specifies the number and type of the interaction facets it supports. So, it should be easy to establish direct relations (the FIO mapping) between abstract UI elements (interaction facets from the abstract UI) and FIO interaction elements (interaction facets from FIOs).

Regarding the specification of the modalities and physical properties of the FIOs, we have designed a very simple mechanism which is used in a very similar way for the two descriptions. The kind of shape or modality is specified by a type, selected from a limited set of modalities and shapes supported, and a value called *granularity*. This value, which is used in a different way for each case, ranges from 0.0 to 10.0, with the lower values representing a coarser granularity, and the higher values a finer granularity.

For modalities, the granularity represents an indication of the skills required

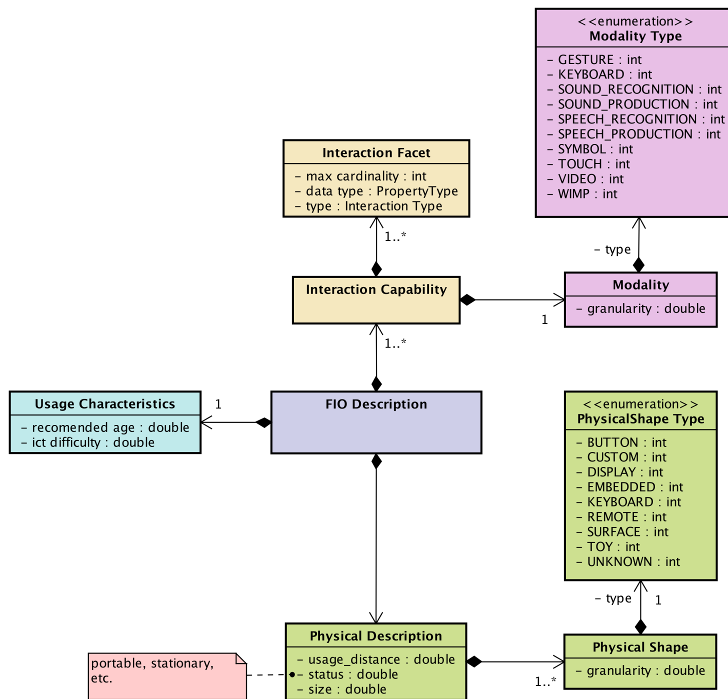


Figure 6.7: UML class diagram showing the concepts, and their relations, of the FIO description model.

to use the specified type of granularity. Thus, in the case of a keyboard modality, a coarser granularity will mean a keyboard with a small number of large keys, while a fine granularity will mean a keyboard with small keys and a large number of them. In the case of gesture-based modality, a fine granularity would suppose the use of a large number of different gestures, and probably more detailed ones (with the fingers), while a coarser granularity would mean a small set of gestures, probably using only the arms or hands, depending on the value.

For the specification of the physical shape, the granularity value represents how close the shape is to the specified shape type.

6.4.1.2 Ideal FIO Specification Model

The Ideal FIO Specification Model and the FIO Description Model share many similarities, including the use of the same concepts. This similarity is necessary because, during the second phase, different instances of them are going to be compared, this is, each FIO available in a physical environment is going to be compared to the Ideal FIO specification in order to know how alike they are.

As can be seen in Figure 6.8, the Ideal FIO is basically composed of a description of what modalities and physical shapes are considered good candidates for a particular usage scenario. The *adequateness factor* of each characteristic (modality, physical shape, usage characteristic) is specified by a *granularity* value, which ranges from 0.0, that indicates that the characteristic is not desirable at all, to 10.0, that is the highest possible degree of adequateness.

6.4.1.3 Ideal FIO Generation

While the UIB is designed to support different strategies for the generation of the Ideal FIO specification, for demonstration purposes we have implemented one strategy using Fuzzy Logic, in particular, it uses three Fuzzy Inference Systems (FIS) as displayed in Figure 6.9. Each one of them is in charge of generating one of the three different aspects of the specification. There is a FIS dedicated to selecting the most adequate modalities for each usage scenario, another one dedicated to selecting the best physical characteristics, including shape, and a last one dedicated to specifying the usage characteristics.

Those FIS have been implemented using the JFuzzyLogic library [Cingolani and Alcala-Fdez, 2012, Cingolani and Alcala-Fdez, 2013], a Java library that facilitates the implementation of fuzzy inference systems, and their rules have been programmed using the Fuzzy Control Language (FCL) [IEC, 1997]. The FCL language provides a standard way to specify fuzzy sets and fuzzy variables,

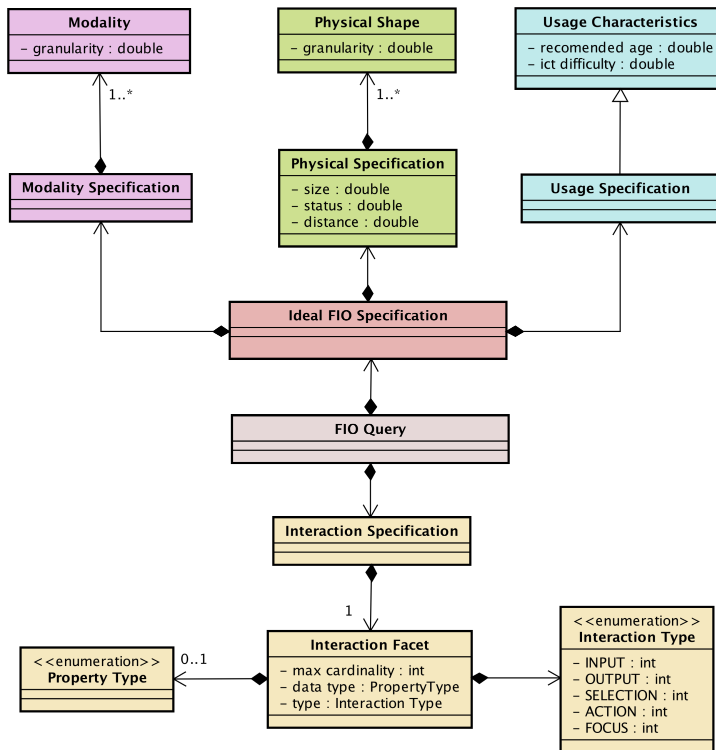


Figure 6.8: Conceptual class diagram displaying the components, and their relations, of the FIO Specification and FIO Query models.

as well as the fuzzy rules that will provide the knowledge required by the fuzzy inference systems to operate.

As previously introduced, the Ideal FIO is composed of three different groups of specifications:

- *Modality Specification.* The modality specification is a set of pairs of modality type and granularity values, that specifies to what extent a modality is supported by a specific usage scenario. This value can also be interpreted as a measure of how adequate is each modality for the usage scenario, because it can be assumed that if the granularity is well supported, it must be a good candidate for the scenario. The modality specification is generated by a series of small fuzzy inference systems, each one of them specialized in a particular modality. These FIS systems receive different properties of the user, environment and scene as input, and produce a granularity value for the modality as output. 6.1 displays a small example of one of those FIS, in particular, the one in charge of providing a granularity value of the SYMBOL modality.
- *Physical Specification.* The physical specification includes physical properties of the IR, like its size, and a set of pairs of physical shape and granularity values specifying how adequate each physical shape is considered. The physical specification is generated by two different FIS, one specialized in providing values of the physical properties, and the other one which uses the context models and the modality specification information to infer an adequateness value (granularity) for each physical shape.
- *Usage Specification.* This specification is generated by one FIS that relies on the information provided by the user model in order to infer the required usage characteristics for the IRs, like the ICT abilities required by the user or the user's recommended age.

As can be seen, the *Ideal FIO* is a kind of fuzzy wish list of FIO characteristics, which is generated by the UIB for each usage scenario, this is, for each particular combination of user, environment and usage characteristics. This specification does not include user interaction characteristics, apart from the modalities, thus it is independent from the concrete user interaction primitives supported by the IRs, and therefore, the same *Ideal FIO* specification is valid for all the Abstract Interaction Units and all the interaction facets of the abstract UI.

In order to continue with the FIO selection process, in the second step, a FIO Query is generated for each interaction facet and AIU that makes up the abstract UI. This FIO Query, as shown in Figure 6.9, is a combination of the

Algorithm 6.1 Fuzzy rules for the SYMBOL modality selection FIS.

```
RULE 1:
    IF user_visual_acuity IS normal AND
    user_cognitive_abstract_signs IS normal
    THEN modality_symbol IS high_granularity;
RULE 2:
    IF user_visual_acuity IS somewhat_impaired AND
    user_cognitive_abstract_signs IS normal
    THEN modality_symbol IS low_granularity;
RULE 3:
    IF user_cognitive_abstract_signs IS somewhat_impaired
    OR user_cognitive_abstract_signs IS impaired OR
    user_cognitive_abstract_signs IS severely_impaired
    THEN modality_symbol IS not_supported;
RULE 4:
    IF user_visual_acuity IS severely_impaired OR
    user_visual_acuity IS impaired
    THEN modality_symbol IS not_supported;
RULE 5:
    IF env_visibility IS low
    THEN modality_symbol IS medium_granularity;
RULE 6:
    IF env_visibility IS very_low
    THEN modality_symbol IS low_granularity;
```

Ideal FIO specification with a user interaction specification that describes the interaction primitives (input, output, etc.) required by a specific interaction facet. This query is sent to the FIO Repository that, as will be described in the next subsection, generates a list of the available FIOs that better match the characteristics of the *Ideal FIO* and the interaction requirements of the interaction facet.

6.4.2 Selecting Adequate FIOs

As indicated in Figures 6.6 and 6.9, the *Ideal FIO* is used to generate queries to ask the FIO repository about the FIOs, available in the physical environment, that better match the needs of the abstract UI and the usage scenario.

A FIO query is composed of two different kinds of information. On the one hand, the *Ideal FIO*, that provides a fuzzy specification of how a FIO should be in order to be considered adequate for a specific usage scenario. On the other hand, an Interaction Specification, that provides a description of the interaction

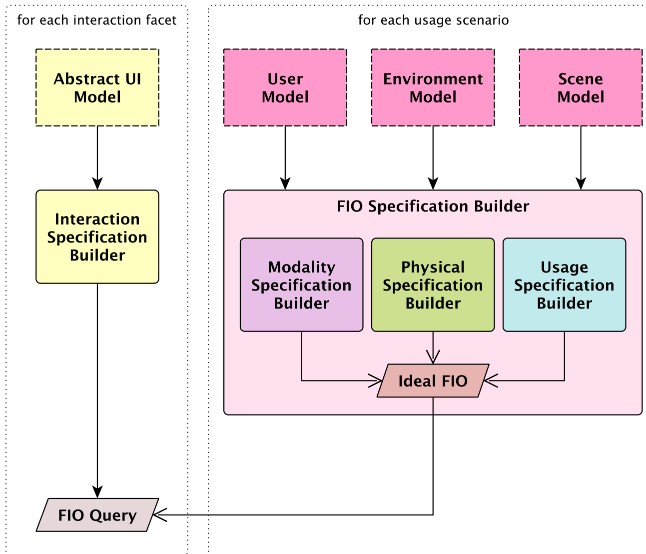


Figure 6.9: The Ideal FIO specification is generated for each usage scenario, while the interaction specification is generated for each one of the interaction facets included in the abstract UI.

capabilities required by the UI, like the type of interaction primitive, the type of the data, or its cardinality, as displayed in Figure 6.8.

The *Ideal FIO* is fixed for each particular usage scenario (combination of user profile, environment profile and scene profile), while the Interaction Specification changes for each interaction facet described in the abstract UI. Thus, in order to provide a FIO mapping for each interaction facet, and build the complete final UI, the UIB must generate one FIO Query for each interaction facet.

The FIO repository keeps a database of the FIOs available in the different physical environments of an HIE. This database is composed of a collection of FIO descriptions associated to each physical location of the HIE. When the repository receives a FIO query, the goal of the repository is to find those FIOs that better match the specification provided by the query. For that purpose, it uses a series of metrics to compare the *Ideal FIO* and the interaction specifications of the query to the descriptions of all the FIOs deployed in a particular location.

These metrics provide the repository with a measure of how similar is each particular FIO to the specifications of the *Ideal FIO*. As the *Ideal FIO* is a description of a FIO considered to be very adequate for a specific usage scenario,

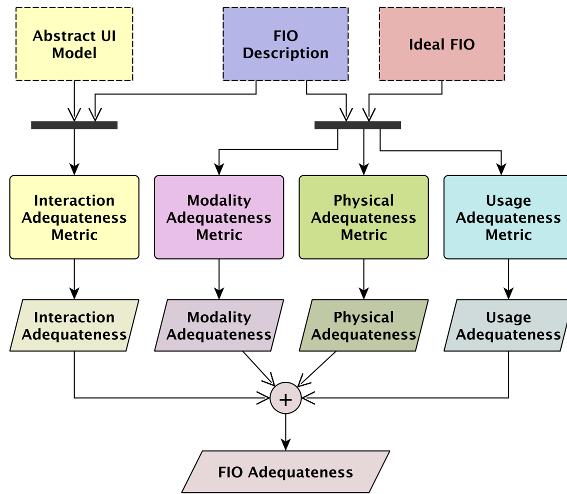


Figure 6.10: The FIO adequateness metric is calculated by an aggregation of four independent adequateness measures (Interaction, Modality, Physical and Usage similarities) between the query specifications and a FIO description.

these metrics provide a measure of how adequate a FIO is for a particular usage scenario. Therefore, we call these values the *FIO adequateness* for a scenario.

As shown in Figure 6.10, the adequateness measure is a result of the combination of multiple specific measures of each one of the aspects of a FIO:

- *Modality adequateness.* Provides a measure of how adequate is a FIO description, in terms of modalities, to the *Ideal FIO* specification.
- *Physical adequateness.* Provides a measure of how adequate is the physical description of the FIO with regards to the *Ideal FIO*.
- *Usage adequateness.* Provides a measure of how adequate are the usage characteristics of the FIO compared to the *Ideal FIO*.
- *Interaction adequateness.* Provides a measure of how adequate is the support for user interaction of the FIO compared to a particular abstract interaction facet.

This division of the FIO adequateness into different factors allows us to implement it as a weighted function of the different individual measures, thus allowing us to give more importance to some particular aspects. Furthermore, as will be seen in the next subsection, this division allows the implementation of Final UI building strategies capable of reasoning using the CARE properties for assessing usability in multimodal interfaces [Coutaz et al., 1995].

The repository calculates the FIO adequateness factor of each FIO deployed in the location specified by the FIO query, and then, it answers the query with a list of FIO descriptions sorted by their adequateness.

The FIO Repository is designed to support multiple different *FIO adequateness* metric implementations, permitting the UIB to select one or another by specifying the metric-id in each FIO query. For illustration and demonstration purposes, we have provided one implementation of a FIO adequateness metric that uses the Fuzzy Geometric Model, a fuzzy object comparison method proposed by Bashon in [Bashon et al., 2010, Bashon, 2013]. This metric is described in detail in the subsection bellow.

6.4.2.1 FIO Adequateness Calculation using the Fuzzy Geometric Model

As previously introduced, we have defined the FIO Adequateness factor as the similarity between the Ideal FIO specification and a FIO Description. Calculating similarity or compatibility between objects is a fundamental aspect of human reasoning, consequently, it has also become an important factor in the development of classification and decision systems, resulting in the development of multiple similarity or distance metrics for different purposes [Deza and Deza, 2009].

Both the Ideal FIO and the FIO Description are objects made up of fuzzy attributes, therefore, we need a mechanism to compare fuzzy objects, i. e. a similarity metric capable of dealing with uncertain and imprecise information. While there exist multiple different similarity measures for fuzzy data [Cross and Sudkamp, 2002], we decided to apply the Fuzzy Geometric Model (FGM) proposed by Bashon [Bashon et al., 2010, Bashon, 2013], because it matches very well the characteristics of the objects we want to compare.

The FGM is a generalization of the Euclidean distance adapted for comparing fuzzy sets. It evaluates the similarity between two objects by comparing their fuzzy attributes. An important characteristic is that the two objects must have the same number of attributes, and each pair of them must share the same fuzzy domain, i. e. they must have the same units and the same range of values. As can be seen, our comparing objects, the Ideal FIO and the FIO description, match these characteristics very well, as the two models share the same attributes and fuzzy domains.

We use the FGM, in combination with specific heuristics, for the calculation of the Modality, Physical, and Usage similarities, but not for the Interaction similarity, because the interaction specification does not use fuzzy attributes. In that case the Euclidean distance is used if the interaction type is the same,

in any other case, the interaction similarity is 0.0.

The Fuzzy Geometric Model performs the calculation of similarity between fuzzy objects in two steps:

1. Calculate the similarity among each corresponding pair for attributes.
2. Aggregate the similarities of all the pairs of attributes in order to give a final judgement of how similar the objects are.

For the first step, the calculation of the similarity between each pair of attributes, the FGM uses the next three equations.

The distance between two fuzzy sets $A_{ij}, B_{ij} \epsilon F(U_j)$ is defined by the mapping $dis : F(Uj) \times F(Uj) \to [0, 1]$ shown in equation 6.1, where U_j stands for the domain of the jth attribute, and $i = 1, 2, ..., m_j$ stands for the number of linguistic terms defined by the membership functions $\mu_{A_{ij}}$ and $\mu_{B_{ij}}$ (in our case: $\mu_{A_{ij}}(x) = \mu_{B_{ij}}(x)$ for all $x \epsilon U_j$). Equation 6.1 calculates the distance between the values of each pair of attributes ($x_1, x_2 \epsilon U_j$) taking into account only one of the specific linguistic terms of the fuzzy set for any.

$$dis(A_{ij}, B_{ij}) = |\mu_{A_{ij}}(x_1) - \mu_{B_{ij}}(x_2)| \tag{6.1}$$

Equation 6.2 shows a normalized generalization of the Euclidean distance, $d_F : at_{F_1} \times at_{F_2} \to [0, 1]$, that calculates the distance between two pair of fuzzy attributes by taking into account the shape of the membership function that characterizes their membership to the fuzzy set. This is done by including the attribute distances regarding all the different linguistic terms (m_j).

$$d_F(a_j, b_j) = \frac{\sqrt{\sum_{i=1}^{m_j} dis(A_{ij}, B_{ij})^2}}{\sqrt{m_j}} \tag{6.2}$$

Finally, the similarity, $S_F : at_{F_1} \times at_{F_2}$, between each corresponding pair of fuzzy attributes a_j and b_j is defined by equation 6.3, where k_j (≥ 0) is a weight that can be used to customize the contribution of each attribute to the final calculation of the object similarity.

$$S_F(a_j, b_j) = \frac{1 - d_F(a_j, b_j)}{1 + k_j d_F(a_j, b_j)} \tag{6.3}$$

Once we have calculated the similarity between each of the corresponding attributes, step two consists in aggregating all of them to produce an overall measure of how similar the two objects are. This can be done in multiple ways, but we have decided to use the weighted average of attribute similarities, because

it allows us to customize the importance of some attributes in the final similarity metric. The overall fuzzy object similarity metric is provided by equation 6.4, where $\alpha_j \epsilon\, [0, 1]$:

$$FuzSim(F_s, F_t) = \frac{\sum_{j=1}^{r} \alpha_j S_F(a_j, b_j)}{\sum_{j=1}^{r} \alpha_j} \tag{6.4}$$

As previously introduced, the FGM, i. e. the $FuzSim$ function, in combination with some simple heuristics is used to implement the Modality, Physical, and Usage similarities. For each particular attribute of each similarity we have used different weights to customize their importance in the final similarity calculation:

- The modality adequateness is calculated using a combination of heuristics and FGM similarity. It compares the granularity of each modality specified by the *Ideal FIO* to the granularities of the modalities supported by the FIO. If the granularity of the scenario is greater than, or equal to, the granularity of the FIO, the adequateness is 1.0, because the granularity of the *Ideal FIO* represents the maximum complexity supported by the scenario, and it supports a more complex modality than the one specified by the FIO. If the granularity of the scenario is smaller than the granularity of the FIO, we use the FGM to compare the two values according to their fuzzy definition as shown in equation 6.5, where g_s and g_f are the granularity values of modalities M_s and M_f respectively. Finally, the modality adequateness value is adjusted using a simple heuristic (see equation 6.6) that penalizes the complexity of modalities.

$$ModAdq(M_s, M_f) = FuzSim(g_s, g_f) \tag{6.5}$$

$$ModAdq(M_s, M_f) = ModAdq(M_s, M_f) - 0.01 * g_f \tag{6.6}$$

- The physical similarity is in fact calculated as the weighted aggregation of the application of the $FuzSim$ function to two different objects: the shape of the FIO and its physical characteristics (size, etc.). Its definition can be seen in equations 6.7 and 6.8, where sp stands for the shape granularity, Ch for the physical characteristics object, sz for the size attribute, st for the status attribute, and d for the usage distance attribute of the physical characteristics.

$$PhAdq(P_s, P_f) = 0.35 FuzSim(sp_s, sp_f) + 0.65 PhCSim(Ch_s, Ch_f) \tag{6.7}$$

$$PhCSim(Ch_s, Ch_f) = 0.5 S_F(sz_s, sz_f) + 0.35 S_F(st_s, st_f) + 0.15 S_F(d_s, d_f) \tag{6.8}$$

- The usage similarity is calculated using the $FuzSim$ function with the weights shown in equation 6.9, where Ug stands for the usage characteristics object, age for the recommended age attribute, and ict for the ICT literacy attribute.

$$UsgAdq(Ug_s, Ug_f) = 0.2S_F(age_s, age_f) + 0.8S_F(ict_s, ict_f) \tag{6.9}$$

Finally, the overall FIO adequateness is calculated using the next equation if the interaction similarity is greater than or equal to 0.25. When the interaction similarity is less than 0.25, $Adequateness = 0.0$.

$$\begin{aligned} Adequateness \quad = \quad & 0.10IntrAdq(I_s, I_f) + 0.65ModAdq(M_s, M_f) + \\ & 0.15PhAdq(P_s, P_f) + 0.10UsgAdq(Ug_s, Ug_f) \end{aligned} \tag{6.10}$$

As can be seen, we have considered the modality adequateness as the more relevant factor for the calculation of the overall adequateness because we think that modality is the factor that has more impact on the natural interaction perception of the UI.

Regarding interaction adequateness, we have set a limit for its lower value. If the interaction similarity is too low (less than 0.25), the adequateness is 0.0, because if a FIO does not match the interaction requirements of an interaction facet, it will be useless.

6.4.3 Building the Final User Interface

As previously introduced in this chapter, the last step in the Dandelion UI adaptation to context process is to build a Final UI by establishing a new mapping between the interaction elements of the abstract UI and the FIOs selected by the User Interface Builder. In the previous two subsections, we have seen how the UIB uses the FIO repository to obtain a list of FIOs available and measure how adequate they are to build a final UI for a specific usage scenario. In this subsection, we are going to describe how the UIB selects, for each particular abstract interaction facet, one specific FIO and then updates the FIO mapping in the User Interface Controller.

Like in the case of the *Ideal FIO* generation and the FIO adequateness metrics, the UIB is designed to support different implementations of FIO selection strategies. Nevertheless, for demonstration purposes we have provided just one implementation of a FIO selection strategy.

The implemented strategy takes advantage, as indicated in the previous

section, of the division of the FIO adequateness metric into four sub-factors to rely on the CARE properties [Nigay and Coutaz, 1994, Coutaz et al., 1995] to drive the selection process.

The CARE properties (Complementarity, Assignment, Redundancy and Equivalence), as described by Coutaz et al. [Coutaz et al., 1995], provide a way to characterize different aspects of multimodal interaction and facilitate the reasoning about how the multiple modalities can be combined to produce usable multimodal user interfaces:

- *Equivalence*. It occurs when two different modalities (or more) are enough, considered individually, to implement a particular user interaction primitive.
- *Redundancy.* It occurs when a UI must use multiple different, but equivalent, modalities to implement the same user interaction primitive in different ways. In contrast to complementarity, each one of the modalities is enough to implement the action.
- *Complementarity.* It is the characteristic of an UI of using a combination of different modalities for the implementation of one particular user interaction primitive. In contrast to redundancy, when two or more modalities are used in a complementary way, all of them are necessary to implement the interaction primitive.
- *Assignment.* Assignment is the complementary of redundancy. It indicates that there is only one possible modality to implement a particular user interaction primitive.

Regarding the CARE properties, in Dandelion we consider FIOs as modalities and the interaction facets and Abstract Interaction Units as the interaction primitives. Therefore, looking at the different metrics that make up the FIO adequateness measure, we can conclude that:

- The Interaction similarity measure can be considered as a measure of the equivalence between two different FIOs. Two FIOs with a high value of interaction similarity will implement the same interaction facets, thus providing the same interaction capabilities.
- The Modality, Physical, and Usage similarities can be used to reason about the selection of redundant FIOs. By using the interaction similarity to select equivalent FIOs, we can use the other three similarities to select FIOs that provide the same interaction capabilities while using different modalities, different physical shapes or usage characteristics.

- The same as above can be done to reason about complementarity. When multiple FIOs are required to implement a particular interaction (for example, an Abstract Interaction Unit with multiple interaction facets), we can select FIOs with high degrees of Modality, Physical, and Usage similarities, but lower degrees of interaction similarities.

The selection process applies the CARE properties to select FIOs with the objective of producing an usable multimodal user interface:

- It relies mainly in the FIO adequateness value, so that it tends to select those FIOs with the higher values in adequateness.
- When multiple interaction facets belong to the same Abstract Interaction Unit, they are logically grouped and they are probably part of the same complex interaction action. Therefore, the system applies the Complementarity property in order to look for FIOs that are similar regarding modalities, physical shape, and usage characteristics, but are not equivalent, i. e. they provide different interaction capabilities.
- The Redundancy property is applied when the developer specifies that one interaction facet is required to have multiple simultaneous implementations. In such case, the UIB can select different FIOs with a high degree of Equivalence, but different modalities, physical shapes, or usage characteristics.

Once a FIO has been selected for each one of the interaction facets of the abstract UI, the final step is to update the FIO mapping, and thus, provide a new Final UI for the application. The update process is managed by the UIB, which uses a remote interface provided by the User Interface Controller (accessible though a messaging protocol using STOMP and JSON). The mapping is updated at run-time, without affecting the application, thus, once the update is finished, the UIC starts routing the GIP messages to the new FIOs, and the application automatically starts using the new Final UI without interruption.

6.5 Demonstration Examples and Summary

In this subsection, we are going to present three examples to demonstrate the capabilities of the Dandelion framework to implement context-adaptive UIs.

In subsection 6.5.1, we use the Environmental Music Player system, already introduced in previous chapters, to show an example of UI adaptation to context changes in the environment of use. Section 6.5.2 is dedicated to adaptation of UIs to the characteristics of different users. And finally, in section 6.5.3, we

use the EvacUI system to provide a complete example of a physical distributed UI capable of adapting itself to different users interacting with the system in a variety of physical environments and situations.

6.5.1 Environment Adaptation: Environmental Music Player

In this subsection, we are going to demonstrate the capabilities of the Dandelion framework to adapt, at run-time, a Physical User Interface to changes in the physical environment. Therefore, we are going to focus exclusively on physical environment adaptation, thus only the environment characteristics, i. e. the environment and scene profiles, are going to change dynamically, while the user profile is going to remain the same for all the scenarios.

For that purpose, we are going to use the Environmental Music Player example, the UI of which was already introduced in subsections 4.2.2 and 5.6.2.

For this demonstration example, we have envisioned four different usage scenarios:

- *The living-room.* The user is listening to music in the living-room while she is doing any leisure activity, like reading.
- *The kitchen.* The user is listening to music while she is cooking in the kitchen.
- *The car.* The user is listening to music while she is driving to her work-place.
- *Outdoors.* The user is listening to music while she is doing some sport activity.

Table 6.1 shows the detailed environment profile information of each usage scenario, and table 6.2 shows the scene profiles for each different usage scenario. With regard to the user profile, we have configured a user with the maximum value (10.0) in all physical and motor characteristics and a medium ICT literacy value (6.0), so that the user does not impose any limitation on the selection of FIOs.

As the EMP abstract UI is quite big, in order to keep the explanation of the adaptation process simple, we have selected four particular interaction facets of the EMP abstract UI introduced in section 5.6.2. For each one of this facets, we have executed the Dandelion context adaptation process using the previously mentioned profiles as inputs. Furthermore, we have provided the FIO Repository with a set of FIOs and their descriptions, in order to let

	Noise	Visiblity			Motion		Climate			Type	Situation	Space
	ambient	visibility	contrast	lighting	motion	vibration	temp.	humidity	wind	type	situation	space
Livingroom	2,0	7,0	2,0	2,0	0,0	0,0	20,0	65,0	0,0	7,0	0,0	4,5
Kitchen	7,0	7,0	4,0	7,0	0,0	0,0	21,0	55,0	0,0	5,5	0,0	3,0
Car	5,5	2,0	7,0	5,0	7,0	5,5	18,0	70,0	0,0	4,0	5,5	1,5
Outdoors	4,0	5,5	8,0	5,0	5,5	1,25	10,0	75,0	15,0	1,25	10,0	9,0

Table 6.1: Values of the different environment profiles associated to each usage scenario of the EMP example.

	Scene						
	usr. count	act. type	act. mode	act. style	hands	vision	attention
Livingroom	1,0	10.0	8,0	0,0	0,0	0,0	1,0
Kitchen	1,0	10.0	5,5	0,0	7,0	4,0	3,0
Car	1,0	10.0	4,0	0,0	8.5	8,0	5,0
Outdoors	1,0	10.0	1,25	0,0	1,0	2,0	2,0

Table 6.2: Values of the different scene profiles associated to each usage scenario of the EMP example.

Dandelion select among them those that are more adequate for each combination of interaction facet and physical environment.

In the following subsections, we are going to present the results obtained by the Dandelion autonomous UI building system for the different interaction facets selected as examples.

Control Actions like Play, Pause, Next Song, etc.

The EMP UI allows the user to perform many different actions, like change the song (next and previous actions) or control the playing of music (play, pause, and stop). Each one of these actions is modeled in the abstract UI as an interaction facet with cardinality 1 and a trigger interaction type.

For demonstration purposes, we have implemented a small set of FIOs capable of supporting the different interaction requirements of the EMP UI (some of them are shown in Figure 6.11):

- A remote controller with support for 10 interaction facets of type trigger.
- A Kinect [Microsoft, 2015] gesture recognition IR with support for 7 interaction facets of type trigger.

- A Leap Motion [Leap-Motion-Inc., 2015] gesture recognition IR with support for 7 interaction facets of type trigger and 1 interaction facet of type selection.
- A button-based IR, which uses home automation wall switches supporting 4 interaction facets of type trigger and 1 interaction facet of type selection.
- A smartphone IR, which uses an Android GUI application to show buttons and text fields, supporting 7 interaction facets of type trigger, 1 interaction facet of type selection, 3 interaction facets of type output, and 1 interaction facet of type input.
- A speech recognition IR providing 5 interaction facets of type output and 1 interaction facet of type input.
- A display IR using a JAVA GUI application that supports 4 interaction facets of type output.
- A speech synthesizing IR using Festival [Taylor et al., 2006, of Edinburgh, 2015] that supports 3 interaction facets of type output.

These FIOs have been described using the FIO Description model and registered in the FIO repository, so that Dandelion can use them during the FIO selection process. Table 6.3 shows the detailed description of each one of those FIOs supporting ACTION interaction facets. For example, the remote controller uses a keyboard modality with a granularity of 4.0 (it only has 10 keys), it has a small size of 1.0, and a recommended distance of usage of 50 cm. Another example is the Kinect-based FIO, which uses a gesture modality with a granularity of 5.0 and it has a recommended usage distance of 300 cm.

While in normal circumstances not all of those FIOs would be available in every physical environment, for illustration purposes we are going to assume that the six FIOs are accessible in every one of the four physical environments of the example.

Recalling section 6.4, the Dandelion context-adaptation process consists in selecting a set of FIOs adequate for a particular combination of abstract UI and usage scenario. The first step in this selection process is to generate the Ideal FIO specification for each scenario. This specification indicates what kind of modalities, physical shapes, and usage characteristics (and to what level of granularity) are supported by a specific usage scenario. As previously introduced in this chapter, this specification is generated by a set of Fuzzy Inference Systems that receive the scenario models as inputs. Table 6.4 shows the Ideal FIO specifications generated for the different scenarios of this example. As can be seen, there are big differences between each scenario. Modalities and

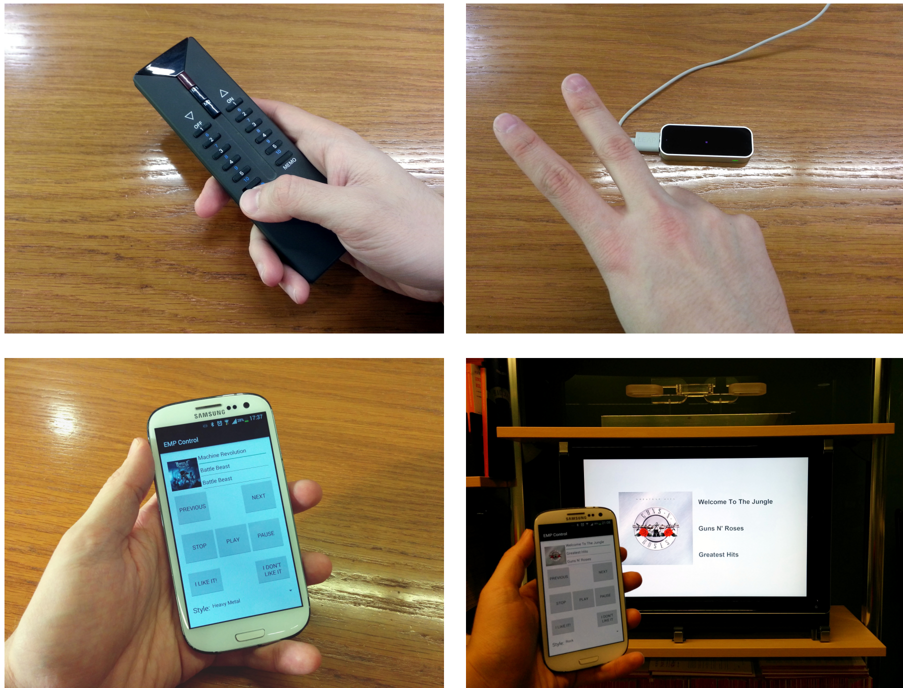


Figure 6.11: Four of the FIOs used for the EMP example. First, a KNX home automation remote controller, connected to Dandelion using UniDA. Second, a Leap Motion device for finger and hand gesture recognition. Third, a custom Android GUI application. Fourth, a TV display for output with a Kinect camera for input (using gesture recognition).

shapes are specified in the range [0..10], but it is important to note that they are values inside a fuzzy domain where values below 2,5 must considered as "not supported", thus values below 5,0 can be considered as "very low" or "very coarse" granularity. Furthermore, it is also important to highlight that the granularity value of modalities and shapes can be interpreted, at the same time, from two different points of view.

On the one hand, it represents the level of complexity (granularity) supported for a particular modality, for example, for the case of the keyboard modality, in the Living-room environment it is supported up to a granularity of 9,025, while in the car it is supported up to 5,104. This indicates that in the Living-room it is possible to use a very complex keyboard, with many and small keys, while in the car only simple keyboards, with few and large keys, will be well supported.

On the other hand, the granularity can be also considered as a good indication of whether a modality, or shape, is adequate for a particular usage

		Remote	Kinnect	Motion Leap	Buttons	Smartphone	Speech Rec.
INTER.	cardinality	**8**	**10**	**10**	**4**	**8**	**5**
	type	**TRIGGER**	**TRIGGER**	**TRIGGER**	**TRIGGER**	**TRIGGER**	**TRIGGER**
MODALITIES	sound_prod	0,0	0,0	0,0	0,0	0,0	0,0
	speech_rec	0,0	0,0	0,0	0,0	0,0	4,0
	speech_prod	0,0	0,0	0,0	0,0	0,0	0,0
	touch	0,0	0,0	0,0	0,0	7,0	0,0
	wimp	0,0	0,0	0,0	0,0	0,0	0,0
	gesture	0,0	5,0	8,0	0,0	0,0	0,0
	keyboard	4,0	0,0	0,0	2,0	0,0	0,0
	symbol	0,0	0,0	0,0	0,0	0,0	0,0
	video	0,0	0,0	0,0	0,0	0,0	0,0
PHYSICAL	size	1,0	8,0	4,0	1,0	3,0	8,0
	status	1,0	10,0	8,0	10,0	1.5	3,0
	distance	50,0	300,0	100,0	60,0	80,0	200,0
SHAPES	display	0	0	0	0	1	0
	button	1	0	0	1	1	0
	remote	1	0	0	0	1	0
	toy	0	0	0	0	0	0
	embedded	0	1	1	1	0	1
	keyboard	1	0	0	0	0	0
	surface	0	0	0	0	0	0
USAGE	age	20	20	20	20	20	20
	ict	2,0	4,0	6,0	2,0	6,0	5,0

Table 6.3: Detailed description of the EMP FIOs using the FIO Description model.

		Livingroom	Kitchen	Car	Outdoors
MODALITIES	sound_prod	9,025	4,986	6,455	9,025
	speech_rec	9,025	4,986	6,455	9,025
	speech_prod	9,025	4,986	6,455	9,025
	touch	9,025	6,007	4,341	6,377
	wimp	6,911	3,593	3,593	3,593
	gesture	9,025	4,986	4,91	7,444
	keyboard	9,025	4,677	5,104	7,444
	symbol	9,025	9,025	7,107	8,027
	video	9,025	5,14	2,964	2,08
PHYSICAL	size	7,913	4,999	2,604	2,08
	status	6,249	3,309	1,506	0,969
	distance	250,465	58	57,142	25,555
SHAPES	display	9,025	6,249	4,798	4,443
	button	3,749	7,732	8,12	6,045
	remote	7,822	0	0	0
	toy	3,749	3,749	3,749	3,749
	embedded	9,025	8,997	8,828	9,025
	keyboard	6,911	5,623	6,249	7,588
	surface	6,947	6,249	5,104	6,672
USAGE	age	25	25	25	25
	ict	6	6	6	6

Table 6.4: Ideal FIO specifications generated for the four usage scenarios of the EMP example.

scenario. If the scenario has a high level of support for a modality, it seems fair to assume that the modality should be considered adequate by the user in that specific environment and scene.

Therefore, looking at the results shown in table 6.4, we can see that the kitchen and car environments are a little bit conflicting. They both have low or medium values in almost all modalities. This is due to the fact that both scenes have a high level of "ambient noise" and a high level of "hand-occupancy" (7,0 and 8,50 respectively), thus they support only "coarse granularity" modalities to receive input from the user. The contrary happens with the living-room environment, where the user are hands-free, so she can use almost any modality at its highest level, except from the "wimp" (Windows-Icons-Mouse-Pointer) modality, which has the lowest value because it is not considered by Dandelion as a good option, in part because of the 6,0 value of the user ICT literacy, and in part due to the recommended interaction distance of 250,465 cm and other scene constraints, like the fact that the user is performing a leisure activity.

Once we have the Ideal FIO specifications for a particular usage scenario, the next step is to check all the available FIOs looking for those that better

		Remote	Kinnect	Leap M.	Buttons	Smartph.	Speech R.	Speech S.	Display
Kitchen	**Adequateness**	**0,894**	**0,912**	**0,408**	**0,905**	**0,733**	**0,915**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,995	0,273	0,98	0,69	0,96	0,96	0,95
	Physical	0,52	0,487	0,582	0,511	0,617	0,66	0,564	0,454
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92
Car	**Adequateness**	**0,923**	**0,893**	**0,395**	**0,914**	**0,562**	**0,901**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,972	0,262	0,98	0,378	0,96	0,97	0,714
	Physical	0,719	0,465	0,541	0,571	0,828	0,569	0,43	0,341
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92
Livingroom	**Adequateness**	**0,877**	**0,929**	**0,824**	**0,908**	**0,871**	**0,928**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,95	0,92	0,98	0,929	0,96	0,96	0,95
	Physical	0,407	0,798	0,553	0,528	0,501	0,746	0,746	0,701
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92
Outdoors	**Adequateness**	**0,934**	**0,878**	**0,731**	**0,922**	**0,79**	**0,899**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,95	0,79	0,98	0,767	0,96	0,97	0,292
	Physical	0,792	0,461	0,49	0,62	0,659	0,55	0,425	0,311
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92

Table 6.5: This table displays the FIO adequateness values calculated for each combination of FIO and usage scenario. The green rows show the overall FIO adequateness values, while the other rows show the values of different submetrics that make up the overall adequateness using the equation 6.10.

match the requirements of the scenario. This is done by using the Ideal FIO specifications as a query for the FIO repository, which will answer the query with a list of available FIOs accompanied by their level of adequateness for the usage scenario. As previously introduced, this adequateness value is calculated using the FIO descriptions (see table 6.3) and the Ideal FIO specifications (see table 6.4) as inputs for the equations explained in subsection 6.4.2.1.

The adequateness results for the trigger interaction facets are shown in table 6.5. It is important to mention that, in this example, only the physical environment and the scene profiles are changing, while the user profile remains the same, thus the adequateness value is only affected by environment and scene characteristics. This is why, as can be seen, there are only slight differences between the results of the different scenarios. The most relevant differences are that FIOs with high granularity values in modalities requiring the use of hands, like the Leap Motion or the Smartphone (due to its touch interface), have the lowest adequateness values for the Kitchen and Car scenarios.

As in the case of granularity values, the adequateness values must be con-

sidered inside their fuzzy domain, where values bellow 0,25 must be considered as "not supported", and values below 0,5 must be considered as "low adequateness".

Looking at the disaggregated values of Interaction, Modality, Physical, and Usage adequateness, we can also see some interesting results. For example, only the Speech Synthesizing and Display FIOs have a value of 0,0 in Interaction adequateness, while all the others have a value of 1,0. This is due to the fact that those two FIOs do not support trigger interaction facets, thus they are completely inadequate, while the others support multiple trigger interaction facets with cardinality 1, thus they are considered very good candidates. The interaction adequateness, as we will see in the next subsections, can have also intermediate values when the interaction facets have the same type but different cardinalities.

Another interesting result is to confirm the high relevance that modalities have on the adequateness values. We can see how FIOs with higher degrees of physical and usage adequateness have lower values of overall adequateness because they have low values of modality adequateness. This is the case of the Smartphone FIO in the Car and Kitchen environments. We think that this is a good behavior of the FIO adequateness metric, because modality is the characteristic that has the biggest impact in the natural interaction perception of a user interface.

The last step in the FIO selection process would be to build the Final UI by selecting the most adequate FIO for each interaction facet. This selection is performed using the strategy introduced in subsection 6.4.3. Therefore, bearing in mind the results shown in table 6.5, the FIOs selected for the music control actions will be:

- Speech recognition for the Kitchen environment.
- The remote controller keyboard for the Car environment, with the buttons (which will be more probably available in that environment) in the second place.
- The Kinect gesture-based FIO for the living-room environment.
- The remote controller keyboard (for example the remote controller in a hands-free headset) for the outdoor environment.

Music Style Selection

In this subsection we are going to explore how the Dandelion ICA performs the adaptation process of the EMP Abstract Interaction Unit in charge of man-

		Leap M.	Smartphone	Buttons
INTERACT.	cardinality	5	3	3
	data type	text	text	text
	type	SELECTION	SELECTION	SELECTION
MODALITIES	sound_prod	0,0	0,0	0,0
	speech_rec	0,0	0,0	0,0
	speech_prod	0,0	0,0	0,0
	touch	0,0	7,0	0,0
	wimp	0,0	0,0	0,0
	gesture	6.5	0,0	0,0
	keyboard	0,0	0,0	3,0
	symbol	0,0	0,0	0,0
	video	0,0	0,0	0,0
PHYSICAL	size	4,0	3,0	2,0
	status	8.0	1,5	10,0
	distance	100,0	70,0	60,0
SHAPES	display	0	1	0
	button	0	0	1
	remote	0	0	0
	toy	0	0	0
	embedded	1	0	1
	keyboard	0	0	0
	surface	1	0	0
USAGE	age	20	20	20
	ict	3	6	2

Table 6.6: Detailed description of the FIOs available for the implementation of the music selection abstract interaction facet.

aging the selection of the music style. This AIU, as shown in algorithm 5.2, uses one interaction facet of type selection, with a variable cardinality depending on the number of music styles available in the user's music collection.

We have implemented three different FIOs with support for selection interaction facets:

- A Leap Motion device, which allows the user to select among five different styles using her fingers.
- A smartphone touch interface, which allows the selection between three different styles.
- Three physical buttons allow the selection of three different styles.

The detailed description of these three FIOs can be seen in table 6.6.

As previously explained in section 6.4.1, the Ideal FIO specifications remain the same for all the interaction facets, only the interaction specifications, used in the FIO Query, change. Therefore the Ideal FIOs for each environment continue to be the same one already shown in table 6.4.

Table 6.7 shows the adequateness results obtained by the FIO repository for the available FIOs in each one of the four usage scenarios of this example.

		Remote	Kinnect	Leap M.	Buttons	Smartph.	Speech R.	Speech S.	Display
Kitchen	Adequateness	**0**	**0**	**0,615**	**0,899**	**0,721**	**0**	**0**	**0**
	Interaction	0	0	0,875	1	0,875	0	0	0
	Modality	0,96	0,995	0,61	0,97	0,69	0,96	0,96	0,95
	Physical	0,52	0,487	0,582	0,511	0,617	0,66	0,564	0,454
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92
Car	Adequateness	**0**	**0**	**0,597**	**0,908**	**0,549**	**0**	**0**	**0**
	Interaction	0	0	0,875	1	0,875	0	0	0
	Modality	0,96	0,972	0,593	0,97	0,378	0,96	0,96	0,496
	Physical	0,719	0,465	0,541	0,571	0,828	0,569	0,428	0,339
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92
Livingroom	Adequateness	**0**	**0**	**0,821**	**0,901**	**0,859**	**0**	**0**	**0**
	Interaction	0	0	0,875	1	0,875	0	0	0
	Modality	0,96	0,95	0,935	0,97	0,929	0,96	0,97	0,95
	Physical	0,407	0,798	0,553	0,528	0,501	0,746	0,746	0,701
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92
Outdoors	Adequateness	**0**	**0**	**0,812**	**0,915**	**0,777**	**0**	**0**	**0**
	Interaction	0	0	0,875	1	0,875	0	0	0
	Modality	0,96	0,95	0,935	0,97	0,767	0,96	0,97	0,292
	Physical	0,792	0,461	0,49	0,62	0,659	0,555	0,394	0,311
	Usage	0,92	0,92	0,435	0,92	0,92	0,92	0,92	0,92

Table 6.7: FIO adequateness values calculated for each FIO and usage scenario for the EMP music selection interaction facet.

As can be seen, in this case, not all the FIOs have a value of 1,0 in Interaction adequateness. We are querying the repository for FIOs supporting a selection facet with cardinality of 5, but the Buttons and Smartphone FIOs only support 3, thus they have a lower value in Interaction adequateness. Apart from that, we can see that the Buttons are considered the best option in all the four cases, mainly because they have a very low requirement of keyboard modality and ICT literacy.

6.5.2 User Adaptation: OMNI Virtual Assistant

In the previous section, we have used the Environmental Music Player application to explore an example of how Dandelion can adapt a Physical UI to changes in the physical environment and usage scene. In this section, we are going to use the OMNI Virtual Assistant example, already introduced in subsections 4.2.1 and 5.6.1, to present an example of Physical UI adaptation to user characteristics variation.

For illustration purposes, we have taken only a subsystem of the OMNI UI example: the user notification subsystem UI, which is in charge of notifying

Algorithm 6.2 UsiXML definition of the OMNI notification abstract UI.

```
<!– notification dialog –>
<aui:AbstractUIModel>
    <aui:AbstractInteractionUnit id="NotificationDialog" ... >
        <aui:DataInputOutputFacet id="NotificationMessage"
        minCardinality="1" maxCardinality="1" dataFor-
        mat="string" inputSupport="false" outputSup-
        port="true" ... >
            <aui:dataType>text</aui:dataType>
        </aui:DataInputOutputFacet>
        <aui:TriggerFacet id="YesAction">
            <aui:triggerType>operation</aui:triggerType>
        </aui:TriggerFacet>
        <aui:TriggerFacet id="NoAction">
            <aui:triggerType>operation</aui:triggerType>
        </aui:TriggerFacet>
    </aui:AbstractInteractionUnit>
</aui:AbstractUIModel>
```

events to the user. This UI, as shown in algorithm 6.2, is composed of only one Abstract Interaction Unit that includes three interaction facets: one OUTPUT facet for showing messages (or questions) to the user, and two TRIGGER facets to receive the response of the user (Yes or No). As already explained in subsection 4.2.1, in a classic GUI, it will take the form of a dialog box with a label and two buttons, but as we will see, in an Ambient Intelligence system, using Physical Distributed UIs, it can take many different forms.

In contrast to the previous example, in this case, the environment and scene profiles are going to remain the same for all scenarios, while the user profile is going to change. We have used the Living-room example from the previous section as environment and scene profiles (see tables 6.1 and 6.2), and we have defined six user profiles, representative of the variety of user types to which the UI must be adapted:

- *Deaf User.* A user with a severe hearing impairment.
- *Blind User.* A user with a severe visual impairment.
- *Elderly 1.* An elderly user with hearing and visual impairments and high ICT literacy.
- *Elderly 2.* An elderly user with mild hearing and visual impairments and some motor difficulties affecting her ability to move her arms, hands, and fingers.

		Deaf U.	Blind U.	Elderly 1	Elderly 2	Kid	Arms Imp.	Cogn. Imp.
Hearing	hearing	2,00	10,00	5,50	8,00	10,00	10,00	10,00
Vision	vision_ability	10,00	2,00	5,50	7,00	10,00	10,00	10,00
	vision_field	10,00	1,50	4,50	7,00	10,00	10,00	10,00
Cognitive	language_rec	10,00	10,00	7,50	8,50	7,00	10,00	4,00
	language_prod	10,00	10,00	9,00	9,00	6,50	10,00	4,00
	abstract_symbol	10,00	10,00	10,00	10,00	7,00	10,00	6,00
	attention	8,50	8,50	8,50	9,00	6,00	10,00	4,00
	processing_speed	10,00	10,00	10,00	8,50	10,00	10,00	5,00
	work_mem	10,00	10,00	10,00	7,50	10,00	10,00	6,00
	long_term_mem	10,00	10,00	10,00	9,00	10,00	10,00	6,00
Motor	speech_artic.	6,00	10,00	7,50	9,00	7,00	10,00	4,00
	finger_preciss.	9,00	9,00	9,00	6,50	7,50	2,00	10,00
	hand_preciss.	9,00	9,00	9,00	7,00	7,50	2,50	10,00
	arm_preciss.	9,00	9,00	9,00	7,50	7,50	3,50	10,00
	contact_grip	9,00	9,00	9,00	7,50	8,00	2,50	10,00
	pinch_grip	9,00	9,00	9,00	6,50	8,00	2,00	10,00
	clench_grip	9,00	9,00	9,00	7,00	8,00	3,50	10,00
	hand-eye_coord	8,00	8,00	8,00	6,00	7,00	2,50	5,00
ICT	ict_literacy	7,00	7,00	7,00	2,50	3,50	7,50	2,00
	ict_anxiety	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Personal	age	50,00	50,00	75,00	75,00	5,00	40,00	30,00

Table 6.8: User profile definitions for the six different users included in the OMNI example for UI adaptation to user characteristics.

- *Kid.* A five year old kid, so she has some difficulties with language production, as well as hand and finger precision.
- *Upper Body Impaired.* A user with severe impairments that affect the movement of her arms, hands, and fingers.
- *Cognitive Impaired.* A user with severe impairments affecting her cognitive abilities.

The detailed description of the six user profiles is shown in table 6.8.

In the next subsections, we are going to describe the details of the FIO selection process for the three previously mentioned interface facets, but before that, we are going to explore the first step in the FIO selection process, the generation of the Ideal FIO specifications, which only depends on the usage scenarios profiles. We have six users, all of them operating the system in the same environment and scene, thus we have six different usage scenarios as shown in table 6.9.

As can be seen in the table, there are important differences between the Ideal FIOs generated for each scenario. For example, we can clearly see that the system indicates that sound-based modalities are a very bad option for the

		Deaf U.	Blind U.	Elderly 1	Elderly 2	Kid	Arms Imp.	Cogn. Imp.
MODALITIES	sound_prod	0,969	9,025	3,064	7,456	9,025	9,025	9,025
	speech_rec	0,969	9,025	1,071	9,025	6,249	9,025	0,969
	speech_prod	0,969	9,025	2,102	6,761	3,749	9,025	0,969
	touch	7,456	7,456	7,456	4,358	4,999	0,969	0,969
	wimp	5,889	3,593	6,007	3,042	3,749	2,080	2,080
	gesture	9,025	9,025	9,025	3,749	3,749	0,969	9,025
	keyboard	9,025	6,911	7,128	6,249	4,999	0,969	3,749
	symbol	9,025	0,969	2,100	3,749	0,969	9,025	1,018
	video	9,025	0,969	2,537	0,969	9,025	9,025	9,025
PHYSICAL	size	7,913	7,913	7,913	7,913	7,913	7,913	7,913
	status	6,249	6,249	6,249	6,249	6,249	6,249	6,249
	distance	250,465	250,465	250,465	250,465	250,465	250,465	250,465
SHAPES	display	8,904	4,960	6,860	2,925	7,913	4,986	4,986
	button	3,909	6,085	5,584	8,109	7,913	0,000	6,249
	remote	7,822	7,538	7,717	7,822	7,822	6,249	7,727
	toy	3,749	3,749	3,749	3,749	9,025	3,749	3,749
	embedded	9,025	9,025	7,913	7,969	7,969	9,025	9,025
	keyboard	6,895	6,225	6,484	6,249	6,249	0,969	3,749
	surface	6,895	3,749	6,895	4,827	5,017	2,060	2,060
USAGE	age	50	50	75	75	5	40	30
	ict	7,0	7,0	7,0	2,5	3,5	7,5	2,0

Table 6.9: Ideal FIO specifications generated for each usage scenario considered in the OMNI example.

Deaf User and the Elderly 1, because they have severe hearing impairments. Gestures are a bad option for the Elderly 2, the Kid, and the user with arm and hands impairments. Language-based modalities are not adequate for the user with cognitive impairments. On the other hand, sound-based modalities seem very adequate for the Blind User or the user with upper body impairments.

Regarding other aspects of the results, it can be seen that the physical characteristics of the FIOs depend mainly on environmental and scene characteristics, as they all remain the same for all the scenarios. The results in physical shapes are very related to the results in modalities, with, for example, shapes like keyboard or display having high values when keyboard and video modalities are high respectively and vice versa.

Showing Notifications

The job of the OMNI notification subsystem UI is to show event messages to the user, they can be either events from the agenda or questions about the

		Display	Smartphone	Speech Prod.
INTERACT.	cardinality	1	1	1
	data type	text	text	text
	type	OUTPUT	OUTPUT	OUTPUT
MODALITIES	sound_prod	0	0	0
	speech_rec	0	0	0
	speech_prod	0	0	6
	touch	0	0	0
	wimp	0	0	0
	gesture	0	0	0
	keyboard	0	0	0
	symbol	0	0	0
	video	5	7	0
PHYSICAL	size	6	3	8
	status	10	1.5	3
	distance	200	70	200
SHAPES	display	1	1	0
	button	0	0	0
	remote	0	0	0
	toy	0	0	0
	embedded	0	0	1
	keyboard	0	0	0
	surface	0	0	0
USAGE	age	20	20	20
	ict	1	6	3

Table 6.10: Detailed description of the FIOs available for the implementation of the notification message output abstract interaction facet.

operation of the system, because, as previously introduced in subsection 4.2.1, the OMNI system operates by proactively asking the user what action she wants to perform next.

In order to find the adequate FIOs for the notification message interaction facet, we need to query the Repository for FIOs that support, at least, one interaction facet of type output, and that have a set of characteristics similar to the specifications described by the Ideal FIOs, displayed in table 6.9. For this example, we have extended some of the FIOs already defined for the EMP example with support for one interaction facet of type output (see table 6.10). The results of the query to the FIO repository can be seen in table 6.11.

There are only three FIOs capable of supporting output interaction facets, and among those three FIOs, there are huge differences in adequateness depending on the user abilities. For example, in the case of the Deaf User, speech recognition has a value of 0.0 in modality adequateness, thus it has a very low value in FIO adequateness for the Speech Recognition IR. On the contrary, the blind subject has a very high value of adequateness for the Speech Recognition IR and very low values for Display and Smartphone IRs, because they depend a lot on the vision ability of the user.

Another interesting result is the one of the Elderly 1. She is a user with

		Remote	Kinnect	Leap M.	Buttons	Smartph.	Speech R.	Speech S.	Display
Deaf U.	Adequateness	**0**	**0**	**0**	**0**	**0,867**	**0**	**0,299**	**0,91**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0,96	0,95	0,935	0,98	0,929	0	0	0,95
	Physical	0,407	0,798	0,553	0,528	0,497	0,746	0,742	0,697
	Usage	0,882	0,882	0,551	0,882	0,882	0,882	0,882	0,882
Blind U.	Adequateness	**0**	**0**	**0**	**0**	**0,255**	**0**	**0,902**	**0,272**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0,96	0,95	0,935	0,98	0	0,96	0,96	0
	Physical	0,397	0,798	0,553	0,528	0,449	0,746	0,604	0,559
	Usage	0,882	0,882	0,551	0,882	0,882	0,882	0,882	0,882
Elderly 1	Adequateness	**0**	**0**	**0**	**0**	**0,254**	**0**	**0,568**	**0,538**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0,96	0,95	0,935	0,98	0	0	0,432	0,396
	Physical	0,403	0,759	0,514	0,489	0,455	0,707	0,67	0,625
	Usage	0,864	0,864	0,534	0,864	0,864	0,864	0,864	0,864
Elderly 2	Adequateness	**0**	**0**	**0**	**0**	**0,198**	**0**	**0,873**	**0,259**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0	0,656	0,348	0,98	0	0,96	0,96	0
	Physical	0,96	0,761	0,516	0,491	0,469	0,709	0,533	0,488
	Usage	0,417	0,476	0,179	0,864	0,276	0,394	0,693	0,864
Kid	Adequateness	**0**	**0**	**0**	**0**	**0,812**	**0**	**0,892**	**0,902**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0,96	0,656	0,348	0,98	0,929	0,96	0,924	0,95
	Physical	0,41	0,761	0,516	0,491	0,462	0,709	0,707	0,662
	Usage	0,854	0,731	0,216	0,854	0,385	0,58	0,854	0,954
Upper Body Impaired	Adequateness	**0**	**0**	**0**	**0**	**0,854**	**0**	**0,904**	**0,89**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0	0	0	0,636	0,929	0,96	0,96	0,95
	Physical	0,352	0,798	0,553	0,528	0,404	0,746	0,605	0,56
	Usage	0,892	0,892	0,614	0,892	0,892	0,892	0,892	0,892
Cognitive Impaired	Adequateness	**0**	**0**	**0**	**0**	**0,803**	**0**	**0,251**	**0,892**
	Interaction	0	0	0	0	1	0	1	1
	Modality	0,924	0,95	0,935	0,98	0,929	0	0	0,95
	Physical	0,404	0,798	0,553	0,528	0,456	0,746	0,605	0,56
	Usage	0,905	0,436	0,23	0,905	0,301	0,377	0,603	0,905

Table 6.11: Results of the FIO query looking for FIOs with output support and high level of adequateness regarding the Ideal FIO specifications of table 6.9.

Figure 6.12: The OMNI UI adapted to the Deaf User scenario. The Kinect and the wall buttons can be used to control system (change channels, answer questions). The display is used to output notification messages, and the colored lights are used to request the focus of the user when a notification is displayed.

many difficulties regarding hearing and vision abilities, thus the result obtained for Dandelion is that neither of the IRs is going to be very adequate, because the modality requirements (granularity) of all of them are higher than the modality support of the usage scenario. For example, the Ideal FIO for the Elderly 1 has a value of 2,537, which is in the edge of being considered as "not supported", thus, even if the Display FIO requires a low video granularity, it has a value of 5,0, so finally, the overall adequateness value for the Display is 0,538, which means "low adequateness". Unfortunately, in this case, the resulting UI is not going to be very adequate for the scenario, but as there are no more FIOs available, Dandelion is going to build the least bad UI possible with the available resources.

Answering Notifications

OMNI operates by proactively asking the user what to do next, thus, once the OMNI system shows a message to the user, she is requested to provide an answer which can be either a "Yes" or a "No". This interaction is modeled in the abstract UI as two different trigger interaction facets, thus, when the user select "Yes" or "No", the OMNI system logic will receive different callbacks triggering the user action.

Like in the previous case, we have reused the FIOs shown in table 6.3 that we have already used for the EMP example. Table 6.12 shows the FIO list resulting from querying the FIO repository for a list of FIOs that support output and are adequate FIOs for each one of the different OMNI usage scenarios.

As can be seen, as in the case of the Ideal FIO specifications, there are important differences in adequateness between the multiple FIOs and each usage scenario. Some usage scenarios, like the Blind User, have very high adequateness in all the FIOs, because none of them depend on modalities that require a high level of visual abilities. The contrary happens with the user with Upper Body impairments. Almost all of the FIOs use touch or keyboard modalities, which, looking at table 6.9, are not well supported by the scenario, thus the Speech Recognition FIOs has the greatest value of adequateness for that scenario.

6.5.3 Environment and User Adaptation: EvacUI

As previously introduced in section 4.2.3, the EvacUI user interface is in charge of providing guidance to the passengers of a ship during an emergency evacuation process.

A passenger ship is a very large and complex Human Interaction Environment, where the EvacUI must deal with many different physical environments (cabins, corridors, decks, etc.), changing conditions during the evolution of an emergency (fire, smoke, flooded areas, etc.), and a large variety of users (elderly people, disabled people, kids, etc.). The goal of the EvacUI is to provide those users (the passengers) with accurate directions of the path to follow to leave the ship safely, and for that purpose, it uses a series of interaction devices distributed along the ship, like information displays, symbolic signals, sound signals, speech synthesizing through the public address system (PA), or even the passenger's smartphone displays.

A key characteristic of EvacUI is that, during the evolution of an emergency, such as a fire or a flooding, the conditions of the ship are constantly changing, thus, an UI adequate for one usage scenario can be rendered invalid due to, for example, a lighting failure or the presence of smoke. Because of that, EvacUI must use those devices complementarily and redundantly in order to provide directions regardless of the conditions of the physical environment and the abilities of the users.

For illustration and demonstration purposes, and in order to keep this description simple, we are going to explore only a small subset of the EvacUI in charge of notifying the next direction each user must follow. Furthermore, we have limited our example to just four different physical environments and five representative user profiles, so that combining all of them, we have twenty different usage scenarios.

For the physical environments, we have imagined a ferry ship, and we selected four types of physical environments to which the passengers can access (see the tables 6.13 for detailed information about the environment and scene

		Remote	Kinnect	Leap M.	Buttons	Smartph.	Speech R.	Speech S.	Display
Deaf U.	**Adequateness**	**0,873**	**0,925**	**0,936**	**0,904**	**0,867**	**0,3**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,95	0,92	0,98	0,929	0	0	0,95
	Physical	0,407	0,798	0,553	0,528	0,497	0,746	0,742	0,697
	Usage	0,882	0,882	0,551	0,882	0,882	0,882	0,882	0,882
Blind U.	**Adequateness**	**0,873**	**0,925**	**0,836**	**0,904**	**0,861**	**0,924**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,95	0,92	0,98	0,929	0,96	0,96	0,618
	Physical	0,397	0,798	0,553	0,528	0,449	0,746	0,604	0,559
	Usage	0,882	0,882	0,551	0,882	0,882	0,882	0,882	0,882
Elderly 1	**Adequateness**	**0,871**	**0,917**	**0,828**	**0,896**	**0,865**	**0,292**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,95	0,92	0,98	0,929	0	0,432	0,685
	Physical	0,403	0,759	0,514	0,489	0,455	0,707	0,625	0,625
	Usage	0,864	0,864	0,534	0,864	0,864	0,864	0,864	0,864
Elderly 2	**Adequateness**	**0,873**	**0,688**	**0,195**	**0,897**	**0,445**	**0,869**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,656	0	0,98	0,381	0,96	0,96	0,434
	Physical	0,417	0,761	0,516	0,491	0,469	0,417	0,533	0,488
	Usage	0,864	0,476	0,179	0,864	0,276	0,864	0,693	0,864
Kid	**Adequateness**	**0,871**	**0,714**	**0,199**	**0,896**	**0,535**	**0,888**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,96	0,656	0	0,98	0,504	0,96	0,924	0,95
	Physical	0,41	0,761	0,516	0,491	0,462	0,709	0,707	0,662
	Usage	0,854	0,731	0,216	0,854	0,385	0,58	0,854	0,854
Upper Body Impaired	**Adequateness**	**0,242**	**0,309**	**0,244**	**0,682**	**0,249**	**0,925**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0	0	0	0,636	0	0,96	0,96	0,95
	Physical	0,352	0,798	0,553	0,528	0,404	0,746	0,605	0,56
	Usage	0,892	0,892	0,892	0,892	0,892	0,892	0,892	0,892
Cognitive Impaired	**Adequateness**	**0,851**	**0,88**	**0,804**	**0,906**	**0,198**	**0,249**	**0**	**0**
	Interaction	1	1	1	1	1	1	0	0
	Modality	0,924	0,95	0,92	0,98	0	0	0	0,95
	Physical	0,404	0,798	0,553	0,528	0,456	0,746	0,605	0,56
	Usage	0,905	0,436	0,23	0,905	0,301	0,377	0,603	0,905

Table 6.12: FIO list provided by the FIO Repository for the query looking for FIOs supporting one ACTION interaction facet and adequate to the Ideal FIOs of the different usage scenarios of the OMNI example.

	Noise	Visiblity			Motion		Climate			Type	Situation	Space
	ambient	visibility	contrast	lighting	motion	vibration	temp.	humidity	wind	type	situation	space
Cabin	2,0	7,0	4,0	5,0	0,0	0,0	20,0	65,0	0,0	7,0	0,0	1.5
Corridor	4,0	5,0	3,0	6,0	4,0	4,0	20,0	80,0	0,0	7,0	3,0	2.5
Deck	6,0	5,0	7,0	6,0	6,0	6,0	15,0	90,0	20,0	5,0	5,5	6,5
Vehicle Room	8,0	3,0	3,0	3,0	5,0	6,0	18,0	75,0	0,0	7.5	0,0	8,0

	Scene						
	usr. count	act. type	act. mode	act. style	hands	vision	attention
Cabin	1,0	10,0	8,0	0,0	0,0	0,0	4,0
Corridor	1,0	10,0	5.5	7,0	0,0	0,0	4,0
Deck	1,0	10,0	4,0	7,0	0,0	0,0	4,0
Vehicle Room	1,0	10,0	8,0	0,0	0,0	0,0	4,0

Table 6.13: Environment and scene profiles for the EvacUI ferry ship Human Interaction Environment example.

profiles):

- *Cabin.* Some passengers are in private cabins.
- *Corridor.* The different corridors of the ship that connect other physical environments between them.
- *Deck.* The open-air deck of the ship.
- *Vehicle Room.* The room where the passenger vehicles are parked inside the ship.

Regarding the user profiles (see table 6.8), we have reused four of the users from the previous example, and we have added one more user profile, in this case representing what we consider the standard user that will probably account for the majority of the passage.

For demonstration purposes, in this example, the environment characteristics are static for each usage scenario, but it is important to bear in mind that in a real scenario, with Dandelion operating at run-time to adapt the UI to context changes in real-time, the environment characteristics will change dynamically with the state of the ship and the emergency. As we have seen in section 6.4, it is the responsibility of the system to feed Dandelion with information about those context changes by providing updated and accurate context models.

With this context in mind, table 6.15 displays the Ideal FIOs that Dandelion would generate for each one of the twenty example usage scenarios of the EvacUI. Examining the results, we can see notable changes in all the aspects (modalities, physical, and usage specification) when the usage scenario is

		Deaf U.	Blind U.	Elderly 1	Cogn. Imp.	Stndrd.
Hearing	hearing	2.0	10.0	5.5	10.0	8.5
Vision	vision_ability	10.0	2.0	5.5	10.0	8.5
	vision_field	10.0	1.5	4.5	10.0	8.5
Cognitive	language_rec	10.0	10.0	7.5	4.0	10
	language_prod	10	10.0	9.0	4.0	10
	abstract_symbol	10.0	10.0	10.0	6.0	10
	attention	8.5	8.5	8.5	4.0	10.0
	processing_speed	10.0	10.0	10.0	5.0	10
	work_mem	10.0	10.0	10.0	6.0	10
	long_term_mem	10.0	10.0	10.0	6.0	10
Motor	speech_artic.	6	10	7.5	4.0	10
	finger_preciss.	9.0	9.0	9.0	10.0	10
	hand_preciss.	9.0	9.0	9.0	10.0	10
	arm_preciss.	9.0	9.0	9.0	10.0	10
	contact_grip	9.0	9.0	9.0	10.0	10
	pinch_grip	9.0	9.0	9.0	10.0	10
	clench_grip	9.0	9.0	9.0	10.0	10
	hand-eye_coord	8.0	8.0	8.0	5.0	8.5
ICT	ict_literacy	7.0	7.0	7.0	2.0	5.0
	ict_anxiety	0.0	0.0	0.0	0	0
Personal	Age	50	50	75	30	40

Table 6.14: Detailed description of the user profiles defined for the EvacUI example.

modified. For example, the recommended distance of usage varies considerably from the Cabin usage scenarios to the Corridor ones, mainly because of the size of the environment and the differences in scene modality and style. There are also notable differences in the modalities specifications, for example, the video modality has a granularity of 9,025 for the Deaf User in the Cabin environment, but it has a value of 4,677 for the the same user in the Deck environment, a large outdoor environment where a display with high detail information can be difficult to see.

As previously indicated, in this example, we are exploring a small subset of the EvacUI user interface, the part that is in charge of informing the users about the next immediate direction they must follow. We have implemented five different examples of FIOs that could be used for that purpose in the EvacUI:

- *Smartphone.* It uses the display to show direction arrows to the user.
- *Sound Production.* It uses speakers distributed along the ship to produce different sound signals that indicate the direction to follow.
- *Speech Synthesizing.* Implemented with Festival. It uses speech to notify users about the direction to follow.

		Cabin					Corridor				
		Deaf U.	Blind U.	Elderly 1	Mental	Standard	Deaf U.	Blind U.	Elderly 1	Cogn. Imp.	Standard
MODALITIES	sound_prod	0,970	9,025	3,065	9,025	8,377	0,970	9,025	3,065	9,025	8,377
	speech_rec	0,970	9,025	1,071	0,970	9,025	0,970	9,025	1,071	0,970	9,025
	speech_prod	0,970	9,025	2,102	0,970	8,377	0,970	9,025	2,102	0,970	8,377
	touch	7,457	7,457	7,457	0,970	8,377	7,457	7,457	7,457	0,970	8,377
	wimp	5,890	3,593	6,008	2,080	6,681	5,890	3,593	6,008	2,080	6,681
	gesture	9,025	9,025	9,025	9,025	9,025	9,025	9,025	9,025	9,025	9,025
	keyboard	9,025	6,912	7,128	3,750	9,025	9,025	6,912	7,128	3,750	9,025
	symbol	9,025	0,970	2,102	1,018	7,827	7,913	3,078	3,660	3,310	7,229
	video	9,025	0,970	2,538	9,025	7,187	9,025	0,970	2,538	9,025	7,187
PHYSICAL	size	7,913	7,913	7,913	7,913	7,913	9,025	9,025	9,025	9,025	9,025
	status	6,250	6,250	6,250	6,250	6,250	6,250	6,250	6,250	6,250	6,250
	distance	208,239	208,239	208,239	208,239	208,239	106,667	106,667	106,667	106,667	106,667
SHAPES	display	8,905	4,964	6,860	4,987	8,396	8,905	4,964	6,860	4,987	8,396
	button	3,909	6,086	5,585	6,250	3,750	3,909	6,086	5,585	6,250	3,750
	remote	7,898	7,520	7,699	7,709	7,898	7,519	7,519	7,519	7,519	7,519
	toy	3,750	3,750	3,750	3,750	3,750	3,750	3,750	3,750	3,750	3,750
	embedded	9,025	9,025	7,913	9,025	9,025	9,025	7,913	7,913	7,914	9,025
	keyboard	6,896	6,226	6,485	3,750	6,912	6,896	6,226	6,485	3,750	6,912
	surface	6,861	6,861	6,861	2,079	6,913	8,494	8,494	8,494	1,189	8,607
USAGE	age	50	50	75	30	40	50	50	75	30	40
	ict	7	7	7	2	5	7	7	7	2	5

		Deck					Vehicle Room				
		Deaf U.	Blind U.	Elderly 1	Mental	Standard	Deaf U.	Blind U.	Elderly 1	Cogn. Imp.	Standard
MODALITIES	sound_prod	0,970	5,328	2,759	5,328	5,143	0,970	5,796	3,065	5,796	5,568
	speech_rec	0,970	5,328	1,018	0,970	5,328	0,970	5,796	1,071	0,970	5,796
	speech_prod	0,970	5,328	1,927	0,970	5,143	0,970	5,796	2,102	0,970	5,568
	touch	6,044	6,044	6,044	2,053	6,666	6,044	6,044	6,044	2,053	6,666
	wimp	3,593	3,593	4,143	2,080	4,409	5,890	3,593	6,008	2,080	6,681
	gesture	6,963	6,963	6,963	6,963	6,963	9,025	9,025	9,025	9,025	9,025
	keyboard	6,763	6,728	6,763	4,974	6,763	7,940	6,728	6,872	4,974	7,940
	symbol	7,913	3,078	3,660	3,310	7,229	7,913	3,078	3,660	3,310	7,229
	video	4,677	2,080	2,080	4,677	4,409	9,025	0,970	2,538	9,025	7,187
PHYSICAL	size	6,912	6,912	6,912	6,912	6,912	9,025	9,025	9,025	9,025	9,025
	status	3,750	3,750	3,750	3,750	3,750	6,250	6,250	6,250	6,250	6,250
	distance	201,394	201,394	201,394	201,394	201,394	253,030	253,030	253,030	253,030	253,030
SHAPES	display	5,623	4,107	4,107	4,006	6,337	7,913	3,078	5,000	5,798	7,817
	button	7,588	7,617	7,588	8,967	6,778	6,250	7,617	7,486	8,967	5,289
	remote	7,588	7,617	7,588	7,901	7,588	7,818	7,638	7,507	7,818	7,818
	toy	3,750	3,750	3,750	3,750	3,750	3,750	3,750	3,750	3,750	3,750
	embedded	9,025	7,913	7,567	7,914	9,025	9,025	7,913	7,913	7,914	9,025
	keyboard	7,360	7,317	7,360	6,180	6,497	7,913	7,317	7,481	6,180	7,190
	surface	5,000	5,000	5,000	2,719	5,883	5,020	5,020	5,020	2,699	5,910
USAGE	age	50	50	75	30	40	50	50	75	30	40
	ict	7	7	7	2	5	7	7	7	2	5

Table 6.15: Ideal FIO specifications generated by Dandelion for the different EvacUI usage scenarios associated to the Cabin, Corridor, Deck, and Vehicle Room physical environments.

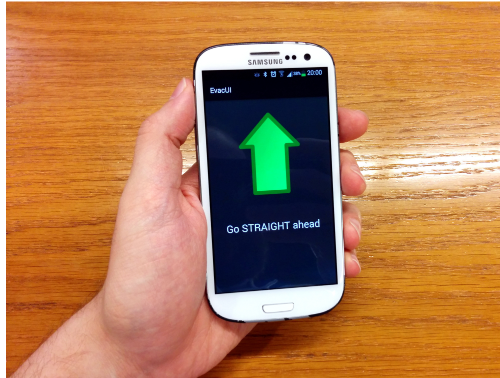


Figure 6.13: An example of FIO for the EvacUI user interface example. It uses a smartphone display to show direction information using symbol and language production modalities. Furthermore, it also provides another interaction facet using speech production. In the second image, there are two different FIOs.

- *Display*. It uses public information displays distributed along the ship to show direction information to the users.
- *Signal*. They are very low resolution and highly bright displays capable of showing only five different direction signals (forward, reverse, left, right, and forbidden).

The detailed description of those five FIOs is shown in table 6.16.

The abstract UI definition of the part of EvacUI in charge of providing immediate directions is quite simple, it has just one Abstract Interaction Unit with two abstract interaction facets, one of type output and one of type focus. EvacUI relies on multiple instances of this abstract UI, one for each physical environment that exists inside the whole HIE (the ship). Therefore, Dandelion would be managing multiple UIs simultaneously, adapting each one of them to a particular combination of user, environment and scene profiles.

Furthermore, as the information provided by this UI is critical, it is important that it can operate with redundant resources whenever is possible. As introduced in previous chapters, using the Dandelion UI Controller API, it is possible to provide Dandelion with different "Interaction Hints" that allow developers to customize, to some extent, the behavior of Dandelion, like customizing the operation of particular FIOs, or customizing the result of the FIO selection process. In this case, as shown in the code snippet displayed in algorithm 6.3, we are indicating Dandelion to implemented the output interaction facet using redundancy, so that Dandelion will try to select more than one FIO when building the Final UI. Dandelion will take advantage of the disaggregated metrics in

		Smartph.	Sound P.	Speech S.	Display	Signal
INTERACT.	cardinality	1	1	1	1	1
	data type	-	-	-	-	-
	type	FOCUS, OUT	FOCUS, OUT	FOCUS, OUT	FOCUS, OUT	FOCUS, OUT
MODALITIES	sound_prod	0,0	5.0	0,0	0,0	0,0
	speech_rec	0,0	0,0	0,0	0,0	0,0
	speech_prod	0,0	0,0	6,0	0,0	0,0
	touch	0,0	0,0	0,0	0,0	0,0
	wimp	0,0	0,0	0,0	0,0	0,0
	gesture	0,0	0,0	0,0	0,0	0,0
	keyboard	0,0	0,0	0,0	0,0	0,0
	symbol	0,0	0,0	0,0	0,0	7.0
	video	7.0	0,0	0,0	4.0	0,0
PHYSICAL	size	2.0	8,0	8,0	5.5	4,0
	status	1.5	3,0	3,0	10,0	10,0
	distance	80,0	300,0	300,0	150,0	300,0
SHAPES	display	1	0	0	1	1
	button	0	0	0	0	0
	remote	0	0	0	0	0
	toy	0	0	0	0	0
	embedded	0	1	1	0	1
	keyboard	0	0	0	0	0
	surface	0	0	0	0	1
USAGE	age	30	20	20	30	30
	ict	7,0	1,0	4,0	3,0	2,0

Table 6.16: Detailed descriptions of the FIOs used for the EvacUI demonstration example.

order to select multiple complementary FIOs according to the CARE properties.

Within EvacUI, when a context change is detected, Dandelion uses the associated context profiles to query the FIO Repository for a list of adequate FIOs that support output and focus interaction facets. Table 6.17 shows the results returned by the FIO Repository for the twenty different usage scenarios proposed in this example.

As can be seen, the results are coherent with the previous results of the Ideal FIOs. For example, the Display and Signal FIOs are considered the best option for the Deaf User in all physical environments, while sound based FIOs are very discouraged. Almost the contrary happens with the user with a severe visual impairment. In the case of the Elderly user, due to her multiple impairments, there are no FIOs with high adequateness values, but the FIO repository indicates that the best options would be to use the Sound Signaling and Display FIOs with low granularity. Finally, an interesting result is to see that the standard user has a high value of adequateness in almost all FIOs and usage scenarios, because she does not have any impairment affecting her capacity to interact with the system. Nevertheless, there are some minor differences between each physical environment. For example, while Sound Production and Speech Synthesis FIOs are considered the best options for the Cabin, the Display and

Algorithm 6.3 Code snippet showing an example of how to use the Interaction Hints mechanism to indicate Dandelion that one particular interaction must be implemented in a redundant way.

```
AbstractInteractionUnit directionAIU =
    app.getAbstractUI().getAbstractInteractionUnitById("DirectionMessage");

DataInputOutputSupport dirOutput =
    (DataInputOutputSupport) lblMsg.getInteractionSupportElementById("direction");

HashSet<FuzzyVariable> interactionHints =
    new HashSet<FuzzyVariable>(1);
interactionHints.add(
    new FuzzyVariable("redundancy", "redundancy_level", 10.0));
uic.manageAbstractInteractionUnit(
    dirOutput, directionAIU, interactionHints);
```

Signaling FIOS are the most adequate options in the other three environments, mainly due to ambient noise constraints.

Finally, the last step in the adaptation to context process would be to build the final UI. This is performed by selecting, among the FIOs returned by the repository, those that are considered more adequate for the scenario, and then establish a mapping in the EvacUI controller.

As previously indicated, we have requested Dandelion to produce a final UI with redundancy for the output interaction facet used to notify directions to the users. Therefore, the User Interface Builder is going to try to select more that one FIO. The UIB will use the CARE properties and the disaggregated metrics to select FIOs that employ different modalities and have different physical and usage characteristics. For example, for the Elderly and Cognitive Impaired user, it will select the Sound Production and the Display FIOs. For the Deaf and Blind users redundancy cannot be supported because the FIOs that use complementary modalities have very low (not supported) adequateness values.

As a final note regarding the ICA implementation of Dandelion, it is important to highlight that it not only is able to modify the UI at run-time, thanks to the decoupling provided by the IMA+ILA, but also in real-time. The median time required to select and configure a FIO for a particular interaction facet is about 24-30 ms, with 18-23 ms required to build the Ideal FIO, 3-5 ms dedicated to query the FIO repository, and 1-2 ms to establish the FIO mapping. Therefore, for example, in the case of the EMP UI, which has 14 interaction facets, it will take about 400 ms to adapt the UI. For a more complex UI, with

		Cabin					Corridor				
		Smartph.	Sound P.	Speech S.	Display	Signal	Smartph.	Sound P.	Speech S.	Display	Signal
Deaf U.	Adequateness	**0,870**	**0,299**	**0,299**	**0,900**	**0,876**	**0,866**	**0,283**	**0,283**	**0,893**	**0,871**
	Interaction	1,000	1,000	1,000	1,000	1,000	1	1	1	1	1
	Modality	0,930	0,000	0,000	0,960	0,930	0,93	0	0	0,96	0,93
	Physical	0,506	0,741	0,741	0,583	0,549	0,479	0,633	0,633	0,537	0,515
	Usage	0,892	0,882	0,882	0,882	0,892	0,892	0,882	0,882	0,882	0,892
Blind U.	Adequateness	**0,244**	**0,917**	**0,910**	**0,255**	**0,272**	**0,24**	**0,895**	**0,888**	**0,248**	**0,402**
	Interaction	1,000	1,000	1,000	1,000	1,000	1	1	1	1	1
	Modality	0,000	0,950	0,940	0,000	0,000	0	0,95	0,94	0	0,213
	Physical	0,368	0,741	0,741	0,445	0,549	0,342	0,594	0,594	0,399	0,496
	Usage	0,892	0,882	0,882	0,882	0,892	0,892	0,882	0,882	0,882	0,892
Elderly 1	Adequateness	**0,252**	**0,629**	**0,292**	**0,621**	**0,264**	**0,248**	**0,613**	**0,276**	**0,614**	**0,434**
	Interaction	1,000	1,000	1,000	1,000	1,000	1	1	1	1	1
	Modality	0,000	0,519	0,000	0,551	0,000	0	0,519	0	0,551	0,265
	Physical	0,434	0,702	0,702	0,512	0,510	0,408	0,594	0,594	0,465	0,496
	Usage	0,872	0,865	0,865	0,865	0,872	0,872	0,865	0,865	0,865	0,872
Cognitive Imp.	Adequateness	**0,792**	**0,919**	**0,255**	**0,851**	**0,282**	**0,788**	**0,897**	**0,233**	**0,844**	**0,425**
	Interaction	1,000	1,000	1,000	1,000	1,000	1	1	1	1	1
	Modality	0,930	0,950	0,000	0,960	0,000	0,93	0,95	0	0,96	0,236
	Physical	0,369	0,741	0,741	0,446	0,549	0,342	0,594	0,594	0,4	0,476
	Usage	0,325	0,906	0,436	0,603	1,000	0,325	0,906	0,436	0,603	1
Standard	Adequateness	**0,816**	**0,918**	**0,911**	**0,898**	**0,879**	**0,812**	**0,874**	**0,891**	**0,895**	**0,902**
	Interaction	1,000	1,000	1,000	1,000	1,000	1	1	1	1	1
	Modality	0,930	0,950	0,940	0,960	0,930	0,93	0,93	0,96	0,94	0,95
	Physical	0,488	0,741	0,741	0,565	0,549	0,462	0,515	0,519	0,633	0,633
	Usage	0,383	0,893	0,893	0,893	0,922	0,383	0,922	0,893	0,893	0,893

		Deck					Vehicle Room				
		Smartph.	Sound P.	Speech S.	Display	Signal	Smartph.	Sound P.	Speech S.	Display	Signal
Deaf U.	Adequateness	**0,543**	**0,293**	**0,293**	**0,886**	**0,868**	**0,859**	**0,289**	**0,289**	**0,883**	**0,876**
	Interaction	1	1	1	1	1	1	1	1	1	1
	Modality	0,442	0	0	0,96	0,93	0,93	0	0	0,96	0,93
	Physical	0,446	0,698	0,698	0,489	0,495	0,437	0,669	0,669	0,471	0,551
	Usage	0,892	0,882	0,882	0,882	0,892	0,892	0,882	0,882	0,882	0,892
Blind U.	Adequateness	**0,248**	**0,905**	**0,81**	**0,531**	**0,396**	**0,229**	**0,9**	**0,892**	**0,234**	**0,404**
	Interaction	1	1	1	1	1	1	1	1	1	1
	Modality	0	0,95	0,805	0,427	0,213	0	0,95	0,938	0	0,213
	Physical	0,393	0,659	0,659	0,436	0,456	0,268	0,63	0,63	0,302	0,512
	Usage	0,892	0,882	0,882	0,882	0,892	0,892	0,882	0,882	0,882	0,892
Elderly 1	Adequateness	**0,246**	**0,575**	**0,284**	**0,529**	**0,426**	**0,237**	**0,619**	**0,281**	**0,6**	**0,436**
	Interaction	1	1	1	1	1	1	1	1	1	1
	Modality	0	0,448	0	0,427	0,265	0	0,519	0	0,551	0,265
	Physical	0,393	0,647	0,647	0,436	0,444	0,335	0,63	0,63	0,369	0,512
	Usage	0,872	0,865	0,865	0,865	0,872	0,872	0,865	0,865	0,865	0,872
Cognitive Imp.	Adequateness	**0,478**	**0,907**	**0,243**	**0,849**	**0,422**	**0,791**	**0,903**	**0,238**	**0,844**	**0,43**
	Interaction	1	1	1	1	1	1	1	1	1	1
	Modality	0,442	0,95	0	0,96	0,236	0,93	0,95	0	0,96	0,236
	Physical	0,389	0,659	0,659	0,432	0,456	0,363	0,63	0,63	0,397	0,512
	Usage	0,325	0,906	0,436	0,603	1	0,325	0,906	0,436	0,603	1
Standard	Adequateness	**0,463**	**0,785**	**0,871**	**0,89**	**0,912**	**0,808**	**0,907**	**0,856**	**0,883**	**0,879**
	Interaction	1	1	1	1	1	1	1	1	1	1
	Modality	0,391	0,756	0,93	0,96	0,95	0,93	0,95	0,872	0,96	0,93
	Physical	0,471	0,698	0,495	0,514	0,698	0,434	0,669	0,669	0,468	0,551
	Usage	0,383	0,893	0,922	0,893	0,893	0,383	0,893	0,893	0,893	0,922

Table 6.17: FIO adequateness results for the different EvacUI usage scenarios related to Cabin, Corridor, Deck, and Vehicle Room environments.

for example 50 independent interaction facets, it will take about 1.2-1.5 seconds to build a new final UI adapted to a new context.

6.5.4 Summary

The end goal of the Interaction Context Abstraction layer proposed by the TIAF is to isolate AmI user interfaces from the particular context in which they are being executed. In this chapter, we have presented how we have implemented this layer in the Dandelion framework.

In this section, we have seen that, by taking advantage of the features provided by the IMA and ILA layers, the job of the ICA is reduced to performing a adequate selection of Interaction Resources to fulfill the interaction requirements of an abstract UI and the natural interaction constraints of the usage scenario. For that purpose, the Dandelion ICA implementation relies on a series of fuzzy inference systems that processes a set of models with fuzzy information about the scenario and the user interaction requirements. The result of this processing is a selection of IRs that are considered adequate for a particular combination of abstract UI and usage scenario. In order to build a final adapted to the scenario, this particular selection of IRs is notified to the UI Controller at run-time, so that it can start routing the GIP events to the new set of IRs, and therefore, modify the UI at run-time without affecting the business logic of the application.

It is important to mention that the selection of IRs, and the final UI of an application, depends exclusively on information that is decoupled from the application code: interaction requirements of the application (the abstract UI), the IRs available in the physical environment (accessible through the FIO Repository), and the dynamic characteristics of the usage scenario (user, environment, and scene profiles). Therefore, it is possible to build a final UI for any kind of usage scenario (as long as it is supported by the models), even for scenarios not predicted or imagined during the development of the application.

The UI adaptation examples shown in section 6.5 have demonstrated the viability of the implementation of an ICA layer that allows AmI UIs to react to context changes and modify their UI at run-time, adapting their user interfaces to the usage constraints introduced by the new context. A key aspect of this adaptation process is that it allows very deep adaptations, because it is possible to change the physical IRs that perform each particular user interaction primitive, thus permitting dramatic changes in the shape of the UI, including modalities, physical shape and look, or even the physical location.

In this regard, it is important to note that the implementation of the ICA described in this chapter must be understood as a reference implementation

with the objective of demonstrating the validity of the TIAF concept as a complete solution for the development of context-adaptive Physical User Interfaces for Ambient Intelligence systems. Thus, while it is a complete and functional implementation of the ICA, it would be possible to extend the models and algorithms so that Dandelion can provide better adaptation support in many more contexts of use.

Finally, it is important to mention that, apart from enabling Dandelion UIs to adapt, at run-time, to context changes, this implementation of the ICA provides four additional results that are relevant by themselves:

- A model to describe physical interaction resources using fuzzy logic, including their modalities and physical shapes.
- A model to specify the interaction constraints and requirements of a particular usage scenario.
- A metric to compare the adequateness of one particular physical interaction resource regarding a specific usage scenario. Furthermore, the proposed adequateness metric allows the utilization of the CARE properties to reason about the multimodality of the IRs used to build a final UI.
- The adequateness metric can also be used as a similarity metric in order to compare physical interaction resources.

Chapter 7

Conclusions and Future Work

"The beginning of knowledge is the discovery of something we do not understand."

Frank Herbert

To conclude this dissertation, the first section of this chapter summarizes the main results and original contributions of this research work, with special emphasis on the conclusions that can be drawn from them. Finally, the last section introduces different questions and topics that have been raised during the realization of this PhD. thesis and are yet open to research and development.

Conclusions

In chapter 2, we introduced the main goal of this PhD work: to improve the support for ubiquitous operation and user mobility in Ambient Intelligence (AmI) and Ubiquitous Computing (UC) systems. In that same chapter, we also divided that broad goal into two fundamental sub-goals:

- To enhance the migrability of distributed physical user interfaces, so that they can be more easily deployed on a variety of scenarios and migrated between them.
- To facilitate the development of DPUIs capable of adapting their interaction with the user to the requirements and characteristics of different

scenarios (users, physical environments, etc.).

In order to fulfill these two sub-goals, this thesis has produced two main results. First, the Threefold Interaction Abstraction Framework (TIAF), a conceptual abstraction framework addressing those two sub-goals in order to provide a technological solution for the development of ubiquitous user interfaces for AmI and UC systems. Second, the Dandelion framework, a reference implementation of the TIAF conceptual proposal, providing a complete and functional development framework that facilitates the development of highly portable and plastic AmI user interfaces and Distributed Physical User Interfaces.

Within the TIAF conceptual framework, we have addressed those two sub-goals by defining three different levels of abstraction required to decouple AmI UI developers from the complexities and particularities of Distributed Physical User Interfaces and facilitate the development of UIs capable of operating in different scenarios, with different constraints, while providing an adequate user interaction experience to the users:

- The first abstraction level, known as the Interaction Modality Abstraction level (IMA),is in charge of decoupling AmI UI developers and their code from the diverse modalities, technologies, and even APIs used to build a user interface. Thanks to this abstraction, the same UI code can be reused in different scenarios where different interaction resources must be used. Furthermore, it is important to mention that this abstraction makes UI code completely agnostic of the modalities used by the interaction resources, thus facilitating the usage of very different physical devices as interaction resources.
- The second abstraction level, known as Interaction Location Abstraction level (ILA), is dedicated to decouple AmI UI code from the physical distribution of the user interface. It allows the transparent distribution of any single element of the UI in such a way that each element of the UI can be realized by any remote interaction resource available.
- The third abstraction level, known as Interaction Context Abstraction level (ICA) is built on top of the two previous ones, and it is in charge of providing autonomous plasticity capabilities to AmI UIs. It decouples UI developers and code from the different contexts or usage scenarios. This decoupling is achieved by allowing the change, at run-time and in real time, of the interaction resources used to render each one of the UI elements.

With the Dandelion framework [Varela, 2013, Varela et al., 2014], we have provided a complete and functional implementation of each one of the three

TIAF abstraction levels. This particular implementation, as has been presented in chapters 5 and 6, addresses not only those two big sub-goals, but also the list of concrete objectives established in chapter 2.

Regarding the objectives of conceiving and implementing mechanisms to support the design and implementation of AmI interaction and business logic with low or no knowledge about the technologies, modalities, protocols, and APIs used by the IRs, section 5.6 has shown that the development framework provided by the IMA abstraction layer, and its Dandelion reference implementation, is generic enough to support very different modalities and devices, even unknown or unpredicted ones. This framework allows the implementation of user interfaces by declaratively describing its interaction requirements using a high-level abstraction model and implementing the UI logic on top of those abstract interaction components. The proposed abstraction model manages interactions concepts that are modality and technology agnostic, thus making the UI definition and behavior completely independent of the final implementation.

Furthermore, this capacity of the proposed solution to completely isolate the developers, and their code, from the particular implementation of the UI is what enables it to allow the deployment of the same AmI system with different realizations of the UI. As presented in subsection 5.3.3, the final UI implementation is not decided at development-time, but at deployment-time (or run-time if using the ICA), thus it is possible to deploy different UIs in different scenarios. Moreover, the implementation is easily performed by establishing a mapping, in an XML file, between abstract interaction elements and real interaction resources.

In regard to the objective of facilitating the development of distributed UIs capable of operating independently of the physical location of the IRs, the examples shown in section 5.6 have demonstrated that the Generic Interaction Protocol (GIP), together with the concept of using the Final Interaction Objects (FIO) as proxy-like abstraction components, allows the distribution of a Physical UI up to a fine-grained level. The provided implementation of the ILA abstraction layer and its core element, the GIP, allows the distribution of every single interaction element of the UI to one, or many, different interaction resources, which could be deployed in any physical location. Furthermore, the particular implementation of the GIP, using the STOMP messaging technology, makes the system easily compatible with a large number of hardware and software platforms, thus facilitating the integration of interaction resources into the framework. In addition, along with Dandelion, we have developed a device abstraction technology, UniDA, which, integrated in Dandelion, enables the UIs to be directly compatible with a large range of existing physical devices.

With reference to the goal of facilitating the prototyping and testing of different physical configurations of an UI, thus possibilitating a reduction in the costs of developing distributed physical user interfaces adapted to different use cases, the examples provided in section 5.6 have shown that Dandelion allows the easy change (just a mapping in an XML file), at deployment-time, of each physical element that builds up an UI. This feature can be exploited by developers, or installers, to easily test different implementations of an UI. Furthermore, the previously mentioned abstraction characteristics, together with this ability to easily change the implementation, allows the easy implementation of very different prototypes of an UI without modifying the system and UI code, thus effectively reducing the cost of developing user interfaces adapted to particular usages scenarios.

This easiness to change the implementation of Ambient Intelligence UIs, and in particular, physical distributed user interfaces, allows us to fulfill another one of the sub-goals established in chapter 2: provide support for the migration of AmI applications from one scene to another without requiring modifications in the system or interaction logic code. An AmI developer can setup one FIO-mapping file for each scenario, thus when the user changes from one scenario to another, the migration of the UI is easily performed by loading a new mapping file that changes, at run-time, the implementation of UI, so that the AmI application automatically starts using the new interaction resources available in the new physical environment.

Regarding the final sub-goal settled in chapter 2, to reduce the costs, in time and effort, of supporting different scenarios with the same AmI application by automatically adapting the UI to each scenario, the results presented in section 6.5 have shown that the TIAF conceptual design, and its Dandelion implementation, provide integrated support for autonomous UI plasticity, including adaptation to user, environment, and device characteristics. On the one hand, the UI decoupling capabilities previously introduced enable developers of AmI UIs, implemented with Dandelion, to modify the UI shape at run-time and without affecting the application. On the other hand, the ICA abstraction level design and implementation shown in subsection 4.3.2.3 and in chapter 6, allows Dandelion to react to context changes by modifying the shape of a DPUI without human intervention, thanks to an autonomous IR selection mechanism.

As previously introduced, the TIAF and GIP design simplify the process of implementing a DPUI to the point of just having to specify a mapping between abstract and physical interaction resources. Therefore, the job of a developer or installer building a final UI for a Dandelion system would consist in selecting, among the IRs available, one or many IRs for each interaction facet declared in

the abstract UI definition. This selection is the last point of coupling between developers/installers and the context, thus, in order to decouple developers from the interaction and the context characteristics, the ICA implementation is in charge of autonomously managing that mapping.

As shown in chapter 6, the Dandelion ICA implementation uses models and Computational Intelligence algorithms to autonomously drive the IR selection process. The whole process implementation is build on top of fuzzy logic technologies. First, three different context models, a user profile, an environment profile, and a scene profile, provide input information about the context using fuzzy characteristics. Second, a set of Fuzzy Inference Systems are used to exploit the information provided by the context models in order to generate a fuzzy specification of the characteristics that a FIO should have to be considered adequate for a particular scenario. And third, a fuzzy comparison measure, named FIO Adequateness metric, is introduced in subsection 6.4.2.1, providing a way to measure how adequate is to use a FIO in one particular scenario. As it is shown in section 6.5, by using this fuzzy FIO selection process, Dandelion is able to autonomously generate or adapt, at run-time and in real-time, a DPUI for a particular context of use.

In section 3.4, we have provided an extensive comparison of AmI and DPUI development frameworks, drawing the main conclusion that there were a lack of support for the particular problematics of Distributed Physical User Interfaces in the context of Ubiquitous Computing and Ambient Intelligence. The TIAF and Dandelion have been specifically designed and implemented to enable developers to build DPUIs for AmI systems, with special attention to be compliant with the features established in chapter 3 as critical characteristics for AmI UIs. Therefore, recalling the comparison table previously introduced in section 3.4, we have updated it with a new entry for Dandelion in order to show how it copes with the existing competition. This new table is shown in table 7.1 and, as can be seen, Dandelion, as a reference implementation of the TIAF conceptual solutions, performs quite well in almost all the characteristics evaluated.

As can be seen, Dandelion only presents poor performance in one of the characteristics analyzed, UI customization, and it is a result of the high level of abstraction introduced by the TIAF. Developers are so isolated from the final shape of the UI that it is difficult to provide them with capabilities to customize and modify that shape. While Dandelion provides some capabilities of this kind, mainly through the Interaction Hints mechanism, it only allows developers to specify a small set of customizations.

Regarding the other analyzed characteristics, as previously introduced in this chapter, section 6.5 has shown the UI plasticity characteristics of the TIAF

	Autonomous Plasticity			User Interface					Physical UI	
	User	Env.	Devices	Distribution	Dev.	Abstraction	Generation	Custom.	Modalities	Devices
MASP	No	No	Yes	Medium	Yes	High	Yes	Low	Low	None
Dynamo-AID	No	No	Yes	Medium	Yes	High	Yes	Low	Low	None
Egoki	Yes	No	Yes	None	Yes	High	Yes	Low	Low	None
SUPPLE	Yes	No	Yes	None	Yes	Low	Yes	Medium	Very Low	None
ICrafter	No	No	Yes	Low	No	Medium	Yes	Medium	Low	None
PUC	No	No	Yes	Low	No	Medium	Yes	Medium	Low	None
MARIA	Yes*	Yes*	Yes*	None	Yes	Medium	No	Medium	Low	None
UsiXML	Yes*	Yes*	Yes*	Low	Yes	Medium	No	Medium	Low	None
H. Automation	No	No	No	Low	No	Low	No	N/A	Very Low	Low
HomeKit	No	No	No	Low	No	Low	No	N/A	Low	Medium
Phidgets	No	No	No	None	No	Low	No	N/A	Medium	Low
VoodooIO	No	No	No	None	Yes	Low	No	Low	Low	Low
iStuff	No	No	No	Medium	Yes	Medium	No	Medium	High	Medium
EIToolkit	No	No	No	Medium	No	Medium	No	N/A	High	Medium
Dandelion	Yes	Yes	Yes	High	Yes	High	Yes	Low	High	High

*Plasticity is supported by transformation processes usally performed manually by the developers

Table 7.1: Comparison of Dandelion against the main available solutions for the development of Plastic Distributed Physical User Interfaces reviewed in chapter 3.

and Dandelion, demonstrating their capability to dynamically adapt the UI to the context. Furthermore, with the examples provided in section 5.6, we have demonstrated the UI distribution and modality decoupling capabilities of the solutions, allowing the developing of highly-distributed, modality and device agnostic AmI User Interfaces.

Finally, besides the evident results achieved with the TIAF and Dandelion, the work in Plastic DPUIs has produced various results that are relevant by themselves.

First, we have introduced a model, using fuzzy characteristics, to describe physical interaction resources, including their modalities, physical shapes, and user interaction capabilities. This model is independent from the rest of the solution, and it can be easily reused for other purposes. For example, the manufactures of interaction devices and software could use the model to provide their customers with a machine readable description of the IRs, which could be exploited by the customer's applications at run-time in order to, for example, recognize the IRs connected to the application. Moreover, we have also provided an independent FIO repository component that can be used to build a remotely accessible database of FIOs, thus enabling applications to detect the IRs available in one location, or even search for IRs that present some particular characteristics.

Second, we have designed a model to specify the interaction constraints and interaction resource requirements of a particular usage scenario. As in the previous case, this model can be reused outside the proposed solution, for example by AmI application developers in order to provide system installers with interaction resource requirements information.

And third, we have introduced a metric that, using the two previous cited description models, enables the calculation of how adequate is one particular physical interaction resource for its use in a specific usage scenario. This metric, in combination with the FIO repository, can be used by AmI system installers to receive advice of what IR to use in each case. Furthermore, it can also be used, for example, to guide developers during the prototyping of AmI UIs, providing them with information about what kind of IRs would be better for each scenario analyzed. Finally, it is relevant to highlight that this adequateness metric, and the FIO description model, are designed to support the utilization of the CARE properties, thus enabling prototype developers to reason about the multimodality of the IRs used to build a final UI. Moreover, the proposed metric can be easily used as IR similarity metric, allowing the comparison between different physical interaction resources.

Finally it is important to mention that all the software solutions implemented within this doctoral thesis have been released under an open source software license, and are publicly available for download, study, compare and use at http://github.com/GII/Dandelion and http://github.com/GII/UNIDA. We think that is important for software engineering thesis to provide functional implementations of the solutions proposed, not just because the direct availability of the solutions is a great contribution to the research and development community, in this case the Ambient Intelligence and Ubiquitous Computing community, but also because making the implementation available is the only way to enable the community to compare the proposed solution with others, either previously existing, or new ones.

Future Work

As in almost every research work, is not feasible to cover all the different topics at once, not only due to time limitations, but also because new problems are raised while others are being solved. In this PhD work we have designed, implemented and demonstrated an integral solution for the development of Plastic Physical Distributed User Interfaces for Ambient Intelligence systems. Nevertheless, while functional and complete, we have identified many different aspects where further research and development would be required.

Improve UI Customization Capabilities

First of all, UI abstraction technologies usually share a common drawback: a lack or reduced support for UI customization.

While UI abstraction technologies can considerably reduce the efforts, and thus the cost, of UI development and even, in cases like the solution proposed in this work, are able to enable UIs to operate in non predicted scenarios, they usually impose severe limitations on the capacities of developers to produce a fine customized user experience.

The solution proposed in this work incorporates some basic support for UI customization by using a "hints" mechanism, allowing developers to indicate small UI customizations like colors, sizes, or even the redundancy level of a particular interaction facet. Nevertheless, in order to achieve fine grained customization, developers are required to implement specific FIOs, designed with one or various particular scenarios in mind.

Additional research would be required in order to design new techniques to support a finer level of customization without compromising the decoupling between developers, their code, and the particular implementation of the UI for each scenario.

Improve UI Generation and Adaptation Capabilities

In this PhD work we have presented a complete conceptual framework supporting the autonomous generation and adaptation to context, at run-time, of UIs for Ambient Intelligence systems. Furthermore, we have provided and demonstrated a reference implementation of that conceptual framework. While has been demonstrated that this implementation is able to produce viable UIs for very different scenarios and UI requirements, it is true that its implementation, using Fuzzy Inference Systems, limits its capacity of generation and adaptation to the expert knowledge available in the different rule databases used.

An interesting future research regarding this topic would be to introduce some techniques for the autonomous generation of the rule databases. We think that a rule-based system is a good approximation for UI generation systems, because it allow developers and installers to get reasoned information about why a final UI has been implemented in one way or another. Nevertheless, the production of rules by experts is and arduous and prone to error process. The idea would be to generate the rule database using some Computational Intelligence techniques like Artificial Neural Networks or Evolutionary Computation. In this regard, it is important to highlight that, as presented in chapter 6, Dan-

delion has been designed to be very decoupled from the UI adaptation engine, thus making very easy the integration of new engines.

Another interesting research line in this topic would be to add learning capabilities to the adaptation engine. It would be possible to receive, or infer, feedback information from the users and exploit that information in order to produce more finely customized UIs in other scenarios. For example, if a user has liked one particular modality, the system could try to use FIOs with that modality in other scenarios or even, if the system detects a user with a similar profile, it could assume that the modality would be also liked by the new user. This learning capabilities could also help to provide more consistent UIs, taking in mind the previous experience of the users, and thus trying to minimizing the problems raised by the introducing dramatic changes in the UI from one scenario to the next.

Another important aspect related to the UI adaptation and generation capabilities are the context models. The proposed TIAF conceptual framework suggest the utilization of at least three different context models, but it does not imposes any limitation on the number or kind of information used. Our reference implementation uses only those three models (User, Environment and Scene), but it would be possible to extend the adaptation engine to support new additional models, or even to extend the available models with more information in order to support new scenarios, new adaptation capacities or even new application fields.

The usability and perceived quality of the generated/adapted user interfaces has been deliberately left out of this work. Not only because it is a very broad and complex topic on its own, but because the objectives of this work, as established in chapter 2, were to enable AmI and UC UIs, and in particular DPUIS, to support multiple and variated usage scenarios without affecting the business or interaction logic. The solution presented in chapter 6 has demonstrated the capability to generate and adapt AmI UIs at runtime, but it mostly ignores the quality and usability of the generated UI. A required, and very interesting, future research line would be to analyze the quality and usability of the UIs, and use the information retrieved in order to improve the UI generation capabilities.

Support for Task-based UI Development

Another important topic that has been raised during the development of this work is that the current implementation of the UI generation and adaptation system uses a dialog-based approach. It considers an UI as a set of unrelated dialogs or displays (the developers must provide one abstract UI model for

each one) and, when requested, it generates a particular UI for one dialog and then, at another different time, when requested, it generates the UI for another. Each of this generation/adaptation processes are individual and isolated, thus they can produce inconsistent user interfaces for each dialog, thus hindering the natural interaction experience of the users.

An interesting research line would be to improve Dandelion to exploit task information during the generation/adaptation process. This information could be used to generate consistent multi-dialog UIs and to manage autonomously the life-cycle of the UI in order to perform a particular task.

In this regard, it could also be interesting to introduce formal verification capabilities in order to autonomously verify the consistency and completeness of the generated UIs regarding a specified task.

Apéndice A

Resumen en castellano

El objetivo final de la Inteligencia Ambiental (AmI) consiste en desarrollar sistemas que asistan a las persona en su vida diaria, mejorando así su calidad de vida. Esta tesis doctoral se centra en la interacción entre este tipo de sistemas y sus usuarios, y más en particular, en el problema de proporcionar una interacción adaptada al contexto particular de cada uno de los posibles escenarios en los que un sistema AmI pueda ser utilizado.

La Inteligencia Ambiental es un área tan amplia que resulta difícil proporcionar una definición precisa del concepto. Sin embargo, dado que los sistemas de Inteligencia Ambiental van a ser el elemento central de esta tesis doctoral, resulta necesario establecer, antes de comenzar, una definición de Inteligencia Ambiental que sirva de marco para todo el trabajo. Una de las definiciones más comúnmente aceptada es la proporcionada por J. C. Augusto y P. McCullang en su artículo "Ambient Intelligence: Concepts and Applications" [Augusto and McCullagh, 2007]:

> «La idea básica detrás de la Inteligencia Ambiental es que, mediante el enriquecimiento de los entornos con tecnología (sensores y otros dispositivos conectados en red), se puede construir un sistema que capaz de tomar decisiones que beneficien a los usuarios, basándose para ello en información obtenida en tiempo real, y datos históricos acumulados. La Inteligencia Ambiental hereda aspectos de muchas áreas de las ciencias de la computación, pero no debe ser confundida con ninguna de ellas en particular. Redes, Sensores, Interacción-Hombre Máquina, Computación Ubicua e Inteligencia Artificial, todas son relevantes y están interrelacionadas con AmI, pero ninguna de ellas la abarca conceptualmente de forma completa. La Inteligencia Ambiental orquesta todos estos recursos para propor-

cionar a los usuarios servicios inteligentes y flexibles. La Inteligencia Ambiental se encuentra en la línea de investigación del ordenador oculto.» J. C. Augusto y P. McCullagh, 2007.

Esta definición de AmI, aunque muy amplia, ya establece los dos aspectos principales de todo sistema AmI:

1. *El objetivo.* Mejorar la calidad de vida de las personas haciendo su vida más confortable y facilitando sus tareas diarias.

2. *El como.* Enriqueciendo los entornos físicos con tecnología.

Aunque esta definición nos proporciona un objetivo global y un camino genérico para conseguirlo, no nos proporciona ningún detalle acerca de las características que debería de presentar un sistema AmI. Una forma de comenzar a explorar esas características es dividir el término Inteligencia Ambiental en sus componentes individuales. Por un lado, el término Inteligencia nos habla acerca del tipo de respuesta que los usuarios esperan del sistema, como por ejemplo proactividad, previsibilidad y comportamiento adaptado al contexto. Por otro lado, el término Ambiental está relacionado con la ubicuidad del sistema, y la operación natural y poco intrusiva del mismo. El primer término está directamente relacionado con la aplicación de técnicas de Inteligencia Computacional para dotarlo de comportamiento autónomo, mientras que el segundo término está relacionado con la Computación Ubicua, y el objetivo establecido por Mark Weisser [Weiser, 1991] de desarrollar sistemas que *«se integren en la vida diaria hasta hacerlos indistinguibles de ella»*.

Como se puede extraer de esta especificación de características, el alcance y los ámbitos de aplicación de los sistemas AmI los hace muy diferentes a los sistemas software tradicionales. Si bien es cierto que comparten muchas similitudes, e incluso tecnologías, con sistemas ampliamente estudiados como los sistemas distribuidos y los sistemas de Inteligencia Ambiental, los sistemas AmI presentan una serie de características propias que hacen que su diseño e implementación sea muy diferente.

A diferencia de los sistemas software tradicionales, que normalmente operan bajo demanda y de forma reactiva, los sistemas AmI deben operar de forma proactiva y transparente, mostrando un comportamiento autónomo e inteligente que se anticipe a las necesidades de los usuarios, y además, interactuando con estos a través de los mecanismos más adecuados en cada contexto.

Para complicar las cosas un poco más, los escenarios de uso de los sistemas AmI son normalmente mucho más complejos que los escenarios de los sistemas tradicionales. Los sistemas AmI operan en entornos que, en este trabajo, hemos llamado Entornos de Interacción Humana (HIE) [Varela et al., 2011], que deben

ser entendidos como entornos extendidos que abarcan los múltiples lugares en los que una persona lleva a cabo su vida diaria, incluyendo por ejemplo, su trabajo, su vida familiar u ocio. Así pues, un HIE integra lugares como la oficina, el hogar, el coche o incluso lugares públicos, como estadios o centros comerciales. Como se puede imaginar, los escenarios de uso de un sistema AmI pueden ser muy diversos, y sobre todo, muy diferentes entre si. Estos pueden presentar diferentes características en su entorno físico (iluminación, ruido, privacidad, etc.), diferentes restricciones de uso (no es lo mismo utilizar un sistema mientras se conduce, que mientras se está viendo una película en el salón, etc.), usuarios con diferentes características (habilidades, capacidad de visión, oído, etc.) y por último, pero no por ello menos importante, cada escenario puede presentar recursos de interacción (dispositivos hardware o elementos software) diferentes con los que interactuar con los usuarios.

Aparte de estas complejas condiciones operativas, los sistemas de Inteligencia Ambiental deben de presentar buenas características en dos aspectos principales [Cook et al., 2009]: interacción natural adaptada a los usuarios [Abascal et al., 2011b, Abascal et al., 2011a, Blumendorf and Albayrak, 2009, Blythe et al., 2005, Kranz et al., 2010] y operación ubicua [Aizpurua et al., 2013, Blumendorf, 2009, Luyten et al., 2006, Luyten and Coninx, 2005].

La interacción natural trata de obtener una experiencia de usuario donde la interfaz de usuario sea casi invisible, integrada en el entorno, y funcionando de forma muy poco intrusiva [Fishkin et al., 1999, Harrison et al., 1998, Ballagas et al., 2003, Xie et al., 2008]. Las interfaces de usuario naturales (NUIs) aprovechan nuestros sentidos y nuestro conocimiento acerca de los objetos cotidianos, la física, y el mundo que nos rodea, para construir interfaces de usuario que liberen a los usuarios de tener que aprender nuevos conceptos e ideas para interactuar con los sistemas informáticos [Ishii and Ullmer, 1997, Ullmer and Ishii, 2000, Sitdhisanguan et al., 2012]. Las NUIs sustituyen objetos cotidianos por dispositivos conectados, y utilizan las modalidades de interacción más adecuadas a cada entorno con el objetivo de hacer invisibles las interfaces de usuario.

La ubicuidad persigue la idea de que un sistema este disponible en cualquier momento y en cualquier lugar [Ranganathan et al., 2004, Ranganathan et al., 2005, Satoh, 2005]. Con respecto a los sistemas AmI y los HIEs, la ubicuidad es una característica que permite a un sistema AmI proporcionar sus funcionalidades en cualquiera de los lugares físicos incluidos en un HIE particular, proporcionando a sus usuarios la capacidad de moverse dentro sus HIEs mientras continúan haciendo uso, u obteniendo los beneficios, de los sistemas AmI.

La ubicuidad, combinada con las Interfaces de Usuario Naturales, hacen

que sea necesario desplegar los sistemas AmI con implementaciones diferentes de la IU para cada escenario [Sousa and Garlan, 2002, Ranganathan et al., 2005]. Cuando un usuario se mueve de un sitio a otro, el escenario de ejecución del sistema AmI cambia, y con el, los dispositivos disponibles, las características del entorno, e incluso los usuarios. Dada la gran diversidad de escenarios que pueden existir, resulta prácticamente imposible predecirlos durante la fase de diseño y desarrollo del sistema. Debido a esto, la mayoría de sistemas AmI son diseñados e implementados para un único escenario particular, o en todo caso, un conjunto concreto de escenarios, con unos dispositivos y tipos de usuario preestablecidos. Desplegar este tipo de IUs en nuevos escenarios, que han sido predichos anteriormente, resulta muy complejo, y habitualmente requiere importantes modificaciones en el sistema, afectando enormemente a la capacidad portar y desplegar un mismo sistema AmI en diferentes entornos u escenarios, y por tanto, afectando a la movilidad de los usuarios de dichos sistemas.

Disponer de una interfaz de usuario natural, y adaptada al contexto, es una de las características más importantes para que un sistema de Inteligencia Ambiental sea aceptado por los usuarios [Ranganathan et al., 2005, Abascal et al., 2008, Cook et al., 2009, Pavan Dadlani, 2011, Sitdhisanguan et al., 2012, Zuckerman and Gal-Oz, 2013]. Un área habitual para la aplicación de la Inteligencia Ambiental es mejorar la calidad de vida de personas con discapacidades o dependencia, personas mayores, o niños. Tres colectivos en los que disponer de una Interfaz de Usuario adaptada puede marcar la diferencia entre un sistema exitoso y otro fallido.

Los desarrolladores de sistemas AmI suelen utilizar NUIs para construir IUs personalizadas para cada tipo de usuario, y para ello, suelen emplear diferentes recursos de interacción (IRs) adaptados a las características de cada escenario de uso (usuario y entorno) [Pavan Dadlani, 2011]. Estos dispositivos provienen de diferentes fabricantes, usan protocolos y APIs heterogéneas, y en algunos escenarios, incluso se utilizan IRs especialmente desarrollados para ese caso. Por si fuera poco, estos IRs suelen ser dispositivos empotrados y estar distribuidos por el entorno físico, formando interfaces de usuario físicas y distribuidas. Soportar esta variedad de dispositivos y modalidades de interacción no es fácil, y puede introducir mucha complejidad en un sistema. Además, desarrollar IUs específicas para cada escenario incrementa enormemente el tiempo y el coste de desarrollo. Sin mencionar que en numerosos casos resulta inviable predecir de antemano todos los posibles escenarios de uso de un sistema.

El objetivo principal del trabajo llevado a cabo en esta tesis doctoral es mejorar la capacidad de operación ubicua de los sistemas de Inteligencia Ambiental, y con ello, facilitar la movilidad de sus usuarios. Como se ha indicando anteriormente, uno de los principales obstáculos para conseguir sistemas AmI

ubicuos es soportar la inmensa variedad de técnicas y dispositivos de interacción necesarios para producir IUs adaptadas a cada contexto de uso. Para abordar este problema, esta tesis doctoral propone incrementar el nivel de desacoplamiento entre el software de sistema y los recursos de interacción que forma la IU. El objetivo pasa por mejorar la portabilidad de los sistemas AmI, facilitando su despliegue en diferentes escenarios, y por tanto, mejorando su capacidad para adaptar su interacción a diferentes usuarios en múltiples y variados entornos.

Este trabajo introduce una mejora a tres niveles en el desacoplamiento entre desarrolladores, software de sistema y recursos de interacción. Primero, a nivel lógico la desacopla a los desarrolladores y su código fuente, de las modalidades de interacción, APIs, y tecnologías utilizadas por cada uno de los IRs que forman la IU. Segundo, a nivel físico, es aísla al software de sistema de la localización física de los IRs. Y tercero, para obtener una operación realmente ubicua y facilitar el despliegue de sistemas AmI en múltiples escenarios, incrementa el aislamiento entre los desarrolladores de la IU y el conjunto concreto de IRs que forman la IU en cada escenario.

Un mejor aislamiento entre la lógica de negocio y control de un sistema AmI, y el aspecto final de su interfaz de usuario, facilita enormemente el despliegue de la misma lógica con diferentes realizaciones de la IU, posibilitando así la instalación y despliegue de sistemas AmI en HIEs que incluyan diferentes localizaciones y múltiples usuarios. En este sentido, el aislamiento de las modalidades y APIs de los IRs permite a los desarrolladores de AmI cambiar la implementación y tipo de los IRs sin afectar al código del sistema, de forma que es posible utilizar el mismo código fuente en diversos escenarios, con diferentes IRs en cada caso.

Así pues, esta tesis doctoral propone la utilización de técnicas de ingeniería basada en modelos (MDE) para mejorar el nivel de desacoplamiento entre los sistemas AmI y sus interfaces de usuario. Las técnicas de MDE permiten a los desarrolladores construir sistemas utilizando modelos de alto nivel, y posteriormente utilizar métodos de transformación que conviertan esos modelos en otros modelos más concretos, así hasta llegar a obtener un sistema final y funcional.

Las técnicas de MDE has sido aplicadas anteriormente con éxito al campo de las interfaces de usuario. Por ejemplo, Thevenin y Coutaz propusieron ya en 1999 el uso de técnicas MDE para dar soporte a la plasticidad en interfaces de usuario [Thevenin and Coutaz, 1999, Coutaz, 2010], esto es, la capacidad de una IU para mantener sus nivel de usabilidad ante cambios importantes en las condiciones y características del sistema y/o entorno de interacción. Esta propuesta cosechó un importante éxito en la comunidad de investigadores en HCI (Human-Computer Interaction), y han sido numerosos los autores que la han utilizado como base para sus propias aproximaciones a la adaptación de

IUs al contexto [Collignon et al., 2008, Blumendorf et al., 2010, Luyten, 2003]. Existiendo incluso algunas aproximaciones dentro del ámbito del Ambient Assisted Living (AAL), la Computación Ubicua y la Inteligencia Ambiental [Abascal et al., 2008, Blumendorf and Albayrak, 2009, Blumendorf, 2009, Abascal et al., 2011a, Abascal et al., 2011b]. Sin embargo, todas estas aproximaciones se han centrado en interfaces de usuario gráficas, de voz, o en todo caso gestuales, incorporando un soporte reducido, o nulo, para otro tipo de modalidades. Por el contrario, el trabajo desarrollado en esta tesis doctoral proporciona un framework de desarrollo de IUs basado en modelos y con soporte integrado para la construcción de interfaces de usuario multimodales, físicas y distribuidas, capaces de adaptarse al contexto en tiempo real.

Como se ha indicado previamente, la propia naturaleza de los sistemas AmI requiere que sus interfaces de usuario se construyan sobre un conjunto de dispositivos físicos distribuidos a lo largo del entorno [Pavan Dadlani, 2011, Luyten and Coninx, 2005, Luyten et al., 2006]. Un incremento en el desacoplamiento entre los sistemas AmI y la localización física de los IRs permitiría mejorar la portabilidad de los sistemas AmI, facilitando su despliegue en diferentes entornos, con diferentes distribuciones de dispositivos. Este trabajo integra en el framework MDE mencionado anteriormente, un capa de abstracción de IRs distribuidos. Esta capa se encarga de encapsular las tecnologías de red y protocolos utilizados por cada IR, posibilitando así el desarrollo de IUs físicas y distribuidas sin tener conocimiento de dichos protocolos y tecnologías, y permitiendo que el mismo código de pueda utilizar diferentes dispositivos, desplegados en diferentes lugares en cada caso.

El desarrollo de IUs distribuida utilizando técnicas MDE ha sido también previamente estudiado por la comunidad investigadora, existiendo dos aproximaciones destacadas: El modelo conceptual de referencia para el desarrollo de interfaces de usuario adaptativas, distribuidas y migrables, proporcionado por Cameleon-rt [Balme et al., 2004]; y el framework conceptual para interfaces multimodales del W3C [W3C, 2003]. En esta tesis doctoral nos hemos inspirado en el modelo propuesto por Cameleon-rt, pero ampliándolo y mejorándolo para incluir un mejor soporte para interfaces de usuario físicas y distribuidas. Primero, en este trabajo el modelo de IU abstracta se transforma directamente en una IU final adaptada al contexto, a diferencia de Cameleon, donde debe pasar por varios estados intermedios. Segundo, la capa de plataforma ha sido modelada e implementada como una capa distribuida de abstracción de recursos de interacción, y además, soporta cualquier tipo de IR, tanto físicos como digitales.

Apoyándose en estas dos soluciones de abstracción o desacoplamiento, los desarrolladores pueden diseñar e implementar interfaces de usuario para sis-

temas AmI con muy poco, o incluso nulo, conocimiento de las modalidades, protocolos, tecnologías de red, localización física y APIs de los IRs utilizados finalmente para interactuar con los usuarios. Es más, los diseñadores e instaladores de las UIs pueden utilizar los IRs que consideren más adecuados para cada escenario, sin preocuparse de que los desarrolladores del sistema tengan que modificar su código para soportarlo, pues la solución proporcionada en esta tesis permite posponer hasta el momento del despliegue, la decisión final de qué dispositivos (IRs) concretos utilizar en cada escenario.

Sin embargo, estas dos soluciones no son suficientes para soportar completamente la operación ubicua de los sistemas e interfaces de usuario AmI. Para ello es necesario que las IU puedan operar de forma natural, y adaptada, en cada uno de los escenarios incluidos en un HIE, y por tanto, las IUs deberían de ser capaces de cambiar de forma y aspecto a medida que los usuarios se mueven de un entorno a otro dentro del HIE.

Como se ha mencionado anteriormente, un HIE podría incluir diversos lugares y escenarios, cada uno de ellos requiriendo el uso de unas modalidades y/o dispositivos diferentes, lo que hace que diseñar e implementar una IU particular para cada uno de ellos sea inviable debido a los costes en tiempo y dinero que implicaría. Así pues, en este trabajo se proporciona un tercer nivel de abstracción que permite aislar a los desarrolladores, e instaladores, del conjunto específico de IRs necesarios o adecuados para cada escenario.

La idea principal detrás de este tercer nivel de abstracción es posponer el ensamblaje de la IU hasta el tiempo de ejecución del sistema, y utilizar modelos del contexto para automatizar dicho ensamblaje, eligiendo autónomamente el conjunto de IRs que conformarán la IU final en cada escenario. Esta selección de los IRs más adecuados para cada las características de cada escenario, es llevada a cabo por el propio sistema, que los elige entre los IRs disponibles en cada entorno, y lo conecta a la lógica de control del sistema a través de las otras dos capas de abstracción.

Los problemas abordados por estos tres niveles de abstracción, adaptación, migración y portabilidad de interfaces de usuario en aplicaciones AmI, han sido identificados previamente por diferentes autores [Miñón and Abascal, 2012, Aizpurua et al., 2013]. Por ejemplo, en [Blumendorf, 2009, Blumendorf et al., 2010, Blumendorf and Albayrak, 2009], Blumendorf et al. presentan la problemática asociada a la interacción con el usuario en sistemas de Ambient Assisted Living (AAL), que son un subconjunto de los sistemas AmI. El artículo presenta un framework de desarrollo de IUs que también utiliza técnicas MDE e información del contexto para adaptar las IUs en tiempo de ejecución. En [Abascal et al., 2008, Abascal et al., 2011a, Abascal et al., 2011b] Abascal et al. destacan la necesidad de adaptación a los usuarios en sistemas AmI, debi-

do a que las diversas capacidades de estos pueden afectar enormemente a la percepción, usabilidad y utilidad del sistema. Sin embargo, estas soluciones se han focalizado principalmente en solo dos modalidades, IUs gráficas y basadas en voz, y no incorporan soporte alguno para interfaces físicas que requieren de la utilización de dispositivos distribuidos y utilizando gran variedad de modalidades. Así mismo, aunque Egoki [Abascal et al., 2011a] soporta la adaptación de la IU a las características y preferencias de los usuarios, ninguna de ellas soporte la adaptación a las características del entorno y sus dispositivos, lo cual resulta de vital importancia en sistemas AmI y UC, donde como hemos mencionado, un cambio de contexto y/o localización, puede cambiar drásticamente las condiciones del entorno, y los dispositivos disponible.

Así pues, en esta tesis doctoral se ha estudiado, diseñado y desarrollado una solución que permite integrar estos tres niveles de abstracción en cualquier sistema de Inteligencia Ambiental, aumentado así el nivel de desacoplamiento entre este, y la implementación final de su interfaz de usuario, y por tanto, mejorando la capacidad de operación ubicua de dicho sistema AmI. Esta solución se ha estudiado y diseñado primeramente desde un punto de vista conceptual y arquitectónico, proporcionando un marco teórico de referencia, llamado el Threefold Interaction Abstraction Framework (TIAF), que da respuesta a los principales problemas asociados a las interfaces de usuario en Inteligencia Ambiental, y proporciona soporte teórico para los tres niveles de abstracción expuestos anteriormente. A continuación, en este marco teórico se a abordado desde un punto de vista de implementación, obteniéndose en esta tesis doctoral una implementación de referencia del mismo, llamada Dandelion framework, proporcionando así un una implementación funcional y operativa que da soporte real a los tres niveles de abstracción propuestos en este trabajo. De esta forma, Dandelion proporciona un conjunto de herramientas y librerías que facilitan la utilización de cualquier tipo de dispositivo y/o modalidad de interacción sin afectar al código fuente de la interfaz de usuario. Facilitando así el desarrollo de interfaces de usuario físicas y distribuidas capaces de operar de forma natural y adapta en diferentes escenarios, incluso adaptándose autónomamente, y en tiempo real, a cambios en las características del escenario.

References

[Abascal, 2004] Abascal, J. (2004). Ambient intelligence for people with disabilities and elderly people. In *ACM's Special Interest Group on Computer-Human Interaction (SIGCHI)*, pages 1–3.

[Abascal et al., 2011a] Abascal, J., Aizpurua, A., Cearreta, I., Gamecho, B., Garay, N., and Miñón, R. (2011a). Some issues regarding the design of adaptive interface generation systems. In *Lecture Notes in Computer Science*, volume 6765 LNCS, pages 307–316.

[Abascal et al., 2011b] Abascal, J., Aizpurua, A., Cearreta, I., Gamecho, B., Garay-Vitoria, N., and Miñón, R. (2011b). Automatically generating tailored accessible user interfaces for ubiquitous services. In *Proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility, ASSETS (XIII)*, pages 187–194.

[Abascal et al., 2008] Abascal, J., Fernández de Castro, I., Lafuente, A., and Cia, J. M. (2008). Adaptive interfaces for supportive ambient intelligence environments. In *ICCHP '08 Proceedings of the 11th international conference on Computers Helping People with Special Needs*, pages 30–37.

[Abrams et al., 1999] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E. (1999). UIML: an appliance-independent XML user interface language. *Computer Networks*, 31:1695–1708.

[Aizpurua et al., 2013] Aizpurua, A., Cearreta, I., and Gamecho, B. (2013). Extending in-home user and context models to provide ubiquitous adaptive support outside the home. *User Modeling and Adaptation for Daily Routines*, pages pp 25–59.

[Antle et al., 2009] Antle, A. N., Corness, G., Droumeva, M., and Bevans, A. (2009). Exploring Embodied Metaphors for Full Body Interaction. *Technology*, pages 1–4.

[Apache-Foundation, 2015] Apache-Foundation (2015). Activemq. URL: http://activemq.apache.org.

[Apple-Inc., 2015] Apple-Inc. (2015). Apple homekit. URL: https://developer.apple.com/homekit/.

[Augusto and McCullagh, 2007] Augusto, J. C. and McCullagh, P. (2007). Ambient intelligence: Concepts and applications. *Computer Science and Information Systems*, 4(1):228–250.

[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*, number 5, pages 537–544. ACM.

[Balme et al., 2004] Balme, L., Demeure, A., Barralon, N., Coutaz, J., and Calvary, G. (2004). CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. *Ambient Intelligence*, 3295:291–302.

[Bandara et al., 2004] Bandara, A., Payne, T. R., de Roure, D., and Clemo, G. (2004). An Ontological Framework for Semantic Description of Devices. pages 2–3.

[Bandelloni and Paternò, 2004] Bandelloni, R. and Paternò, F. (2004). Flexible interface migration. *Proceedings of the 9th international conference on Intelligent user interface - IUI '04*, page 148.

[Bashon et al., 2010] Bashon, Y., Neagu, D., and Ridley, M. (2010). A new approach for comparing fuzzy objects. In Hüllermeier, E., Kruse, R., and Hoffmann, F., editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Applications*, volume 81, pages 115–125. Springer Berlin Heidelberg.

[Bashon, 2013] Bashon, Y. M. (2013). *CONTRIBUTIONS TO FUZZY OBJECT COMPARISON AND APPLICATIONS*. PhD thesis, University of Bradford.

[Belkin-International-Inc., 2014] Belkin-International-Inc. (2014). Wemo home automation. URL: http://goo.gl/JMdzfq.

[Berti, 2005] Berti, S. (2005). Migratory MultiModal Interfaces in MultiDevice Environments. In *Proceedings of the 7th international conference on Multimodal interfaces*, pages 92–99.

[Berti et al., 2004] Berti, S., Correani, F., and Mori, G. (2004). TERESA: a transformation-based environment for designing and developing multi-device interfaces. *CHI'04 extended abstracts on Human factors in computing systems*, pages 793–794.

[Blumendorf, 2009] Blumendorf, M. (2009). *Multimodal interaction in smart environments: a model-based runtime system for ubiquitous user interfaces.* PhD thesis.

[Blumendorf and Albayrak, 2009] Blumendorf, M. and Albayrak, S. (2009). Towards a framework for the development of adaptive multimodal user interfaces for ambient assisted living environments. *Universal access in human-computer interaction. Intelligent and ubiquitous interaction environments*, pages 150–159.

[Blumendorf et al., 2008a] Blumendorf, M., Feuerstack, S., and Albayrak, S. (2008a). Multimodal User Interfaces for Smart Environments: The Multi-Access Service Platform. *Computer*, pages 478–479.

[Blumendorf et al., 2010] Blumendorf, M., Lehmann, G., and Albayrak, S. (2010). Bridging models and systems at runtime to build adaptive user interfaces. In *Symposium on Engineering Interactive Computing Systems*, pages 9–18, New York, New York, USA. ACM Press.

[Blumendorf et al., 2008b] Blumendorf, M., Lehmann, G., Feuerstack, S., and Albayrak, S. (2008b). Executable models for human-computer interaction. *Interactive Systems.*, pages 238 – 251.

[Blythe et al., 2005] Blythe, M. A., Monk, A. F., and Doughty, K. (2005). Socially dependable design: The challenge of ageing populations for HCI.

[Bodart et al., 1995] Bodart, F., Hennebert, A., Leheureux, J., Provot, I., Sacre, B., and Vanderdonckt, J. (1995). Towards a Systematic Building of Software Architectures: the Trident methodological guide. In *Proc. of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95*, volume 95, pages 262–278.

[Bonino et al., 2008] Bonino, D., Castellina, E., and Corno, F. (2008). The DOG gateway: Enabling ontology-based intelligent domotic environments. *IEEE Transactions on Consumer Electronics*, 54(4):1656–1664.

[Bonino and Corno, 2008] Bonino, D. and Corno, F. (2008). DogOnt - Ontology Modeling for Intelligent Domotic Environments. *The Semantic Web-ISWC 2008*, 19(2):790–803.

[Borchers et al., 2002] Borchers, J., Ringel, M., Tyler, J., and Fox, A. (2002). Stanford interactive workspaces: A framework for physical and graphical user interface prototyping. *IEEE Wireless Communications*, 9(6):64–69.

[Bouillon et al., 2004] Bouillon, L., Vanderdonckt, J., and Chow, K. C. (2004). Flexible re-engineering of web sites. In *Proceedings of the 9th international conference on Intelligent user interface - IUI '04*, page 132.

[Bouillon et al., 2002] Bouillon, L., Vanderdonckt, J., and Souchon, N. (2002). RECOVERING ALTERNATIVE PRESENTATION MODELS OF A WEB PAGE WITH VAQUITA. In *Computer-Aided Design of User Interfaces III*, pages 311–322.

[Brault, 2008] Brault, M. W. (2008). Americans with Disabilities: 2005. Technical report, US Census Bureau.

[Brault, 2012] Brault, M. W. (2012). Americans with disabilities: 2010. Technical report, US Census Bureau.

[Bristish-Standards-Institute, 2008] Bristish-Standards-Institute (2008). Ergonomics data and guidelines for the application of iso/iec guide 71 to products and services to address the needs of older persons and persons with disabilities. Technical report, Bristish Standards Institute.

[Bticino, 2015] Bticino (2015). Openwebnet library and tools. URL: https://openwebnet.codeplex.com.

[Calvary et al., 2007] Calvary, G., Coutaz, J., Favre, J., Vanderdonckt, J., and Stanciulescu, A. (2007). A language perspective on the development of plastic multimodal user interfaces. *Journal on Multimodal User Interfaces*, (October):1–12.

[Calvary et al., 2001a] Calvary, G., Coutaz, J., and Thevenin, D. (2001a). A Development Process for Plastic User Interfaces. In *Proceedings of the CHI2001 Workshop on Transforming the UI for Anyone, Anywhere*, page 349. Citeseer.

[Calvary et al., 2001b] Calvary, G., Coutaz, J., and Thevenin, D. (2001b). Supporting context changes for plastic user interfaces: a process and a mechanism. *People and Computers XV Interaction without Frontiers*, pages 349–363.

[Calvary et al., 2003] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15(3):289–308.

[Calvary et al., 2002] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., and Vanderdonckt, J. (2002). Plasticity of user interfaces: A revised reference framework. *Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design*.

[CBSR, 2014] CBSR (2014). Cubesensors. URL: http://cubesensors.com.

[Cingolani and Alcala-Fdez, 2012] Cingolani, P. and Alcala-Fdez, J. (2012). jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation. *2012 IEEE International Conference on Fuzzy Systems*, pages 1–8.

[Cingolani and Alcala-Fdez, 2013] Cingolani, P. and Alcala-Fdez, J. (2013). jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming. *International Journal of Computational Intelligence Systems*, 6(sup1):61–75.

[Clerckx et al., 2004] Clerckx, T., Luyten, K., and Coninx, K. (2004). DynaMo-AID : a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In *Lecture Notes in Computer Science 3425 Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems, EHCI-DSVIS'2*, pages 11–13.

[Clerckx et al., 2008] Clerckx, T., Vandervelpen, C., and Coninx, K. (2008). Task-based design and runtime support for multimodal user interface distribution. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4940 LNCS, pages 89–105. Springer.

[Clerckx et al., 2006] Clerckx, T., Vandervelpen, C., Luyten, K., and Coninx, K. (2006). A task-driven user interface architecture for ambient intelligent environments. *Proceedings of the 11th international conference on Intelligent user interfaces - IUI '06*, page 309.

[Collignon et al., 2008] Collignon, B., Vanderdonckt, J., and Calvary, G. (2008). Model-driven engineering of multi-target plastic user interfaces. In *Proceedings - 4th International Conference on Autonomic and Autonomous Systems, ICAS 2008*, pages 7–14. Ieee.

[Cook et al., 2009] Cook, D. j., Augusto, J. C., and Jakkula, V. R. (2009). Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298.

[Coutaz, 2010] Coutaz, J. (2010). User Interface Plasticity: Model Driven Engineering to the Limit! In *Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–8.

[Coutaz et al., 1995] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M. (1995). Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties. In *Proc. Of IFIP Int. Conf. on Human-Computer Interaction Interact'95, Chapman & Hall, London*, number June, pages 115–120.

[Coyette et al., 2006] Coyette, A., V, J., and Limbourg, Q. (2006). SketchiXML: An Informal Design Tool for User Interface Early Prototyping. *In Proceedings of RISE'2006 Workshop on Rapid User Interface Prototyping Infrastructures Applied to Control Systems.*

[Coyette and Vanderdonckt, 2010] Coyette, A. and Vanderdonckt, J. (2010). Prototyping Digital, Physical, and Mixed User Interfaces by Sketching. In *1st Int. Workshop on User Interface eXtensible Markup Language UsiXML'2010*, volume 2010, pages 27–36.

[Cross and Sudkamp, 2002] Cross, V. V. and Sudkamp, T. a. (2002). Similarity and Compatibility in Fuzzy Set Theory. 93.

[Deza and Deza, 2009] Deza, M.-M. and Deza, E. (2009). *Enclyclopedia of Distances.* Springer Berlin Heilderberg.

[Dibowski and Kabitzsch, 2011] Dibowski, H. and Kabitzsch, K. (2011). Ontology-Based Device Descriptions and Device Repository for Building Automation Devices. *EURASIP Journal on Embedded Systems*, 2011:1–17.

[ECMA-International, 2013] ECMA-International (2013). The json data interchange format. URL: http://goo.gl/hVk2ct.

[Edlin-White et al., 2010] Edlin-White, R., D'Cruz, M., Floyde, A., Riedel, J., Cobb, S., Broadley, S., Marsá, V. S., Hernández, J. A., Gacimartín, C., and Bruikman, H. (2010). MyUI: Requirements for User Interfaces Adaptation. Technical report.

[FIPA, 2011] FIPA (2011). Fipa device ontology specification. URL: http://www.fipa.org/specs/fipa00091/PC00091A.html.

[Fishkin et al., 1999] Fishkin, K. P., Moran, T. P., and Harrison, B. L. (1999). Embodied user interfaces: Towards invisible user interfaces. In *Proceedings of the IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction*, pages 1–18, Deventer, The Netherlands, The Netherlands. Kluwer, B.V.

[Gajos et al., 2004] Gajos, K., Hoffmann, R., and Weld, D. (2004). Improving user interface personalization. *Supplementary Proceedings of UIST*, 4.

[Gajos and Weld, 2008] Gajos, K. and Weld, D. S. (2008). Automatically Generating Personalized User Interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 1–210.

[Gajos et al., 2008a] Gajos, K. Z., Weld, D. S., and Wobbrock, J. O. (2008a). Decision-theoretic user interface generation. *Proc. of the 22 nd AAAI Conf. on Artificial Intelligence*.

[Gajos et al., 2010] Gajos, K. Z., Weld, D. S., and Wobbrock, J. O. (2010). Automatically generating personalized user interfaces with Supple. *Artificial Intelligence*, 174(12-13):910–950.

[Gajos et al., 2008b] Gajos, K. Z., Wobbrock, J. O., and Weld, D. S. (2008b). Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1257–1266, New York, New York, USA. ACM Press.

[Girolami et al., 2008] Girolami, M., Lenzi, S., Furfari, F., and Chessa, S. (2008). SAIL: A Sensor Abstraction and Integration Layer for Context Awareness. *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pages 374–381.

[Greenberg et al., 2001] Greenberg, S., Greenberg, S., Fitchett, C., and Fitchett, C. (2001). Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, volume 3, pages 209 – 218.

[Harrison et al., 1998] Harrison, B. L., Fishkin, K. P., Gujar, A., Mochon, C., and Want, R. (1998). Squeeze Me, Hold Me, Tilt Me! An Exploration of Manipulative User Interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '98*, (April):17–24.

[Hayes et al., 1985] Hayes, P. J., Szekely, P. A., and Lerner, R. A. (1985). Design alternatives for user interface management sytems based on experience with COUSIN.

[Holleis, 2007] Holleis, P. (2007). Programming Interactive Physical Prototypes. In *Proc. 1st International Workshop on Design and Integration Principles for Smart Objects*.

[IEC, 1997] IEC (1997). IEC 61131 - Part 7 - Fuzzy Control Programming. Technical report.

[Ishii and Ullmer, 1997] Ishii, H. and Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people, bits and atoms. *Proceedings of the SIGCHI conference on Human factors in computing systems*, (March):241.

[Janse et al., 2005] Janse, M., Ramparany, F., Kladis, B., Rozendaal, L., Broens, T., and Eertink, H. (2005). Specification of the Amigo Abstract System Architecture. Technical report.

[Janssen et al., 1993] Janssen, C., Weisbecker, A., and Ziegler, J. (1993). Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, pages 418–423.

[Johanson et al., 1999] Johanson, B., Fox, A., and Winograd, T. (1999). The Interactive Workspaces Project : Experiences with Pervasive Computing Magazine. pages 1–17.

[JSON.org, 2015] JSON.org (2015). Json (javascript object notation). URL: http://www.json.org.

[KNX-Association, 2010] KNX-Association (2010). Konnex association official web. URL: http://www.knx.org.

[Kranz et al., 2010] Kranz, M., Holleis, P., and Schmidt, A. (2010). Embedded Interaction: Interacting with the Internet of Things. *IEEE Internet Computing*.

[Lazaro-Ramos, 2010] Lazaro-Ramos, J. P. (2010). Reference Architecture and information model for service infrastructure final version. Technical report, ITACA.

[Leap-Motion-Inc., 2015] Leap-Motion-Inc. (2015). Leap motion. URL: https://www.leapmotion.com/product.

[Lonmark-International, 2010] Lonmark-International (2010). Lonmark international official web. URL: http://www.lonmark.org.

[Luo et al., 1993] Luo, P., Szekely, P., and Neches, R. (1993). {M}anagement of interface design in {H}umanoid. In *{INTERCHI}'93*, pages 107–114.

[Luyten, 2003] Luyten, K. (2003). Runtime transformations for modal independent user interface migration. *Interacting with Computers*, 15(3):329–347.

[Luyten and Coninx, 2005] Luyten, K. and Coninx, K. (2005). Distributed user interface elements to support smart interaction spaces. *Multimedia, Seventh IEEE International*.

[Luyten et al., 2006] Luyten, K., Van den Bergh, J., Vandervelpen, C., and Coninx, K. (2006). Designing distributed user interfaces for ambient intelligent environments using models and simulations.

[Miñón and Abascal, 2012] Miñón, R. and Abascal, J. (2012). Supportive adaptive user interfaces inside and outside the home. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7138 LNCS, pages 320–334.

[Michotte and Vanderdonckt, 2008] Michotte, B. and Vanderdonckt, J. (2008). GrafiXML, a Multi-target User Interface Builder Based on UsiXML. *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*, pages 15–22.

[Microsoft, 2015] Microsoft (2015). Kinect for windows. URL: http://goo.gl/yzRaj8.

[Miori et al., 2006] Miori, V., Tarrini, L., Manca, M., and Tolomei, G. (2006). An open standard solution for domotic interoperability. *IEEE Transactions on Consumer Electronics*, 52(1):97–103.

[Monson-Haefel and Chappell, 2000] Monson-Haefel, R. and Chappell, D. (2000). *Java Message Service*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

[Mori et al., 2004] Mori, G., Paterno, 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.

[MyTech-IA and GII, 2014] MyTech-IA and GII (2014). Unida framework. URL: http://www.gii.udc.es/unida.

[Nichols, 2006] Nichols, J. (2006). Automatically Generating High-Quality User Interfaces for Appliances. (December):1–358.

[Nichols et al., 2006] Nichols, J., Myers, B. A., Rothrock, B., and Nichols, J. (2006). UNIFORM : Automatically Generating Consistent Remote Control User Interfaces.

[Nigay and Coutaz, 1994] Nigay, L. and Coutaz, J. (1994). Multifeature Systems : The CARE Properties and Their Impact on Software Design The Design Space : Physical Devices and Interaction Languages. (Nigay).

[Obrenović and Starcević, 2004] Obrenović, Z. and Starcević, D. (2004). Modeling multimodal human-computer interaction. *Computer*, 37(9):65–72.

[Obrenović et al., 2007] Obrenović, Z., Troncy, R., and Hardman, L. (2007). Vocabularies for Description of Accessibility Issues in Multimodal User Interfaces. In *International Classification*, number 2002.

[of Edinburgh, 2015] of Edinburgh, U. (2015). The festival speech synthesis system. URL: http://www.cstr.ed.ac.uk/projects/festival/.

[Olsen et al., 2000] Olsen, D. R., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P. (2000). Cross-modal interaction using XWeb. In *Proceedings of the 13th annual ACM symposium on User interface software and technology - UIST '00*, volume 2, pages 191–200.

[Paternò et al., 2000] Paternò, F., Mancini, C., and Paterno, F. (2000). Model-based design of interactive applications. *intelligence*, 11(4):26–38.

[Paternò et al., 2009] Paternò, F., Santoro, C., and Spano, L. D. (2009). MARIA: A universal, declarative, multiple abstraction-level language for service-oriente applications in ubiquitous environments. *ACM Transactions on Computer-Human Interaction*, 16(4):1–30.

[Pavan Dadlani, 2011] Pavan Dadlani, Joser Pergrín Emparanza, P. M. (2011). *Distributed User Interfaces in Ambient Intelligence Environments*, volume 1, chapter Exploring Distributed User Interfaces in Ambient Intelligent Environments, pages 161–168. Springer London.

[Peissner et al., 2011] Peissner, M., Häbe, D., and Schuller, A. (2011). MyUI: Mainstreaming Accesibility through Synergistic User Modelling and Adaptability.

[Philips-Electronics-N.V., 2015] Philips-Electronics-N.V. (2015). Philips hue developer program. URL: http://www.developers.meethue.com/.

[Physical-Graph-Corporation, 2014] Physical-Graph-Corporation (2014). Smartthings. URL: http://smartthings.com/explore/.

[Ponnekanti et al., 2001] Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. (2001). ICrafter: A service framework for ubiquitous computing environments. *Ubiquitous Computing.*

[Ranganathan et al., 2004] Ranganathan, a., Chetan, S., and Campbell, R. (2004). Mobile polymorphic applications in ubiquitous computing environments. *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.*, pages 402–411.

[Ranganathan et al., 2005] Ranganathan, A., Shankar, C., and Campbell, R. (2005). Application polymorphism for autonomic ubiquitous computing. *Multiagent and Grid Systems*, 1(2):109–129.

[Roscher et al., 2009] Roscher, D., Blumendorf, M., and Albayrak, S. (2009). A Multimodal User Interface Model For Runtime Distribution. In *CHI*, pages 1–4.

[Rui et al., 2009] Rui, C., Yi-bin, H., Zhang-qin, H., and Jian, H. (2009). Modeling the Ambient Intelligence Application System: Concept, Software, Data, and Network. *IEEE transactions on systems, man, and cybernetics. Part C.*, 39(3):299–314.

[Rui et al., 2007] Rui, C., Yi-bin, H., Zhang-qin, H., Yong, Z., and Hui, L. (2007). Framework for Local Ambient Intelligence Space: The AmI-Space Project. *Computer Software and Applications Conference Annual International*, 2(Compsac):95–100.

[Satoh, 2005] Satoh, I. (2005). Mobile Applications in Ubiquitous Computing Environments. *EICE transactions on communications*, 88(5):1026–1033.

[Sendín, 2007] Sendín, M. (2007). *Infraestructura Software de Soporte al Desarrollo de Interfaces de Usuario Plásticas bajo una Visión Dicotómica*. PhD thesis, Universitat de Lleida.

[Sendín and Lorés, 2004] Sendín, M. and Lorés, J. (2004). Plasticity in mobile devices: a dichomotic and semantic view. volume 5.

[Sendín et al., 2003] Sendín, M., Lorés, J., Montero, F., and López-Jaquero, V. (2003). Towards a framework to develop plastic user interfaces. *Human-Computer Interaction with Mobile Devices and Services*, 3:428–433.

[Sitdhisanguan et al., 2012] Sitdhisanguan, K., Chotikakamthorn, N., Dechaboon, A., and Out, P. (2012). Using tangible user interfaces in computer-based training systems for low-functioning autistic children. *Personal and Ubiquitous Computing*, 16:143–155.

[Sousa and Garlan, 2002] Sousa, J. P. and Garlan, D. (2002). Aura: An architectural framework for user mobility in ubiquitous computing environments. *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, 25(August):29–43.

[Stanciulescu et al., 2005] Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. (2005). A transformational approach for multimodal web user interfaces based on UsiXML. In *Proceedings of the 7th international conference on Multimodal interfaces*, pages 259–266. ACM.

[STOMP, 2012] STOMP (2012). Stomp (simple text-orientated messaging protocol) specification. URL: https://stomp.github.io/stomp-specification-1.2.html.

[Sukaviriya et al., 1993] Sukaviriya, P., Sukaviriya, P., Foley, J. D., Foley, J. D., Griffith, T., and Griffith, T. (1993). A second generation user interface design environment: the model and the runtime architecture. *Conference on Human Factors in Computing Systems*.

[Supermechanical, 2014] Supermechanical (2014). Twine. URL: http://supermechanical.com.

[Taylor et al., 2006] Taylor, P., Black, A. W., and Caley, R. (2006). The Architecture of the Festival Speech Synthesis System.

[Thevenin and Coutaz, 1999] Thevenin, D. and Coutaz, J. (1999). Plasticity of user interfaces: Framework and research agenda. *Proceedings of INTERACT'99*.

[Thevenin et al., 2003] Thevenin, D., Coutaz, J., and Calvary, G. (2003). A Reference Framework for the Development of Plastic User Interfaces. In *Multi-Device and Multi-Context User Interfaces: Engineering and Applications Frameworks*, pages 27–51.

[Ullmer and Ishii, 2000] Ullmer, B. and Ishii, H. (2000). Emerging frameworks for tangible user interfaces. *IBM systems journal*, 39(3):1–15.

[UPnP-Forum, 2011] UPnP-Forum (2011). Upnp device architecture 1.1. URL: http://goo.gl/qXK3zW.

[Vanderdonckt et al., 2004] Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., and Florins, M. (2004). USIXML : a User Interface Description Language for Specifying Multimodal User Interfaces The Reference Framework used for Multi-Directional UI Development. In *Proceedings of W3C Workshop on Multimodal Interaction WMI*, pages 19–20.

[Vanderdonckt et al., 2008] Vanderdonckt, J., Mendonca, H., and Massó, J. (2008). Distributed user interfaces in ambient environment. *Constructing Ambient Intelligence*, pages 121–130.

[Vandervelpen and Coninx, 2004] Vandervelpen, C. and Coninx, K. (2004). Towards model-based design support for distributed user interfaces. *Proceedings of the third Nordic conference on Human-computer interaction - NordiCHI '04*, pages 61–70.

[Varela, 2013] Varela, G. (2013). Autonomous adaptation of user interfaces to support mobility in ambient intelligence systems. In *EICS 2013 - Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 179–182.

[Varela et al., 2011] Varela, G., Paz-Lopez, A., Becerra, J. A., Vazquez-Rodriguez, S., and Duro, R. J. (2011). UniDA: Uniform device access framework for human interaction environments. *Sensors*, 11(10):9361–9392.

[Varela et al., 2013a] Varela, G., Paz-Lopez, A., Becerra Permuy, J. A., and Duro, R. J. (2013a). Decoupled Distributed User Interface Development Framework for Ambient Intelligence Systems. In *Proceedings of the 3rd Workhsop on Distributed User Interfaces: Models, Methods and Tools, In conjunction with ACM EICS 2013*, London.

[Varela et al., 2013b] Varela, G., Paz-Lopez, A., Becerra Permuy, J. A., and Duro, R. J. (2013b). The Generic Interaction Protocol: Increasing portability of distributed physical user interfaces. *Romanian Journal of Human - Computer Interaction*, 6(3):249–268.

[Varela et al., 2014] Varela, G., Paz-Lopez, A., Becerra Permuy, J. A., and Duro, R. J. (2014). Prototyping Distributed Physical User Interfaces in Ambient Intelligence Setups. *Distributed, Ambient, and Pervasive Interactions*, pages 76–85.

[Varela et al., 2007] Varela, G., Paz-López, A., Vázquez-Rodríguez, S., and Duro, R. J. (2007). HI3 project: Design and implementation of the lower level layers. In *Proceedings of the 2007 IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems, VECIMS 2007*, number June, pages 36–41.

[Veloso and Sendín, 2005] Veloso, M. and Sendín, M. (2005). Infrastructure for Plastic User Interfaces under a Dichotomic View. In *Proceeding of the International Congress Communicating Naturally Through Computers (Interact 2005)*.

[Villar et al., 2006] Villar, N., Gilleade, K. M., Ramduny-Ellis, D., and Gellersen, H. (2006). VoodooIO Gaming Kit: A real-time adaptable gaming controller. *Computers in Entertainment (CIE)*, 5(3):7.

[W3C, 2003] W3C (2003). W3c multimodal interaction framework. URL: http://www.w3.org/TR/mmi-framework/.

[W3C, 2007] W3C (2007). Cc/pp information page. URL: http://www.w3.org/Mobile/CCPP/.

[W3C, 2011a] W3C (2011a). Owl 2 web ontology language document overview. URL: http://www.w3.org/TR/owl2-overview/.

[W3C, 2011b] W3C (2011b). Resource description framework (rdf): Concepts and abstract syntax. URL: http://www.w3.org/TR/rdf-concepts/.

[Weiser, 1991] Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 265(3):94–104.

[Wiecha et al., 1990] Wiecha, C., Bennett, W., Boies, S., Gould, J., and Greene, S. (1990). ITS: a tool for rapidly developing interactive applications. *ACM Transactions on Information Systems*, 8(3):204–236.

[Wolf et al., 2011] Wolf, P., Strnad, O., Haller, H., Schmidt, A., Peibner, M., Hernandez, J. A., and van de Korput, R. (2011). MyUI: Context Ontology, User Modelling Concept and Context Management Architecture. Technical report.

[X10-Europe, 2010] X10-Europe (2010). X10 europe official web site. URL: http://www.x10europe.com.

[Xie et al., 2008] Xie, L., Antle, A. N., and Motamedi, N. (2008). Are tangibles more fun? *Proceedings of the 2nd international conference on Tangible and embedded interaction - TEI '08*, pages 191–198.

[Zuckerman and Gal-Oz, 2013] Zuckerman, O. and Gal-Oz, A. (2013). To TUI or not to TUI: Evaluating performance and preference in tangible vs. graphical user interfaces. *International Journal of Human Computer Studies*, 71(7-8):803–820.

List of Figures

3.1 Various examples of Phidgets physical devices. 19

3.2 Examples of VoodooIO devices. 20

3.3 Examples of iStuff devices. 21

4.1 Examples of Yes/No interaction with the OMNI system. 42

4.2 Picture of the OMNI remote controller. 43

4.3 Sketched UI of the Environmental Music Player using WIMP user interfaces. 45

4.4 Example of various interaction possibilities for the volume selection in the Environmental Music Player application. 46

4.5 Example view of a possible EvacUI running different UIs, for a variety of users, in different locations of a ship, during an evacuation process. 47

4.6 The IMA layer facilitates the implementation of different UIs by decoupling the UI control logic from the interaction resources. On the left, the application without using the IMA layer. On the right, the application when using the IMA layer. 51

4.7 Detailed conceptual block diagram of the TIAF with support for the IMA layer. The User Interface is defined by developers using the concepts provided by the Abstract Interaction Model. Instances of those concepts are used in the UI Control Logic to implement the particular behavior of the UI and its interaction with the system's business logic. 52

4.8 The ILA allows developers to use remote IRs without any knowledge about the required specific networking protocols. On the left, an application without the ILA has to access the IRs directly using their specific protocols. On the right, an application with the ILA is isolated from the knowledge of specific networking protocols. 54

4.9 Detailed conceptual block diagram of the TIAF with support for the IMA+ILA layers. 56

4.10 Without the ICA, developers and/or installers have the responsibility to select the adequate IRs for each scenario. With the ICA, it is the system itself, based on context models, who, autonomously, selects the most adequate IRs for each scenario. . . . 57

4.11 Detailed conceptual block diagram of the complete TIAF framework with support for the IMA+ILA+ICA layers. 59

4.12 Class conceptual model of the Abstract Interaction Model directly inspired by the UsiXML Abstract User Interface Model. . 63

5.1 Dandelion deployment block diagram. UI Control logic is decoupled from the interaction resources by the Dandelion UI management system. Furthermore, Dandelion relies on UniDA to support a wide range of physical devices (sensors, actuators, appliances, etc.) from different manufacturers. 71

5.2 Dandelion detailed architecture block diagram. 72

5.3 Dandelion portable UIs development process with IMA+ILA layers. In chapter 6, we will see how Dandelion implements the ICA to manage, at run-time, the mapping between abstract and final components, liberating developers/installers from that responsibility. 76

5.4 Sketched GUI version of the OMNI notification subsystem UI. . . 78

5.5 Sketched UI of the Environmental Music Player using WIMP user interfaces. Already shown in Figure 4.3. 79

5.6 External interface of the Dandelion User Interface Controller as seen by developers and UI control logic code. 83

5.7 Many different UIs can be deployed for the same Abstract UI definition and the same UI Control Logic. The installer of the system only has to specify different mappings, between abstract and final UI elements, for each scenario. 86

5.8 Detailed overview of the ILA implementation in Dandelion. The UI Controller operates as a router of abstract interaction operations from the UI Control Logic to the Final Interaction Objects, using the GIP as transport protocol. 87

5.9 Each FIO creates its own topic in the broker and then uses it to publish its generated GIP events. DUICs of all the applications using that FIO are subscribed to that topic. 89

5.10 FIO's registration process in the FIO repository. FIO descriptions are stored in a repository that can be used by the installers, or the ICA, in order to set up the abstract to final mappings. . . 91

5.11 Without UniDA, an HIE application using heterogenous devices is exposed to the particular characteristics and APIs of each device technology. 93

5.12 By using UniDA, HIE applications are isolated from device APIs and their particular characteristics. All the devices are accessed through the same channel and API, and all the devices of the same type (independently of their technology) are interfaced in the same way. 93

5.13 UniDA allows us building FIOs that support many devices of the same kind but with different technologies and APIs. 95

5.14 Class diagram showing the different concepts and their relations that populate the UniDA Device Network Model. 98

5.15 This diagram shows an architecture and deployment diagram of a system (the FIOs of Dandelion) using the UniDA Library and Gateways. The FIO logic interacts exclusively with the UniDA library through the Device Model and the Device Operations and Management. UniDA relies on a set of Gateways to translate the Device Model concepts to the particular APIs and protocols of each specific device technology. 103

5.16 OMNI system IRs/FIOs for the different scenarios. 111

5.17 Two examples of FIOs for the OMNI notification user interface. First example: a smartphone with two output interaction facets, one using the screen, and one using voice synthesizing, and two action facets, for the yes/no actions. Second example with three FIOs:a TV display, with one output interaction facet using the screen; a voice synthesizing FIO using a Festival server; and a notification FIO that uses colored lights to notify the presence of a message. 116

5.18 Two different possibilities for the final implementation of the music style selection interaction primitive. First, a touch UI using a smartphone, useful for mobile and outdoor environments. Second, a leap motion controller allows the selection of the style using hand and finger gestures. 120

6.1 The ICA allows Dandelion to autonomously select, at run-time, which set of FIOs to use for each usage scenario. This way, Dandelion is able to react to context changes at run-time, modifying the Final UI to keep it operating within the natural interaction constraints required by Ambient Intelligence UIs. 126

6.2 Overview of the Dandelion UI adaptation to context process. The UIB creates a query to ask the FIO repository for a list of FIOs that comply with the requirements of the usage scenario. This list is used to configure, in the UI controller, the mappings between FIOs and abstract UI elements. 128

6.3 UML class diagram showing the different concepts, and their relations, in the Dandelion User Profile Model. 133

6.4 UML class diagram displaying the different concepts, and their relations, that make up the Dandelion Environment Profile Model. 136

6.5 UML class diagram showing the concepts, and their relations, of the Dandelion Scene Profile Model. 138

6.6 The FIO selection process is performed in four different phases. First, the models are used to create a specification of the characteristics required of a FIO to be considered adequate for the usage scenario. Second, the specification is used to select a list of FIOs from all the ones available in a particular physical environment. Third, one FIO is selected for each interaction facet. Finally, the FIO mapping is updated in the UIC. 140

6.7 UML class diagram showing the concepts, and their relations, of the FIO description model. 143

6.8 Conceptual class diagram displaying the components, and their relations, of the FIO Specification and FIO Query models. 145

6.9 The Ideal FIO specification is generated for each usage scenario, while the interaction specification is generated for each one of the interaction facets included in the abstract UI. 148

6.10 The FIO adequateness metric is calculated by an aggregation of four independent adequateness measures (Interaction, Modality, Physical and Usage similarities) between the query specifications and a FIO description. 149

6.11 Four of the FIOs used for the EMP example. First, a KNX home automation remote controller, connected to Dandelion using UniDA. Second, a Leap Motion device for finger and hand gesture recognition. Third, a custom Android GUI application. Fourth, a TV display for output with a Kinect camera for input (using gesture recognition). 159

6.12 The OMNI UI adapted to the Deaf User scenario. The Kinect and the wall buttons can be used to control system (change channels, answer questions). The display is used to output notification messages, and the colored lights are used to request the focus of the user when a notification is displayed. 171

6.13 An example of FIO for the EvacUI user interface example. It uses a smartphone display to show direction information using symbol and language production modalities. Furthermore, it also provides another interaction facet using speech production. In the second image, there are two different FIOs. 177

List of Algorithms

5.1 XML code describing the OMNI notification UI with UsiXML. . 79
5.2 XML code describing the player controls part of the Abstract UI model for the EMP. 81
5.3 XML code describing the audio metadata part of the Abstract UI model for the EMP. 82
5.4 UI Control Logic code required to set the business logic actions that must be called when the user triggers the actions to start playing music or to stop it in the EMP example. 84
5.5 UI Control Logic code required to output the song title string to the user. It is executed each time the song changes. 85
5.6 Container skeleton of the OMNI abstract UI. 107
5.7 Control part of the OMNI abstract UI. It contains the abstract interaction primitives required to power on/off the system, change the channel, and change the audio volume. 108
5.8 Small code snippet showing an example implementation of a user action callback (IDandelionActionCallback interface). In this case, this code corresponds to the OMNI action to change the channel to the next one available. 109
5.9 Example of how the DUIC can be used to output information to the users. As can be seen, it is very easy to use, and it operates completely at the abstract level, without exposing the code to any particularities of the modalities and technologies of the UI. . 110
5.10 Example source of the OMNI display FIO for notifying messages. It uses one implementation of the IOutputAction interface in order to support one output interaction facet. 112
5.11 Example of mapping file for the OMNI system where the notification dialog message is associated to the OMNI display FIO, and the 'yes' and 'next channel' actions are associated to hand gestures. 114

5.12 How to use interaction hints to provide some level of customization to the final UI. In this case, we are suggesting the color of the notification message. 115
5.13 Code snippet of the EMP abstract UI dedicated to the selection of music style. 117
5.14 Code snippet of the EMP UI control logic to show the dynamic list of music styles as a selection 118
5.15 EMP implementation of the selection callback to change the music style. 119
5.16 Example of abstract UI code defining an output interaction primitive for images. 120
5.17 Example of how to use Dandelion to output an image. The image data must be first converted to PNG format. Dandelion automatically encodes the raw data of the image in base64 to send it to the FIOs using the GIP. 121
5.18 Example of output action of a FIO that uses an Android smartphone screen to show the album art of the EMP to the user. . . . 122
6.1 Fuzzy rules for the SYMBOL modality selection FIS. 147
6.2 UsiXML definition of the OMNI notification abstract UI. 166
6.3 Code snippet showing an example of how to use the Interaction Hints mechanism to indicate Dandelion that one particular interaction must be implemented in a redundant way. 179

List of Tables

3.1 Comparison of solutions for the development of Plastic Distributed Physical User Interfaces. 34

6.1 Values of the different environment profiles associated to each usage scenario of the EMP example. 157

6.2 Values of the different scene profiles associated to each usage scenario of the EMP example. 157

6.3 Detailed description of the EMP FIOs using the FIO Description model. 160

6.4 Ideal FIO specifications generated for the four usage scenarios of the EMP example. 161

6.5 This table displays the FIO adequateness values calculated for each combination of FIO and usage scenario. The green rows show the overall FIO adequateness values, while the other rows show the values of different sub-metrics that make up the overall adequateness using the equation 6.10. 162

6.6 Detailed description of the FIOs available for the implementation of the music selection abstract interaction facet. 164

6.7 FIO adequateness values calculated for each FIO and usage scenario for the EMP music selection interaction facet. 165

6.8 User profile definitions for the six different users included in the OMNI example for UI adaptation to user characteristics. 167

6.9 Ideal FIO specifications generated for each usage scenario considered in the OMNI example. 168

6.10 Detailed description of the FIOs available for the implementation of the notification message output abstract interaction facet. . . 169

6.11 Results of the FIO query looking for FIOs with output support and high level of adequateness regarding the Ideal FIO specifications of table 6.9. 170

6.12 FIO list provided by the FIO Repository for the query looking for FIOs supporting one ACTION interaction facet and adequate to the Ideal FIOs of the different usage scenarios of the OMNI example. 173

6.13 Environment and scene profiles for the EvacUI ferry ship Human Interaction Environment example. 174

6.14 Detailed description of the user profiles defined for the EvacUI example. 175

6.15 Ideal FIO specifications generated by Dandelion for the different EvacUI usage scenarios associated to the Cabin, Corridor, Deck, and Vehicle Room physical environments. 176

6.16 Detailed descriptions of the FIOs used for the EvacUI demonstration example. 178

6.17 FIO adequateness results for the different EvacUI usage scenarios related to Cabin, Corridor, Deck, and Vehicle Room environments. 180

7.1 Comparison of Dandelion against the main available solutions for the development of Plastic Distributed Physical User Interfaces reviewed in chapter 3. 188