Vrije Universiteit Brussel

Faculty of Sciences and Bioengineering Sciences
Department of Computer Science

eSPACE

Conceptual Foundations for End-User Authoring of Cross-Device and Internet of Things Applications

PhD Candidate:
Audrey Sanctorum

Promoter:
Prof. Dr. Beat Signer

Dissertation presented in fulfilment of the
requirements for the degree of Doctor of Sciences

Jury:

Prof. Dr. Olga De Troyer	Vrije Universiteit Brussel (Chair)
Prof. Dr. Johan Loeckx	Vrije Universiteit Brussel (Secretary)
Prof. Dr. Beat Signer	Vrije Universiteit Brussel (Promoter)
Prof. Dr. Bart Jansen	Vrije Universiteit Brussel
Prof. Dr. Gaëlle Calvary	Grenoble Institute of Technology
Prof. Dr. Jean Vanderdonckt	Université catholique de Louvain

July 2020

Abstract

Over the last few decades we have witnessed an increasing number of smart devices, ranging from smartphones and tablets to wearable devices such as smartbands, smartwatches and smart glasses. Despite the fact that people start using multiple smart devices simultaneously and perform some of their tasks by navigating from one device to another, it is still a challenge to support these kind of interactions between devices. Interaction between devices is often limited to data synchronisation via the cloud or restricted to dedicated applications only running on devices from the same manufacturer. Therefore, various research has been carried out to explore different solutions for facilitating cross-device interaction (XDI) and the distribution of data or user interfaces (UIs) across multiple devices.

Next to smart devices, in recent years we observe the rise of Internet of Things (IoT) solutions, allowing digital user interfaces to communicate with enhanced everyday objects. These so-called IoT objects or *things* are equipped with some hardware allowing them to communicate using different wireless protocols. Popular IoT devices are smart light bulbs, thermostats as well as speakers. The plethora of new devices lead to additional challenges including (but not limited to) interoperability, privacy and security of IoT devices. Further, since most of these IoT devices come with their own dedicated application, people end up with a fragmentation of control where many different applications are necessary to manage their devices.

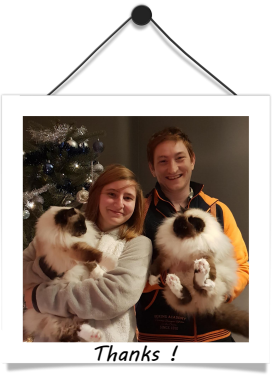
While existing research has mainly focused on either cross-device solutions or the Internet of Things, the unification of these two closely related research domains is often neglected, resulting in solutions designed only for smart devices and systems specifically developed for IoT objects only. However, with the pool of smart technologies that users have to deal with in the present day, they need solutions that can communicate, manage, and control *all* these different types of devices and *things*. Additionally, since the users' needs are difficult to anticipate in advance due to their complex and evolving nature (together with technological advances), developers cannot satisfy the diverse and frequently changing user needs. Therefore, we see emerging end-user

development solutions, which allow so-called end users without programming experience to tailor or create new software artefacts according to their specific needs.

In this dissertation, we aim to improve end users' control over their smart environments that include smart devices as well as IoT devices, by developing a solution that *unifies the advantages of both the XDI and IoT research domains*, and provides the *appropriate abstractions based on the users' mental models of cross-device and IoT interactions*. Thereby, we enable end users to create and modify their own XD and IoT applications based on their *existing knowledge*. The unification of XDI and IoT further allows users to manage all their devices in a single place rather than having a fragmented control over multiple applications, while the appropriate abstractions will help them to more easily become familiar with our solution. In order to develop the necessary conceptual foundations, we investigated related work on end-user development in the domains of cross-device interaction and the Internet of Things. Based on this detailed analysis we derived a number of requirements for the end-user authoring of XD and IoT applications, which have been augmented with additional requirements resulting from a number of use case scenarios involving XD and IoT interactions.

Since our goal is to allow end users to create their own applications and user interfaces, we opted for a model-based approach to facilitate the UI development process. We analysed existing model-based approaches, helping us to finalise our requirements, and design a reference framework and conceptual model. Our eSPACE reference framework structures the UI development process for user-defined applications while the conceptual model introduces all the necessary components for this development process. In order to get a better understanding of end users' mental models when it comes to cross-device and IoT interactions, we performed an elicitation study to explore which metaphors are most used by people to visualise these interactions. Based on related work and our study, we then developed a set of design guidelines for the creation of cross-device and IoT end-user authoring solutions. Given the requirements, our reference framework, the conceptual model and design guidelines, we finally came up with the design of a cross-device and IoT end-user authoring solution that allows end users to create and modify their XD and IoT applications and user interfaces. The resulting eSPACE authoring tool fulfils most of the presented requirements, uses the concepts introduced by our model, follows the UI development process of our reference framework to construct user-defined applications and complies to all our design guidelines.

Acknowledgements



First and foremost, I would like to thank my promoter Prof. Dr. Beat Signer, for supporting and guiding me during my PhD. Throughout this process you have given me endless feedback and taught me valuable lessons that I will carry along the rest of my academic career. I would also like to express my gratitude towards Prof. Dr. Olga De Troyer without whom this research could not have been completed.

Next, I would like to thank all members of the jury, Prof. Dr. Olga De Troyer, Prof. Dr. Johan Loeckx, Prof. Dr. Bart Jansen, Prof. Dr. Gaëlle Calvary and Prof. Dr. Jean Vanderdonckt, for their efforts, input and validation. Their participation led to fruitful discussions which positively contributed to the final version of this dissertation. The research described in this dissertation has been funded by the Research Foundation of Flanders (FWO), whom I also thank.

I further thank my colleagues and friends for helping me during my PhD; their coding advice and brainstorming sessions were very valuable to me, thank you Reinout, Renny and Jan. Moreover, I would also like to thank Jon and Suzanne for assisting me during the elicitation study, as well as all participants who volunteered for my user studies. During my PhD I had the opportunity to supervise six master thesis students, whom I thank for having given me new perspectives on the topic.

Without some healthy distractions my PhD days would not have been the same, therefore I want to thank all my friends. The list is too long to mention each one of you, many thanks to all my friends from the AI lab, secretariat, IAESTE, Infogroep, high school and friends I met during my studies at the university. Thank you all for the game nights, the fun ladies at the movies events, the long talks, the drinking sessions and the trips we shared together!

Last but not least, I would like to thank my family that has always been there for me. I thank my parents, brother and grandfather for the moral support which gave me the courage to go on. A special thanks to my mother for helping me with interview transcripts and for bringing me food when needed. Next, I want to express my

gratitude towards my boyfriend, Jon, for keeping up with my mood swings, for the dinners he prepared and for encouraging me during my PhD. Three years ago I got the chance to adopt two wonderful cats, whom I thank for inspiring me when writing the use case scenario and who gave me some necessary distractions. Finally, I would like to acknowledge my grandmother for being with me in my heart and thoughts throughout the years.

Contents

Acronyms

AC	Active Component
API	Application Programming Interface
AUI	Abstract User Interface
BLE	Bluetooth Low Energy
CLI	Command-Line Interface
CRF	Cameleon Reference Framework
CSS	Cascading Style Sheets
CTT	ConcurTaskTree
CUI	Concrete User Interface
DComp	Distributed Component
DeUI	Distributable User Interface
DIY	Do-It-Yourself
DS	Design Science
DSL	Domain-Specific Language
DSRM	Design Science Research Methodology
DUI	Distributed User Interface
ECA	Event-Condition-Action
eSPACE	end-user Smart PIACE
EUD	End-User Development
FUI	Final User Interface

GUI	Graphical User Interface
HCI	Human-Computer Interaction
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IFML	Interaction Flow Modeling Language
IO	Interaction Object
IoT	Internet of Things
IS	Information System
JPA	Java Persistence API
LoRa	Long Range Radio
MBD	Model-Based Design
MB-IDE	Model-Based Interface Development Environment
MBUID	Model-Based User Interface Development
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Multi-Display Environment
MQTT	Message Queuing Telemetry Transport
NUI	Natural User Interface
OMG	Object Management Group
ORM	Object Role Model
OS	Operating System
PC	Personal Computer
PSSUQ	Post-Study System Usability Questionnaire
PUI	Physical User Interface
QoS	Quality of Service
REST	Representational State Transfer
RQ	Research Question

RSL	Resource-Selector-Link
SIGCHI	Special Interest Group on Computer-Human Interaction
SVG	Scalable Vector Graphics
UI	User Interface
UIDL	User Interface Description Language
UIe	User Interface Element
UIML	User Interface Markup Language
UsiXML	USer Interface eXtended Markup Language
WoT	Web of Things
WoX	Web of Topics
WYSIWYG	What You See Is What You Get
XD	Cross-Device
XDI	Cross-Device Interaction

Chapter 1

Introduction

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

The proliferation of devices, such as smartphones, tablets and smartwatches, has changed the way we interact with the world around us. While nowadays at least 70% of the total population own a smartphone, the device ecosystem continues to grow with diverse and innovative Internet-connected devices. As a consequence, on average people now use more connected devices than a few years ago. Belgians, for instance, had on average 2.2 connected devices in 2013, compared to 2.9 in 2017, which is a compound annual growth rate (CAGR) of 7.15% over 4 years[1]. In such a multi-device world, people constantly have to juggle between multiple devices. A study by Google [112] with 11 964 participants in 2016 showed that 57% of the U.S. users use more than one type of device and that 21% are concurrent usages[2]. A larger study even depicts that on average 21% of the people in 30 different EU countries use 5 or more devices, as shown in Figure 1.1 [113]. Only 29% of this population do not use more than one device.

In 2012, Google defined two categories of multi-device usage, including sequential and simultaneous usage [111]. Figure 1.2 illustrates these two categories. 90% of sequential usage is done to accomplish a task over time, where users move from one device to another to accomplish a task. Online tasks are often initiated on a

[1] `https://www.thinkwithgoogle.com/intl/en-gb/advertising-channels/mobile/consumer-barometer-study-2017-year-mobile-majority/`

[2] Concurrent usage is defined in this study as the use of the computer browser and another device within the same hour at home.

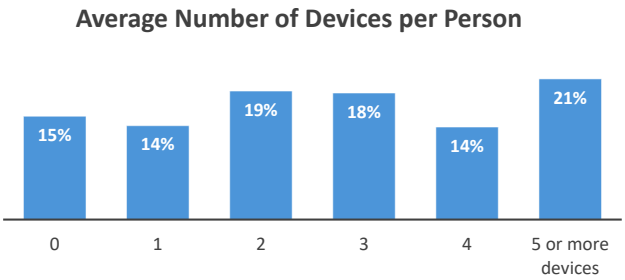
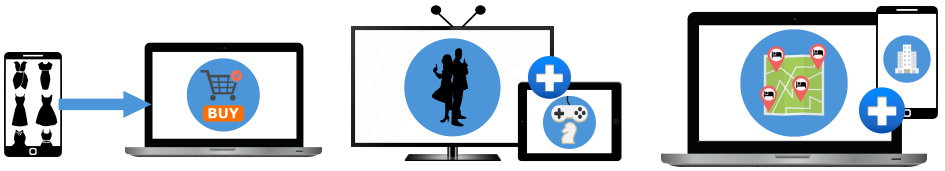


Figure 1.1: Average number of devices per person in EU countries

smartphone but then finished on a computer. To move from one device to another, people often rely on *search* (63%). Others directly navigate to the corresponding website (52%) or send themselves the link to the website via email (49%). Next, simultaneous usage is defined as using more than one device at the same time. There are two categories of simultaneous usage. The first one is multitasking, where devices are used for an unrelated activity, such as gaming on a phone while watching TV. The second type is complementary usage, where devices are used for a related activity, such as watching a movie on the TV and looking up an actor on the tablet. 78% of simultaneous usage is dedicated to multitasking and the remaining 22% consists of complementary usage.



(a) Sequential multi-device usage, finishing a task (related activity)

(b) Simultaneous multi-device usage, multitasking (unrelated activity)

(c) Simultaneous multi-device usage, complementary usage (related activity)

Figure 1.2: Categories of multi-device usage

Similar multi-device behaviours have been identified in a study by Microsoft [139], who distinguished four kinds of multi-device behaviours. The most popular behaviour is *content grazing* where users use two or more screens to access unrelated content at the same time, which corresponds to simultaneous multitasking. The second most common behaviour is *investigative spider-webbing* where users use one device to search information related to an activity performed with another device. This corresponds to the simultaneous complementary usage identified by Google. The third behaviour is the *quantum journey*, where people use multiple devices to finish a task, which corresponds to sequential usage. The least common behaviour, still

performed by 39% of the users, is *social spider-webbing*, where users share content of what they have achieved on other devices. For example, this includes the sharing of the scores from a game console using their phones or tablets. Note that, this could also still be seen as sequential usage.

Multi-device interaction, or cross-device interaction (XDI) as more commonly referred to in research has become a part of our daily life. Therefore, researchers working in this domain try to facilitate cross-device interactions by providing new kinds of XD gestures, frameworks and user interfaces to interact with data across devices. The field of cross-device interaction has also led to research in distributed user interfaces (DUIs), allowing user interface components to be distributed across devices. A DUI is defined as follows by Elmqvist:

Definition 1.1 – Distributed User Interface

"*A distributed user interface is a user interface whose components are distributed across one or more of the dimensions input, output, platform, space, and time*" [76].

The five dimensions mentioned in Definition 1.1 are discussed in the following. The *input* dimension stands for managing input on one or more devices. The graphical *output* can be tied to one device or display but it can also be distributed across multiple devices. The interface can be executed on one or more computing *platforms*. Next, the *space* dimension refers to the restriction of an interface to be in the same physical (and geographic) space or distributed over remote interactive spaces. Finally, the *time* dimension represents interface elements that execute simultaneously (synchronously) or distributed in time (asynchronously).

Cross-device interaction and distributed user interfaces offer the functionality for facilitating sequential and simultaneous multi-device usage, as both XDI and DUIs reduce the complexity of interacting across devices. For example, one could use a cross-device gesture for transferring content from one device to another in order to seamlessly execute a sequential task, which is faster than sending the data via email or transferring it by using a USB stick. Further, an interface could be distributed and synchronised across multiple devices to perform a simultaneous task. An example of such a task could be "booking a hotel" by showing a map with all the choices on a TV while exploring each hotel's details on the smartphone.

In addition to smart devices, an increasing number of smart objects or "things" started to emerge, extending Internet connectivity beyond standard devices and giving ordinary objects (with embedded technology) the ability to communicate and interact over the Internet. The term Internet of Things (IoT) was coined by Kevin Ashton

in 1999 for describing a system where the "*Internet is connected to the physical world via ubiquitous sensors*" [13]. Since then, IoT went a long way and is now present in different sectors ranging from smart homes, energy management, elderly care and healthcare to transportation and agriculture.

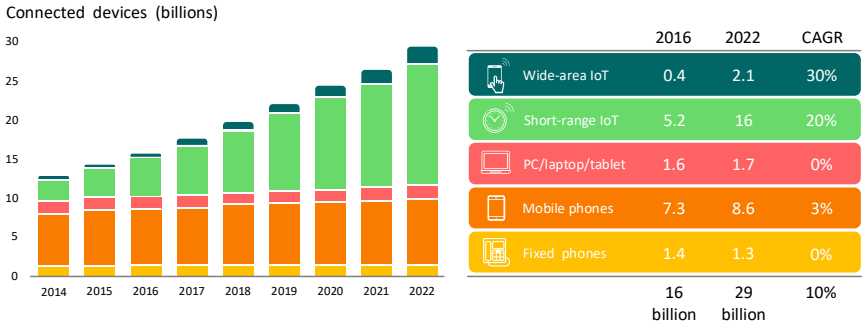


Figure 1.3: Worldwide total of connected devices [46]

Nowadays more "things" are connected to the Internet than people and as can be seen in Figure 1.3, the number of connected devices is still growing. The figure has been taken from a mobility report (2016) with an IoT forecast by Ericsson [46], which divided the IoT into short-range IoT and wide-area IoT. The short-range IoT category includes all devices connected through Wi-Fi, Bluetooth, Zigbee and other typical protocols with a range of up to 100 meters. The long-range IoT category contains devices using cellular connections and unlicensed low-power technologies such as LoRa.

A recent definition of the IoT has been provided by Guinard and Trifa [100]:

Definition 1.2 – Internet of Things

"*The Internet of Things (IoT) is a system of physical objects that can be discovered, monitored, controlled or interacted with by electronic devices which communicate over various networking interfaces, and eventually can be connected to the wider Internet*" [100].

While the IoT presence is growing, coping with this large range of devices is challenging on many levels. There is still no consensus on the (network) architecture, the scale and complexity are challenges as well and for end users there is the "baskets of remotes" problem [90], since every object or set of objects has its own remote controller, either physical or through an application on the user's phone. In this dissertation we mainly focus on this last problem for end users.

Considering the pool of smart devices and smart *things* that users have to deal with today, there is a need for a solution to control, manage and cope with interaction across these smart technologies. Such a solution should **unify** the interaction across devices and *things* so that users can control their entire smart environment with one solution rather than having the control fragmented across different applications. In addition, since we live in a world where technology keeps evolving over the years, a solution should also be flexible so that it can incorporate new technologies. Moreover, beyond technological changes, the solution must face the complex and evolving nature of the end users' needs. However developers or researchers cannot anticipate all user needs in advance, in particular if these change over time. In order to cope with such needs, the concept of end-user development (EUD) emerged [163] and Lieberman et al. [126] defined EUD as follows:

Definition 1.3 – End-User Development

"*A set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact*" [126].

Many people work on tasks that rapidly vary on a yearly, monthly or even daily basis, which results in software needs that are diverse, complex and frequently change. While a developer cannot directly meet all these needs, an end user could respond faster to their evolving requirements since they have more domain knowledge [165]. By applying EUD, end users can create and tailor their applications to their needs more closely than any developer ever could.

In a nutshell, the intended solution we mentioned earlier should unify research artefacts from the fields of XDI, DUI and IoT in order to support and control all smart technologies in a user's smart environments, and apply a flexible, extensible EUD approach as a mean to support changing and evolving user needs and technologies. Note that throughout this dissertation the term *smart technologies* will be used to refer to both smart devices and smart *things*. indexsmart technologies

1.1 Research Questions

Over the years various new smart technologies have been introduced and research in the domain of cross-device interaction, distributed user interfaces and the Internet of Things has emerged to help developers, designers and end users to cope with the broad spectrum of connected devices and *things*. Further, additional applications have been brought to the market for end users to interact with these smart technologies. However, no solution provides control over all these smart technologies

in one place, which causes the *fragmentation problem*, forcing users to have different configurations and applications—which also come with different interfaces—to control their smart environments. Moreover, the proposed applications often provide *limited functionality* and *customisability* to their users. Therefore, research on end-user development aims to provide more power to end users as mentioned earlier in this chapter. This research has led to end-user authoring tools for creating tailored cross-device interaction and IoT applications. However, only few of these tools offer the possibility to control all smart technologies. More importantly, they introduce new techniques and abstractions to create user interfaces and applications with *no consensus on which guidelines* to follow for the creation of these kinds of end-user authoring tools.

The fragmentation, lack of uniformity and limited functionality problems as well as the lack of consensus about appropriate guidelines for end-user authoring tools for smart technologies can be formalised in the following problem statement:

Problem Statement *As new smart technologies emerge in large numbers, it is not only difficult for developers to keep up with these evolving technologies, but even more for end users. This is caused by the end user's lack of a flexible, unified and extensible solution for controlling all smart technologies (in one place) that can cope with their individual and evolving needs.*

In this dissertation we propose the end-user development of cross-device and IoT user interfaces to address the given problem statement. The user-defined XD and IoT user interfaces would of course support interaction with all kinds of different smart technologies. By using the appropriate abstractions for hiding the technical details, end users would be able to create and tailor such user interfaces without the need for any programming skills. We therefore formulate our main research question for resolving the problems stated above as follows:

Main Research Question *How can we enable end users to have a better control over their smart technologies by using appropriate abstractions to hide the complex technical details when creating and modifying unified cross-device and IoT user interfaces specifically made to control these smart technologies?*

In order to tackle this main research question, we defined four subquestions which will contribute to the overall solution. In a first phase, we will investigate the requirements of end-user authoring tools providing cross-device and IoT interaction functionality by carrying out some background research in end-user development approaches for cross-device and IoT solutions, leading to the following research question:

Research Question 1 (RQ1) *What are the main requirements for the end-user authoring of unified cross-device and IoT user interfaces?*

This question will be answered by performing a thorough examination of related work in cross-device and IoT end-user development research. As a result, a summary and discussion about the existing solutions, their limitations and the different abstractions or metaphors that are used in these solutions to hide technical details will be provided. Based on this analysis, a number of requirements will be derived and further refined with the help of a use case scenario for cross-device and IoT interactions.

In order to facilitate the development process of our XD and IoT EUD authoring solution, we might explore a model-based approach. A model-based approach would allow us to accurately define all important concepts of XD and IoT applications, which could be reused and eventually extended by other developers. Therefore, we will also examine related work on existing model-based solutions, which might lead to some additional requirements and brings us to our next research question:

Research Question 2 (RQ2) *What are the necessary concepts and methods to address the requirements resulting from answering RQ1?*

As discussed later, it is not easy to reuse, adapt and extend existing solutions to support the requirements resulting from RQ1. Therefore, we will build on the knowledge gained from our background research to present new conceptual foundations supporting all identified requirements, which will lay the basis for the development of EUD authoring solutions for cross-device and IoT applications.

The second to last phase of this dissertation focusses on the design of the frontend of the authoring solution to answer the following question:

Research Question 3 (RQ3) *Which metaphors or abstractions should be used on top of our conceptual foundation to allow end users to visualise and create their unified cross-device and IoT interactions?*

In order to answer this question, in a first instance we conduct a user elicitation study with as objective to find the most appropriate metaphors for IoT and cross-device interaction fitting the mental models of users. Further, we will also analyse existing guidelines for the creation of EUD authoring tools and look back at the different metaphors used by the end-user authoring solutions investigated for RQ1. Based on the background research and user study, we will define guidelines involving metaphors for the creation of cross-device and IoT end-user authoring tools, leading to the last part of this dissertation to address the following research question:

Research Question 4 (RQ4) *How can we design a unified cross-device and IoT EUD authoring tool given the requirements from RQ1, the conceptual foundations from RQ2 and the guidelines including the appropriate metaphors found in RQ3?*

The objective of this last part is to create a compliant proof-of-concept EUD authoring solution based on the artefacts resulting from addressing the previous research questions. The functionality of the tool will be shown through the presentation of different use case scenarios later in this dissertation. An initial user study will be performed to get some insights on end users' first thoughts about our proof-of-concept authoring solution and to have a first indication about the user satisfaction of our end-user authoring tool for the creation of XD and IoT applications.

1.2 Research Approach and Methodology

In this section we describe the research methodology that we persued in order to answer the previously mentioned research questions. We decided to follow the *Design Science Research Methodology* (DSRM) for information systems research that has been defined by Peffers et al. [169]. The DSRM is widely used by design science researchers to guide them when producing, presenting/structuring and evaluating research outcomes. This methodology fits our described research, as we conduct design science (DS) research in information systems (IS).

The result of design science research in information systems is an *IT artefact* created to solve an important problem which can be either a theoretical artefact, such as constructs, models and methods applied in the development of IS, or a computer-based artefact, such as instantiations in the form of software frameworks and prototype systems [104]. In this dissertation, our research outcome will be in the form of some theoretical artefacts such as our reference framework, conceptual model and design guidelines, and a concrete artefact represented by a proof-of-concept prototype of a unified XD and IoT end-user authoring tool.

The DSRM process model is based on prior design science research principles and includes six activities. The first activity is the *problem identification and motivation* (1). The next activity involves the *definition of the objectives of a solution* (2). The third one is the *design and development* (3) followed by the *demonstration* (4), *evaluation* (5) and finally the *communication* (6). In the remainder of this section we provide an overview how these different DSRM activities have been followed in our dissertation.

Activity 1: Problem Identification and Motivation

We identified the main problems in the previous section as a lack of a solution providing control over all smart technologies, leading to the fragmentation of control, since users have to use various dedicated applications and UIs in order to control their smart environments. In addition to the lack of a unified solution allowing cross-device as well as IoT interactions, current applications are limited in terms of flexibility, customisation and extensibility, making them not suitable to cope with evolving technologies and user needs.

In order to provide a solution to these problems, we investigated existing research in the domains of end-user development for cross-device and Internet of Things solutions, which aim at empowering end users to create and tailor their own XD or IoT applications according to their needs. The analysis of XDI and IoT EUD approaches allowed us to identify the requirements for an EUD solution that provides end users with a better control over their smart environments. A few additional requirements were derived from a use case scenario that involves XD and IoT interactions. We further opted for a model-based approach to simplify the development process. Therefore, existing model-based research in our domains of interest has been analysed as well, leading to some last requirements forming part of the answer to RQ1. The final set of requirements and investigation of related work, have been necessary to design a conceptual foundation for offering a flexible, customisable and extensible basis for XD and IoT EUD authoring solutions. Lastly, we also identified a problem at the visualisation layer, which is the lack of consensus and design guidelines for the developers of such EUD authoring solutions regarding which abstractions or metaphors to use for allowing end users to create and modify their XD and IoT applications. Therefore, in order to introduce design guidelines that are best suited for representing XD and IoT interactions, we analysed the users' mental models when dealing with such interactions in an elicitation study.

Activity 2: Objective of the Solution

With the objectives presented below, we aim to solve the problem stated earlier on by ultimately providing end users a usable solution that allows them to better manage their smart technologies and make these smart devices and *things* interact with each other more easily.

- Our first objective is the formulation of requirements for an end-user authoring solution allowing the creation of cross-device and IoT applications. The requirements should be derived from existing work as well as use case scenarios. Note that this objective will pave the way towards our final objective.

- The requirements resulting from the previous objective should serve as baseline for building a conceptual foundation that unifies and supports all of these requirements by providing abstract common concepts, which are found in cross-device and IoT authoring tools. In addition, this conceptual solution should be flexible, reusable, extensible and therefore futureproof.
- An important objective once we have our conceptual solution is to find the right abstractions in order to best make it available to end users. There is further a need for design guidelines for the correct use of the abstractions and/or metaphors in an end-user authoring environment.
- Finally, after having accomplished the previous objectives, a last objective is to develop a flexible and extensible end-user authoring tool that supports a wide variety of devices and enables cross-device and IoT interactions. Thereby, the XD and IoT end-user authoring tool is going to be informed by the defined requirements, conceptual foundations and design guidelines.

While we provided a summary of our objectives, they are further motivated and detailed throughout this dissertation.

Activity 3: Design and Development

As a next step, we created several research artefacts in order to complete the objectives stated in Activity 2. First, we introduced the eSPACE reference framework which unifies the concepts present in cross-device and IoT user interfaces. The reference framework decomposes the UI development process into four layers of abstractions to facilitate the UI design process. Next, based on the concepts proposed in the reference framework, we present the eSPACE conceptual model, a domain-specific model for customised cross-device and IoT user interfaces. Just as our reference framework, the conceptual model provides reusability, extensibility and flexibility, mainly by building on top of the RSL hypermedia metamodel [203]. Both, the framework and model are described in Chapter 4. The model is stored using the RSL link server [185], which has been extended to fully support our needs, as further explained in Chapter 4. Finally, as discussed in the Activity 2, we bring the properties and functionality of the reference framework and conceptual model to the visualisation layer, by developing the eSPACE end-user authoring tool for the creation of cross-device and IoT applications. Note that this tool has been designed based on the design guidelines that resulted from an elicitation study aiming to better understand an end user's mental model when dealing with XD and IoT interactions. The study can be found in Chapter 5, while the tool's design and implementation are detailed in Chapter 6.

Activity 4: Demonstration

We demonstrate the utility and value of our different research artefacts in multiple ways. For example, in order to illustrate the power of our conceptual model, we expose its functionality via a number of show cases in Chapter 4. The same holds for the reference framework, where the usefulness of each components is expressed through examples taken from a use case scenario presented in Chapter 3. By comparing the conceptual model with the requirements defined for our first objective to address RQ1, we further illustrate how our model and framework support reusability, flexibility and extensibility. Last but not least, we use the concepts introduced in our model and framework to build the eSPACE authoring tool, which serves as final demonstration of the potential of the introduced concepts.

Concerning the authoring tool, a detailed explanation of its design and functionality is presented in Chapter 6. The tool is further demonstrated through some usage scenarios and used by end users during an initial user study, as explained in the next activity.

Activity 5: Evaluation

As required by the chosen DSRM, we evaluated our research artefacts using different methods depending on the type of artefact. Note that, for some of the artefacts the demonstration process also served as an evaluation of this artefact. In the case of the eSPACE conceptual model, we compare the model's functionality with the requirements derived from our first objective and research question, and illustrate how our model fully complies to these requirements. We further integrate this model into the RSL-based link Server, which is later used as a running information system for our eSPACE authoring tool. For the eSPACE reference framework, we validated the modelled approach by implementing the authoring tool using the concepts introduced in the model together with the UI development process proposed by the framework. More precisely, user-defined applications developed with the authoring tool are built and decomposed based on the different layers defined in the reference framework. In doing so, we evaluate the validity of our model and framework.

Finally, in order to evaluate our design guidelines that resulted from the XD and IoT elicitation study presented in Chapter 5, we designed the eSPACE authoring tool according to those guidelines and based on our requirements. The tool has later been evaluated by end users in the form of an initial user study in order to gain more insights about limitations and potential future improvements of the tool based on the participants' feedback.

Activity 6: Communication

The last activity included in the DSRM is the communication activity, which consists of the presentation of this research project and resulting artefacts to the research community. During the course of this PhD, parts of the contributions listed earlier have been published at international peer-reviewed conferences and journals. A full list of these publications can be found in Section 1.4. By presenting our work at those conferences, we gained more insights about our research domain and got feedback about our ideas and future plans, which also served as input for this final dissertation. Note that, we also presented our work during the EICS conference workshop on cross-device user interfaces (XDUI) in 2016, in order to gain more visibility and meet leading as well as upcoming researchers in the domain of XDI. Part of the presented work is available in the form of a website that classifies XDI-related work based on multiple criteria[1]. Further, we supervised multiple Master's thesis students in the domain of XDI, IoT interaction and end-user development and they provided ideas, different visions and information about the current state of related work in these domains. Finally, our research contributions and artefacts are also presented throughout this dissertation.

1.3 Contributions

We briefly summarise this dissertation's main contributions in Figure 1.4 and describe them in the following:

- We present a **set of requirements** for a unified XD and IoT end-user authoring tool providing end users more control over their smart technologies. The requirements resulted from an investigation of related work in the domain of end-user development of XD and IoT applications as well as related work in model-based solutions in the XDI and IoT fields. Related work for the EUD of XD and IoT applications can be found in Chapter 2, while related work of model-based solutions is described in Chapter 4. The requirements derived from our analysis of related work were further refined by exploring a use case scenario involving cross-device and IoT interactions, which is presented in Chapter 3.
- A next contribution is the **eSPACE reference framework**, facilitating the creation of user interfaces by structuring the UI development process into multiple layers of abstraction. The work on this framework has been inspired by related work on model-based solutions as further outlined in Chapter 4. Note that, this contribution is represented as one of our *methodological* contributions in Figure 1.4.

[1] `https://dui.wise.vub.ac.be/classification`

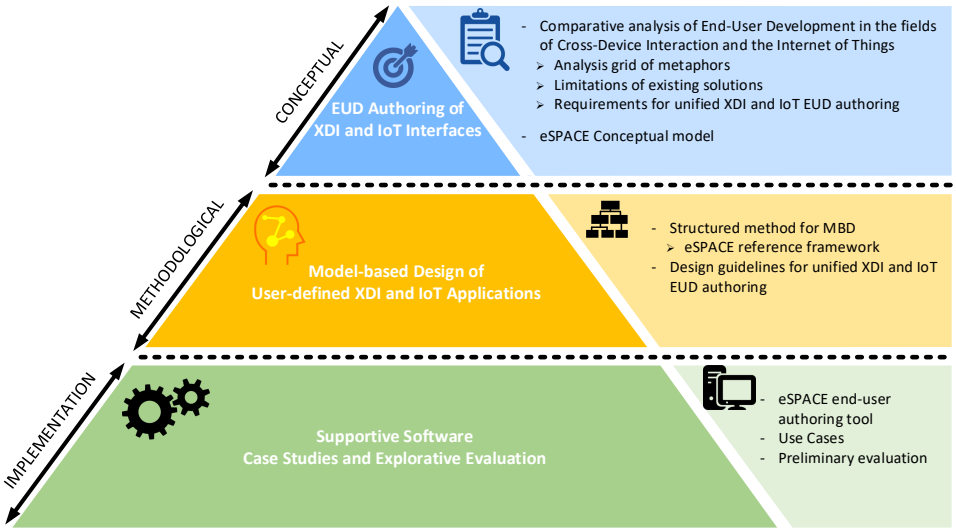


Figure 1.4: Summary of contributions

- We materialised the concepts introduced in the reference framework by providing a **conceptual model** based on the RSL hypermedia metamodel. Our domain-specific model allows the modelling of cross-device IoT applications and provides reusabilty, flexibility and extensibility of its components. The model is designed to satisfy all requirements stated earlier.
- While the model serves for the information storage on the RSL-based link server by providing support on the data level, we also introduced **design guidelines** for supporting the visualisation level. Based on an elicitation study, we analysed the mental models of people when referring to interaction across devices and IoT appliances, which resulted in a set of design guidelines that are presented in Chapter 5. Note that these guidelines are also based on related work in similar research domains.
- We designed and developed a proof-of-concept XD and IoT **end-user authoring tool**, called eSPACE, to demonstrate the potential of our reference framework, conceptual model and design guidelines. Even though the tool only represents an initial prototype, it already fulfils a good number of requirements that resulted from RQ1 and can easily be extended to support the remaining requirements. A full description of the eSPACE authoring tool is presented in Chapter 6.

1.4 Publications

Some of the contributions of this dissertation have been validated through different international peer-reviewed conferences and journal papers over the course of the last four years.

Related Publications

The list of publications related to this dissertation consists of the following articles.

- A. Sanctorum and B. Signer, June 2016. "*Towards User-Defined Cross-Device Interaction*". Proceedings of DUI 2016, Workshop on Distributed User Interfaces, pages 179–187, Lugano, Switzerland. https://doi.org/10.1007/978-3-319-46963-8_17.
- A. Sanctorum and B. Signer, May 2019. "*A Unifying Reference Framework and Model for Adaptive Distributed Hybrid User Interfaces*". Proceedings of RCIS 2019, International Conference on Research Challenges in Information Science, pages 1–6, Brussels, Belgium. https://doi.org/10.1109/RCIS.2019.8877048.
- A. Sanctorum and B. Signer, 2019. "*Towards End-User Development of Distributed User Interfaces*". Universal Access in the Information Society, 18(4):785–799, 2019. https://doi.org/10.1007/s10209-017-0601-5.
- A. Sanctorum, S. Kieffer and B. Signer, October 2020. "*User-driven Design Guidelines for the Authoring of Cross-Device and Internet of Things Applications*". Proceedings of NordiCHI 2020, Nordic Conference on Human-Computer Interaction, Tallinn, Estonia. (accepted for publication)

Unrelated Publications

We would also like to briefly mention the following unrelated publication, as this dissertation is presented in pursuit of a doctoral degree.

- S. Trullemans, A. Sanctorum and B. Signer, June 2016. "*PimVis: Exploring and Re-finding Documents in Cross-Media Information Spaces*". Proceedings of AVI 2016, International Working Conference on Advanced Visual Interfaces, pages 176–183, Bari, Italy. https://doi.org/10.1145/2909132.2909261.

1.5 Thesis Outline

This dissertation comprises eight chapters which together provide answers to all of our research questions. A compact representation of the dissertation's outline is shown

using a flow chart in Figure 1.5. The current chapter has introduced the general concepts, research questions and contributions. In Chapter 2 we present related work in the domain of end-user development in the context of XDI and IoT research. Based on this related work we derive a number of requirements to answer our first research question. We further present a use case scenario in Chapter 3, from which additional requirements originated. After that, we investigate model-based approaches in the XDI and IoT research domains that allowed us to finalise the requirements for RQ1. The investigation of model-based approaches and the defined requirements led to the design of a reference framework and conceptual model for cross-device and IoT user interfaces, which are discussed in Chapter 4. The reference framework and conceptual model provide a way to answer our second research question.

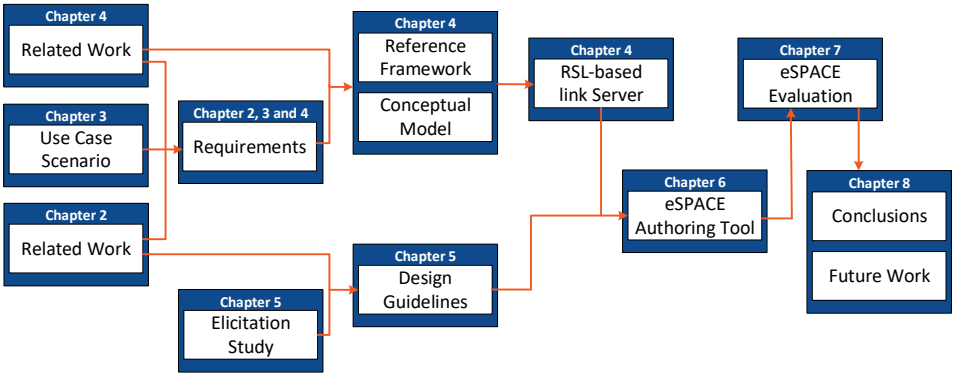


Figure 1.5: Thesis outline

In a next step, we present our end-user elicitation study in Chapter 5, to better understand user's mental models when referring to cross-device and IoT interactions, allowing us to answer RQ3. Based on the participants' vision, we infer design guidelines for a potential end-user authoring solution for the development of XD and IoT applications. While our model is validated by its integration with the RSL-based link Server serving as backend of the authoring solution, we adopt the development process described by our reference framework to construct user-defined applications in our authoring tool. In addition, the development of the end-user authoring tool takes the design guidelines from our user study into account as well. The tool's design and functionality is illustrated in Chapter 6. Our eSPACE authoring tool is further evaluated with an initial user study to complete our answer to the last research question, this evaluation is described in Chapter 7. Last but not least, Chapter 8 provides a summary of our work, some conclusions as well as potential future directions based on our research.

Chapter 2

Background

We live in a time full of opportunity for imaginative individuals. In our lifetime, we will witness the emergence of more and varied forms of human-computer interaction than ever before.

Learning and Thinking with Things
by O'Reilly Media

In this chapter we introduce the evolution of user interfaces (UIs) as well as the concept of end-user development in general and in the context of cross-device and Internet of Things interactions. We start with a brief overview of the history, which will highlight how user interfaces and technology as well as their users, evolved over time. Next we introduce and review the importance of end-user development and its role in our society, and more specifically in the context of cross-device and IoT systems. Last but not least, in order to answer our first research question we conclude with a discussion about the derived requirements for an authoring tool for end-user development of XD and IoT applications based on our analysis of related work.

2.1 History of User Interfaces and Their Users

User interfaces have been around for decades and have evolved together with technology and people. Before delving into the history of user interfaces, let us first define what a user interface is[1]:

Definition 2.1 – User Interface

A user interface is the means by which a person can interact with a software application or a hardware device.

User interfaces have not always been as we know them today and the way we interact with computer systems has evolved over the years. We start our history of user interfaces when digital electronic computers first appeared in the 1950s [141]. The history of this evolution can be broken into three main eras: batch interfaces (1945–1968), command-line user interfaces (1969–1983) and graphical user interfaces (1984–present) [180], as shown in Figure 2.1.

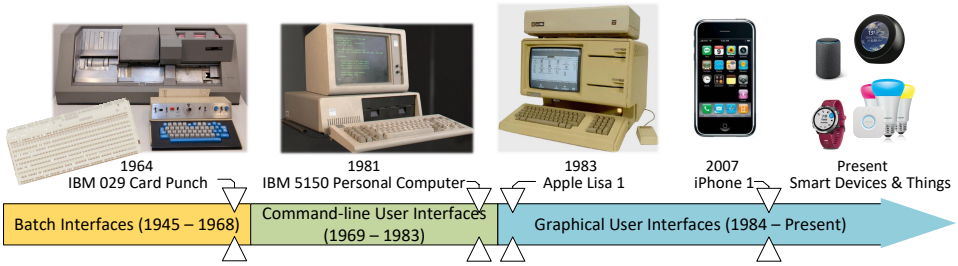


Figure 2.1: Milestones in the history of user interfaces

The first user interface appeared in the mid 20th century with the first batch machines, where users had to input a program through a punch-card system. The data was encoded on paper cards in the form of holes and then fed to the machine, which mechanically read each card in order. This process could take hours or days before finishing. With the rise of command-line interfaces (CLIs), the latency was taken from days or hours to seconds, because the user interface was a series of request-response transactions, expressed as textual commands in a specialised vocabulary [180]. As result of the lower latency, users could change their mind about later stages of a transaction depending on near real-time feedback from earlier transactions. Software could be interactive as never possible before. In the mid-1970s, the widespread adoption of video-display terminals has cut the latency even further by showing the characters on a screen instead of paper. However, these interfaces, just as for batch machines, required significant training from users in order to learn to master them. Therefore,

[1] `https://techterms.com/definition/user_interface`

users of these computer systems were still engineers and programmers. At that time, computers were expensive too and thus not affordable for households. This started to change in the late 1970s and early 80s, when computers became smaller and cheaper. In 1981, the first IBM 5150 personal computer (PC) has been introduced targeting homes and small business use. Note that such computers had already been in use since 1973, the year when the Xerox Alto has been developed. The Xerox Alto was the first computer designed to support an operating system based on a graphical user interface. It featured a bitmap display and a mouse. It was not meant for public use but inspired the development of future generations of personal computers [180].

By popularising the IBM PC to the public, "*a new phenomenon occurred: non-experts began using computers*" [141]. This group of "end users" did not want to spend time understanding complex user interfaces and software, but were expecting to use computers as a tool to assist them in their daily life, similar to a phone or car. Therefore, manufacturers started to consider "user-friendly" user interfaces, making them more easy to use. Research addressing this problem started to emerge, leading to a Special Interest Group on Computer-Human Interaction (SIGCHI) with the goal to promote the use of human factors in the human-computer interaction (HCI) process [30]. HCI has been defined as follows:

Definition 2.2 – Human-Computer Interaction

"*A discipline concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them*" [105].

Research was shifting focus from helping programmers-as-users towards helping programmers develop better interfaces for non-programming end users [98]. Even though the original IBM PC only had a single monochrome text mode with graphics capability available as an extra-cost option and no mouse, these PCs were widely "cloned", which led to the creation of a broad ecosystem of software, peripherals, and other products that could be used with the platform. This resulted in the introduction of systems incorporating technologies that are still present in today's personal computer, such as the Apple Lisa Office System 1 in 1983, which was the second commercial personal computer with a graphical user interface (GUI) inspired by the Xerox Star 8010 Information System. While both computers were not a commercial success, a year later the Macintosh became the first successful mouse-driven computer with a graphical user interface. These PCs provided a combination of GUI and command-line interfaces. In contrast to CLIs, which were purely textual, graphical user interfaces also includes images (e.g. icons, windows, scrollbars and sliders), the main thing that was still missing was colour. These first graphical user interfaces were based on the

desktop metaphor to allow users to interact more easily with the computer [180]. The desktop metaphor treats the computer screen as a "desktop" that contains documents, folders of documents and pictures that can be opened, moved around and otherwise manipulated. Over the next 10 years, GUIs have begun to incorporate features such as colour, better screen resolution and increased processing power, but their design has remained relatively consistent. The involvement of the user, however, still evolved over time, being more and more included into the development process by modelling the user's goals and requirements. This user-centred design takes end users into account from the very beginning of the development process in order to create systems that satisfy the user needs and expectations. As the individual users change, the set of requirements vary and evolve. Therefore, already in the 1990s, work has been done in making systems more flexible not only in the design phase but also during usage. This increased the user's efficiency and gave them greater freedom [161]. One way to obtain flexibility was by making it possible to change the system characteristics and tailor the system according to their personal tasks and needs. In such *adaptable systems* the interface presentation, naming and dynamics can, for example, be changed through special menus or macro mechanisms. Another way of making systems more flexible was by making them *adaptive*, such systems are able to change their own characteristics automatically depending on the user's needs.

The fact that people were having their own machines made it possible to tailor the machine's software settings without impacting the computing environment of other users [37]. In addition, as a result of better hardware, graphics cards and usability advances in GUIs, users had the possibility to explore novel programming tools designed to meet their needs. Spreadsheet systems, such as Excel, were the first end-user development (EUD) programming environments where users could create first-order functional programs using formulas [173]. The widespread use of personal software systems and the evolution of the World Wide Web into the so-called Web 2.0, determine new roles for users, evolving from information consumers into information producers [58]. The Web supports diverse "programming" activities, going from simple parameter customisation to variation and assembly of components, creating simulations, blog posts and games. This opened up diverse and powerful opportunities for end users, who are now willing to modify or create software artefacts. Over time, research about end-user development emerged which tried to find a balance between the application complexity and learning effort [81]. For this purpose, graphical design languages and design patterns were created and refined [163]. Intuitive metaphors were introduced, such as the jigsaw puzzle or pipeline metaphor for linking basic programming elements together, hereby representing the data flow of a program [163].

While people were getting familiar with their personal computers, these PCs were also influencing the way mobile technology looked like and vice versa. After making the

screens bigger, processors faster and the memory bigger the next frontier was getting computers more portable. In 1991 the Apple PowerBook was released as a first laptop meeting the needs of the market. A bit less than 10 years later, in 2000, the very first camera phone was brought to the market by SoftBank. From then on the evolution of smartphones took off. Their user interfaces reused elements from the desktop GUI, such as the WIMP components, being Windows, Icons, Menus and Pointer. However, due to constraints in space and available input devices, different unifying metaphors were used and new interaction techniques started to emerge, resulting in so-called post-WIMP GUIs. As UI designers had to rethink interface designs for smaller handheld devices, a few years later, the first Apple iPhone came out which may have proposed the best phone user interface so far. The iPhone included a multi-touch GUI enabling new interaction techniques including gestures with more than one finger, such as "pinching" to zoom in or out. Further, it featured functionality in the form of applications that could be implemented by 3rd party developers to create Web 2.0 apps which looked and behaved like built-in phone applications (having access to the iPhone's services). A year later, this led to the creation of the App Store and the Google Android market (Google Play Store). This "app revolution" also influenced the computer UIs with Windows 8 as notable example, which incorporated features from the smartphone or tablet. The proliferation of mobile devices into our daily life has resulted in HCI research involving the interaction between people and various device configurations and ecologies [35]. New research fields emerged, such as multi-display environments (MDEs), cross-device interaction (XDI) and distributed user interfaces (DUIs). The main purpose of cross-device research is to facilitate the use of multiple devices for developers as well as for end users by creating new cross-device languages, tools and interaction techniques.

Today, next to smartphones, other smart technologies emerged, such as tablets, smart thermostats, smart doorbells and some wearable devices as well including smartbands, smartwatches, smart glasses and smart jewellery. These smart technologies are connected with other devices and networks through various wireless protocols, such as Wi-Fi, Bluetooth or NFC. In recent years a new trend started, making any traditional "dumb" or non-Internet-enabled device and everyday object "smart" by extending it with Internet connectivity. These devices embedded with technology, able to interact over the Internet via dedicated applications are called Internet of Things (IoT) objects[1]. The rise of wearable devices and IoT objects raised new challenges in the design of UIs for even smaller (touch) screens and new physical UIs. Again good designs were inspired by what people already know and are familiar with. For example, the Nest Learning Thermostat[2] is based on visual cues taken from the original thermostat design and the way people use dials (think of a speedometer in a car), as

[1] `https://internetofthingsagenda.techtarget.com/definition/IoT-device`
[2] `https://store.google.com/gb/product/nest_learning_thermostat_3rd_gen`

shown in Figure 2.2. The design of user interfaces for these new devices is not the only challenging part. Due to the various communication protocols used by smart devices, communicating with all of them in a secure way has become very challenging [9]. Therefore, research has not only emerged on how to design smart devices and applications but also on how to enable fluid communication between them and how to provide privacy and security. Further, research has also been done on how to facilitate the use of IoT objects for end users by proposing applications to connect, communicate and configure their smart environment.

Figure 2.2: Nest learning thermostat

User interfaces have gone a long way and are still evolving today with the adoption of voice and gestural interaction. Some authors speak about Natural User Interfaces (NUIs) as the next era [177]. These interfaces are often invisible and make the user act and feel natural depending on the context and should lead to skilled practice. Using a voice interface for giving commands in a car is an example of a NUI.

2.2 End-User Development

As we have seen over the course of the evolution of user interfaces, computer users have shifted from expert programmers to non-expert users, also called end users. Over the years, end users gained more control over their user interfaces and applications, going from simple modification and personalisation to the creation of new functionality and applications. Many solutions have been proposed to do so in the field of end-user development (EUD). Next to this we have also seen the proliferation of new kinds of smart devices and *things* over time, making it hard to find appropriate end-user solutions to control all these smart technologies. In this section we will focus on EUD solutions in the fields of cross-device interaction (XDI) and the Internet of Things (IoT), as our main objective is to allow end users to easily manage and control all smart technologies in their smart environments by using a solution that allows them to create their own UIs and smart apps through appropriate metaphors which hide the technical details. Therefore, we aim at unifying both the XDI and

IoT research domains and find the metaphors that best match the mental models of end users when thinking of cross-device and IoT interaction.

2.2.1 Cross-Device Interaction

With the increasing number of smart devices over the last two decades, the need for designing applications going beyond the bounds of a single device emerged. Research in the fields of cross-device interaction, multi-device environments and distributed interfaces surfaced, trying to find new ways of interacting with digital content across various devices. Brudy et al. [35] unified these fragmented research domains under the umbrella of *cross-device computing*. Figure 2.3 shows a simplified overview of the author's ontology of cross-device research terminology. The top part of the figure categorises the key terms of cross-devices subdisciplines using a nested structure with cross-device, multi-device and distributed covering the broadest scope. Note that a large amount of the terms are used interchangeably by many researchers. The bottom part of the figure shows a list of the different focus areas of research in these diverse (sub)disciplines of cross-device computing.

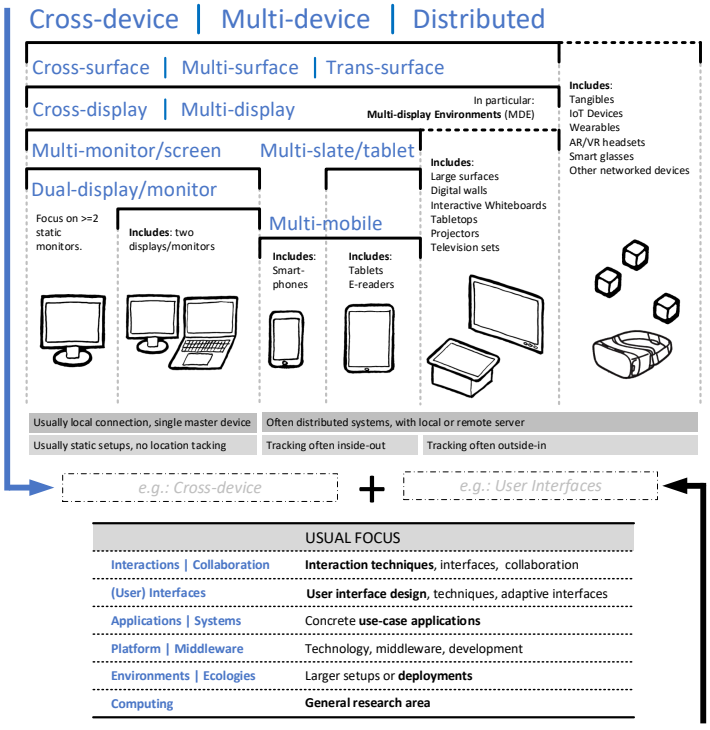


Figure 2.3: Simplified view of the cross-device research terminology [35]

Some researchers focus on proposing novel interaction techniques, such as Di Geronimo et al. [91] who introduced the Cross-Tilt-and-Tap (CTAT) framework to support the combination of touch and tilt interactions in cross-device scenarios. Another interaction technique which is often used is the *portals* technique allowing content to be moved between devices by dragging an item towards the tinted edge of the device and releasing it to move it to the device next to it [198, 78]. The *portals* technique is also sometimes used in combination with other interaction techniques, such as cross-device pinch-to-zoom or tilt-to-preview [138]. An overview of the various interaction techniques and devices used in cross-device research can be found in [193]. While focussing on the interaction techniques, some authors also focus on the collaborative aspect of a cross-device environment, such as Marquardt et al. [138] who explored how groups of people position themselves, orient and tilt their devices during collaborative settings to find better interaction techniques. Other authors focus on unifying the interaction with applications and documents at a certain location to allow multiple users to collaborate using different devices and displays [16, 23]. Cross-device interaction requires dedicated user interfaces that can sometimes be distributed across multiple devices. Researchers such as Husmann et al. [107] provide a tool for distributing a web interface across multiple devices, and going a step further, Nebeling [154] provides semi-automatic generation of cross-device web interfaces. Nebeling et al. [156] also introduced a GUI builder designed to support the development of cross-device web interfaces. A non-web-based solution to design distributed user interfaces has been provided by Melchior et al. [145], which allows any UI element to be distributed to another device by describing a distribution scenario. This approach has been showcased with a distributed Pictionary, Minesweeper, Snake and a combination of those games as the Game of the Goose. Sometimes *ready-to-use* systems are introduced as well, often building on previously defined XDI research, such as aCrossETH, which is an application developed making use of CTAT [91]. Other examples are the Scrapbook and Fotobook applications that have been created to demonstrate the potential XDKinect [157], a development toolkit that facilitates the development of cross-device applications using the Kinect to track interactions between multiple devices and users. A few researchers focus on creating new middleware that aims at assisting the development of applications for smart spaces by coordinating software entities and networked devices contained in those spaces. An example of such middleware platform is Gaia, introduced by Román et al. [186], which also provides a framework to develop user-centric, context-sensitive, resource-aware, multi-device and mobile applications. Going a step further by including IoT devices as well, Ajam and Mu [2] also present ongoing work on a middleware to enable immersive and interactive multi-device TV experiences. In order to allow certain cross-device interactions, many researchers augment tables [178] or rooms [31] with cameras, projectors and sensors to better track the different devices and users in the

interactive environment. However such environments are then often limited in terms of interaction space [193].

In [193], we present a rich body of work in the domain of cross-device and distributed user interfaces. Through a critical analysis we could make the following conclusions. Research on cross-device computing has shown that great benefits can be gained by breaking the confinements of solitary devices and users [35]. However, since most cross-device solutions propose new interaction techniques to manipulate and share data across various devices, frequently applications provide their own set of interactions, often without taking into account existing interaction techniques introduced by other systems. This results in inconsistent interaction techniques for performing similar actions across devices and applications. For example, one system allows the sharing of UI elements by tilting the device, while another solution might require users to use a swipe interaction instead, causing confusion for users using both systems. In addition, different modalities might be used as well, such as speech and/or gestures, confusing users even more. Moreover, some systems might allow only data transfer while others allow the transfer of user interfaces as well as individual user interface elements, such as buttons. This distribution is further sometimes limited to certain devices or device types depending on the operating system and protocol that is used. In addition, the interaction might be limited in terms of space too, since some systems require a fixed setup involving the need for cameras and sensors that have to be placed in the interactive environment.

Conveying the limitations and interaction possibilities in cross-device systems and applications to the users represents quite a challenge. There is a need for new guidelines, UI patterns, feedback and feedfoward mechanisms designed for cross-device computing [35]. However, to address these needs, more research has to be done to investigate the mental models of end users of smart devices, cross-device interaction and large device ecologies in order to provide empirical ground for new cross-device research. According to user experience studies, we notice that already many people struggle with multi-device fragmentation [48, 69, 194] and it remains to be determined to what extent users will adopt new multi-device systems and their new interaction techniques [35], as they often suffer from the effects of legacy bias. This phenomenon occurs when users resort to well-known interaction styles even when more effective and novel techniques are available and has been documented in multiple studies, such as cross-device sensemaking [174], note-taking [114], and curation tasks [36]. In order to promote the adoption of cross-device interaction by users and to better understand their mental models, we chose the path of end-user development. As we have seen in the previous section, the role of users has changed a lot throughout history. Recently, this shift is particularly visible in the online community where users are becoming more and more content creators rather than just content consumers.

The "Do-it-Yourself" (DIY) culture is gaining popularity in multiple domains, ending the monopoly of mass manufactoring just as the Internet terminated the monopoly of mass media [12]. Therefore, we will mainly focus on the user interface part of cross-device computing, but in contrast to the work of Husmann et al. [107], Nebeling et al. [154, 156] and Melchior et al. [145] that focusses on making it easier for designers and developers to control their cross-device environments, we want to facilitate this task for end users. In the following section, we review existing work in the domain of end-user development in cross-device computing.

2.2.2 Authoring of Cross-Device Applications

We will focus on tools for end users that provide cross-device interaction at design time rather than tools just providing the possibility of only performing cross-device interaction at runtime. Similar to Paternò et al. [165], we will identify the *metaphors* used by the proposed XDI EUD solutions.

One early solution that aimed at empowering end users has been introduced by Han et al. [102]. The authors presented WebSplitter, an XML framework that allows a web page to be distributed to different users and displays. WebSplitter can further be used to split views of a web slideshow presentation over multiple devices. The presenter can, for example, navigate their slides using a PDA while students can follow the presentation on their own devices, such as a PDA or a laptop. An XML metadata policy file is used to manage access privileges allowing different users to have a different view of a web page presentation. Through this file, an end user can decide which XML tags or web components should be distributed to which user groups or devices by defining mapping *rules* (for each XML tag). As consequence, the author of the policy file can limit some user groups by giving them access to only a partial view of a web page. Note that for each web page, a separate policy file has to be defined. Once the system is running, the different views generated for the web pages based on the policy file cannot be changed by the user, which is quite a limitation. Most of the recent systems offering UI distribution give users more freedom concerning the distribution of data and applications across devices.

New web-based systems emerged, such as Ghiani et al.'s platform [94], which allows end users to manage migration of web application in a multi-device environment together with the associated privacy policies. Web pages or parts of them can be migrated across devices while preserving their state using the push and pull modality. By using the push modality, web pages can be forwarded from the device the user is currently using to another device, while a pull allows users to select a web page from another device and send it to the currently used device. The authoring is done through a visual environment via a point-and-click user interface. A few years later,

Ghiani et al. [95] proposed the MashupEditor for EUD of Web mashups that are built from existing Web components. The editor uses the *copy-paste* metaphor to create a connection between different web components, such as an input field, a Google Maps and a weather component, once a connection made the name of the city entered in the input field will be shown on the map component and the city's temperature will be shown on the weather component. The MashupEditor environment is meant to be compatible with a wide range of devices by using web technologies, but does not offer multi-device mashups yet, which means that so far a mashup cannot be split across several devices.

Another tool based on web technologies that allows end users to create mashups has been presented by Cappiello et al. [44], who introduced a UI-centric approach for creating mashups on multiple devices. The proposed environment follows the WYSIWYG (What You See Is What You Get) paradigm, by showing real time changes to the mashup application. In contrast to the MashupEditor, connecting UI components is done using the *drag-and-drop* metaphor. The authors used a model-driven approach, where each action of the user on their compositions are transformed into descriptive models describing the logic of the mashup [45]. These models are automatically parsed to instantiate the respective applications on the fly so that users can interactively try out their mashup composition. The mashup can run on multiple devices as standalone app.

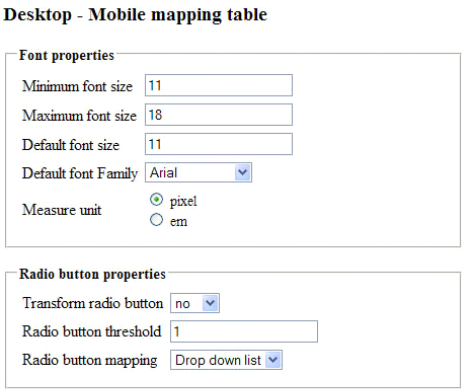
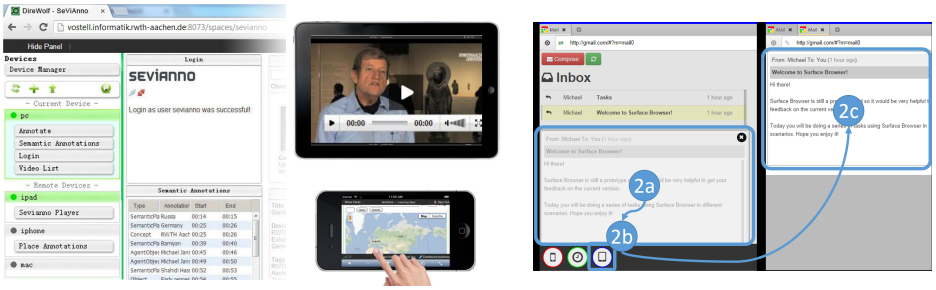


Figure 2.4: Part of a desktop to mobile mapping table [167]

Model-driven and model-based approaches are used to facilitate the UI development process by generating UIs from declarative models, but such approaches often focus at making it easier for designers and developers rather than end users [64, 143]. Although there are a few exceptions, such as the previously introduced system and a tool presented by Paternò and Zichitella [168], which is based on the MARIA language [167] and allows end users to customise desktop-to-mobile adaptation via

a mapping table, as shown in Figure 2.4. However, their solution only supports basic customisations and does neither offer support to modify the distribution of UI components across devices nor cross-device interactions.

Other web-based tools are SmartComposition [123] and DireWolf [122], which in contrast to previously introduced tools offer distribution of components across different devices. The components are presented as pre-built widgets which can be arranged in a grid-based layout using *drag-and-drop*. Both systems support inter-widget communication allowing cross-device cooperation and use peer-to-peer communication between devices. Figure 2.5a provides an example of a web mashup using DireWolf.



(a) Widget distribution of DireWolf [122]

(b) UI distribution of an email using XDBrowser 2.0 [154]

Figure 2.5: DUI solutions

Some more advanced approaches provide automation, such as XDBrowser 2.0 [154] which provides semi-automatic distribution of UI components based on the results of a study that analysed how participants manually customise web pages for cross-device use. The tool enables end users to re-author web pages across devices by allowing them to select parts of a web page and copy or move the selection between devices, as shown in Figure 2.5b.

Going into the domain of multi-device UI design, Meskens et al. [147] presented Jelly, a design environment where UIs can be designed for multiple platforms in parallel by *copy and pasting* parts of a user interface from one device to another. Every time an element is copied to another device, the designer can select from a list of available widgets how the element should look like on the target device. Jelly also proposes *linked editing* which keeps the content of UIs consist across the different devices. However, like most multi-device design tools [62, 128, 148], Jelly is meant to be used by designers rather than end users.

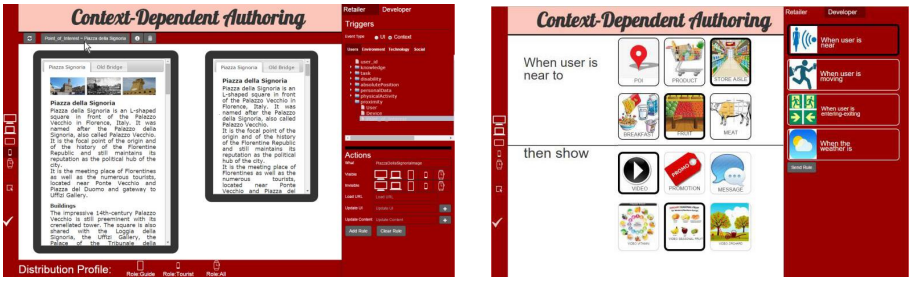
Ghiani et al. [92] introduced an authoring environment for the development of context-dependent cross-device user interfaces using *trigger-action rules* to define

context-dependent behaviour. The rules metaphor has been defined and used in the literature as follows:

Definition 2.3 – Rules metaphor

By using rules, the system's behavior is represented using if-then statements. Each statement expresses the way the system should behave given specific situations [165].

In Ghiani et al.'s authoring environment [92], a part is dedicated to rule composition where the trigger can be a UI or contextual event and the action can have an effect on the visibility of a UI component on a certain device. An example could be, "*if user is near the widescreen, then assign 'shopping list content' to the widescreen device and unassign it from the mobile device*". The rules can be previewed at any time to see the effect on the UI(s) in the main area. Unfortunately, the tool was only usable by professional developers, therefore the authors provide a domain-dependent extension for people working in the smart retail area with a simplified view of the trigger-action rules, as shown in Figure 2.6.



(a) Original environment for authoring context-dependent UIs

(b) The domain-dependent environment for the smart retail

Figure 2.6: Ghiani et al.'s authoring environments [92]

Some authors focussed on customising the cross-device interaction itself, such as Chen et al. [50] who presented Improv, an input framework that allows end users to define custom cross-device inputs by demonstration. The defined gestural inputs are used to manipulate existing applications on the fly. First the target behaviour is demonstrated on the computer, then the new input method is demonstrated on the smartphone or another input device. Once that both are demonstrated and recorded, Improv will perform the target behaviour when the input gesture is recognised.

Instead of using gestures, Messer et al. [149] introduced InterPlay, which allows end users to use pseudo-English sentences to control heterogeneous home networked devices. The pseudo sentence represents a task and consists of a verb, a subject and

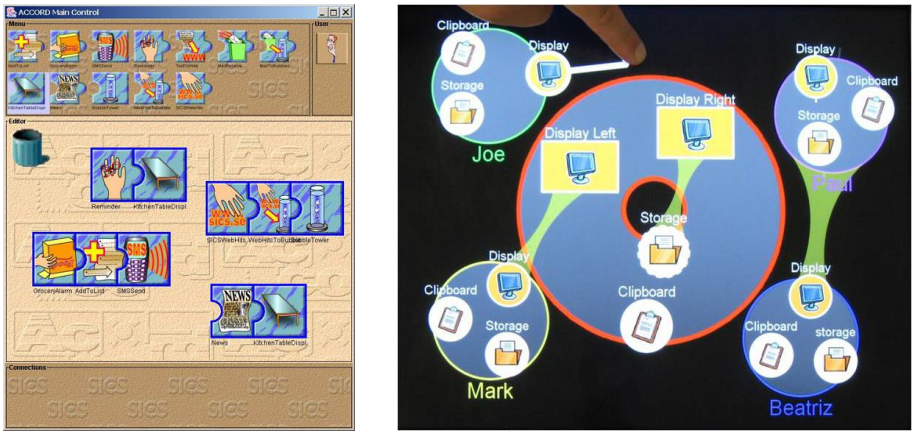
one or multiple target device(s). An example sentence could be: "*Play, The Matrix, Living Room DTV*". The advantage of using pseudo sentences is their similarity with natural language which helps users to understand the intent without the need to refer to a manual.

Already taking a step towards the authoring of Internet of Things applications, Humble et. al. [106, 183] presented an editor that allows end users to configure their ubiquitous computer environments. The editor uses the *jigsaw puzzle* metaphor to assemble heterogeneous components, such as connecting and configuring lightweight sensors, computer devices and services. It would thus allow to create some basic cross-device interaction as illustrated in Figure 2.7a. The *jigsaw puzzle* metaphor has been defined as follows:

Definition 2.4 – Jigsaw puzzle metaphor

By using the jigsaw puzzle metaphor, applications are represented as a combination of multiple puzzle pieces. Each piece of a puzzle usually corresponds to a service, while the shape of the puzzle provides cognitive clues to discern the possible actions [67].

While the jigsaw puzzle metaphor is one of the most used metaphors [67], its main disadvantage is that its expressiveness is limited by the number of sides of a puzzle piece.



(a) Humble et. al. [106, 183]'s editor using the jigsaw puzzle metaphor (b) Platform Composition [172, 171] that uses the join-the-dots metaphor

Figure 2.7: XDI tools

A variation of the jigsaw puzzle metaphor is the *join-the-dots* metaphor used by Pering et al. [172, 171] in their Platform Composition, where the GUI shows the devices as large circles enclosing smaller circles representing the device's core services, as shown in Figure 2.7b. To create a connection between a service and a device, users simply have to draw a line from the service to the target device. Disconnecting is done by dragging a line across the service (metaphorically "cutting" the connection). The use of the *join-the-dots* metaphor has been described in the literature as follows:

Definition 2.5 – Join-the-dots metaphor

By using the join-the-dots metaphor, the environment is represented using circles or clusters that can be connected with each other. Each cluster represents a device containing services that are represented by smaller circles or dots. Connections between devices and services are created by drawing a line between the corresponding circles [67].

The main advantage of the join-the-dots metaphor is its simplicity, since users receive a graphical overview of all available devices and their services, hereby outlining the entire state of the system.

2.2.3 Internet of Things

As shortly described in our history section, the rise of the Internet of Things (IoT) resulted in a proliferation of new smart technologies, such as smart fridges, power plugs, light bulbs and thermostats, that were introduced on the market, most of them accompanied by dedicated applications to control them. However, these applications are limited to controlling a range of IoT devices (often from the same manufacturer), with the consequence that users have to install multiple applications to control all these devices. This fragmentation introduces an extra cognitive burden for the users who are trying to find the right application to control the right *thing*.

A lot of research has been done around IoT and various survey papers summarising this research emerged [8, 96, 152]. While IoT brought up new interaction possibilities together with new kinds of augmented everyday objects, it also raised new challenges involving the connectivity, security, privacy and usability of all these new devices.

Due to the wide variety of heterogeneous devices, sensors and actuators with different capabilities, functionality and network protocols, it is already difficult to create applications controlling all of them for programmers. Therefore, frameworks and IoT architectures have been created for developers to deal with challenges related to connectivity and privacy [11, 179]. An interesting vision concerning the future of

IoT connectivity has been presented by Guinard and Trifa [99], who mention that every *smart thing* should provide access to its services using well-known web standards such as REST. The authors present two approaches, one where each device embeds a Web Server offering a RESTful API to access the device's functionality and one where each device communicates with smart gateways that provide a way to access the functionality of more resource-limited devices through a RESTful API as well. By using REST and the Web to communicate with devices, rapid prototyping of IoT applications become possible as these technologies are powerful and offer a lot of flexibility. With their approaches, Guinard and Trifa aim to lay the basis for the future Web of Things (WoT)[1]. The WoT aims at countering the fragmentation introduced by the IoT by connecting devices over the Web in order to make it easier to create applications without requiring knowledge about various IoT technologies and standards. The official definition of the Web of Things has later been given by Guinard and Trifa in their book [100], which is dedicated to this subject:

Definition 2.6 – Web of Things

"*The Web of Things is a refinement of the Internet of Things by integrating smart things not only into the Internet (network), but into the Web Architecture (application) [100]*".

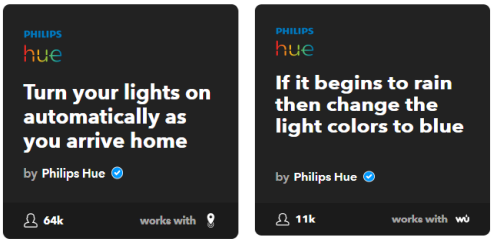
While the WoT might improve the interoperability problem, there is still the need to find a way to show the data produced by these smart *things* as well as to investigate how to manage and interact with them. Many dashboard-like systems[2] were introduced to organise and present the large amount of data produced by IoT devices. Each brand presenting an IoT product also comes with their own application software for end users, often only allowing them to manage IoT devices of this specific brand, forcing the end users to figure out how to use each functionality of the different applications proposed by IoT manufacturers. Furthermore, end users are again stuck with the predefined functionality provided by each application, often leaving no room for any customised functionality. Therefore, research has been carried out in EUD for IoT environments, allowing end users to have more control over their smart spaces. In order to give users more control, meaningful abstractions and metaphors to abstract the low-level details of IoT systems have to be found, to allow end users to only focus on the relevant aspects when dealing with such IoT devices [165].

[1] `https://www.w3.org/WoT/`
[2] `https://ubidots.com/blog/iot-dashboards`

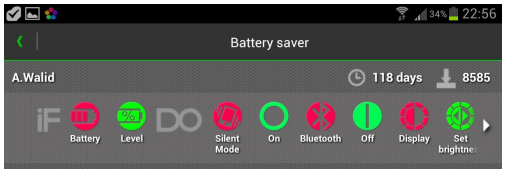
2.2.4 Authoring of Internet of Things Applications

In this subsection we analyse related work in the domain of IoT authoring tools for end users together with the metaphors used by these tools.

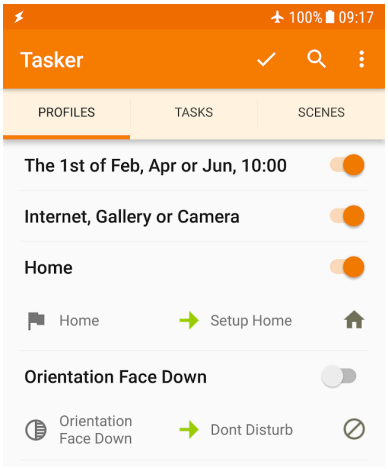
Various commercial solutions have been introduced to help users configure their smart environments based on a set of *Event-Condition-Action (ECA) rules* [75]. A well-known example is IFTTT[1] (if this, then that), a web-based service that allows users to create conditional statements that are automatically executed based on the internal state of apps or other web services. IFTTT already supports different IoT devices in its conditional statements, such as the Philips Hue[2] and Alexa speakers[3]. This shows that the IFTTT approach can address emerging IoT devices, as discussed by Ur et al. [213]. IFTTT uses the *rules* metaphor and the trigger-action ("if, then") programming style, which is also used by similar alternatives, such as Atooma[4], Tasker[5] and others [38]. Figure 2.8 shows an overview of some of these commercial solutions.



(a) Example rules with IFTTT



(b) Example rule with Atooma



(c) Tasker interface for rules

Figure 2.8: Commercial IoT solutions

By using the concept of trigger-action programming as well, Fogli et al. [83] presented ImAtHome, an application to help users configure and manage their home automation accessories by using ECA rules. Contrarily to IFTTT that only supports rules with one event and one action, ImAtHome supports more complex rules that can be created via a different visual interaction language allowing manual and automatic activation

[1] https://ifttt.com
[2] https://www2.meethue.com/en-us
[3] https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011
[4] https://resonance-ai.com/about.html
[5] https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm&hl=en

of a set of actions (scenes). It further promotes reusability of scenes to make the rule creation more efficient. However, its expressiveness is quite limited and the app is restricted to iOS users since ImAtHome is based on the Apple HomeKit framework.

Similarly, Barricelli and Valtolina [17] also introduced a *rule-based* system called the SmartFit Rule Editor that targets the context of the eWellness domain. It helps coaches and trainers to monitor and analyse fitness and wellness data as well as to detect events and specify rules based on their working setting. The architecture of the SmartFit framework is based on the Software Shaping Workshop (SSW) design methodology [57]. The goal of this methodology is to help designing software systems that are customised to the needs of end users, who are recognised as domain experts, so that they can tailor or create their own tool. In SmartFit, domain experts are the coaches and trainers, who will receive appropriate software artefacts needed to perform their activities in order to ease their work with computer technology.

Another more advanced example is TARE [93], where *rules* are also expressed via a *trigger-action* syntax. The rule editor provides continuous feedback of the rule that is being edited in an easy-to-understand language, as illustrated in Figure 2.9. TARE allows end users to create and customise context-dependent behaviour of their smart environment. In contrast to most IoT EUD tools it also supports distribution of a UI from one device to another using rules. However, it is unclear how TARE supports the distribution of parts of the UIs and whether UI design is supported. Further, TARE provides an interactive simulation for previewing the execution of created rules in a simulated context. The authors also included a home controller application showing the sensor values, a map of the smart home and a way for direct control of home appliances, which serve as basis to populate the authoring environment.

Figure 2.9: TARE [93] rule composition interface

Presenting a more visual prototyping system, Dey et al. [74] introduced iCAP, a *visual rule-based* system to prototype context-aware applications for smart environments, which is shown in Figure 2.10. iCAP is based on a user study that the authors performed in order to better understand the mental models of end users concerning context-aware applications. 20 participants with no programming skills were interviewed during 90 minutes, more specifically they were given the description of a smart home and, in a first instance, were asked to describe how, when and where they want music to be played in the smart home. During the last part of the interview, participants could create their own context-aware scenarios that they would find useful in their smart home. The iCAP interface is based on how participants described the different scenarios, which was mainly using rules describing a subject's situation and a command to the house to act on this state. The visual interface has been chosen because visual programming languages are effective in taking advantage of the spatial reasoning skills of users [200]. While this visual interface is more familiar and simple, it is less expressive than a text-based one, but given that the average user-specified rules were not very complex, the authors deemed this an appropriate choice.

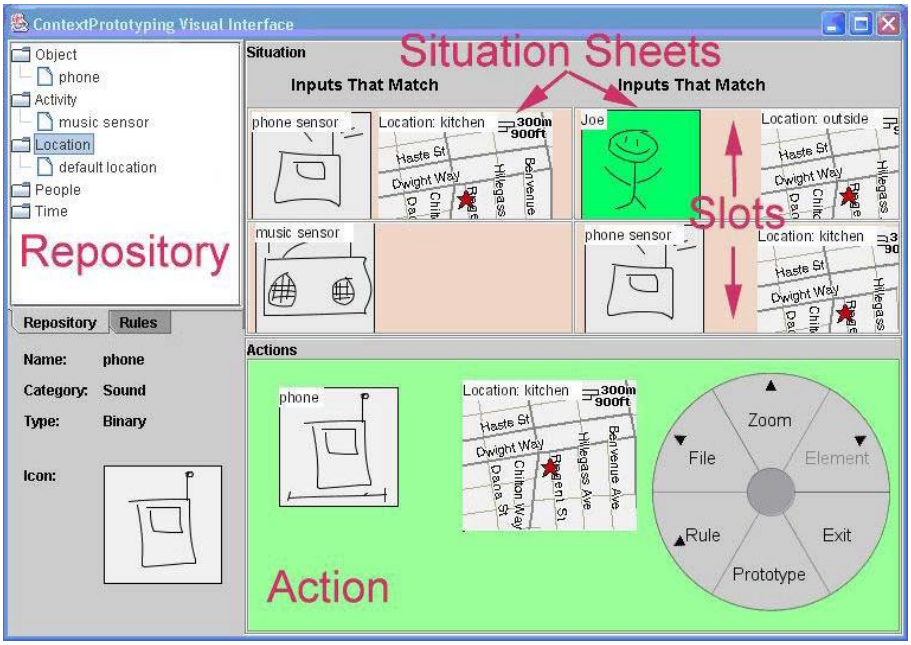


Figure 2.10: iCAP [74] visual interface

Focussing on how to facilitate the description of the "interaction" in particular, Kubitza and Schmidt [124] proposed meSchup, an IoT prototyping platform that simplifies the integration of devices and the creation of smart environments. It provides a web-based scripting IDE that combines traditional text-based programming with

interactive physical demonstration. However, the tool mainly targets people with some programming experience since the users still need to use JavaScript to create rules. Therefore, the authors implemented a simple mobile EUD interface called TouchCompozr [124] as an app for the meSchup platform. TouchCompozr allows end users to form IF-THEN-ELSE *rules* for the compositions of smart things using form elements. The form input fields can be filled in by physically demonstrating a real-world interaction, as shown in Figure 2.11a. The figure depicts a user pressing a button in order to fill in the IF-part of the rule, which means that pressing this button is used as trigger of the current rule. After that, to define the THEN-part the user can demonstrate another action, such as turning on the light, which will then be used as action of the rule.

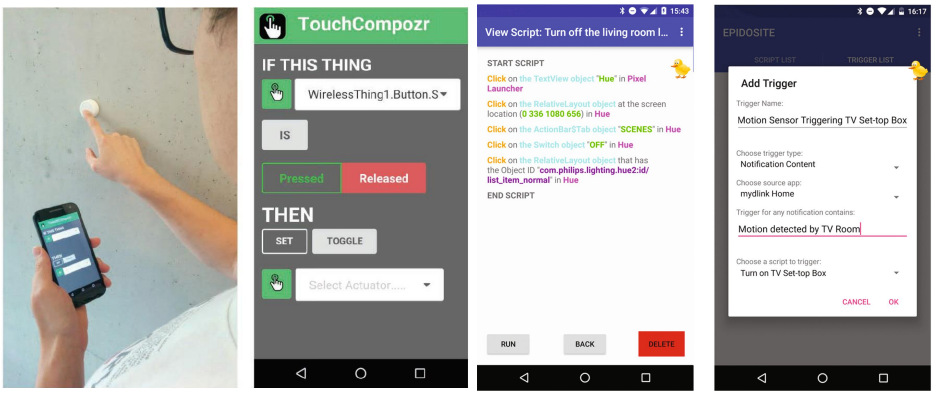
Another tool using programming by demonstration is introduced by Dey et al. [73] in a CAPpella, which supports programming of context-aware behaviour by demonstration. Users demonstrate a behaviour including a situation and its associated action(s) a number of times and based on this demonstrations, a CAPpella will then learn to recognise this situation and perform the demonstrated actions when the demonstrated situation is detected. In order to show the system when the behaviour is demonstrated, users have to indicate the start and end time in the a CAPpella user interface, hereby using the *timeline* metaphor. For the sake of clarity, we provide a definition of the use of the timeline metaphor:

Definition 2.7 – Timeline metaphor

By using the timeline metaphor, temporal references can be provided, typically represented by a line on or above which different elements are (graphically) positioned. The elements correspond to events which are organized in a chronological order [165].

Tackling specifically the problem of users having a plethora of mobile IoT applications scattered on their phones, Li et al. [125] introduced EPIDOSITE, which enables end users to automate various IoT devices by demonstrating the required behaviour through the manipulation of the smartphone's IoT apps. The created automations are stored as scripts in the EPIDOSITE app, as shown in Figure 2.11b. This way, the users have "one app to rule them all". However, the users still have to keep the original IoT apps since EPIDOSITE manipulates the apps as demonstrated by the user when the triggers are launched.

Another smartphone solution is Keep Doing It that has been proposed by Maues et al. [68], which provides users with recommended *rules* to automate smartphone tasks based on how the user uses their phone. This approach is more limited since users cannot define their own rules.



(a) TouchCompozr [124] interface (b) EPIDOSITE [125] interface

Figure 2.11: IoT solutions for smartphones

Still in the smartphone solutions, Heo et al. [103] introduced the IoT Mashup Application Platform (IoT-MAP) that dynamically discovers devices, downloads the necessary software modules and provides a UI, called Versatile, to the end users for the mashup and composition of smart things. The composition UI is based on Node-Red[1] and thus uses the *pipeline* metaphor. The pipeline metaphor is defined and used as follows:

Definition 2.8 – Pipeline metaphor

By using the pipeline or graph metaphor, applications are represented as directed graphs. The nodes of such a graph typically correspond to individual devices, services or activities, while links (i.e. pipelines) allows to connect them. Binary logic operators and filter elements can be used to organize pipelines into more complex structures [67].

Exploring different types of rule creation interfaces, Desolda et al. [72] performed an elicitation study with Computer Science students to identify appropriate visual composition mechanisms for trigger-action rules. More specifically the leading question of the study was "*how to specify events and actions in a rule by answering to the Which, What, When, Where, and Why questions?*". The participants designs gave rise to three main design prototypes, the E-Free, E-Wizard, and E-Wired interactive prototypes. The "E" stands for EFESTO, a mashup platform by the authors [71], which was used as basis for implementing the three prototypes. The E-Free prototype depicts rules as events and actions and is shown in Figure 2.12. The interface is split into two parts, with the triggering events on the left and the actions that will

[1] `https://nodered.org`

be executed on activation of the events on the right. New events and actions can be added by clicking the '+' symbol on the left or right of the interface, which will open a popup wizard where users can define the event or action they want by first selecting *which* service, then *what* the service needs to do, *where* the event/action has to be triggered (optionally) and *when* it should be triggered (optionally). Once the steps of the wizard completed, the event or action will be shown on the appropriate side of the interface.

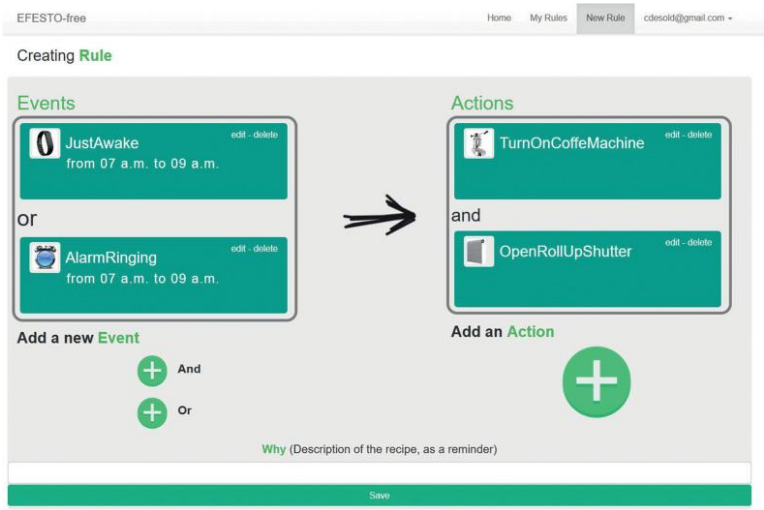


Figure 2.12: E-Free rule creation interface [72]

The E-Wizard interface is the same as the E-Free UI, but it offers a more controlled environment for the user. Instead of having the possibility to create multiple events before starting to define an action, in E-Wizard users have to first define a basic rule with one event and one action. Then they can add additional events or actions by using the '+' symbol. The authors present the E-Wizard prototype as incremental compared to the E-Free prototype which provides a "freer" environment. The last prototype, E-Wired, uses the *pipeline* metaphor, where nodes represent services that are involved in a rule. The directed links between services represent the "cause-effect" relationships. The UI of the E-Wired prototype depicts a sidebar on the left containing the available services and a workspace area on the right where services can be linked to create a rule. When a connection is created between two services, by drawing an arrow from one service to another, a popup window will appear asking users to fill in *what* the event is, with optionally *when* and *where* it should be triggered, and then the same information is asked for the action. The authors carried out a comparative study with 40 participants (27 computer science, 10 business and 3 physiotherapy students) comparing the three prototypes with IFTTT. The results of the study

pointed out that E-Free was favoured by the participants. Further E-Free also had a higher satisfaction and performed better compared to other prototypes and IFTTT. Lastly, the authors performed a study to evaluate the E-Free prototype with 15 home automation experts. These studies resulted in design implications for future EUD rule creation UIs.

Including a 'tangible' aspect in the rule creation process, Russis and Corno [190] synthesised existing work on end-user programming of smart homes to come up with a set of guidelines for designers of EUD interfaces and tools. Based on these guidelines, the authors propose the HomeRules prototype, a tablet-based tangible interface for EUD of smart homes using ECA rules. An evaluation of two paper prototypes steered the design of the interactive HomeRules prototype which UI comprises a 'rule panel' and a 'devices' sidebar on the right. The rule panel shows the rules as 'event + condition -> action'. A rule can be composed of multiple events, conditions or actions. If a user, for instance, wants to add an extra action, the device which will perform the action should be selected in the devices sidebar or the user can choose the 'interactive learning' option, which will wait for the user to perform any operation in the home and report it in a popup screen where the user can select the operation as 'action'. The action will then appear under the existing action in the rule panel. The same method can be used for adding events and conditions as well.

(a) The hardware toolkit (b) Choosing a trigger

Figure 2.13: T4Tags 2.0 [19]

In contrast to previous systems, Bellucci et al. [19] included self-made tangible tokens to the IoT environment and provide an end-user toolkit, called T4Tags 2.0, for programming those tokens using trigger-condition-action rules. The tokens can easily be placed on physical objects to program intelligent behaviour. For example, by placing a token on a door, one can detect when this door is being opened by someone. The authors provide a ready-to-use DIY toolkit comprising of a wireless base station, 4 BLE physical tokens that can be labelled and are embedded with sen-

sors (temperature, luminosity and accelerometer), 2 Philips Hue[1] light bulbs, 2 smart Fibaro Wall Plug[2] smart power outlets, a regular chalk pen and 18 adhesive squares for attaching tokens to objects. The different hardware is illustrated in Figure 2.13a. The behaviour of these smart objects can be programmed using the T4Tags 2.0 UI, a mobile-oriented web application, whose menu resembles the one of Atooma, providing the IF or THEN options in a circular menu with icons, as shown in Figure 2.13b. The created rules can be made public on a social platform and are viewable in a tile-based interface, such as IFTTT rules shown in Figure 2.8a, but instead of using only text, the tiles are more graphical, including icons and pictures of involved services.

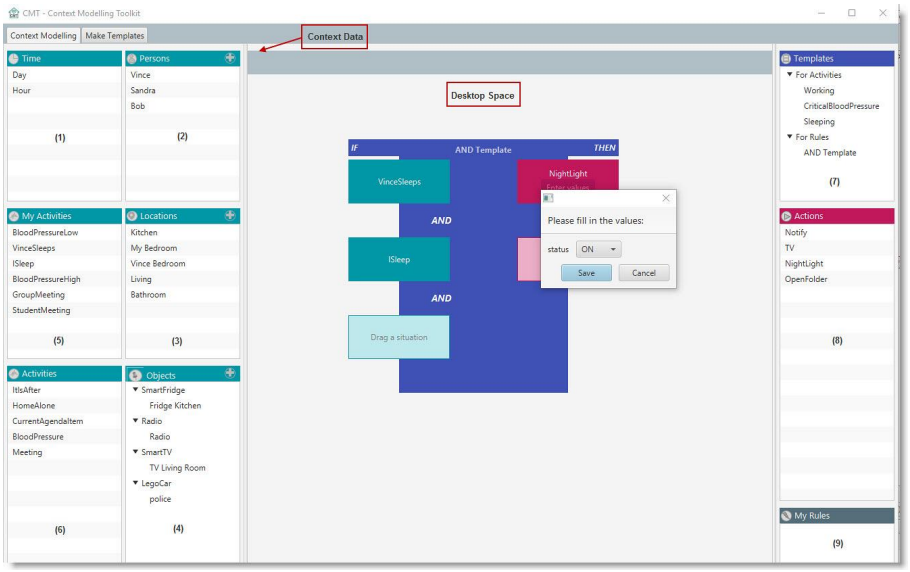


Figure 2.14: CMT rule creation using the "AND template" [212]

Focussing on the context awareness aspect of recent smart environments, Trullemans et al. [212] presented the Context Modelling Toolkit (CMT), allowing end users to control context-aware solutions. CMT presents a visual authoring environment based on "IF situation or event THEN situation or action" rules, which is depicted in Figure 2.14. The figure shows the main rule creation interface with an "AND template" as skeleton for defining a rule, where the IF side can contain multiple situations or events, while the THEN sides can contain multiple situations or actions. In contrast to existing solutions, CMT focuses on the reusability of situations of a rule. A situation can be defined by using templates, similar to the "AND template" shown in Figure 2.14. These templates can be created using a more advanced authoring view for more experienced end users. CMT distinguishes between three types

[1] `http://www2.meethue.com/en-US`
[2] `http://www.fibaro.com/en/the-fibaro-system/wall-plug`

of users being the end users, expert users and programmers. Depending on the type of users different authoring views are made available.

Following a component-based web mashup approach, Koren and Klamma [120] presented a conceptual extension of the DireWolf framework [122] that integrates heterogeneous Web of Things (WoT) devices by including UI components served directly by WoT devices. The UI components can be aligned in a tree structure and separated into different application spaces which can be located on different types of devices. These application spaces can be embedded in arbitrary HTML pages. The authors however still have to evaluate the scalability and usability of their approach.

A more complete tool, called AppsGate, has been presented by Coutaz and Crowley [60]. It provides an EUD environment to help end users control and augment their home. Users can program their smart home devices in the syntax editor by creating *rules*, actions and conditional statements using a *pseudo-natural language*, as illustrated in Figure 5.17. AppsGate supports debugging as well, by allowing users to run programs using a virtual date and time. Monitoring of the smart home can be done using the *timelines* and through a *dependency graph* which shows the relations between entities.

Figure 2.15: AppsGate rule creation interface [61]

Another metaphor used for EUD that we already discussed in the previous section is the *jigsaw puzzle* metaphor. Danado and Paternò [66] presented Puzzle that supports the development of IoT applications on smartphones. Each puzzle piece represents some functionality that can be composed by connecting the pieces via a drag-and-drop interface. The shape and colour of the puzzle piece indicates the number of inputs and outputs as well as the data that can be exchanged. A commercial solution using the jigsaw puzzle metaphor is the Zipato Rule Creator[1], an online tool to create rules for the Zipato Home Automation System.

[1] `https://www.zipato.com`

2.2.5 Discussion and Limitations

We have presented a rich body of work on end-user development in the domains of cross-device interaction and the Internet of Things. While some tools focus on the interaction between smart devices or *things* and their connection with each other, others focus on the elements that can be distributed among these smart technologies, ranging from simple commands to parts or entire user interfaces transferred between devices and *things*.

We have seen many different approaches and metaphors allowing end users to create interactions between their smart technologies. Since the main goal of an end-user authoring tool is to hide the complexity of the underlying technologies, intuitive metaphors are needed to highlight the main conceptual aspects and abstract the technical details. For example, in the case of defining a data transfer between two devices, it is important to hide the communication protocol by using appropriate metaphors that show when and how the data is transferred from one device to the other without requiring the technical knowledge of a developer [166].

Choosing the right abstractions and metaphors is not an easy task, given that they all come with their advantages and disadvantages. Various metaphors have been used in the field of XDI and IoT EUD, ranging from simple drag-and-drop to rules, timelines, pipelines, join-the-dots or the jigsaw puzzle metaphor. In Table 2.1 we provide an overview of the most dominant metaphor(s) used by related work described in this chapter. EUD solutions in the field of XDI are noted down in black while EUD solutions in the IoT field are coloured in blue.

Metaphor	Systems
Timeline	a CAPpella [73], AppsGate [60]
Rules	AppsGate [60], Atooma, CMT [212], E-Free [72], E-Wired [72], E-Wizard [72], Ghiani et al. [92], HomeRules [190], iCAP [74], IFTTT, ImAtHome [83], Keep Doing It [68], SmartFit [17] T4Tags 2.0 [19], TARE [93], Tasker, TouchCompozr [124]
Pipeline	AppsGate [60], E-Wired [72], Versatile [103]
Jigsaw	ACCORD [106, 183], Puzzle [66], Zipato
Join-the-dots	Platform Composition [171]
Drag-and-drop	DashMash [44], DireWolf [121], DireWolf 3.0.0 [120], SmartComposition [123]
Copy/paste	Jelly [147], MashupEditor [95], XDBrowser 2.0 [154]
None	EPIDOSITE [125], Ghiani et al. [94], Improv [50], InterPlay [149], Paternò and Zichitella [168], WebSplitter [102]

Table 2.1: Summary of used metaphors with existing solutions in alphabetical order

Most of the presented IoT EUD solutions use the rule metaphor [17, 19, 60, 72, 83, 93, 124, 68, 190, 212], with the exception of a CAPpella [73] using the timeline metaphor, Puzzle [66] using the jigsaw puzzle metaphor, DireWolf 3.0.0 [120] using drag-and-drop, Versatile [103] using pipelines and EPIDOSITE [125] using no metaphors at all. AppsGate [60] applies multiple metaphors, including the rule, timeline and pipeline metaphors. On the other hand, in XDI EUD solutions,

metaphors are less present and more scattered between drag-and-drop [123, 121, 44], copy/paste [95, 147, 154], rule [92], join-the-dots [171] and jigsaw puzzle [106, 183].

As mentioned earlier, metaphors abstract from low-level details, but each metaphor comes with its own limitations. Scalability can, for example, be an issue with the pipeline metaphor where the diagram becomes hard to interpret once the amount of elements and connections increases. While the jigsaw puzzle metaphor supports more complex configurations than the pipeline metaphor, by allowing structures (e.g. clusters of pieces) to act as one entity [67], its scalability might be an issue as well, since the screen easily gets cluttered with an increasing amount of puzzle pieces. The jigsaw puzzle metaphor is also limited by the number of sides of a puzzle piece, restricting the set of possible combinations, i.e. restricting its expressiveness [67]. Some metaphors allow easy composition of events and actions, such as rules where events/actions can be composed using Boolean operators, such as AND, OR and NOT. Rules can be visualised in different ways, as seen in related work. For example, Desolda et al. [72] presented different prototypes using the rule metaphor, where one prototype visualises rules using the pipeline metaphor. The drawback of using rules is that they can be easily broken and it is difficult to detect conflicting actions [74]. Some systems, such as T4Tags 2.0 [19], therefore restrict the choice of triggers combinations, guide users while creating rules to avoid inconsistent or invalid states and prompt users informing them when a specific behaviour cannot be defined, explaining the cause. Others [55] focus on providing ways to debug trigger-action rules.

With the different possibilities of metaphors and their limitations in mind, researchers try to find the right metaphor to use in their tools for letting end users define cross-device or IoT interactions, sometimes combining multiple metaphors as seen in E-Wired [72] and in AppsGate [60]. The choice of metaphor is often based on related work, observations of commercial solutions and own experience, as for instance explicitly mentioned in HomeRules [190], which has been designed based on guidelines that were derived out of such an analysis. However, it is important to also consider the way people think about cross-device and IoT interaction, since they might have a different view compared to a developer in terms of algorithmic computation and require representations and concepts that are more suitable for them [166]. In other words, we need to investigate the end user's mental models when thinking about cross-device and IoT interactions for selecting the most intuitive metaphor(s) which will affect the way people will include EUD activities in their everyday lives.

Taking the mental models of end users into account is often omitted by existing solutions. While Desolda et al. [72] performed an elicitation study to find appropriate metaphors, the participants of this study were computer science students instead of regular end users. To the best of our knowledge, from the described related work in this chapter, the only researchers that performed a user study with end users to

better understand their mental models before building a solution are Dey et al. [74], who interviewed end users about context aware applications. Lucci and Paternò [132] also performed a study to improve current designs and inform future designs of mobile context-dependent applications, on which some of TARE [93]'s requirements are based on. However, the study was less open than the one of Dey et al. [74] and focused on *smartphone* EUD applications. Lucci and Paternò [131] analysed and compared three Android apps, including Tasker, Locale[1] and Atooma. A usability study of these three applications pointed out a lack in consistency of the terminology used in the applications. Accordingly, in order to improve the user exprerience of EUD apps, the choice of elements, terms and icons should be consistent and unambiguously understandable by end users. This led to the card sorting study in [132], where users classify the concepts supported by the three apps, to identify which ones are more closely related to the users' mental models. Note that, in contrast to preliminary studies, almost all of the presented solutions did evaluate their solution through a study *after* having already chosen a metaphor.

Another important characteristic of related work, next to the metaphors they use, is of course the underlying functionality that these EUD tools provide. As mentioned by Lucci and Paternò [131], commercial solutions use different terms and icons, which is also true for EUD solutions in general. The same functionality can therefore be found under different names in various tools which might be counter intuitive for end users. Further the functionality provided by cross-device EUD tools is rarely combined with functionality of IoT EUD solutions. There is a *lack of unification* between the XDI and IoT domains. Although, some tools, such as the authoring tool of Ghiani et al. [92], are going into the right direction, they revealed to be too difficult for end users. The authors then later presented TARE [93], supporting IoT as well as cross-device functionality, such as user interface distribution using trigger-action rules. It is however unclear, to which extent cross-device UIs can be created. The authors do not mention the granularity of the distribution, which parts of UIs can be distributed and how it is distributed remains unclear. One key advantage of this authoring tool is the real-time preview of a rule, while a disadvantage is the difficulty of finding the right attributes of a rule in the context model hierarchy.

2.2.6 Resulting Requirements

In Chapter 1 we have expressed the need for a unified XD and IoT end-user authoring solution that would solve the problems of fragmented control, lack of uniformity, limited functionality and customisation. Based on studies by Google [111] and Microsoft [139] about the users' behaviour with their devices, we derived some requirements that such a solution should support. Further, by analysing the related

[1] `http://www.twofortyfouram.com`

work presented in this chapter, their system characteristics, user study results and requirements, we identified additional requirements. We collected a large number of different requirements, ranging from very specific requirements including warning mechanisms [72] or means for simulating and debugging rules [93] to more general requirements, such as external service triggers [125] or distribution of Web widgets [121]. From these requirements we kept only the more recurrent general ones. Note that this selection procedure has been mainly based on our own opinion. The remaining functional: R1, R2, R2.1, R2.2, R3 and non-functional requirements: R4, R4.1, R4.2 and R5 are listed below together with some research papers or studies from which they resulted. These requirements serve as part of the answer to our first research question, while additional requirements will complete our answer to RQ1 in the upcoming chapters.

Requirement 1 (R1). Provide an overview of the smart technologies, environments and applications Motivated by many presented solutions in the XDI and IoT domain (e.g. DireWolf [121], AppsGate [60] and TARE [93]), users should be provided an overview of the available devices together with their status. They should further be able to visualise their smart environment as well as the created applications or rules.

Requirement 2 (R2). Interaction support As explained in Chapter 1, the main goal of cross-device interaction and IoT research is to enable smart devices and smart objects to interact with each other. Therefore, this represents our second requirement, which is further subdivided into two subrequirements.

Requirement 2.1 (R2.1). Support for interaction across multiple smart technologies All discussed solutions provide different types of approaches in order to allow users to interact with their devices and make these devices interact with each other, thereby using various metaphors to help users cope with the underlying complexity of such interactions. Users must therefore be able to make smart technologies interact with each other by defining the way in which they should interact. Interaction between devices can be in the form of data transfer (e.g. [171]) or by issuing commands (e.g. [149]). For example, data transfer could be triggered by letting the user define that a picture should be sent to the tablet whenever a user swipes this picture in the direction of the tablet from their smartphone. An example of issuing a command could be that whenever a user presses a certain button on their smartwatch, the smart light bulb should turn on. These kinds of interactions should be easy to define and manage by users. Note that interaction involving data transfer facilitates the *sequential multi-device usage* [111] or *quantum journey* [139] which is often performed by people owning multiple devices. When changing devices, instead of having to search for the same data on another device, the data can simply be transferred—in

the chosen manner of the user—from the first device to the next device, on which the user can finish their task.

Requirement 2.2 (R2.2). Support for creation, customisation and distribution of cross-device and IoT user interfaces In order to further support interactions as seen in the domain of distributed user interfaces, not only data should be easily transferred and distributed across devices but also user interfaces. Therefore, users must be able to create, customise and distribute user interfaces between their devices. While some of the introduced systems apply a mashup approach taking different web interfaces and distributing them across devices [123, 121, 154], some systems, such as [92] also allow the creation of interfaces from scratch. This functionality is often not present in EUD for IoT environments. It is however present in TARE [93], although it is unclear how exactly the UI creation part is performed. The IFTTT tool supports the creation of simple buttons but does neither support the distribution of these buttons across devices nor the creation of more complex user interfaces. Notice that with this requirement we facilitate the *complementary usage* [111] of devices and *investigative spider-webbing* [139] behaviour performed by many users, where a user interface can be split and synchronised across multiple devices.

Requirement 3 (R3). Support for sharing and integration of apps in a central smart apps repository Looking at commercial solutions in the EUD of IoT environments, such as IFTTT and Atooma, we have seen that applications or *rules* created by the users are often stored in a central repository and shared amongst the community. The sharing helps users setting up new rules in a faster way than creating them from scratch. It further sparkles creativity and inspires users to create similar rules adapted to their particular needs [166]. Rule sharing has also been identified as one of the main requirements in TARE [93], where the authors claim that the reuse mechanism of rules was found useful in work settings such as warehouses and retail where efficiency is important. Similar findings about the advantages of sharing and building upon experiences of other users were also supported by Bellucci et al. [19], who also provide a sharing platform in T4Tags 2.0.

Requirement 4 (R4). Extensibility In order to allow a system to evolve over times it is important to support extensibility at several levels. As mentioned by Fischer et al. [81], users should not be presented with closed systems, but they should be provided tools to extend the system in order to fit their needs.

Requirement 4.1 (R4.1). Offer extensibility at the level of communication protocols, devices and user interfaces Given that an EUD authoring tool should support many different types of smart technologies ranging from computers, smartphones and

smartwatches to smart thermostats, smart lights and smart power plugs, it is important to make it easy to add new types of emerging devices. Some XDI tools, such as Improv [50], explain how they could be extended to support IoT objects. Such support for extensibility is important, since, as we have seen in Ericsson's report, *things* are still predicted to grow in popularity over the coming years [46]. Therefore, an authoring solution should not be a closed system where it is difficult for developers to add new devices and thus completely impossible for end users. Next to the flexible support of new devices and *things*, new communication protocols that might come with these new smart technologies and user interface components (e.g. [147]) should be easy to add over time as well.

Requirement 4.2 (R4.2). Enable the integration of third-party applications In order to trigger certain actions on a device, third-party applications might be needed and should therefore be easy to be integrated with the authoring tool. A popular commercial example from related work that provides third-party applications is IFTTT, but this is also supported in non-commercial solutions, such as EPIDOSITE [125] which uses IFTTT as backend and mashup tools such as DashMash [45]. An example of such a third-party application could be a *weather* service providing functionality related to the weather that could potentially be used in a rule that would be triggered depending on the outside temperature.

Requirement 5 (R5). Support for end-user development Given that the unified XD and IoT authoring tool is supposed to be used by end users, the functionality presented in the previous requirements should be accessible to end users whenever possible. End users should therefore be able to create, customise and distribute cross-device and IoT user interfaces (R2.2). They should further be provided means to share their applications with a community of users (R3). In addition, it should be easy for end users to add new devices and user interface components to their authoring tool, as already mentioned in requirement R4.1. Accordingly, end users should neither have to write a single line of programming code nor be required to have particular technical knowledge [126].

Chapter 3

Use Case

We are all apprentices in a craft where no one ever becomes a master.

Ernest Hemingway

In order to illustrate the potential of a unified end-user authoring solution which supports cross-device interaction, user interface distribution and interaction with the Internet of Things, we defined a use case scenario. This use case scenario is the result of an analysis of a larger set of use case scenarios collected from related work (e.g. meetings [73, 156], museums [87, 92], music scenarios [74] and others) as well as own use cases (e.g. scrapbook scenario [192]) involving XDI, DUI and IoT interactions. From this set of use cases we derived additional requirements which will be added to the ones introduced in the previous chapter in order to further address our first research question. In this chapter, we start by presenting our use case scenario that is a reduced version of the set of use cases, and has been created by keeping non-repetitive elements of the different scenarios which still reflect our derived requirements. After that we present the requirements that resulted from the larger set of use case scenarios. Note that the introduced scenario will also be used throughout the remainder of this dissertation to better explain various concepts.

3.1 Scenario

During the week, Lucas wakes up around 7:30 a.m. every morning. In order to support his daily morning routine, he created a number of applications. The first application is the *morning routine* application shown on the smartphone mock-up in Figure 3.1a. The application consists of several components. The upper left part of the screen shows an alarm that is currently set to 7:20 a.m. Next to the alarm, Lucas placed a button to control his coffee machine. The central part of the screen shows a real-time schedule of trains passing by between 7 a.m. and 9 a.m.; the selected train being the one Lucas plans to take. Some 20 minutes before the selected train arrives, the lights in the living room will start flashing as indicated at the bottom right. Further, as shown at the bottom left, the blinds in the bedroom should open when he gets up at 7:30 a.m. In addition, Lucas placed two extra buttons to manually control the blinds and the light.

Figure 3.1: Mock-ups of Lucas' first version of the *morning routine* application (a), his *cats* application on the tablet (b), his *leaving home* application on the smartwatch (c) and Lucy's version of the *cats* application (d)

The second application is the *leaving home* application running on a smartwatch as shown in Figure 3.1c. Once Lucas locks the front door, this application will check whether the lights, the TV as well as the electric iron are off. In the case that one of them is still on, Lucas receives a notification and can turn the device off via the smartwatch user interface. The UI consists of three parts—each containing a button to turn one of the devices on/off—and a swipe up/down gesture can be used to switch between them.

Lucas' next application is the *cats* application. He has two cats for which he bought a smart feeder, a pet Wi-Fi camera and a food storage box with LEDs and an integrated weight sensor. With his self-made application, he can control the feeder and see his cats when he is not at home. Further, the food storage box glows green when relatively full and red when almost empty to remind Lucas when he should buy some cat food. In addition, Lucas added a rule to this application to add the item "cat food" to the grocery list and show a `food low` icon in the UI when the box is glowing red, as shown in the mock-up of the cats application in Figure 3.1b. On the left-hand side of the UI, Lucas set different feeding times, one at 9 a.m. and one at 9 p.m. At any time, Lucas can check what his cats are doing via the video stream on the right, which also contains a `record/snapshot` button to record a small video or take a picture. He also added a button to access the gallery view containing the captured images and videos. Lucas further installed some motion sensors behind the couch to get notified when the cats try to scratch the couch, in order that he can then activate a small spray of water to stop them. The temperature component allows Lucas to see whether it is not too cold in the apartment for the cats. Once the temperature drops to 16 degrees Celsius, he gets a notification and can turn on the heating.

The last application, the *grocery list* application, is composed of two parts. One part is on the fridge while the other part is on Lucas' smartphone. Via the fridge component Lucas can add items to the grocery list as well as leave notes, memos or pictures on the fridge, while the smartphone only shows the synchronised grocery list. Figure 3.2b and Figure 3.2c show the mock-ups of this application. Lastly, in order to manage all the applications, an *administration panel* can be used to edit the applications, manage their rules and distribution properties as well as the user rights.

On a regular day during spring Lucas' alarm goes off at 7:20 a.m. After turning the alarm off, Lucas spends ten more minutes in bed until he finally decides to get up. Usually, the blinds would open now and he would start making his bed. However, it is not a usual day, since Lucas' girlfriend Lucy moved in yesterday evening. Therefore, they made some changes to Lucas' applications so that Lucy can also use them. She created a profile with her preferences and adapted some of his applications to her needs using the administration panel. Given that Lucy's job allows more flexible hours, she has to get up later than Lucas, around 8:30 a.m. Consequently, the blinds now need to be opened when both Lucas and Lucy are out of bed. So they changed this rule in the *morning routine* application accordingly as well as the alarm time which goes off later for Lucy. While Lucas and Lucy use the same applications they each have their own view of the user interface.

Hence, when Lucas got out of bed, the blinds did not open. He turns on the coffee machine with his application and leaves his room to greet the cats. Lucas takes his coffee and goes to the living room for having breakfast while watching the news on

TV. He usually takes the train of 8:30 a.m. which he also configured in his application. After a while, the lights start flashing, which means that Lucas has to leave soon in order to catch his train. He turns off the lights with his smartwatch, prepares to walk to the station, kisses his girlfriend goodbye and leaves his apartment without locking the door.

Lucy's morning routine is very similar to Lucas', however she does not take the train to work. Therefore, the UI of her morning routine application does not include the train schedule. Since she is taking her car to go to work, she added a new rule to the application: *"When there is a lot of traffic jam, make the artificial plants next to the TV glow orange and otherwise blue."* In addition, she also added a new UI element showing a real-time traffic map but did not use Lucas' rule for flashing the lights.

Later, Lucy heads off to work as well. When locking the door, she receives a notification telling her that the TV is still on. Since she does not have a smartwatch, she opens the *leaving home* application on her phone, which adapts to the new device as well as to her preferences and turns off the TV. Note that since Lucy is red-green colour blind, she uses orange and blue buttons instead of the red and green buttons, as shown in Figure 3.2a.

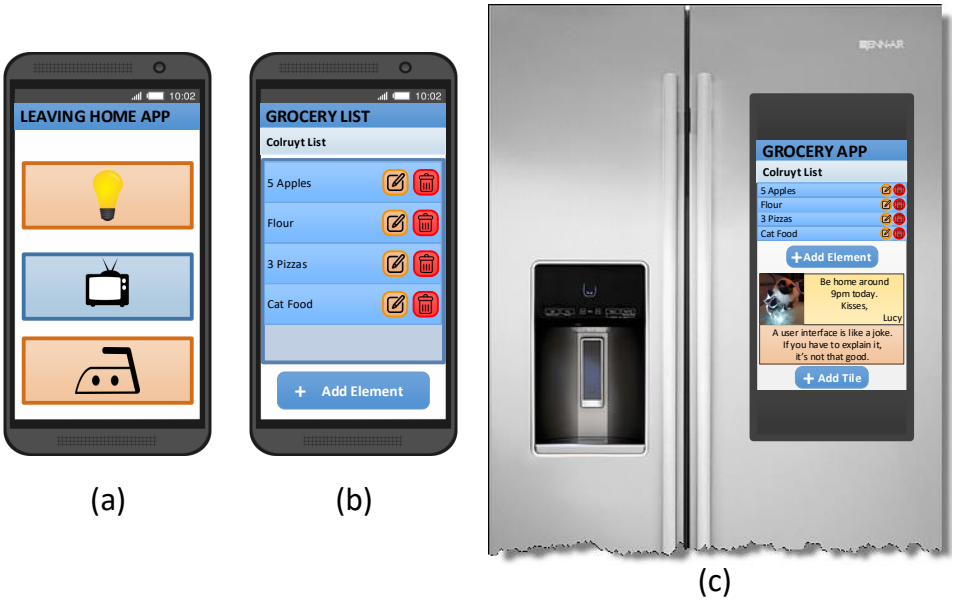


Figure 3.2: Mock-ups of Lucy's *leaving home* application on her smartphone (a), the *grocery list* application on Lucas' smartphone (b) and on the fridge (c)

During the day, Lucas receives a notification that the cats are scratching the couch. He wants to check via the cats application whether they are really scratching the

couch or just passing nearby. After a quick look at the video stream, he realises that the cats are just playing near the couch and therefore does not use the water spray, but takes a few pictures and hits the `funny` button instead. The `funny` button has been added by the couple to notify each other about funny content and is shown in the tablet and phone mock-ups in Figure 3.1b and Figure 3.1d (button with the cat face). Whenever Lucas presses this button, Lucy receives a notification. As illustrated on the smartphone mock-up, Lucy's UI of the *cats* application only contains the video stream, the gallery button, and `funny` button since she did not need all the other features. Further, Lucy shared the video stream UI element to her best friend who also loves cats. In contrast to Lucas, whenever she presses the `funny` button, she notifies Lucas and her best friend. After taking some pictures and sharing them with colleagues, Lucas continues to work.

When using the *cats* application earlier, Lucas saw that the cat food supply was running low. Therefore, he checks his grocery list, sees "cat food" that had been added automatically as well as a list of other products added by Lucy. He decides to go shopping on the way home and removes the products he bought from the list. Finally, once home, the couple shows each other a few pictures by displaying them on the TV using a *touch-and-throw* gesture in the gallery view on their smartphones. They defined this gesture as a throw gesture with their phone while touching the image they want to distribute to the TV.

Note that while users still use different applications to control their smart technology, these applications are organised and used for a specific context according to the user's needs, in contrast to current fragmented applications which are often tied to a certain brand of IoT objects.

3.2 Derived Requirements

The presented scenario illustrates the usefulness of a system allowing cross-device interaction as well as distribution of UIs (e.g. grocery list application), IoT interaction (e.g. using buttons on smartwatch interface) and some adaptations depending on the context which was the user of the application and device on which the application is viewed on, in this case. While in the previous chapter we derived requirements from related work, in this chapter we add requirements resulting from our use case scenario. Some requirements are added as subrequirements of those presented before while others will form new core requirements.

In R3 we expressed the need for sharing applications via a central repository. Based on our scenario we also want to include the possibility of sharing with specific users, more specifically users of the exact same devices, typically other members of a household.

Therefore, we change R3 into a core requirement "shareability" with as subrequirements the previously defined R3 which becomes R3.1 and a new requirement R3.2 formulated as described below. While requirement 3 and its subrequirements are all functional requirements, the remaining requirements introduced below are all non-functional requirements.

Requirement 3 (R3). Shareability Sharing is an important concept as it helps users setting up new rules or applications in a faster way than creating them from scratch. It further sparkles the creativity and inspires users to create similar rules and applications adapted to their particular needs [166].

Requirement 3.1 (R3.1). Support for sharing and integration of apps in a central smart apps repository Looking at commercial solutions in the EUD of IoT environments, such as IFTTT and Atooma, we have seen that applications or *rules* created by users are often stored in a central repository and shared amongst the community. Rule sharing has been identified as one of the main requirements in TARE [93], where the authors claim that the reuse mechanism of rules was found useful in work settings such as warehouses and retail where efficiency is important. Similar findings about the advantages of sharing and building upon experiences of other users were also supported by Bellucci et al. [19], who also provide a sharing platform in T4Tags 2.0.

Requirement 3.2 (R3.2). Enable sharing of applications, user interfaces or parts of a user interface with specific users While the previous requirement has been added with the idea of sharing a user's creations with the community of users, this requirement is specifically targeting users having access to the same device configurations as typically found in households. This means that the applications can be used as they have been created, or tailored to the new user's preferences as done with Lucy in the presented scenario. Such a type of sharing can also be used for guests, in order to provide them (temporary) access to the smart devices in the house.

In Chapter 2 we defined requirement R5 as the *support for end-user development*. Since this requirement is seen as an overarching requirement, we move this requirement to the end of our list and update it depending on the new requirements derived from our use case scenario.

Requirement 5 (R5). Reusability Next to sharing with other users, it is also important to support the reuse of user interfaces and functionality. In order to do so, a clear separation between the user interface elements and the tasks that will be performed by those elements is necessary. That way UI elements can be reused to perform other tasks. Further the same tasks can be executed using other UI elements or rules. By

allowing flexible reuse, users also do not have to start from scratch every time they design new applications. Therefore we define the following subrequirements:

Requirement 5.1 (R5.1). Support for reuse and combination of different user interfaces When designing new user interfaces, a user should be able to reuse the same components in multiple applications and eventually combine these components in different ways. The `on/off` buttons of the *morning routine* application have, for example, been used multiple times and depending on their position in the user interface, the button is associated with different functionality. The button next to the light acts like a light switch, while the button next to the blinds allows the blinds to be opened or closed. Note that the reusability of UI components has also been seen in related work, such as Jelly [147].

Requirement 5.2 (5.2). Support for reuse and combination of different functionality This requirement is different from the previous one in the sense that instead of reusing user interface components, users should also be able to reuse application logic. Further, they should be able to create combined new functionality out of existing ones, as done with the `touch-and-throw` gesture that is a combination of the `touch` gesture and the `throw` gesture recognition functionality. Another example is the use of the `notification` functionality which is used in the *leaving home* application to notify Lucas or Lucy when they close the door if an appliance is still on. This `notification` functionality is reused in the *cats* application with the `funny` button that notifies Lucas' or Lucy's contacts when the button is pressed. Note that the reusability of functionality has also been shown in related work, such as reusability of rules in TARE [93] and in CMT [212].

Requirement 6 (R6). Portability Given that we want to support cross-device interaction, it is also important to make sure that user interfaces and applications can be used (and run) on all compatible devices. By compatible devices we mean that certain applications might need specific characteristics to be present on a device to be able to run on this device. For example, typically a graphical user interface can only be used on devices that have a display.

Requirement 6.1 (R6.1). Offer platform independence While an interface might have been created with a certain platform or device in mind, the application should be able to run on any compatible platform. For example, Lucas designed the *leaving home* application for his smartwatch but Lucy is using it on her smartphone.

Requirement 6.2 (R6.2). Support for adaptation of applications and user interfaces to users and devices Related to what has been explained in R6.1, an application should be able to run on multiple devices, but it should further also be adapted according to the characteristics of the device on which it is accessed. In the use case scenario we have seen that the *leaving home* application, for example, displays larger rectangular buttons when opened on Lucy's smartphone compared to the round buttons shown on Lucas' smartwatch. Further the user interface should also adapt to specific users. For example, the interface should adapt to Lucy by replacing the colours red and green by other colours, as shown for the *leaving home* application where her buttons are shown in blue and orange.

Requirement 7 (R7). Support for end-user development Given that the unified XD and IoT authoring tool is supposed to be used by end users, the functionality presented in the previous requirements should be accessible to end users whenever possible. End users should therefore be able to create, customise and distribute cross-device and IoT user interfaces (R2.2). They should also be provided means to share their applications with a community of users (R3.1) as well as share to specific users (R3.2). Further, it should be easy for end users to add new devices and user interface components to their authoring tool, as already mentioned in requirement R4.1. In addition, end users should be able to reuse existing UIs and functionality (R5). The user-defined applications should further be able to adapt to specific users and devices in the manner described by the end user (R6.2). Note that, accordingly end users should neither have to write a single line of programming code nor require to have any particular technical knowledge to perform all the described functionality [126].

As we will see in the next chapter, we choose to follow a model-based approach in order to fulfil these requirements. The advantages of such an approach is that it facilitates the development process and makes our system more flexible, extensible (R4) and reusable (R5). The details and further advantages will be extensively discussed in the following chapter where we present related work of model-based solutions in the domain of adaptive, cross-device and IoT user interfaces and introduce our conceptual model and reference framework.

Chapter 4

Reference Framework and Conceptual Model

No substantial part of the universe is so simple that it can be grasped and controlled without abstraction. Abstraction consists in replacing the part of the universe under consideration by a model of similar but simpler structure. Models, formal and intellectual on the one hand, or material on the other, are thus a central necessity of scientific procedure.

Arturo Rosenblueth and Norbert Wiener

In the previous chapters we identified a number of limitations and requirements for managing and controlling smart devices as well as smart *things*. In this chapter we present our model-based approach for fulfilling these requirements and addressing existing limitations. First, we explain the benefits of a model-based approach and review some related work of model-based user interface development in the cross-device and IoT research domains. We further look at adaptable and adaptive systems in order to provide the context aware interactions identified in one of our requirements (i.e. R6.2). Based on related work we refine our final requirements and present a reference framework and conceptual model which lays the conceptual foundation for our end-user authoring tool.

4.1 Related Work

In the late 1980s, a large number of modelling languages for the creation of richer and accurate interface models emerged, which led to the rise of *model-based user interface development (MBUID)* [143, 176]. This approach uses declarative models to generate user interfaces (UIs). The aim is to reduce the costs of developing and maintaining UIs (compared to traditional UI development) by following a layered architecture that separates different concerns [64, 84]. The use of models also results in other benefits, including increased communication and control, as exposed by Hutchinson et al. [110]. Many prototype systems started to emerge where developers could use a high-level specification language to write the characteristics of the user interface rather than using a programming library. The specification would then automatically generate the corresponding UI or be translated into an executable program.

Szekely [208] introduced a generic MBUID architecture which shows the typical components of model-based interface development environments (MB-IDE). The main components are the modelling tools, the model, the automated design tools and the implementation tools. In order to build the model, developers use the modelling tools. The model is divided into three levels of abstraction. The highest level contains the *task and domain model*, which represents the tasks users need to perform and the data and operations supported by the application. The next level is called the *abstract user interface (AUI)* specification, which expresses the structure and content of the UI in an abstract way, for example, by expressing the information that will be shown in each window. The third level is the *concrete user interface (CUI)* level, which specifies the rendering style of the presentation units and the layout of all elements of a window. Many MB-IDEs allow designers to only work on certain aspects of a model and use automated design tools to compute the missing elements of the model. Using the implementation tools, the CUI specification can be translated into an executable representation linked to application logic. This source code has later been adopted as the fourth abstraction level and is called the *final user interface (FUI)* level [43]. More details about the model-based interface development process can be found in [208]. During the 90s, different MB-IDEs were introduced using different modelling languages and focusing on different levels of the model. ITS [221] and MASTERMIND [210], for example, required developers to specify all levels of the model, while JANUS [15] only needed the developers to provide the data model and with HUMANOID [209], developers could generate an interface by only specifying the data types of command inputs.

Since then, modelling languages have become more expressive as well as easier to use, and different conceptual frameworks and architectures have been introduced to capture the important aspects of MBUID. However due to the continuous evolution

of UIs, model-based user interface development still remains a challenge. Already in 1996, Szekely [208] stated some challenges that are still valid today. The first one is *task-centred interfaces* (1). Applications offer a lot of functionality and options to users, who often have a hard time figuring out the capabilities of the application in order to perform their task. Some applications try to cope with this problem by introducing *task assistants*, help buttons and tutorials. Although, nowadays users want to be able to use an application on the fly and do not want to spend too much time trying to understand an application. A second challenge is *multi-platform support* (2). With the growing amount of devices in each household, there is a need for UIs that can run on multiple devices and platforms and that can adapt themselves depending on the device and platform. While traditional interface development needs a redesign of each UI element for individual platforms, the model-based approach offers a better alternative, where changes only need to be done in the mapping from the abstract interface objects to the concrete interface objects. For similar devices with different screen sizes a simple scaling factor is necessary. As a third challenge, Szekely introduced *interface tailoring* (3), which refers to the ability of an interface to customise and optimise itself according to the context of use. This ability can take place at different levels of the MBUID process. Developers may provide different versions of the final application depending on some criteria or the tailoring can be done by experienced users. In some cases the interface can adapt by analysing the user's patterns of use. The FUSE [130] system, for example, allows the interface to be tailored according to the level of experience of a user. The last challenge is support for *multimodal interfaces* (4). Input modalities such as pen interaction, gestures, speech, gaze and output modalities such as sound, light, vibration and many more can be used to make richer user interfaces. However, creating UIs that combine all these modalities is difficult for many reasons. The input modalities need to be visible to the user, straightforward and natural to use. Combining multiple modalities can become redundant in some cases. Furthermore, some modalities will be preferred depending on the context of use. An early architecture for the development of multimodal interfaces has been presented by Cohen [51]. The open architecture supports pen, voice and direct manipulation. More recently, Vanderdonckt [215] introduced 20 additional detailled challenges for MBUID. Further, a thorough overview about the history and challenges of MBUID can be found in the work of Meixner et al. [143] and a survey on model-driven engineering tools has been presented by Perez-Medina et. al [170].

The challenges by Szekely [208] are quite well covered by the requirements defined in the two previous chapters. By providing an overview (R1) and expressing the fact that functionality of the authoring solution should be simple for end users (R7), we address the first challenge. Next, the multi-platform support is expressed by our portability requirement and more specifically by requirement R6.1. asking for "platform independence". The third challenge involving interface tailoring is reflected

in R6.2, where we mention the need for adaptation. Only the last challenge involving multimodal support is not directly reflected in our requirements. Enabling interaction across devices could be done in a multimodal way, but in this dissertation we do not focus on how to tackle this last challenge in details. In some of the related work, authors introduce how they added different modalities to their system and in future work we will also discuss how to integrate multimodality into our solution. Note that, at the time when Szekely's challenges were introduced, IoT devices were not yet as popular as today. We believe that a fifth important main challenge for MBUID is the integration of *things* into the UI design process. The inclusion of *things* might be quite challenging since they come with new kinds of physical or digital UIs depending on their characteristics and capabilities.

In the remainder of this section we focus on MBUID in the domain of adaptive and adaptable UIs, cross-device UIs and UIs for IoT environments.

Model-based Adaptive and Adaptable User Interfaces

One of the most challenging parts of developing UIs is to build an interface that fits all of its users. Adaptive user interfaces—also referred to as multi-context or multi-target user interfaces—aim to improve the user experience by adapting to a user's context of use at runtime. Thereby, the context of use is defined by a triple consisting of the *user*, *platform* and *environment* entities [84]. Another variant of adaptive UIs are adaptable UIs, which can be tailored according to predefined options. While adaptive UIs automatically react to changes in context, adaptable UIs usually require an explicit human intervention. A key goal of adaptive UIs is *plasticity*, which is the ability of a UI to preserve its usability throughout multiple contexts of use [59].

The first reference framework for multiple contexts of use applying a model-based approach was introduced by Calvary et al. [41]. An extended version of this framework with additional relationships and definitions was presented later on [42, 43], which gave rise to the CAMELEON reference framework (CRF) [40]. The CRF has now been accepted by the HCI community as a reference to structure and classify model-based UI development processes, supporting multiple contexts of use [142]. Both design and run-time phases are covered by the CRF. Furthermore, support for UI plasticity is proposed via ontological models and the use of observed models at runtime [42]. The adaptation metamodels provide tools to specify the appropriate reaction on context of use changes. The CRF structures the development life cycle into four levels of abstraction which were introduced earlier and are shown in Figure 4.1. In order to use these levels for the model-based development, CRF proposes a four step reification process. The *concepts-and-tasks model* is reified into an *abstract user interface* which is reified to a *concrete user interface* model. Finally, the concrete UI

will be turned into the *final user interface* which is typically done via code generation techniques.

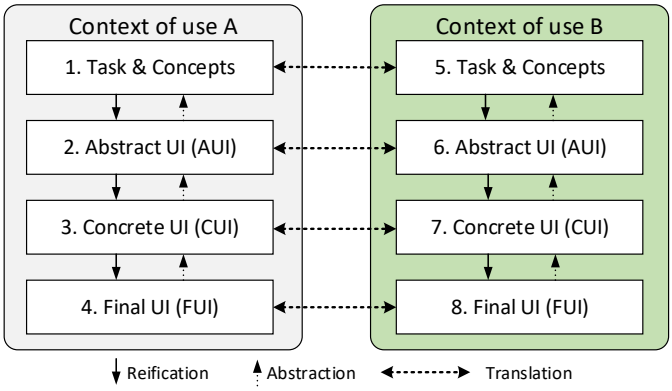
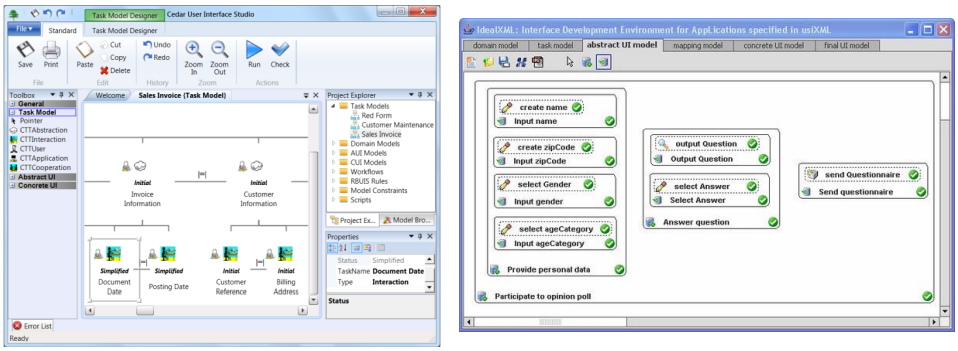


Figure 4.1: Simplified CAMELEON Reference Framework [127]

Over time, different approaches were introduced for the development of adaptive and adaptable UIs. Some approaches focussed on providing a reference architecture, others proposed techniques to achieve adaptive UIs and the last category introduced new tools. We will focus on the model-based systems that introduce a practical technique based on a reference architecture and demonstrate the feasibility of the approach by introducing one or more tools.

A large part of the existing tools propose an IDE style UI to create adaptive and adaptable UIs. An example can be found in Akiki et al.'s work [4], where the authors present Cedar Studio, an IDE to support technical users in developing and maintaining adaptive model-driven applications. The IDE uses the role-based UI simplification (RBUIS) mechanism [5], which is based on the CEDAR reference architecture [3]. By using CEDAR, the system supports direct as well as indirect adaptation and allows its adaptive behaviour to be extended. Furthermore, the architecture follows the levels of abstractions suggested by CAMELEON for representing the UI models, which makes it possible for the IDE to offer control of the UI at all levels of abstraction using visual design tools. Next to visual design tools, Cedar Studio also proposes code editing tools for the extensible adaptive behaviour. It provides support for model design as well as automatic generation and synchronisation between models. The design tool provided for the Task models uses the widely adopted ConcurTaskTree (CTT) [164] notation to model the tasks, as shown in Figure 4.2a.

The first design tool to model CTTs was presented by Mori et al. [151] and called the ConcurTaskTrees Environment (CTTE). It supports the creation, editing and analysis of task models as well as a simulator to analyse the dynamic behaviour of the task models. A few years later, Paternò et al. [167] introduced the MARIA



(a) Task modelling with CEDAR Studio [4] (b) Abstract UI model from IdealXML [206]

Figure 4.2: Adaptive model-based solutions

language and tool. Using MARIA, AUI and CUI models can be generated from a CTT task model. The associated MARIA authoring tool can be used to describe the UIs at different levels of abstraction and has especially been developed for designing and generating interactive web applications based on web services. The tool further allows designers to customise the model-to-model transformations and the rules used to transform the model into a final implementation. The use of the MARIA language has been demonstrated by presenting the migratory Pac-Man application, where the UI is able to follow mobile users as they change devices while maintaining the state of the application. Nonetheless, according to the authors the tool is not yet suited for end-user development as further usability testing is still required and additional components should be added to make the tool usable for non-professional software developers.

Another popular user interface description language (UIDL), called UsiXML[1] (USer Interface eXtensible Markup Language), has been introduced by Limbourg and Vanderdonckt [127]. The language allows designers to apply a multi-path development of their UIs, thereby allowing a user interface to be specified and produced from any level of abstraction. This development method is also based on the CAMELEON reference framework defining the development life cycle for multi-context applications. Many tools were presented supporting UsiXML. For example, UsiComp [89] is a UI development and execution environment based on UsiXML allowing designers to create models which can be modified at design and runtime. It produces UIs that are adapted to different platforms depending on the platform model. The editor shows the different models within a single document-style panel, which makes it less flexible and more difficult to use for large-scaled systems.

[1] `http://www.usixml.org/en/home.html?IDC=221`

The different UsiXML-based tools target different phases of the CRF abstraction levels, meaning different phases of the UI development process. Some tools were introduced for the AUI level, such as IdealXML [206] that creates abstract UI models from task and domain models as illustrated in Figure 4.2b. Other tools focus on the CUI level, including SketchiXML [62, 63] which is used to generate CUI models from hand-drawn sources, VisiXML [216] that generates CUI models from vectorial drawings, and GrafiXML [150] a graphical high-fidelity editor to design CUIs by dragging and dropping widgets. These three tools range from low-fidelity to high-fidelity prototyping. Finally, there is a category of tools for rendering the final UI (FUI), such as FlashiXML [21], which is a UI generator using UsiXML as source language and Macromedia Flash as target language and JaviXML which renders CUI UsiXML files in Java. A complete overview of the UsiXML tools can be found in Vanderdonckt [214]. While UsiComp may still be an early-stage research prototype, it is the only one of the presented UsiXML tools that supports all levels of abstractions. Model-to-model transformations as well as UI adaptation rules are written in the Atlas Transformation Language (ATL) [117]. The rules can, for example, be used to adapt the UI models to different platforms. An ATL transformation is also used to convert the CUI to Java code.

Inspired by multiple research domains and based on the CAMELEON's abstraction levels, the Multi-Access Service Platform (MASP) has been created to address the deployment and run-time issues when developing user interfaces in smart environments [181, 29]. MASP provides support for the creation of adaptable [197], distributed [27], synchronised [26] and multimodal [28] UIs based on an abstract description language. Based on this abstract description of content and interactions, it can generate the user interfaces in languages such as HTML, WML, and VoiceXML. MASP uses CC/PP-based [119] device capability detection to detect the device capabilities at runtime, which allows the generation of UIs by taking screen size, the supported media and user preferences into account. Furthermore, MASP provides a layouting tool [80] for generating the layout models, a task tree editor and a task tree simulator. However, MASP's tools only support basic pre-defined adaptations applied on a box-based layout and do not support the definition of visual and code-based adaptation rules which could allow more layout optimisations.

A model-based approach has been presented in GUMMY [148], a multi-platform GUI builder that can generate initial designs for a new platforms based on existing UIs for the same application. The prototyping of the multi-platform GUIs is done in the same way as in traditional GUI builders (e.g. Visual Studio[1]). While designers prototype their UI, GUMMY builds a platform-independent representation of the UI. This allows GUMMY to hide the levels of abstractions from designers, who can then

[1] `https://visualstudio.microsoft.com`

operate on the CUI level—which they know best—and hereby have more control over the UI. The underlying language used for the generated models is the User Interface Markup Language (UIML) [1], which is an XML-based language for representing CUIs. GUMMY still lacks some important features. It only allows the design of user interfaces with a single screen and there is no way for designers to specify how the UI should behave when resized at runtime. Furthermore, due to the fact that designers can only work on the CUI level, some characteristics of models from other abstraction levels (such as the temporal operations on CTT tasks models) are not easy to deduce.

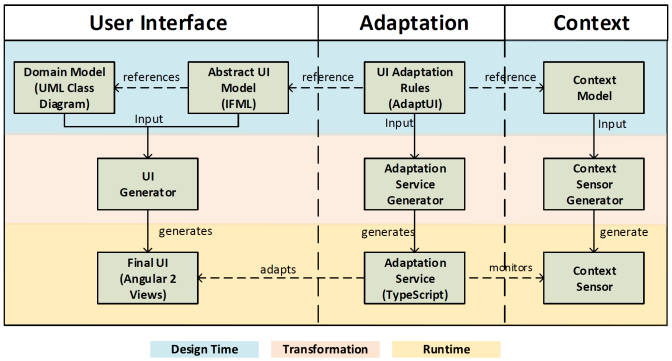


Figure 4.3: Model-driven architecture for self-adaptive UIs [223]

Based on adaptability properties for software systems [191], referred to as self-* properties, Yigitbas et al. [223] present how these properties can be applied to the domain of self-adaptive UIs. Such UIs are able to adapt to the context of use at runtime using these self-* properties. Yigitbas et al. [224] introduced a model-driven architecture consisting of three parallel development paths, which is illustrated in Figure 4.3. The first path addresses model-driven development through the use of an AUI and domain model which are transformed into a final UI. The second path has been added to support adaptation while the last path is meant for context management. The authors further provide an IDE called Adapt-UI for the model-driven development of self-adaptive UIs. The Adapt-UI development environment provides different modelling views. One view is for UI modelling by using IFML[1], another view is for context modelling by using ContextML and the last view is for adaptation modelling with AdaptML. Based on these different models, UI code is generated that is coupled with adaptation services and integrated in an overall UI framework, which allows run-time UI adaptation realised by an automatic reaction to changing context of use parameters [223, 224].

A broader overview about the history of MBUID in the context of adaptive UIs can be found in the work of Akiki et al.[6].

[1] `https://www.ifml.org`

Modelling of Cross-Device User Interfaces

In this section we look at models for cross-device and distributed user interfaces, where UIs can not only be generated for multiple devices—as we have seen in the previous section—but can also be split up and synchronised across various devices. Similar to our previous section, we focus on model-based systems that demonstrated their feasibility by introducing use cases.

Balme et al. [14] presented CAMELEON-RT, one of the first architecture reference models which referred to the term distributed user interfaces. The model can be used to compare and reason about existing run-time infrastructures as well as to create future run-time infrastructures for distributed, migratable and plastic user interfaces. The CAMELEON-RT architecture reference model consists of three layers, the interactive systems, the Distribution-Migration-Plasticity (DMP) middleware and the platform layer. The DMP layer is the core of the architecture and provides mechanisms and services for DMP UIs. Based on this architecture *CamNote* (CAMELEON Note), a distributed slides viewer which is able to run on multiple platforms, and *I-AM* (Interaction Abstract Machine), a platform manager supporting the dynamic configuration of resources across different workstations, have been developed. The architecture targets the three context of use pillars and can be considered as general purpose due to its implementation neutrality. All levels of abstraction are supported since it follows the CAMELEON reference framework.

In the same year and based on the concepts of DUIs and MBUID, Vandervelpen and Coninx [217] presented an approach to add support for UI distribution to their Dygimes model-based runtime environment that dynamically generates user interface for mobile devices and embedded systems. Dygimes [52] has originally been created to generate UIs for different kinds of devices at runtime by using task specifications [164] that are combined with XML-based Abstract User Interface building blocks. The envisioned model extensions consists of adding distributed event controllers, real location transparency, tool support and distributed layout/topology management. A year later, the same authors took a different approach, one that does not rely on a new methodology or ontology, but relies on a suitable description in order to transform functionality of native applications into web interfaces and focusses on the dynamic distribution of these web interfaces [218]. RelaxNG[1] schemas are used to describe the services offered by native applications and from these schemas the XHTML-based user interface is generated. Using light-weight HTTP-based daemon as a distribution manager, the interface can be distributed manually or automatically among heterogeneous devices. This allows designers to distribute web interfaces with a minimum

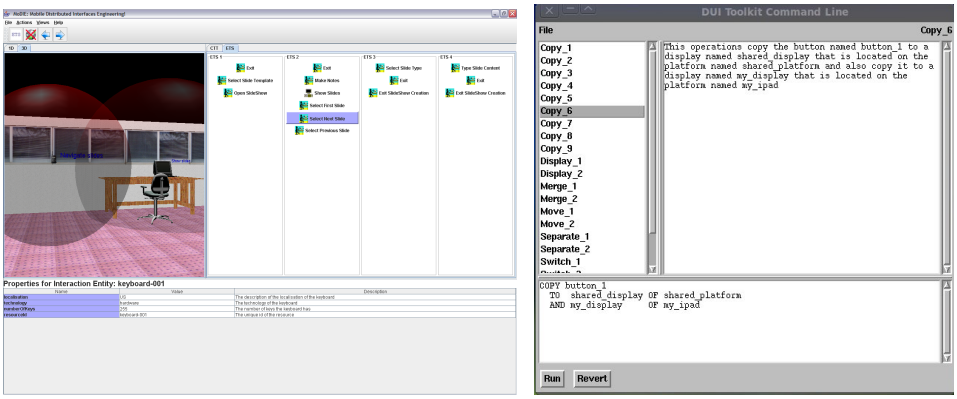
[1] http://www.relaxng.org/spec-20011203.html

effort. The authors [134] later concluded that a structured high-level user interface description language (in this case HTML) is most suitable to create DUIs.

Going to the field of ambient intelligence, Luyten et al. [133] presented the MoDIE (mobile distributable interface engineering) platform [134], a tool to design UIs for ambient intelligent environments by following a task-centred model-based approach. With UI completeness and continuity as main properties in mind, the authors provide tools for the visualisation of the task allocation environment and simulation of the execution of the tasks. The authors introduced the notion of a situated task in order to model how a task can be distributed in subtasks on different platforms in the same environment over time. The task allocation is visualised by the tool in a 3D environment as depicted in Figure 4.4a. Luyten et al. further envisioned to integrate MoDIE with model-driven engineering to create a complete environment for designing and testing distributed user interfaces. Another system focussing on the distribution of tasks is FlowiXML that introduced a series of workflow UI patterns for distributing tasks to one or many resources [88]. In contrast to MoDIE, FlowiXML has already been integrated in a multipath UI development process using UsiXML [127] and validated with a case study.

Some adaptive UI approaches evolved towards systems that include user interfaces that can be distributed across devices. Manca and Paternò [136] extended the MARIA language to provide support for the description of distributed user interfaces. Unlike Vandervelpen and Coninx [217], who focus on the AUI level, Manca and Paternò support the UI specification at the CUI level in order to facilitate the generation of the corresponding implementation. Next to supporting GUIs, devices using other modalities, such as voice and touch, are also supported. A few years later the authors also proposed an authoring tool exploiting the distributed MARIA language to allow the distribution of a UI at different granularity levels [137]. They presented the potential of their solution with a variety of distribution scenarios and by allowing dynamic end-user customisation of the distribution.

A few months before the introduction of the distributed MARIA tool, Melchior et al. [146] already proposed true DUI support based on a model-based approach as well. The approach contains three pillars, the *models* for DUIs along with the specification language to express these models, the *step-wise method* for modelling the DUIs and an envisioned *software* that supports this step-wise method. The platform, user and environment model are used to represent the context of use. In order to model the distribution of UIs, a CUI model which includes the distribution dimensions and is able to express any DUI element in an XML-compliant format is used. Furthermore, a set of distribution primitives are defined which will operate on the CUI models and enable the distribution of UI elements. A formal language to express these primitives is introduced as well and is shown in Figure 4.4b. Using this



(a) Task distribution configuration in MoDIE [133] (b) Extra-UI by Melchior et al. [146]

Figure 4.4: DUI model-based solutions

language, distribution scenarios can be created for automatic and manual distribution via a command line interface serving as extra-UI. Melchior et al. illustrate this method with two DUI applications, a Pictionary and a Minesweeper game, which were then incorporated in a larger DUI Game of the Goose. With this approach the distribution of UIs is supported at design time and run time. A few years later, Melchior [145] introduced his toolkit for creating DUIs, called the JayTk, which is built on top of Beernet [144] to allow peer-to-peer distribution. JayTk provides different levels of granularity for UI distribution, namely action/service, widget, windows and application. Using this toolkit, designers and developers can create applications that support dynamic distributed graphical user interfaces and that can be used on multiple devices. This technology has been used by Melchior's Spin-off called Usidistrib[1], which extends current applications to support distribution and mobility across devices. In contrast to Manca and Paternò [136], Melchior et al. [146] focus on GUIs and did not test other modalities such as voice.

As introduced earlier, a system which supports voice-based interactions is MASP[2] [25]. The authors first introduced a framework for the creation of multimodal UIs for smart home environments. Based on this framework, they proposed a model-based user interface architecture called MASP, whose core is a set of UI models representing the interaction on different levels of abstraction. Based on the context of use, the UI is derived from the run-time model, which is then distributed and adapted across devices that support different modalities. Afterwards, MASP coordinates input and output and keeps all UI parts synchronised [188]. MASP has been deployed in the Service Centric Home Project and different multimodal applications [220], including

[1] http://www.usidistrib.be
[2] http://www.dai-labor.de/ngs/abgeschlossene_projekte/masp

a cooking assistant [28], an energy assistant [188] and a meta user interface [187]. However, according to Melchior [145], MASP is not providing any choice regarding the displayed UI, offers a limited multi-user interaction and it is not possible to control the distribution granularity.

More recently, Tesoriero and Altalhi [211] proposed a model-based approach to develop applications that exploit coupled-displays ecosystem. The authors propose the concept of Distributable User Interfaces (DeUIs), where users can distribute Interaction Objects (IOs) across Interaction Surfaces (ISs). Hence, instead of sharing the whole screen, users can share IOs, such as labels, buttons and frames across displays, which allow more flexibility, privacy, a higher granularity and multitasking. To enable such a UI distribution, Tesoriero and Altalhi extend the CAMELEON Reference Framework (CRF) models and metamodels to integrate UI distribution models. In order to create, edit and validate UI distribution models, a graphical model editor was developed as an Eclipse feature based on the Eclipse Modeling Framework (EMF)[1]. The models are used as input parameters for the model-to-text transformation that generates the source code of the web applications. The code runs on the User Interface Distribution Framework, which serves as extra abstraction layer to isolate the implementation of the UI distribution. This application is called distributable by the authors, because it can either run on the devices as standalone version or be distributed across devices. While distributed UIs require more than one device or screen, DeUIs does not require more than one device to be able to run.

Modelling of IoT Environments

In our background chapter we have seen that a lot of research has been done in the IoT domain in order to promote interoperability among IoT devices. IoT environments are characterised by a hight degree of heterogeneity, which is addressed by certain platforms by proposing models and reference architectures abstracting the functional and non-functional elements of IoT systems [47].

Bassi et al. [18] introduced the IoT Architectural Reference Model (IoT ARM), which provides a common structure and guidelines to handle the core aspects of the development, usage and analysis of IoT systems. The model consists of submodels. The foundation of the IoT reference model is the domain model which is illustrated in Figure 4.5. The domain model introduces the main concepts of the IoT including `Devices`, `Physical Entities`, `Services` and `Virtual Entities`. Virtual entities are the digital representations of physical entities, also referred to as *virtual counterparts*. A service offers the functionality for interacting with network or device `Resources` associated with physical entities. Next to the IoT domain model, the

[1] `https://www.eclipse.org/modeling/emf`

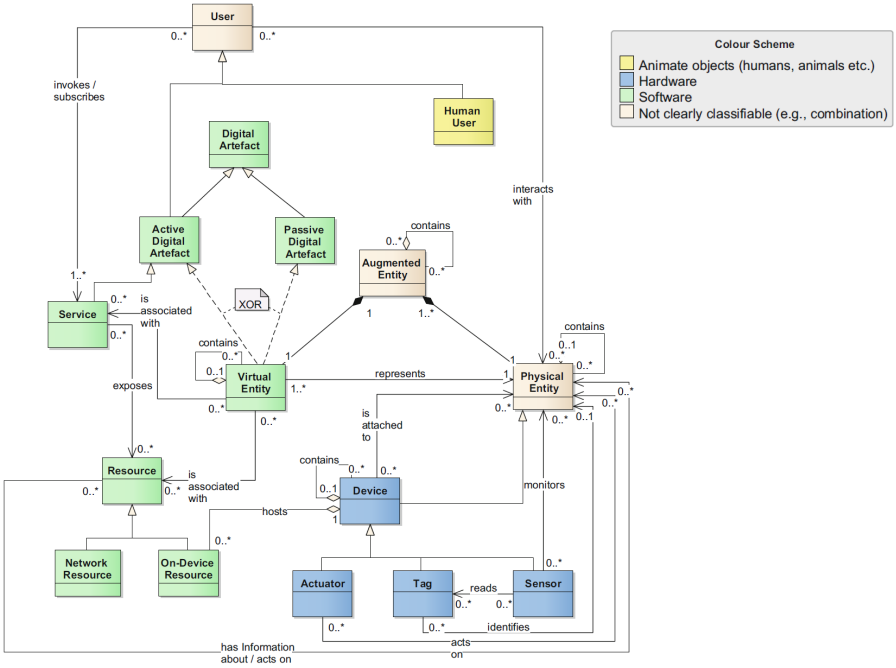


Figure 4.5: IoT ARM domain model [18]

IoT reference model includes the IoT information and functional model. The latter also consists of the IoT communication model and IoT trust, security and privacy model. Based on the IoT ARM, Bassi et al. provide a reference architecture to generate concrete IoT architectures. A process for generating concrete architectures is presented in Chapter 6 of Bassi et al.'s book [18].

A model-driven approach using IoT ARM as starting point, has been presented by Costa et al. [56], who introduced a design and analysis process together with a framework to help designers model their IoT applications and verify the corresponding Quality of Service (QoS) properties. The framework consists of a SysML4IoT and a SysML2NuSMV component. The SysML4IoT is a SysML profile[1] based on the IoT-ARM [18] and is shown in Figure 4.6. Most concepts depicted in the figure come from the IoT-A reference model such as the `Devices` and `Services` that are part of the `System` as well as the `Physical Entities` and `Users`. Other concepts have been added to comply to the ISO/IEC/IEEE 15288 standard[2]. The SysML2NuSMV component is a model-to-text translator which automatically converts a model and its QoS properties into a NuSMV (symbolic model verifier) program. The approach has been evaluated via a proof-of-concept IoT application for energy consumption.

[1] `https://sysml.org`
[2] `https://www.iso.org/standard/63711.html`

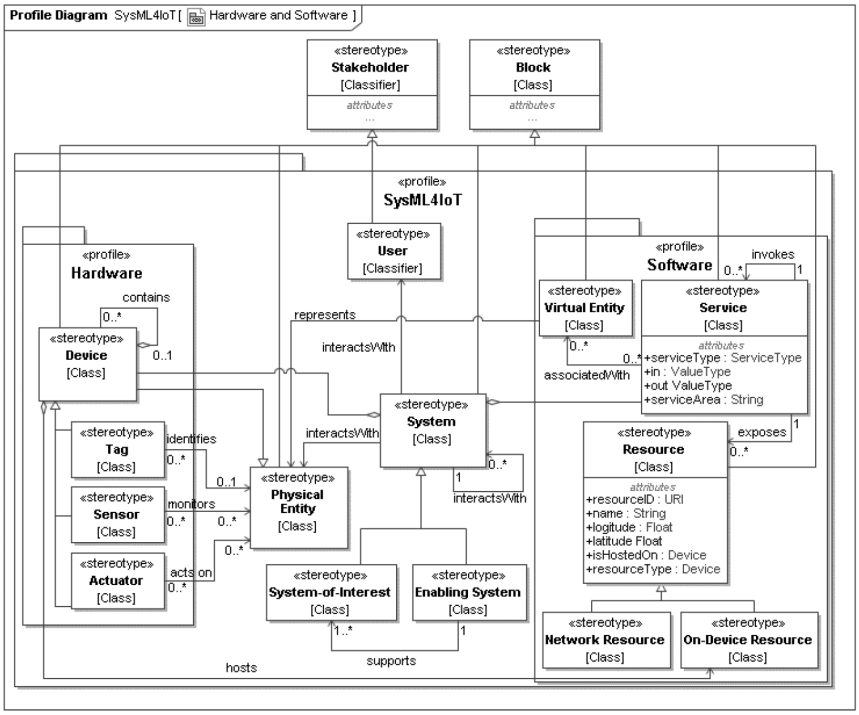


Figure 4.6: Partial SysML4IoT profile representation [56]

While Costa et al. [56] focussed only on one development phase, namely the analysis. Pramudianto et al. [175] propose an MDD approach for generating skeleton code. The authors present a perspective of an IoT architecture that decouples the domain modelling from the implementation of IoT technology, in order to allow the knowledge of the domain to be engineered by domain experts, while technology experts can focus on the implementation. A domain modelling tool has been developed so that domain experts can design domain models which contain virtual objects that can be linked to IoT implementations. Once the virtual objects are linked, prototype code in Java is generated and can be manually refined. The architecture and the tool have been evaluated using a software walk-through technique and have shown potential to ease the development of IoT applications.

Some authors focus on the communication between IoT devices, such as Fleurey et al. [82], who proposed a model-based approach to automatically generate efficient communication APIs in order to exchange messages between resource-constrained devices. The approach is based on a concise ThingML[1] description of the messages.

[1] `http://thingml.org`

Focussing on the communication as well, Mainetti et al. [135] extended the topic approach from the MQTT publish-subscribe protocol [129] by proposing the Web of Topics (WoX), a conceptual model for the IoT. The aim is to shorten the gap between the design and the solution domains in an IoT context. In WoX an IoT entity is defined as a set of Topic-Role couples. The authors focussed on the emerging Web of Things (WoT) paradigm that abstracts the heterogeneity of embedded devices to facilitate their integration and interoperability. WoX can be seen as an abstraction layer between the WoT and the user applications, made to accelerate the development of IoT applications by hiding the communication protocol details. A few years later, the Wox APIs have been published on the WSO2 [86] Enterprise Service Bus to exploit the key benefits of an enterprise architecture (e.g. security, scalability and interoperabilty.) [39].

Conzon et al. [53] identified a lack of toolkits that easily enable developers to create and evaluate IoT prototypes. Therefore, the authors extends the work on the ebbits platform architecture [34]. While the ebbits platform provides features for the integration of *things* by exposing them as internet web services, application developers still need in-depth knowledge of IoT technologies in order create system that uses these *things*. Conzon et al. propose a model-driven development toolkit based on the semantic discovery service that allows developers to dynamically select and locate available resources and devices. It also provides a graphical interface to create mashup applications.

Instead of focussing on the executability of IoT applications like in previously described systems, Brambilla et al. [33] focus on the user interaction. The authors' work is meant to complement such previously described systems. They provide IoT extensions to the IFML language [32], introduce some UI design patterns for the modelling of user interactions with IoT systems and include the implementation of a code generator prototype tailored for IoT application development. An example of the user interaction model is shown in Figure 4.7. While previous MDD IoT solutions could provide the access layer to the devices, this solution can be used for the model-driven specification and execution of the application layer.

Some authors focus on sensor networks, such as Patel and Cassau [162] who propose a development methodology for IoT application development inspired by model-driven development and building upon work in sensor network macroprogramming. The methodology defines a sequence of steps to be followed to develop IoT applications by a separation of concerns. Four concerns have been identified, namely, domain, functional, deployment and platform. The concerns are addressed by dividing the responsibilities of stakeholders into different roles, namely a domain expert, software designer, application developer, device developer and network manager. The development methodology is implemented as a concrete development framework and is

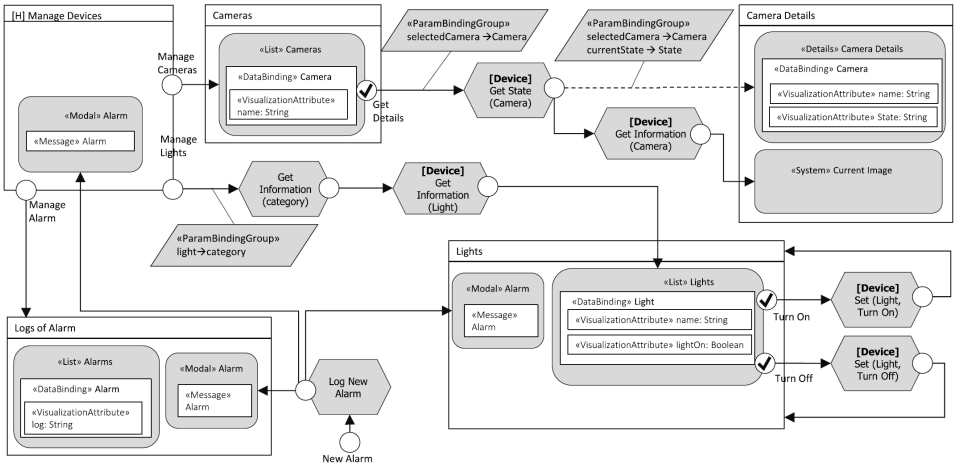


Figure 4.7: Example of a user interaction model of an application from Brambilla et al. [33]

depicted in Figure 4.8. The framework provides three modelling languages for abstracting the complexity of IoT applications and offers automation techniques for different development phases. The latter includes code generation, task mapping, and linking techniques. According to the conducted evaluation, this methodology improves the productivity of the stakeholders of IoT applications and reduces development effort. In contrast to most of the previously described approaches, the approach supports the entire life cycle phases of the application development process.

Likewise, Nguyen et al. [159] also focussed on sensor networks and introduce FRASAD, a FRAmework for Sensor Application Development which implements an MDA approach to manage the complexity of IoT applications and aiming to improve their reusability, flexibility and maintainability. The authors introduce a node-centric, multi-layered software architecture which hides the low-level details and provides a higher level of abstraction. At the highest abstraction level, a rule-based programming model and a domain-specific language (DSL) are used to describe the applications. In order to facilitate the design, implementation, testing and optimising of the IoT applications, a GUI, code generation components and supporting tools have been included. Building upon their previous work [160], the authors created this framework to demonstrate and evaluate their solution. Two case studies illustrated that FRASAD enables a fast way to develop IoT applications by reducing the cost of dealing with heterogeneity of the devices and complexity.

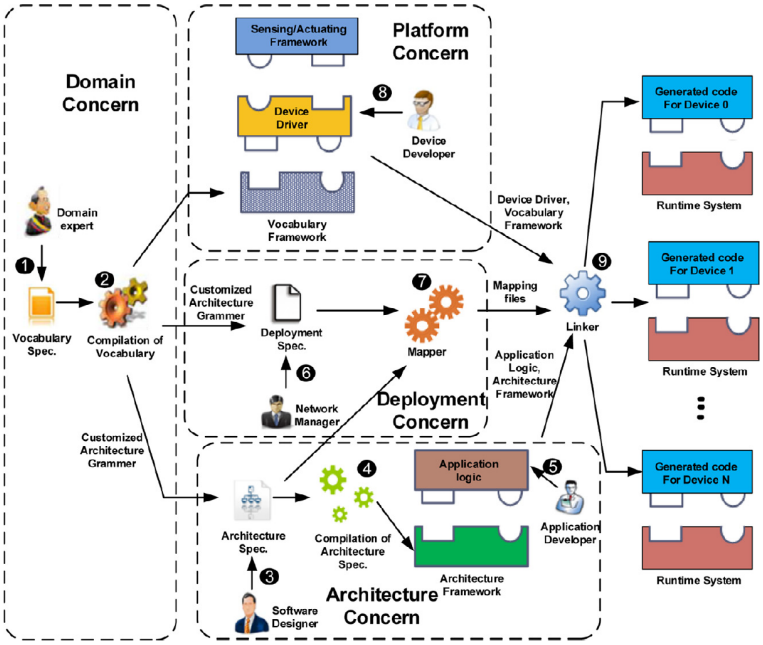


Fig. 3. IoT application development: overall process.

Figure 4.8: Conceptual framework for IoT application development [162]

While the previously described systems sound promising, they all omit the support for the run-time adaptation of the IoT system. Focussing on the adaptivity of IoT systems, Hussein et al. [108] presented a model-driven approach to facilitate the modelling and development of adaptive IoT systems. By following the SysML4IoT profile presented by Costa et al. [56], the design model is created and a publish/subscribe pattern [118] has been adopted to specify environment information. Further, a system management component [191] has been introduced to support the adaptive behaviour. In order to model this runtime adaptive behaviour the state machine approach [109] is adopted. Depending on the design model, an IoT platform model is generated and based on this model some Java code is generated and deployed to the smart environment.

Ciccozzi and Spalazzese [79] also propose a model-driven adaptive approach for IoT applications. The authors introduce MDE4IoT, a framework supporting the modelling of *things* and the self-adaptation of Emergent Configurations (ECs) of connected IoT solutions. Emergent Configuration has been defined by the authors as "*a set of things with their functionalities and services that connect and cooperate temporarily to achieve a goal*". An interesting characteristic to notice from the MDE4IoT framework is that the physical devices can be represented at different

granularity levels. A physical device can be represented either as a black box with a set of available features or as a complex device composed of smaller pieces, such as sensors, actuators and processing units. The granularity level depends on the purpose of the models and their intended use, and the capabilities of the involved model transformations.

While previously mentioned systems do not provide solutions for end users, Johnsson and Magnusson [116] introduced their graphical editor for end users that follows an inverted GUI development approach. The approach focusses on presenting functionality as graphical components in a GUI. In contrast to the classic workflow of conventional graphical editors where users usually first build their graphical interface and then attach functionality to it, the authors' editor lets users first choose the functionality and then attach a GUI component to it from a list of suggestions. Johnsson and Magnusson follow a model-based approach by presenting a language to describe the abstract user interfaces, introducing the editor for concrete user interfaces and implemented interpreters to render the final UIs. In order to facilitate the communication with smart devices, the authors use the PalCom middleware framework [85] allowing to combine services offered by different devices and to send commands to the devices. The graphical editor which is shown in Figure 4.9 therefore only works for creating and editing GUIs for PalCom systems. It has been evaluated and compared to Android Studio by engineering students. The results of the evaluation showed that the authors' editor was more efficient. Further, the scalability has been evaluated by creating mobile applications for advanced home care scenarios. However, the editor has not yet been tested by end users and thus still requires further evaluation.

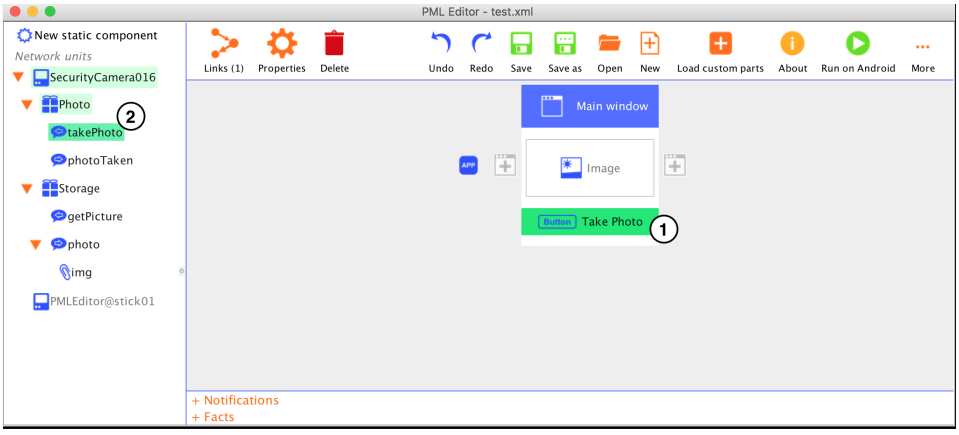


Figure 4.9: Graphical GUI editor building on the PalCom User Interface Markup Language (PML) [116]

Summary and New Requirements

Various systems have been developed to support model-based user interface development in multiple domains. We briefly introduced the ones most relevant to our research, which include MBUID in the adaptive, cross-device and IoT domains. Many solutions in the adaptive and cross-device research domains follows the different levels of abstractions of the CAMELEON Reference framework [40]. Some focus on parts of the UI development process by providing tools that only support certain abstraction levels, while others such as Cedar Studio IDE [4], CAMELEON-RT [14] or MASP [24] provide an approach that supports all abstraction levels. Further, the CRF has sometimes been extended in order to, for instance, support self-adaptive UIs [223] or distributable UIs [211]. In order to model IoT environments, a couple of solutions follow the IoT ARM model, including Costa et al. [56] and Hussein et al. [108]. The WSO2 reference architecture [86] is used as starting point to build some of the IoT systems, as done by Mainetti et al. [135], but most of the remaining IoT systems all provide their own framework and/or model to design IoT solutions.

Overall, a large number of presented solutions focus on a particular research domain (i.e. adaptive UIs, XDI or IoT), with some exceptions such as MASP and the distributed MARIA, which span over multiple research domains including adaptable and distributed UIs. Other exceptions are the MDE4IoT framework [79] and the model-driven approach from Hussein et al. [108], which covers adaptive IoT solutions. However, to the best of our knowledge none of the existing systems covers both the XDI and IoT domains and provides support for XD/distributed and IoT UIs that can be adapted according to the context of use. Therefore, in Section 4.2 and Section 4.3 we will introduce a reference framework and conceptual model that unify the different concepts from frameworks and models of the related work's adaptive, cross-device and IoT solutions. Besides, also note that existing model-based solutions commonly address designers and developers, with less support for *end-user development*, with Johnsson and Magnusson's tool [116] forming an exception.

Based on the presented related work, we now review and finalise our existing requirements. As we have seen with adaptive interfaces, the user interfaces adapt based on a given context of use. In order to express this adaptation to a certain context of use, we will adopt the term *context awareness* as done in the literature. We therefore reformulated our non-functional requirement R6.2 accordingly and thereby enable more than device- and user-based adaptation only. After that, we continue with the introduction of some new non-functional: R4.3 and functional: R2.3 subrequirements.

Requirement 6.2 (R6.2). Support for context awareness A user interface should adapt to the platform (device) on which it is accessed, the user who is using it and the environment in which it is utilised. For example, the interface should adapt to Lucy who is red-green colour blind by replacing the colours red and green with other colours, as shown for the *leaving home* application where her buttons are shown in blue and orange. The buttons are also shown in a larger size, as she uses the application on her phone instead of a smartwatch. Further, the environment could play a role by, for example, making the brightness of the interface lower when the application is used in a darker environment.

Requirement 4.3 (R4.3). Offer extensibility of adaptive behaviour and distribution configurations Related to the *extensibility* core requirement where we already defined extensibility at the level of communication protocols, devices and user interfaces in R4.1, we add extensibility at the level of adaptive behaviour and distribution configurations, which should be extensible as well. As argued by Akiki et al. [6], extensibility is an important concept when it comes to UI development and therefore support extensibility of adaptive behaviour in Cedar Studio, which is also partially supported by UsiComp [89], MARIA [167] and MASP [24]. Extensibility also enables the system to evolve over time instead of having static configurations which cannot be changed. Our solution should therefore provide the possibility to add new adaptive behaviour, and should not be dependent on a fixed configuration of the UI distribution either, but allow run-time UI distribution [145]. For example, a specific UI distribution as we have seen in the *grocery list* application in our use case scenario should be able to be reconfigured at runtime so that if necessary Lucas could distribute the memos from his fridge to his smartphone as well.

Requirement 2.3 (R2.3). Offer fine granularity UI distribution Related to the support for distribution of user interfaces across devices mentioned in R2.2, we specify that this UI distribution should be possible at a fine granularity level. Regarding distribution in the context of CRF's levels of abstractions, Demeure et al. [70], for example, mentioned that any part or whole of the CUI could be distributed. Therefore, graphical containers as well as individual components (e.g. text fields, buttons and other UI elements) are subject to distribution. The authors further describe the notion of *splittable* and *unsplittable* components to indicate whether or not a component could be split across devices. Such distribution at a fine granularity enables a more flexible distribution configuration where not only entire UIs can be distributed, but also smaller parts of a user interface, as supported in the distributed MARIA [136], CAMELEON-RT [14] and Melchior's tool [145].

4.2 The eSPACE Reference Framework

In order to allow end users to build the user interfaces presented in Chapter 2, we now present our multi-layered *end-user Smart PIACE (eSPACE)* reference framework illustrated in Figure 4.10, which unifies cross-device and IoT user interfaces based on our requirements. The framework is further based on our analysis of related work and has mainly been inspired by the CAMELEON reference framework (CRF) [40].

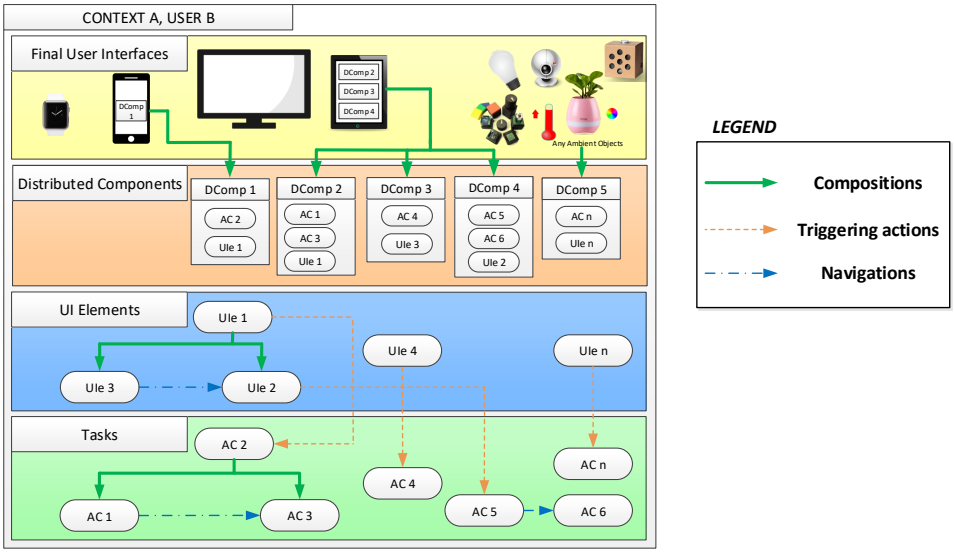


Figure 4.10: eSPACE reference framework

Similar to CRF, we decompose the UI development process into different layers of abstraction to facilitate the UI design process. In contrast to CRF, we do not include transformations between each layer of abstraction but chose a compositional approach [49] which does not require specific mapping techniques for associating elements from one model to their related counterpart(s) in another model. The CRF provides two ways to associate elements of different layers, namely *linking* and *deriving*. With linking associations are made between existing elements of two models, while with deriving elements from one model are constructed based on elements in another model [97]. Compared to the CRF we only provide one way to associate elements from different layers, namely using *relations* which are often in the form of *compositions*, with the *triggering actions* relation forming an exception between the *UIes* and *tasks* layer, as explained later on. The reason behind this choice is that we do not foresee the use of different models per layer of abstraction but rather one model that connects the different layers. The use of one model combining the different layers in order to form the final user interface model, simplifies the task for

designers and developers, as they do not need to learn different types of modelling languages to build their UIs. In addition, designers do not have to pay attention to the mapping of the elements between each of the layers. However, with such an approach models cannot always exist in an independent way (since they are composed out of elements of previous layers) as is the case in the CRF, where models can be created at any layer of abstraction. Note that a last important distinction with the CRF is the FUI layer, which in CRF refers to the source code generated from a CUI, but in the eSPACE reference framework the FUI is part of the model and should still be transformed into an executable program.

Given requirement R5, regarding reuse of UI elements (UIes) and functionality, we aim for a loosely coupled interaction between the UI elements and their functionality or tasks. This is achieved by putting the tasks that a UI should perform and the UI elements in separate layers. By doing so, a UI can be reused for multiple tasks and a task can be coupled with different UI elements depending on the users' preferences. The layer containing the tasks is called the *tasks* layer. We represent tasks as active components (ACs), a concept that has been introduced by Signer and Norrie [205]. Each AC represents a piece of program code that executes a certain action, such as *turning on the light*. A formal definition of an active component is given by Signer and Norrie [204]:

Definition 4.1 – Active Component

"*The concept of active components is used to link to pieces of program code that may either be applications in their own right or bridges to existing applications [204].*"

The advantage of using ACs is that one can trigger some application logic by simply linking to an active component from a UI element or another AC. More importantly, an active component does not have to implement the application logic itself but can also act as a proxy for some third-party applications as well as physical objects (e.g. a light). For example, a number of dedicated active components have been implemented in the PaperPoint presentation tool in order to call some functionality of Microsoft PowerPoint [203]. Another example is the Lost Cosmonaut installation, where the interaction with RFID tagged physical objects would trigger some changes in the ambient light setting as well as the ambient sounds [219]. ACs can be linked to each other rather than to a UI element in order to define and execute more complex interactions. For example, an AC checking the scratching sensor can be linked to an AC that sends notifications, so that Lucas gets notified when the cats might be scratching his couch. In this example we also show that ACs can capture the functionality of both a sensor and an actuator. In order to easily promote the reuse

of such linked ACs or complex tasks, a new AC can be created which is composed of other ACs (e.g. the `scratching sensor` AC and the `notifier` AC can form a `scratching notifier` AC), as illustrated in Figure 4.10 where `AC 2` is composed of `AC 1` and `AC 3`. If Lucas now wants to reuse this new `scratching notifier` AC in other applications, he can just refer to it as `AC 2`. The notifier AC is a generic notifier that can be reused in various applications with different configurations. Another more complex example is the combination of a `touch` AC and a `motion` AC to create the `touch-and-throw` AC, which is used by the couple in our use case scenario to distribute images from their phone to the TV in the presented scenario. Figure 4.11 illustrates how ACs and UIes can be linked with each other in order to create a part of the *cats* application. The figure shows that navigating to the gallery view is possible be using the `gallery button` UIe and that the `touch-and-throw` AC can be used to show content from the `gallery` UIe on TV. Since the `touch-and-throw` AC runs on the smartphone we added a link that represents ownership between this AC and the phone and did the same for the `show` AC which belong to the TV.

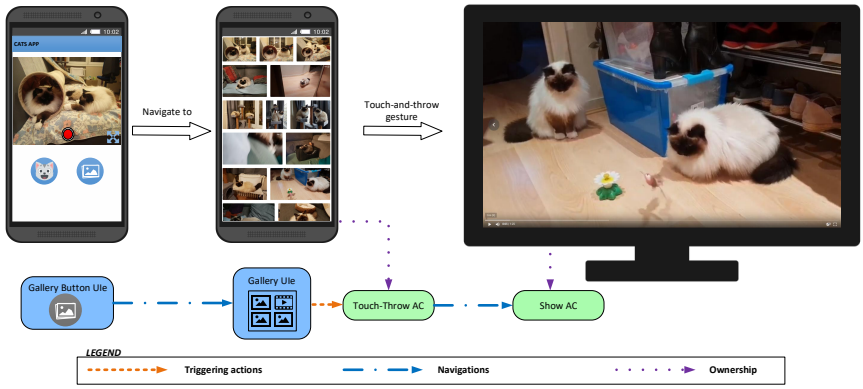


Figure 4.11: Illustration of an interaction created by linking ACs and UI elements

Note that active components can be compared to the tasks in the task models of CRF, but they could also be compared to *services* described by the IoT ARM domain model since services expose the functionality of either network resources or smart devices and *things* resources. In the same way, ACs can be holding functionality of both third-party services and smart technologies. The term services is also used by Johnson and Magnusson [116], who define a clear separation between UI elements and services in their system.

The next layer is the *user interface elements* layer which contains the UI elements. A UI element can, for example, be a text field, a button, a video component or even a combination of different UI elements. These UI elements could be described with a UIDL, in our case we plan on using HTML. As suggested by Luyten et al. [134],

HTML is one of the most suitable languages to create UIs that can be distributed. However, in theory any language can be used to describe UIes. Note that a UIe can also represent a physical UI, such as a light switch, a physical button, slider and so on. Similar to the previous layer, different UI elements can be linked together to form more complex UIs. One can use links to navigate from one UI element to another. This can be used to navigate between different UI views, such as in the *cats* application where when Lucas presses the `gallery` button, a link is followed to the `gallery view`, as shown in Figure 4.11. Links can also be used to arrange UI elements in a certain view. For example, the video player of the *cats* application could be created by linking a *video frame* UI element together with a *button* UI element. Note that, to add functionality to the video player, the *video frame* UI element should be linked with some AC with as functionality *showing the cats video stream* and the *button* UI element should be linked with an AC that can capture images or videos of the video stream. In order to easily reuse such a UI composition, a new UI element can be designed that is made out of a *video frame* UIe and *button* UIe positioned in a certain manner. An example composite UI is shown with `UIe 1` that is composed of `UIe 2` and `UIe 3` in Figure 4.10. Last but not least, links can be used to give functionality to UI elements by linking them to ACs, as explained earlier. For example, linking a *button* UIe to an AC that holds code to turn on the light, will give the button the ability to act as a light switch. While most UIes will be linked to ACs, some UIes do not need to be linked to any functionality. A typical example of such UIes are labels. Note that this *UIe* layer as well as the *tasks* layer are the only two layers from which a designer can start constructing models, as they are not composed out of elements from other layers.

The third layer, is the *distributed components* layer which consisting of different *Distributed Components* (DComps). These DComps consists of compositions created by linking ACs and UI elements with each other. Each DComp is composed of at least one AC or one UI element. It has the property of being distributable in real-time and therefore holds all the information needed to distribute and synchronise its components. We have seen from related work that such information can, for instance, include whether or not the UIe(s) of the DComp are distributable and if they have a distributable state as seen in Tesoriero et al. [211]. Each composition described in the two previous layers will be part of a DComp. For example, the *cats* application consists of many DComps, one contains a *video frame* UI that is combined with a *show video stream* AC. This DComp also contains the *record/capture* button that is linked to some ACs which allows the user to take pictures and record a video. Given its "distribution" properties, the DComp can be distributed to another device, such as a smartphone or TV in order to see the video stream from there. Distribution can happen at different granularity levels. For example, as described in Chapter 3, for Lucas' *grocery list* application, he has chosen to distribute the top part of the fridge

UI (containing only the list of groceries) to his smartphone to make it accessible via his phone. DComps can also represent simple buttons for triggering the light, TV, power plugs and so on. Furthermore, DComps can also be composed uniquely of ACs without any UIes. For example, a *scratching sensor* AC linked to a *notifier* AC. These DComps will not be visible, since they do not contain any UI elements. However, they might, for example, be visible in the form of rules in the system, since we have seen that rules are a popular metaphor among existing IoT solutions. In this case the rule would be "if motion detected by the scratching sensor then trigger a notification to Lucas". The DComps are still device independent, which means that they are not tied to any device at this stage and therefore their look and feel is not yet defined.

The last layer is the *final user interfaces* layer which contains the final user interfaces. Once a DComp is assigned to a device it will get its look and feel depending on the device and become part of the Final User Interface (FUI). The FUI is always associated to one device, but it can be composed of multiple DComps that can be arranged according to a certain layout. For example, the *leaving home* application contains three simple DComps, one to control the lights, one to switch the TV on/off and one to turn on/off the electric iron. Additionally, an application can contain multiple FUIs, such as Lucas' *grocery list* application which contains a FUI for his smartphone and one for his smart fridge. Finally, a FUI can change depending on the context of use. If Lucy would use Lucas' smartwatch, the buttons of the *leaving home* application would adapt to Lucy's colour preferences. Note that, as indicated on top of Figure 4.10, the context of use can have an influence at different layers of our framework. For example, in the *morning routine* application the lights of the living room will start blinking to signal Lucas that he should leave in order to catch his train; so they blink depending on the expected arrival time of his train, which might change daily depending on delays. In this case context is affecting the AC layer since an AC is triggered depending on a certain context. The FUI layer, can be compared to the CUI layer of the CAMELEON reference framework since it is the last layer before transforming the models into executable code.

In this section we have introduced a framework allowing cross-device and IoT interaction depending on the context of use. Our framework supports the *reusability* (R5) of its different components (i.e. `ACs`, `UIes`, `DComps` and `FUIs`) in multiple applications, by either reusing them as they are or by linking them together to form new reusable components. We further illustrated the *flexibility* and *extendibility* (R4) of this approach by allowing these components to be linked together to navigate between them and create new compositions (allowing new adaptation or distribution rules to be created), which is not supported in many existing systems. With a basic set of components created by developers, end users could configure and link the existing components to create new and sometimes unexpected functionality. By following such an authoring

rather than programming approach, the existing set of components can be extended over time by end users. However, it should be noted that appropriate abstractions need to be used to let end users manipulate the components described in our eSPACE reference framework, since we cannot expect them to learn a modelling language. We discussed in Chapter 2 some of the metaphors used in related work to abstract XDI and IoT concepts and will investigate which metaphors to use according to the end users' mental models of XDI and IoT interactions in Chapter 5.

4.3 The eSPACE Conceptual Model

Based on the concepts introduced in the previous sections, we developed our eSPACE conceptual model for unified customised cross-device and IoT user interfaces which can adapt to context of use. The conceptual model is based on the Resource-Selector-Link (RSL) hypermedia metamodel [203] and its concept of Active Components (ACs) [205]. We chose a hypermedia model because our UI development approach described by the eSPACE framework clearly revolves around linking components. Further, by using a hypermedia solution we also profit from concepts introduced by hypertext visionaries such as Douglas Engelbart [77] or Ted Nelson [158]. One of these concepts that is of importance for us is the transclusion introduced by Nelson [158], enabling the reuse of data without duplication. In addition, Engelbart was the first person to demonstrate collaborative work based on a hypertext-enabled real-time editor in "The Mother of All Demos" [77]. Nowadays, similar ideas of real-time editing can for example be found in Google Docs. We have specifically chosen the RSL hypermedia metamodel due to the fact that, in contrast to other hypermedia models, it is not tied to any specific domain or application, as explained in [203]. This unifying RSL hypermedia metamodel comes with attractive properties, such as support for the loose coupling of resources via different types of links, which are treated as first-class citizens. RSL also offers features for user and context management. Finally, it provides extensibility for domain-specific requirements through concept specialisation in the metamodel, which we will use for our domain-specific model that integrates the concepts presented in our eSPACE reference framework and satisfies the requirements presented earlier in this chapter and in Chapter 2 and Chapter 3.

Last but not least, it is import to note that our research lab has been doing research on RSL and thus has expertise with this metamodel. While an implementation of this model has been created in the form of iServer [201], our lab recently implemented a new version offering some benefits as described in Section 4.5.

4.3.1 The RSL Metamodel

The main components of the RSL hypermedia metamodel are the `Resource`, `Selector`, and `Link` entities, which are subtypes of the `Entity` entity type as shown in Figure 4.12. Note that, we choose the ORM modelling language [101] for describing the models, but the original RSL metamodel was based on the OM data model [203]. Given that `Resource`, `Selector`, and `Link` entities share the same parent `Entity`, all three share the same functionality. RSL entities can, for example, have multiple `Properties`, which are key-value pairs that can be used to store application-specific (meta)data. In our case this can, for instance, be used to store the size property of a UI element. A `Resource` is the simplest type of entity that is used to represent entire information units. Since a `Resource` is an abstract concept, it needs to be subtyped for supporting a variety of different resource types such as text, images or videos. Next to resources, RSL introduces `Selectors` that can be used to refer to a piece of a resource and again has to be subtyped to support concrete types of media. A `video` selector can, for instance, be used to address a specific fragment of a video. A selector can only refer to one resource, while a resource can have more than one selector, as shown by the `referred by/refers to` fact type.

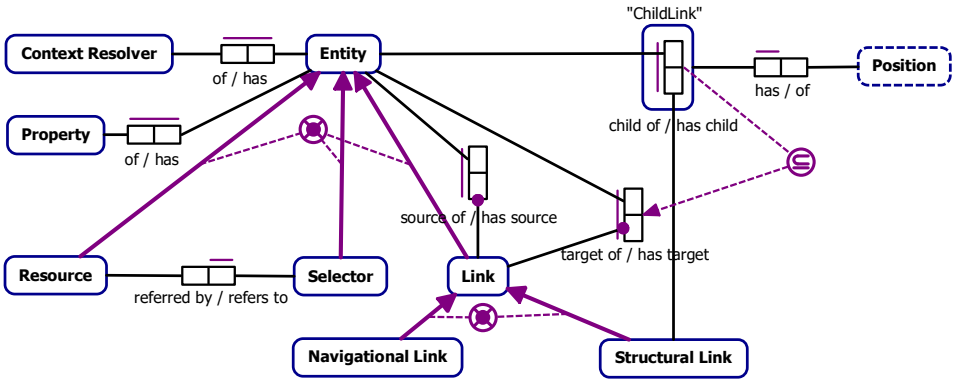


Figure 4.12: Original RSL metamodel in ORM notation [203]

The `Link` entity type is used to represent associations between entities. Since `Links` are subtypes of `Entities`, they can have other links as sources or targets. RSL links can have multiple sources and targets. Further all links are bidirectional. The latter means that it is not only possible to find all link targets for a given source but also to get all link sources given a certain target entity. The RSL core model distinguishes two link subtypes, the `Structural Links` and `Navigational Links`. By using `Structural Links` one can create structures to aggregate content. For example, different UI element resources can be combined using such structural links to form a bigger UI element resource that is structured in a certain way. The targets of

`Structural Links` are ordered, as shown by the objectified fact type `child of/has child` that has a `position` value type, which allows an order in a structure. If we for instance take this dissertation as entity, it could be encoded as a structure containing multiple ordered target links to chapter resources using the order of appearance. The `Navigational Links` are used to describe navigational relationships between entities. They can also be used to navigate to parts of a resource (using selectors) and structures. Other `Link` subtypes can easily be created if other types of links are required.

In addition, RSL offers functionality for the context-dependent handling of entities through the use of `Context Resolvers`. Each entity can be associated with context resolvers that act as gateways determining whether an entity is available or not given a certain context. Context resolvers can also be used on links, to set conditions on when a link needs to be followed. For example, a structural link with multiple targets can have different accessible and inaccessible target entities depending on a given context. If this link would refer to this dissertation, one could change the visibility of certain chapters depending on the person who reads it.

This leads us to the final concepts of RSL, which are related to user management and shown in Figure 4.13. A `User` can either be an individual or a group. Users can have preferences and can be granted or denied access to specific entities. This allow access management combining access rights on individuals and groups, such as "allow this group but without these users". Finally, an entity is created by exactly one individual user, who has full control over its content. A complete description of the RSL metamodel can be found in [203].

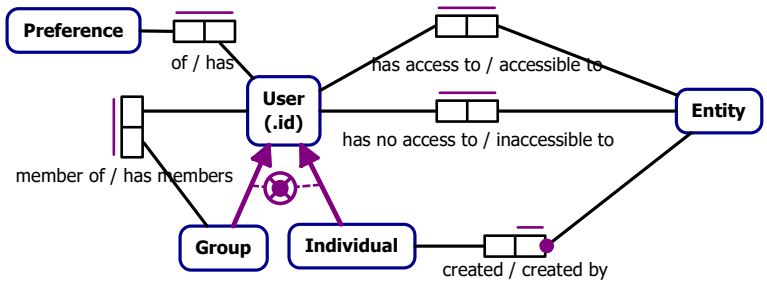


Figure 4.13: Original RSL user model in ORM notation [203]

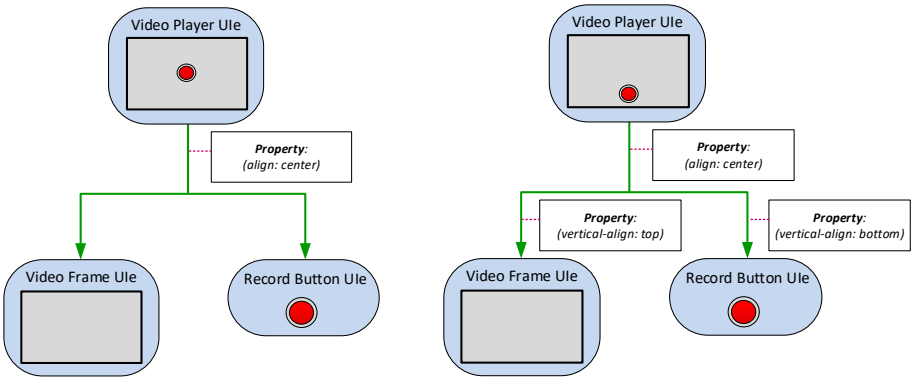
4.3.2 RSL Extensions

While the core RSL metamodel already provides us with a lot of concepts that we need in order to fulfil our requirements and support the components described in our reference framework, there is still some missing functionality. Therefore, we will

make a few small extensions to the original RSL metamodel. We, for example, argue that explicit sharing of the entities with other users is something we need in order to make our authoring tool "futureproof" and to comply to our third requirement related to shareability. Even though users might adapt existing applications based on their individual preferences, as Lucy did with Lucas' applications, it might of course be interesting to see whether some general "ideal" interaction patterns could evolve over time as it can be seen with evolutionary processes in other domains (e.g. the evolution of language). While it is out of the scope of this dissertation to perform these long term studies, we would like to do a first step in this direction by offering the necessary technical support in order to share interaction components with other users. Another extension that we provide is the possibility to add properties on parts of a link, such as specific sources or targets.

Source- and Target-specific Link Properties

The original RSL metamodel allows entities to have a set of properties, hence links being a subtype of an entity also can be associated with properties. Since we want to be able to reuse the different components introduced in our reference framework, we will need to associate some properties of a component to its structural link rather than to the component itself. Further, in some cases certain properties should specifically be assigned to a certain target or source of the composition.



(a) Properties on link using the original RSL metamodel

(b) Target-specific properties on link using the extension to the RSL metamodel

Figure 4.14: Support for source- and target-specific link properties example

Take, for example, a `video player` UI element that is composed out of different UI elements placed in a certain way using properties on a structural link. This structure might have some common properties for its targets, as shown in Figure 4.14a, but it will eventually need target-specific properties to form the desired UI structure, as

depicted in Figure 4.14b. However, with the current metamodel all parts of a link have the same properties, there can be no properties defined over the individual sources or targets of a link in the context of their link membership.

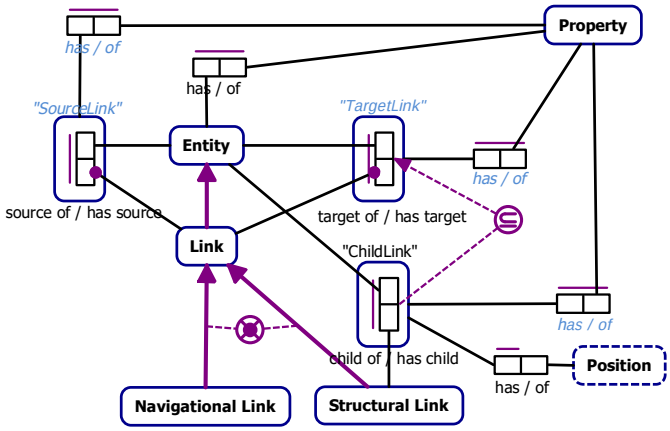


Figure 4.15: Extended RSL properties

In order to allow source- and target-specific link properties, we objectified the `source of/has source` and `target of/has target` fact types between the `Entity` and `Link` entity types, and added a `has/of` fact type representing the relationship between a `SourceLink` or `TargetLink` and a `Property`. The same relationship has been added between `ChildLink` and `Property`. Figure 4.15 shows this extension to the RSL metamodel. Properties can be reused as shown in Figure 4.14 and can now be used in the two ways demonstrated in this figure for source as well as target/child links of both navigational and structural links.

User-Entity Sharing

As explained earlier in this section, we need support for explicit sharing of content across users of the system. While reuse and collaboration is already present in the RSL metamodel, we need to go beyond user access management and ownership. As is, only the creator of an entity can decide to make this entity accessible or inaccessible to certain users. In some cases, one might want to grant more privileges of their created entity to other users. For example, if Lucas has created an application that acts as a remote controller to their TV and shared it with Lucy, Lucy might want to share this application as well with her guests at home when Lucas is not present.

More importantly, we also want to make a distinction between *giving access* to an entity and *sharing* an entity with other users. Imagine the entity being the *leaving home* application that has control of Lucas' smart lock (of his door), TV, electric iron and

lights. Lucas wants to make this application *accessible* to the public since he thinks that others might want something similar. In this case, the control over Lucas' smart devices and *things* should not be made available to the public. On the other hand, when Lucas *shares* this application with Lucy, he wants to preserve the control over the different devices and *things*. This can be used to ensure some privacy, since no sensitive information will be available when users make entities accessible for each other.

For this reason, we added the sharing relationship to the RSL metamodel, as shown in Figure 4.16. The new ternary relationship allows a user to share many entities with many users. Note that if a user A shares an entity with user B this entity should also be accessible to user B. Further sharing is not possible if the user does not have access to the entity.

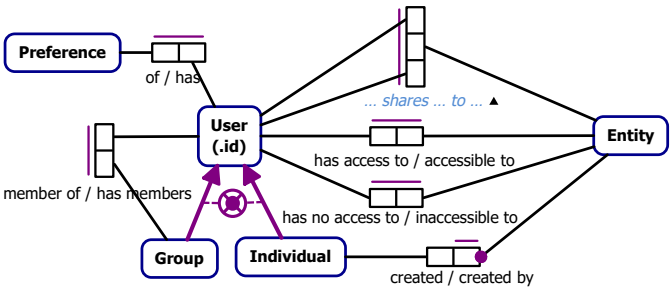


Figure 4.16: Extended RSL user model

4.3.3 Domain-specific Conceptual Model

Based on the concepts we have seen in related work, the reference framework and requirements, we present our domain-specific conceptual model as domain-specific extension of the RSL metamodel. Our model is shown in Figure 4.17, where the core RSL components are shown in black and our extensions are in italics and coloured in blue.

In our domain-specific extension we differentiate between the following types of resources, which are the `Application`, `Final User Interface`, `DComp`, `UI Element`, `Active Component`, `Parameter`, `Property Set`, `Layout`, `Device`, `Service`, `Context` and `Physical Object` resource. Note that we use the term *Physical Object* to refer to *things* as well as physical user interfaces (PUIs). As mentioned in the beginning of Chapter 2, IoT objects can come with new PUIs, such as the Nest Learning Thermostat. Conforming to our reference framework, an `Application` consists of one or more `FUIs`, which in turn are composed of at least one `DComp` that contains at least one `AC` or a `UI Element`. Further, an `AC` can be

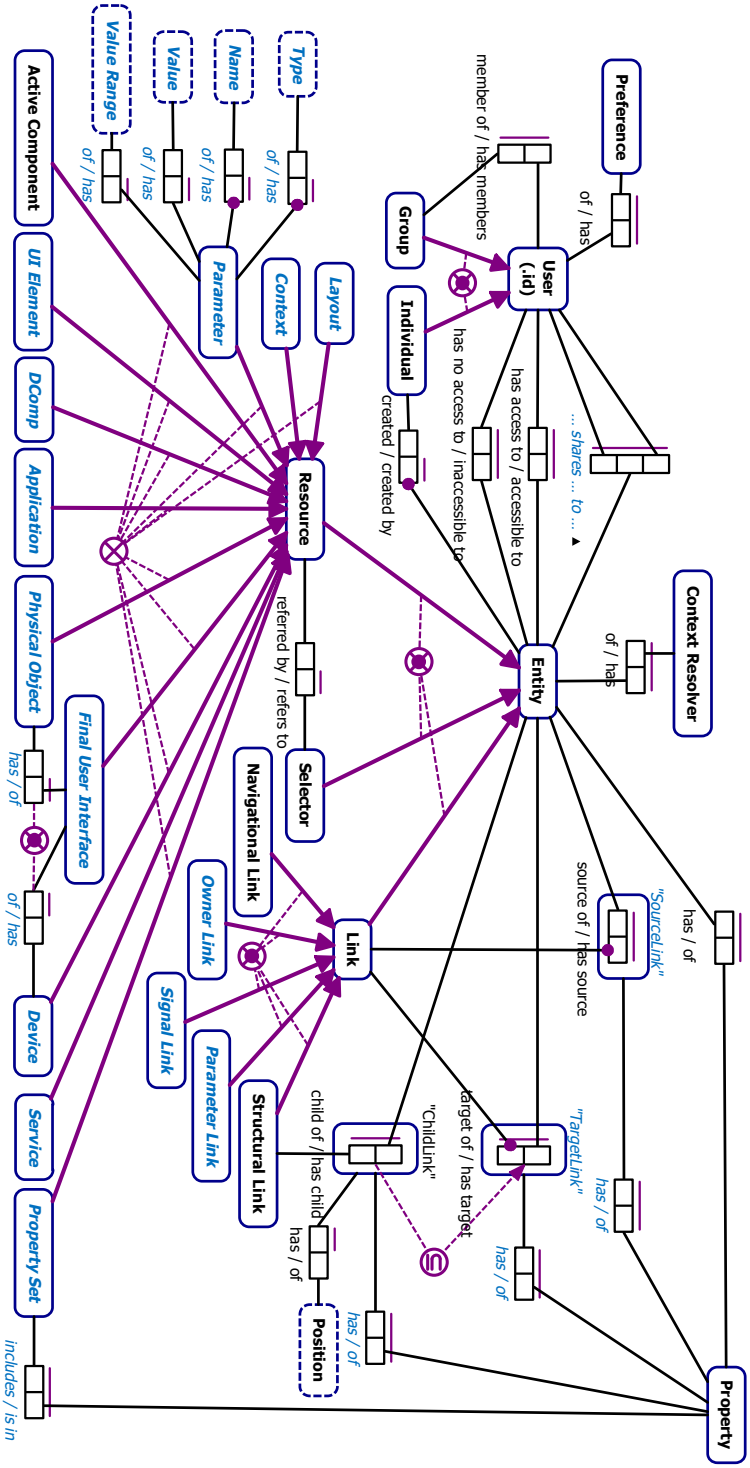


Figure 4.17: Conceptual model based on RSL

composed out of other ACs, the same is true for an UI which can be composed out of other UIs. These constraints on compositions are not visible in our ORM scheme since these associations are instantiated using Structural Links which are used to build all kinds of structures. Instead we provide an overview of the constraints in Table 4.1.

Entity Type	Allowed Targets of the Structural Link	Min	Max
Application	FUI	0	*
FUI	DComp, Layout, Property Set	0	*
DComp	UIe, AC	1	*
UIe	UIe	0	*
AC	AC	0	*

Table 4.1: Contraints in compositions according to our eSPACE framework

Next to that, an Application usually runs on one or multiple Device and/or on Physical Objects. This is inferred by the FUIs the application is composed of, since the FUI is associated to a Device or a Physical Object, as shown by the two fact types and the exclusive-or constraint between these fact types. A Service resource has been added to include third-party functionality. Take, for instance, a *weather* service which offer functionality related to the weather. In order to provide this functionality, the service will be linked to a couple of ACs. An example AC could be a *show weather of 'city'* AC, where city is a parameter, if users want to see the weather of Brussels, that parameter will be set to *Brussels*. While RSL's Context Resolvers are used as gateways to let something happen given a certain context, we still needed a more powerful concept of a "context" in the authoring tool. Therefore, we added the Context resource, which is used to represent the notion of context to the users. The context could be related to user, time, location or other environmental properties. Using this Context resource one can create an interaction rule involving a context as trigger. For example, "*if it is 10:00 p.m., then turn off the light*" can be modelled by linking a time context resource with a turn on AC of the *light* physical object via a signal link. On this link a Context Resolver is associated to follow the link only when it actually is the correct time. The context resource will provide functionality to monitor time, while the context resolver—with metadata "10:00 p.m."—will make sure that the link is only followed at the right time, which will result in turning off the light at 10 p.m. Note that our use of context resolvers can be compared to special kinds of Properties. A similar approach has been provided by Brambilla et al. [33] with their extended IFML notation for the IoT domain, as shown in Figure 4.7.

Further, the Layout resource will be used to set the layout style or structure. This kind of pre-defined structures can be used to align multiple user interface elements in a certain manner, such as in a grid or list layout. Next to the ordering of elements,

style data can, for example, be kept as `Properties`. In order to easily reuse a set of such style properties, we introduced the `Property Set` resource, which allow to group properties into a `Property Set` that can be referred to when these properties need to be applied.

The `Parameter` resource is used to represent the input and output parameters of `ACs`, `UI elements` and `Context` resources. Each `Parameter` contains a name, type, value range and value. By modelling a parameter as a separate resource, it makes our model more flexible and different parameters can be reused by various `ACs`, `UI elements` and `Context` resources as explained later in this section. This is also the reason why we needed *context* as an entity.

While we use the existing `Navigational Link` to, for example, navigate between UI views of an application and `Structural Links` to create different kinds of structures that will ultimately form an application, our domain-specific extension further differentiate between the following types of links. The first one is the `Signal Link` which is used for triggering interaction, so they could have as source a `UI Element` or a `Context` resource and as target an `Active Component`. For example, a `button` UI element can be linked to a `notification` AC with a `Signal Link` to trigger a notification whenever this button is pressed. The second type of link is the `Owner Link` which indicates an ownership relationship. A `Device`, `Physical Object` and `Service` own a range of different ACs. A concrete example could be a smart light bulb that has ACs such as the *change status* AC, *blink* AC and *change colour* AC. The last link type we added is the `Parameter Link` which is used to link `Parameters` to `ACs`, `UI elements` or `Context` resources. As explained before, we want the parameters to be reusable and therefore we defined this special type of link. The `Parameter Link` will hold the value of the parameter in a given structure. A `DComp` can, for instance, be made of a `UI Element` and an `AC` which both hold certain parameters with a given value. This value will be stored in the properties of a `Parameter Link` together with the id of the `DComp`.

Figure 4.18 shows an example that summarises the use of the various types of links that we defined. The *blinds* physical object owns different ACs. One of these ACs is used in a `DComp` that, in this case, could be seen as a "rule" in an application. The `DComp` provides the following functionality, "*open the blinds at a 7:30 a.m.*", which is modelled using a *time* context resource linked through a `Signal Link` to the *change status* AC of the *blinds* physical object. The `Parameter Link` between the `Time` context resource and its `time` parameter holds the value of this parameter (7:30 a.m.) and the `DComp`'s id. The id of the `DComp` is then used to match the right value for each composition. Finally, the `Signal Link` is associated with a `Context Resolver` to make sure that this link is only followed at 7:30 a.m.

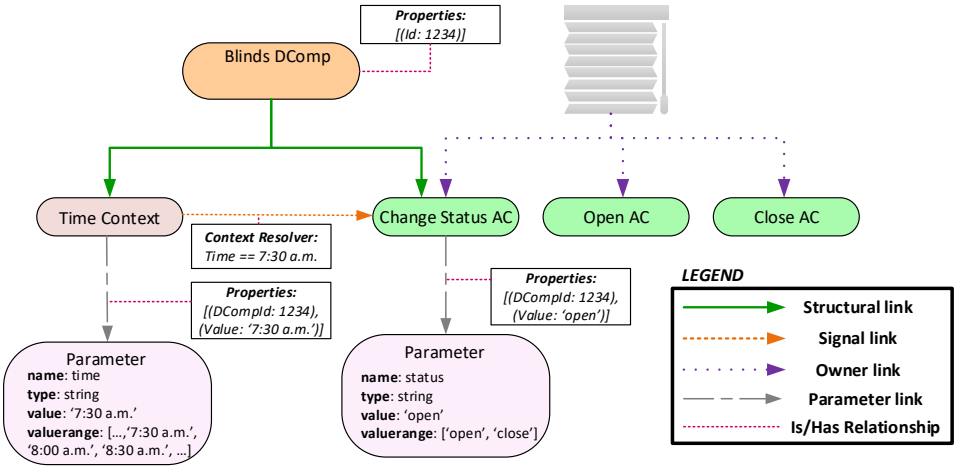


Figure 4.18: Example of the use of the different link types

4.4 Model Functionality and Discussion

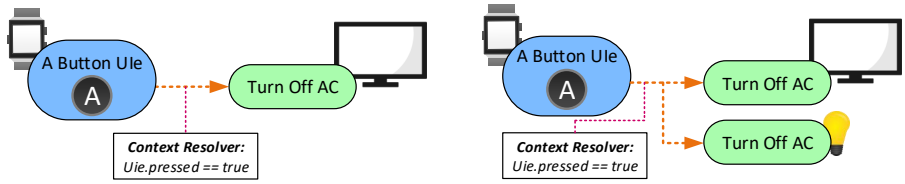
In this section we describe how our model satisfies all the requirements we described earlier on in this chapter as well as in Chapter 2 and Chapter 3. Therefore we briefly explain how each requirement is addressed using our model and some examples. Further, note that all the described model functionality has been implemented, this implementation is detailed in the next section.

Requirement 1 (R1). Provide an overview of the smart technologies, environments and applications By defining the physical objects, devices and applications as specific types of resources and given the built-in user management of RSL, one can retrieve all physical UIs, *things*, devices and applications of a given user. It is then up to the front-end authoring tool to provide an appropriate overview of this data.

Requirement 2 (R2). Interaction support Actions or interactions are reflected in our model in the form of `Active Components`. Each AC can provide different kind of interactions going from simple commands to data transfer between smart devices and *things*.

Requirement 2.1 (R2.1). Support for interaction across multiple smart technologies `Active Components` are either owned by a `Service`, a `Device` or `Physical Object`. In order to trigger an action on a device, we introduced the `Signal Link`, a subtype of the generic RSL `Link`. Actions can be triggered based on context, UI interaction or other actions (ACs). Therefore, in order to make two devices interact with each other, a `Signal Link` can be defined between a `UI element` on one device and an `AC`

on another device as shown in Figure 4.19a. This figure shows that whenever button 'A' is pressed on the smartwatch, the TV should turn off. Since RSL provides multi-target links, we can also use such a link to trigger multiple actions simultaneously as shown in Figure 4.19b, where the button triggers the light to turn off as well. Both figures are used to illustrated how to use the `Signal Link` and we therefore omitted the other links connecting the AC and the UI to their respective devices to keep it clear and simple. Note that by using RSL `Selectors` one can also trigger actions only on certain parts of a resource. For example, a peculiar interface could be created where pressing the upper part of an *image* UI element triggers an action on the tablet while pressing the lower part of the *image* UI element triggers an action on the phone. This can be achieved by selecting the upper part of the *image* UI element using an *image* selector, linking this with an `AC` from the tablet using a `Signal Link` and repeating this for the lower part of the *image* UI element.



(a) Interaction between smartwatch and TV (b) Interaction between smartwatch, TV and smart light bulb

Figure 4.19: Support for interaction across smart technologies by using a signal link

Requirement 2.2 (R2.2). Support for creation, customisation and distribution of cross-device and IoT user interfaces In order to fulfil this requirement, we first described the UI development process with the eSPACE framework, which lay the first stones towards enabling support for the creation, customisation and distribution of cross-device and IoT user interfaces. As a next step we made the concepts introduced in our framework more concrete by integrating them into our conceptual model, as described before. Using these concepts we will now demonstrate how a final user interface can be built. Therefore, we will use Lucy's final user interface of the *cats* application shown in Figure 4.20.

Structural links are used to build the structure of the interface as well as glueing all components together. They are, for example, used to create the *video player* UIe that is composed of a *button* UIe and a *video frame* UIe. A new AC has been created using `Structural Links` as well, being the *touch-and-throw* AC that is a combination of a *touch* AC with an *accelerometer* AC. The *touch-and-throw* AC is used to distribute pictures and videos from the *gallery view* (on the phone) to the smart TV. In order to distribute only certain video excerpts to the TV, a *video* selector is used. In this case the selector allows distribution at a higher granularity level.

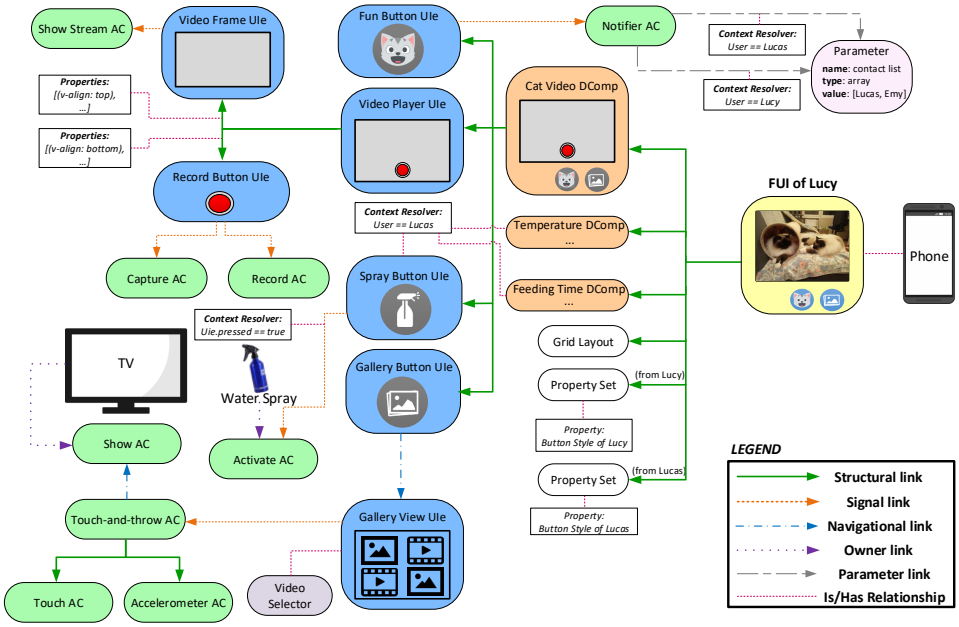


Figure 4.20: Simplified instantiation of our model with the `cats` application FUI

Structural links are further used to structure the `DComps` (portrayed in orange) and the `FUI` (yellow component). Note that the structural link forming the FUI has two `Property Set` targets, one link target is created by Lucas and is not accessible by Lucy and the other is created by Lucy and is not accessible by Lucas. Instead of using access rights, context resolvers can also easily be used to make a resource or link seen only by a specific user. Such a context resolver has been used on the *spray button* UIe, the *temperature* DComp, the *feeding time* DComp and the parameter links of the *notifier* AC. These components are only visible if the context of the context resolver is satisfied. In order to keep the example model more readable, not all details are shown, the parameters of the ACs and UIs are often omitted. We kept the parameter of the *notifier* AC to show an example of how context resolvers can be used to reuse the same parameter but with a different value. As explained before the value of a parameter is kept on the parameter link and in this case will be *[Lucas, Emy]* for Lucy and *[Lucy]* for Lucas. This metadata is however not shown in Figure 4.20.

In Section 4.2 we not only expressed the need to compose ACs and UI elements but also to navigate between them. This is achieved by using the `Navigational Links` already provided by the original RSL metamodel. One can link ACs via navigational links to execute multiple actions after each other, as done between the *touch-and-throw* AC and the *Show* AC. This type of link is also used to navigate between UI views, as illustrated for the *gallery button* UIe and *gallery view* UIe.

Properties are used as already shown earlier in Figure 4.14 and styling properties are grouped into property sets. For Lucy the styling properties would, for example, include that she only wants blue and orange buttons instead of green and red ones.

As explained in the previous requirement, `Signal Links` are used to trigger (inter)action, as depicted by the link between the *spray button* and the *activate* AC of the *water spray* physical object. This link triggers the AC when the button is pressed as shown by the context resolver. Again, note that, for readability purpose, not all context resolvers are shown in the figure. The same is true for `Owner Links` which are only shown for the *TV* and *water spray* but should be present for each AC. In this case all ACs without `Owner Link` are ACs from the *phone* device.

With all the elements described above we ensure the creation, customisation and distribution of user interfaces that can be used for cross-device and IoT interaction. Of course, a next step should be taken in order to enable end users to create such a complex model, therefore a layer of abstraction should be put on top of this model using appropriate metaphors which will be investigated in the next chapter of this dissertation.

Requirement 2.3 (R2.3). Offer fine granularity UI distribution In Section 4.2 we explained that DComps are used to allow UI distribution and synchronisation among devices. By creating a DComp composed of a UI element and an AC, one can distribute at the level of granularity of UI elements. In order to promote even finer granularity of the distribution, selectors can be used as explained in R2.2. An example of finer UI distribution by using selectors is the use of a *UIe* selector, which could distribute parts of a UI element such as the controls of a *media player* UIe.

Requirement 3 (R3). Shareability The original RSL metamodel provides the building blocks for access management. Together with the RSL extension of the sharing relationship, both concepts can be used to provide different kinds of sharing functionality, which are explained in our two subrequirements.

Requirement 3.1 (R3.1). Support for sharing and integration of apps in a central smart apps repository The access management can be used to maintain a central repository with all entities that are accessible to the "public". Figure 4.21 shows how this public repository can be used by a *user Leo* to create his own application reusing two DComps of Lucas' *leaving home* application. In order to reuse these DComps, Leo simply has to create a structural link between his application and the DComps in the public repository, as shown by the green arrow in the figure. He further has to find a device providing the functionality of the two ACs or link the UI elements with other ACs. Remember that an AC basically holds a piece of program code that

executes an action. Whenever an `Owner Link` is created between an AC and a device, a compatibility check should be made to see whether the device can run the program code of the AC.

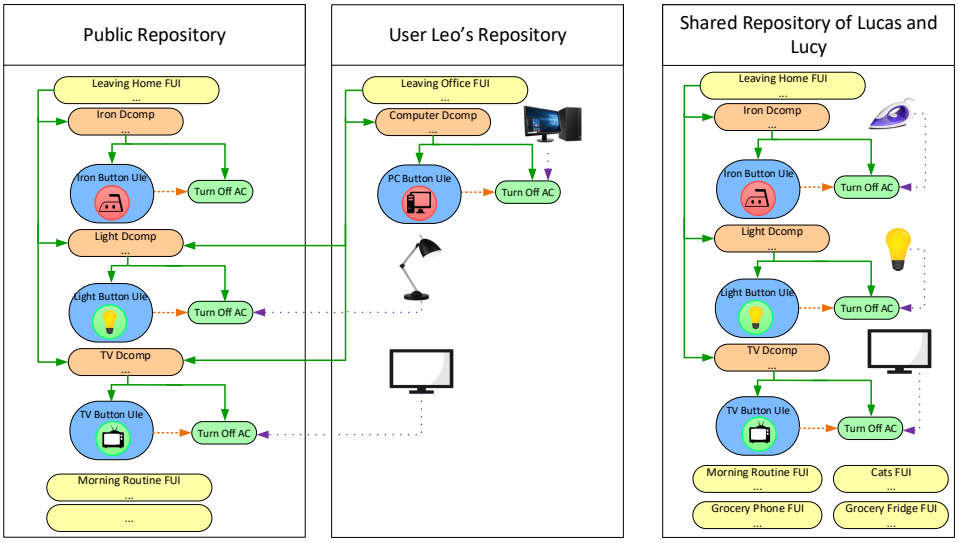


Figure 4.21: Example of a public and a shared repository

Since content is stored at a fine granularity level, users can reuse any part of an existing application. Further, users can also decide to only make public smaller portions of an application ranging from a UI element or DComp to a FUI or an entire Application. Next to this, in order to search the public repository, one can use filters and ask to only show certain entities such as UI elements.

Requirement 3.2 (R3.2). Enable sharing of applications, user interfaces or parts of a user interface with specific users In Section 4.3.2 we explained how we extended the original model to add an explicit sharing relationship between users. In contrast to the previous requirement, when sharing entities with specific users, the `Owner Link` between the AC and the device remains. This type of sharing is particularly interesting for sharing applications between people of the same household as illustrated in Figure 4.21, where we see the shared repository of Lucas and Lucy. Similar to sharing to the public, sharing to specific users can happen at different levels of granularity, which means that specific parts of a user interface can be shared instead of sharing an entire application. The sharing of specific parts of an application can, for instance, be used to make certain appliances accessible for children or guests.

Requirement 4 (R4). Extensibility One of the advantages of a model-based approach is flexibility and extensibility, as our conceptual components can be re-arranged (using links) and extended over time.

Requirement 4.1 (R4.1). Offer extensibility at the level of communication protocols, devices and user interfaces Using our model, any entities can be added at any time by a developer, including support for new devices and user interfaces. Concerning the communication protocol, devices communicate with each other by connecting to the RSL server, which currently supports communication through a RESTful protocol and WebSockets. The RSL server is implemented in such a way that new communication methods can easily be added later on, which will be discussed in the next section. End users should be able to add UIs and devices as well, but this is up to the front-end authoring tool, which will be described in Chapter 6.

Requirement 4.2 (R4.2). Enable the integration of third-party applications In order to support this requirement, we created the `Service` resource which has been introduced earlier in this chapter. The `Service` resource represents third-party applications and owns a certain amount of ACs which represent the (inter)actions that are available by this service. For instance, the *train schedule* service could have a *show scheduled trains* AC with the parameters of a certain station and timespan. New services and thus third-party application access can be added over time which can then be retrieved from the public repository addressed by R3.1.

Requirement 4.3 (R4.3). Offer extensibility of adaptive behaviour and distribution configurations The adaptive behaviour can be added to the model by using context resolvers, context resources and user preferences. Using the context resolvers, the view and/or actions of an application can change given a certain context, as shown in Figure 4.20. The figure shows the use of context resolvers to hide or show parts of the cats application FUI and to change the parameters of the *notifier* AC depending on the user of the application. New context resolvers can be added over time to extend the adaptive behaviour of applications. In order to define adaptive behaviour with as trigger a certain context of use, context resources can be used as illustrated in Figure 4.18. Additionally, user preferences can be used to add adaptations depending on the preferences of a specific user. Note that compared to the CRF, our method to adapt an entire model to context of use requires more "manual" work by the designer, who has to make use of context resolver(s) where necessary. In contrast, the CRF can use model transformation rules which are applied to the entire model and this can be fully automated. Next, distribution configurations can be extended as well. By adding a link between the *touch-and-throw* AC and a *show* AC of another device, one can distribute the images and videos of the *gallery view* to another device. Further

the properties of DComps allow UIes to be distributed to any device at runtime as explained in Section 4.2, which also allows flexible and new distribution configurations. Some extra metadata could also be included in the DComp to specify how to handle the distribution of UI elements to certain devices depending on their characteristics. For example, if one would distribute the *leaving home* UI from Lucy's smartphone to a smartwatch then navigational links should be created between each button so that each button is displayed separately on the screen and so that the user can navigate between the three different views (each containing one button) as done for Lucas' version of the *leaving home* UI. Notice that while we specify this information in the form of metadata of DComps, the CRF uses transformation models or mapping rules to do this.

Requirement 5 (R5). Reusability All concepts introduced in our conceptual model can be reused in the same or different application models. As already shown in Figure 4.20, the same context resolver is reused multiple times to omit certain parts of the FUI for when Lucy is using the cats application. Other conceptual components can be reused in a similar fashion by using links, as shown in Figure 4.21 with Leo's repository.

Requirement 5.1 (R5.1). Support for reuse and combination of existing user interfaces UI elements can be reused multiple times in the same FUI as well as in other FUIs and can be structured in a different way via `structural links`. It is up to the front-end authoring environment to make this functionality available for end users.

Requirement 5.2 (5.2). Support for reuse and combination of existing functionality Similar to previous requirement, ACs can be reused in the same as well as in other applications. Multiple ACs can be combined to create new functionality, such as a *touch-and-throw* AC. Making this functionality available for end users is up to the front-end authoring solution.

Requirement 6 (R6). Portability In order to promote portability, we do not involve devices until the very last step of the UI design process. Only a FUI is linked to a certain device, which will optimise its components for that device. The structure a FUI is made of can be reused by another device, which will have its own FUI tailored for this device.

Requirement 6.1 (R6.1). Offer platform independence As explained above, FUIs are adapted to a certain device and their components can be used on other compatible devices. Checking for compatibility of the FUI's components with a device should be

done using the device's characteristics and making the applications compatible with any platform will depend on the technologies used in the front-end authoring tool.

Requirement 6.2 (R6.2). Support for context awareness The adaptation of applications and UIes to users, platforms and environments can be achieved using context resources and context resolvers, as explained in R4.3. Further, the RSL user preferences can also be used to tailor the UIs of applications to the user that is using them.

Requirement 7 (R7). Support for end-user development The eSPACE conceptual model supports all previous requirements, but whether the functionality mentioned in these requirements is simple to perform for end users is entirely up to the front-end authoring solution, which has to provide appropriate abstractions for end users.

4.5 Implementation

We have used and extended a working RSL-based information system that has been recently developed by our lab in Java 8 [185]. The implementation of this information system demonstrates the viability of the concepts introduced in this chapter in practice. In contrast to previous RSL-based link servers such as iServer [202], the new implementation has made some improvements in terms of reusability, extensibility and maintainability. Since RSL is a common factor in our lab's research, the new version of RSL has been made available on our GitLab together with documentation on how to use it. In this section we shortly explain the new design of RSL as a software framework but more details can be found in [184]. While previous RSL implementations' main issues were unmaintained dependencies and extensibility, the new implementation has been designed to be easily used by our students and other researchers and therefore minimises the complexity of turning a conceptual RSL metamodel into a working application. In order to avoid extensibility issues, the framework is implemented as a set of decoupled modules which can be replaced or extended without the need to refactor the entire codebase. Roels [184] designed the new RSL framework such that it can be used in two ways. It can either be used as a software library or it can be used as an external link service. In the first case, applications can embed the library to make use of the hypermedia functionality provided by RSL. In the second case, the same core RSL library is used, but it is wrapped inside a server component that provides RSL functionality through RESTful or WebSocket interfaces, which is the way we used it. Using this approach, we benefit from better compatibility with application platforms and technologies, and it further allows us to have devices, users and applications to operate on the same content repository [184].

In order to easily add application-specific extensions, the new RSL link server has been implemented as a reusable model-driven information system. This entails that users can load their RSL-based models at runtime and the server will automatically generate the related functionality without the need for adding any code related to hypermedia concepts or persistence (which in our case is the storage of Java objects to the object database). Models are described using a custom JSON-based format, as shown in Listing 4.1 which represents a part of our eSPACE model, more specifically the `Parameter Resource` and `Parameter Link`. Note that the original RSL-based models could not support "list" as a property type. Since we needed this for our eSPACE model, we extended the parser to support this kind of property type.

```
{
  "name": "eSPACE",
  "version": "0.0.1",

  "resources": [
  {
    "name": "ParameterResource",
    "properties": [
          {"name": "type", "type": "string"},
          {"name": "valuerange", "type": "list"},
          {"name": "value", "type": "string"}
    ]
   },
   ...
  ],
  "links": [
  {
    "name": "ParameterLink",
    "sourceRestrictions": {"type": "whitelist", "list": ["ACResource", "
        UIeResource", "ContextResource"]},
    "targetRestrictions": {"type": "whitelist", "list": ["
        ParameterResource"]},
    "properties": [
          {"name": "value", "type": "string"},
          {"name": "dcompId", "type": "int64"}
    ]
   },
   ...
  ]
}
```

Listing 4.1: Parts of the eSPACE model described in JSON format

The JSON model description starts by defining the name of the model followed by its version to keep track of model updates. The remainder of the JSON file specifies the application-specific extensions of the `Resource`, `Selector` and `Link` entities and their properties defined as a set of key/value pairs. Optionally, restrictions can be provided on the sources and targets that a link can have, as shown with the `sourceRestrictions` and `targetRestrictions` whitelist in Listing 4.1. The whitelist specifies the allowed sources and targets for the `ParameterLink`. A blacklist can also be used to specify which sources or targets are not allowed for a certain link type. The general architecture of the RSL library is shown in Figure 4.22 with the components we added to Roels [184] implementation coloured in purple, this includes

all concepts related to user management (see Figure 4.13). The `core classes` module holds all core RSL entities and functionality described by the RSL metamodel, these include but are not limited to the `Resource`, `Selector`, `Link` and `User` classes. A user class has methods to get entities which are accessible, inaccessible or shared by this user. This class further holds the group to which the user might belong to as well as the user's preferences. The original `Entity` class has methods to manage its properties, incoming and outgoing links which is inherited by the `Resource`, `Selector` and `Link` classes. We extended this `Entity` class implementation with methods for user access, ownership and sharing. The `Link` class also has methods to get, remove or modify its sources and targets. Keeping track of the incoming and outgoing links of an `Entity` and the sources and targets of a `Link` allows for efficient bidirectional link traversal, since a link can be followed from its target entity as well as from its source entity. The synchronisation of such bidirectional associations is done automatically and efficiently using the Java Persistence API (JPA), which is also used to persist the data in an ObjectDB[1] object database. The `object persistence` module in Figure 4.22 shows different possibilities for permanent storage of the data. Roels [184] choose an object database mainly for its performance compared to a relational database and due to the fact that a graph database would not treat links as first-class citizens making links to links, for instance, not possible. Since over time one might find a better persistence approach, Roels created an abstract persistence layer that is decoupled and can easily be replaced in the future.

In Listing 4.1, we showed how to represent parts of our eSPACE model using a custom JSON-based format, given this model, the RSL library will use JavaPoet[2] to generate the corresponding Java classes as subtypes of the predefined core RSL classes from which they inherit part of their functionality. For example, for the `Parameter` resource (which is an extension of the resource entity), a `ParameterResource` Java class is generated that inherits the functionality of the `Resource` class and will contain the methods for managing the properties (getters and setters) defined in the JSON model, which are the `type`, `valuerange` and `value` properties. The generated classes are further dynamically compiled and (re)loaded at runtime. In order to interact with instances of the new classes at runtime, high-level methods are provided that use generics and reflection. Such high-level methods include the creation, modification and deletion of instances as well as managing their links and access rights. Note that multiple models can be loaded into the RSL library and can be used side by side. An example of a model used in our lab is the MindXpres model for presentation software [184]. Since all new entities introduced by the models are subtypes of the same three superclasses, all instances from the different models can also be linked with each other and used together in the same application.

[1] `https://www.objectdb.com`
[2] `https://github.com/square/javapoet`

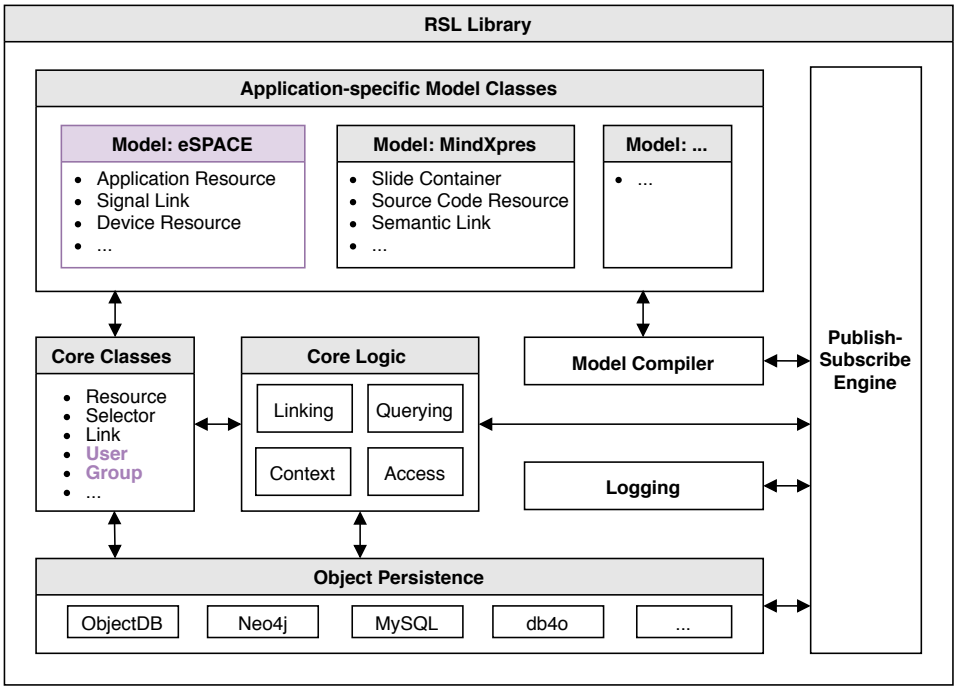


Figure 4.22: Overview of the model-driven RSL link service [185]

The remaining components further provide utility functions related to querying content by crawling the complex graph of linked data. This way content of interest can efficiently be queried based on property values or object types. An example query could be: "get all `Application` resources created by user `Lucas`". Next, there is also the publish-subscribe engine that coordinates various components and manages the flow of events. For example, when an object is created, deleted or persisted, a corresponding event is published to the eventbus in order that other components can react if necessary. Finally, the logging component is used to create detailed logs of these events.

As mentioned earlier, we use the RSL library wrapped inside a server component in order to have a client-server architecture. The server wrapper architecture is depicted in Figure 4.23. It includes the RSL software library from Figure 4.22 and adds extra server functionality on top of it. In order to communicate with the server, an application could use a RESTful interface or a WebSocket depending on the application's needs. Different interfaces are supported by the server and new ones can be added if necessary as the system is built in an extensible manner. More technical details can be found in [184] on how to add new interfaces.

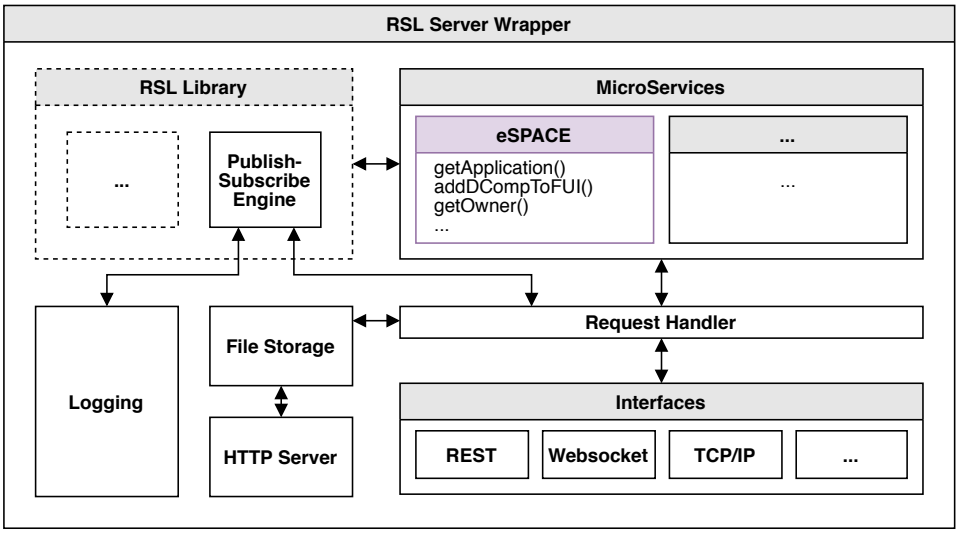


Figure 4.23: Overview of the RSL Server Wrapper [185]

`MicroServices` have been introduced to implement higher-level application-specific functionality on top of the low-level hypermedia functionality provided by the RSL library. We added the `eSPACE` microservice to the server, which provides multiple high-level methods that are used by the eSPACE authoring tool and are related to users and their user-defined applications as well as the applications' structure. By adding these methods as a microservice we also reduce the amount of requests made to the server and thus enhance the performance as well. One of the methods, for instance, adds a `FUI` to a specific `Application` resource. Therefore the method either creates a structural link from application to FUI, or adds the FUI to an existing structural link. While this looks like a simple query it already involves a few requests in order to first verify whether there already is a structural link with the application as source, create this link if it does not yet exist and then set its source the application and the target the FUI. In addition, this extra "layer of abstraction" allows the use of more comprehensive methods using the terms of the application-specific components, instead of generic RSL methods, such as `createEntity()`, `addLinkSources()` or `getAccessibleEntitiesByType()`. These methods can, for example, be renamed appropriately to `createApplication()`, `addOwnerofAC()` or `getAccessibleApplications()`.

Next to microservices, the server also provides file storage. If, for instance, an image is associated to a resource, the image data should be stored in the file storage and the resource class should store a pointer to the file instead of holding the image data itself. The file can then be accessed via a URL that is returned by the built-in HTTP server. Finally, in order to keep track of connecting and disconnecting clients

as well as the requests made to the server, the server-specific logger is used to create detailed logs. Note that thanks to an event-based approach, the requests can be handled in an asynchronous and parallel manner and thus allow the server to handle a large amount of requests simultaneously.

More details about the RSL implementation as well as its limitations are described in [184]. Ideas about how to improve the current implementation are discussed as well. A last thing worth mentioning is that Gradle[1] is used to manage the project and all its dependencies, which means that it can be built and used on any operating system without having to worry about the setup.

Given our conceptual foundations, in the next chapter we will investigate how to make this rich functionality of our data layer accessible to end users on the visualisation layer. In doing so, we also investigate the best way to satisfy our last requirement related to providing support for end-user development.

[1] `https://gradle.org`

Chapter 5

User Study

The broader one's understanding of the human experience, the better design we will have.

Steve Jobs, Apple

In this chapter we focus on answering our third research question which is related to finding the right abstractions or metaphors to visualise cross-device and IoT interactions. Finding the appropriate abstractions is also necessary to satisfy requirement 7 regarding support for end-user development. Therefore we take a similar approach as Dey et al. [74] and McAweeney et al. [140] which consists of engaging users through an exploratory elicitation study with as goal explore the mental models of people when dealing with interaction across smart devices and *things*. Based on these mental representations of XD and IoT interactions we then derive a number of characteristics which we grouped into categories from which we ultimately present design guidelines involving metaphors for the end-user authoring of cross-device and IoT applications. In order to do this, we provided participants with a scenario including both XDI and IoT concepts and asked them to visually represent the interactions described in the scenario.

This scenario-based design has been chosen because scenarios are evocative, promote reflection, analysis and innovative thinking [189]. We recruited users with a technical background—who studied or are studying computer sciences—as well as non-technical participants. The mixed sampling was necessary since we wanted to verify which metaphors would fit the mental models of both, technical as well as non-technical users. In order to maintain an unbiased position, we further followed the grounded theory method [54] for the data collection and analysis.

This chapter will first concretely describe how RQ3 is divided into two subquestions in order to better structure its answer. Further, the study setup will be explained as well as the methodology that we have followed. We then present the results of the study together with some concluding remarks concerning these results. Afterwards, we introduce design guidelines for the design of XD and IoT end-user authoring tools, which are based on the results of our study. Next, we check related work against our design guidelines and finish this chapter with an analysis of people's knowledge regarding cross-device and IoT interactions.

5.1 Research Questions

We define two subquestions in order to provide a better answer to RQ3, which is shown bellow as a recap:

Research Question 3 (RQ3) *Which metaphors or abstractions should we use on top of our model and implementation to allow end users to visualise and create their unified cross-device and IoT interactions?*

This question can be divided into the following subquestions. The first one is addressed in the following sections while the second one is addressed in the Section 5.5.

Research Question 3.1 (RQ3.1) *What are the mental models of people when dealing with XDI and IoT interaction?*

In order to answer this question we performed the elicitation study presented in this chapter. Thereby, we gained insights on how people visualise and understand interactions across smart devices and *things*. The visualisation part is achieved by asking people to draw interactions while the understanding part is accomplished through a semi-structured interview that is carried out at the end of the study with each participant.

Research Question 3.2 (RQ3.2) *How can we interpret and use the identified mental models from RQ3.1 to visualise and create XDI and IoT interaction?*

This question is answered by analysing the participants' drawings and interviews. The analysis sought to find common patterns and differences which allowed us to formulate design guidelines to help developers design end-user authoring tools that represent XDI and IoT interactions in a way that fits the users' mental models. As explained in the introduction, the resulting design guidelines are then compared with existing EUD authoring solutions and used to build our EUD authoring tool. More about this phase of our dissertation is presented in Section 5.5 and Section 5.6.

5.2 Setup

The study setup consisted of a scenario around a student and fitness enthusiast called Alex, who is interacting with various cross-device applications and IoT devices. The scenario was described in a set of PowerPoint slides with a number of questions for the participants on the last slide (see Appendix A.1). One of these slides is shown in Figure 5.1. Another short presentation was used to first introduce participants to the concept of cross-device interaction and the Internet of Things. Furthermore, a consent form, interview questions, a questionnaire as well as a script of the study procedure have been prepared. The scenario and questions have been reviewed in a pilot study with four users (answers were not used in the final study), helping us to improve our study setup. Note that the study has been performed in English as well as in French and therefore all the study material is available in both languages.

After the movie finished, they turn off the TV and go to bed

When turning off the TV, the following actions take place:

- The TV turns off
- The ambient lights at the back of the TV turn off
- The light in the living room turns on

Figure 5.1: Example slide from the Scenario.pptx presentation

5.3 Methodology

Protocol

Each study session included two participants; either two technical persons, two non-technical persons or a technical and a non-technical person. The participants first had to fill in a consent form and then received a short presentation about the concepts of XDI and IoT. Afterwards, the two participants where separated—each of them with an individual observer—and asked to "graphically answer" a few questions based on a presented scenario. After both participants finished drawing their answers, they were asked to sit together and compare their drawings to potentially make some improvements. This was followed by a short semi-structured interview with each participant. Finally, participants were asked to fill in a questionnaire. Each session

has been videotaped for later use during the analysis and the observers were taking notes during the entire study.

XDI and IoT Presentation (10 min)

One of the two observers gave a short presentation introducing the concepts of XDI and IoT, with examples of both XDI (Spotify and Chromecast) and IoT (Philips Hue and Amazon echo). In order to avoid bias in the experiment, participants were not provided any examples of metaphors. After the presentation, participants were explained that they will get a scenario along with some questions. They were further asked to follow the *think-aloud* protocol and speak out their thoughts during the experiment.

We suggested them to act as if they were talking with the observer who was sitting with them, in order to avoid that they felt uncomfortable. They were further told that any answer would be correct and that the study was in no way a test to measure their intelligence.

Scenario Drawing (30–60 min)

During this phase of the user study, the two participants were separated and assigned to an individual observer. Each participant was given a hardcopy of the PowerPoint slides containing the scenario and questions. They also received a number of blank paper sheets as well as some pens and pencils. At any point, participants could ask questions to their observer. In case of comprehension problems of the scenario questions, each observer would reformulate the question according to their script. While observing the participants, the observer could also ask questions to clarify some of their drawings such as *"Are these actions happening at the same time? How did you indicate that?"*. These questions evolved over the course of the study as suggested in [54] and based on our analysis after each session, more directed questions such as *"How did you indicate that the movie playing on Alex' phone is the same one as the movie shown on the TV?"* could be asked.

Note that the time for this phase was originally set to a maximum of 30 minutes, but some participants needed more time to finish their drawings.

Comparison of Drawings (30 min)

After completing the previous phase, both participants were asked to sit together and compare their drawings. This way, we could get more insights on why some participants chose to draw certain concepts in a certain way and see whether by comparing the drawings they could improve their own drawing or make a new one combining

elements from both participants. At any time participants could ask questions to the observers and if they had difficulties starting the comparison, the observers would ask a few questions according to their script.

Interview and Questionnaire (10 min)

Each participant was interviewed separately and asked five questions, with three of the questions related to their drawings and potential difficulties during the study. Another question asked whether they think that they have enough control over existing cross-device and IoT solutions. A final question investigated whether they had further comments about the study. At the end of the session, participants also filled in a questionnaire consisting of 15 questions. Five questions were dealing with demographic data and the participants' education. Nine questions collected information about their exposure to XDI, IoT and technology in general while a final question asked them for feedback about the ease of completing the study.

Participants

Our user study included 30 participants (12 females and 18 males) with an average age of 33.8 years (min=23, max=76, SD=11.4) who participated in pairs of two. As mentioned before, an equal number of participants, with a technical and non-technical background, were recruited as illustrated in Table 5.1. Note that we chose an equal number of each group so that we could compare the drawings made by non-technical people with the ones of more technical people.

Background	Participant
Technical	P1, P2, P8, P9, P11, P13, P16, P21, P22, P25, P26, P27, P28, P29, P30
Non-technical	P3, P4, P5, P6, P7, P10, P12, P14, P15, P17, P18, P19, P20, P23, P24

Table 5.1: Background of participants

Most participants (27) had already heard about the term IoT and 15 participants had some IoT devices at home. Only slightly more than half of the participants (17) knew about the term cross-device interaction. One participant did not possess any smart devices, 10 participants had only a smartphone and 19 participants had two or more smart devices. In the following, we use the format $p(t, n)$ to provide information about participants, with p representing the total number of participants which can be divided in t participants with a technical background and n non-technical participants. Participants with a technical background were all people having studied computer science.

5.4 Results

Throughout the study, data has been collected via notes taken by both observers, video recordings, the participants' drawings, interview transcripts and the answers filled in by the participants in the post-survey questionnaire. By analysing the drawings, a coding guide has been established, which was then used by the two observers to analyse the drawings. Over time, new categories and subcategories have been added to the open coding. In the remainder of this section we review the final categories with their subcategories representing our study findings. A summary of each subcategory is shown at the end of the section in Figure 5.12. Note that some subcategories might overlap and thus do not sum up to the total number of 30 participants for each row in Figure 5.12. From the final categories described in this section we derived the design guidelines presented in the next section.

5.4.1 Data Transfer and Synchronisation

(a) Grouping by participant P15 (b) Use of radio waves by participant P21

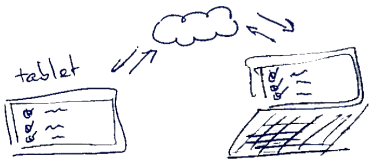
Figure 5.2: Drawing examples of data transfer and synchronisation

Data transfer was sometimes difficult to picture for non-technical participants as mentioned in the interviews. For example, participant P19 explained that it was difficult to represent *"waves"* since you cannot see them. 29 (14,15) participants used arrows at least once to indicate data transfer and interaction between devices and smart objects. One technical user (P22) focussed more on user interfaces on the devices and only added arrows during the comparison phase of the study, after comparing their drawings with the second participant. The arrows that P22 added are meant to show how some events trigger specific actions and were thus neither used for data transfer nor for synchronisation. From the 29 participants using arrows for data transfer, we could see that 20 (11,9) annotated these arrows with data that had to be transferred (18 (10,8)) or wrote/drew the transfer protocol on the arrow (4 (3,1)). As can be seen with the previous numbers, two participants did both wrote the data and transfer protocol on their arrows. 10 (3,7) participants did not write the transferred data on an arrow but rather on the devices (3 (0,3)) or next

to the devices (2 (1,1)). Further, 4 (2,2) participants defined the data as "input" by drawing an arrow from the data to the device, as illustrated for the *'calories'* as data input for the smartphone in Figure 5.2a.

Different kinds of arrows were used by the participants and most did not use them in a consistent way. For instance, participant P28 used dashed and solid arrows and mentioned during the study that they were both used for the same purpose. Some participants used radio waves in combination with arrows, such as illustrated for participant P21 in Figure 5.2b, and a few participants also used double-lined arrows (⇒), such as participant P28 using them for showing causality.

Slightly less than half of the participants (12 (8,4)) did not make a clear distinction between transferring and synchronising data between two devices. 13 (4,9) participants used a double-sided arrow (↔) to indicate synchronisation between two devices, while only 2 (1,1) used a synchronisation symbol (↻). Furthermore, 4 (3,1) participants used the *"sync"* keyword and 2 (1,1) used two arrows going in opposite directions (⇆) as shown in Figure 5.3a, where synchronisation is done between the tablet and the laptop through an intermediary cloud service.



(a) Synchronisation using two arrows going in opposite directions by participant P25

(b) Data transfer using dashed arrow and WiFi symbol by participant P28

Figure 5.3: Drawing examples of data transfer and synchronisation

5.4.2 Expressing State Changes

In the given scenario many devices change state through events and actions, such as switching between an "on" and "off" state. The smartphone, for instance, also changes state between a remote controller and a movie viewer, which has been difficult to show by many participants. Participant P6 had to draw this part multiple times in order to show that the smartphone could change state between showing the movie and being a remote control. Their final version is shown in Figure 5.6a. In order to go from one state to another, many participants (16 (9,7)) used regular arrows labelled with the corresponding command for changing the state (e.g. *"turn on"*), as shown in Figure 5.4. 17 (9,8) participants represented the state changes graphically by, for instance, showing a light bulb emitting some rays of light. Only 4 (0,4) participants did not clearly show any state changes. Overall, participants expressed that it was

easier to show state changes when the "state change" event was triggered by another device rather than when a device was changing its internal state to something else, like switching the phone to "remote mode".

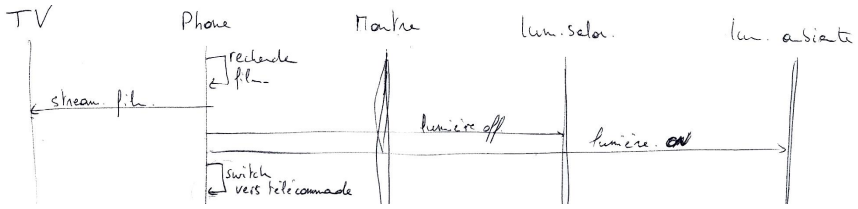


Figure 5.4: Drawing example of state changes by participant P11

5.4.3 Time-based Actions

Throughout the scenario different sequential and concurrent actions take place. Not all participants made a distinction between these time-based actions. Concurrent actions were sometimes grouped together by the participants (11 (5,6)), either by noting the concurrent actions on the same arrow or by grouping the actions in one drawing which was sometimes also framed. An example of grouping on the same arrow is depicted in Figure 5.2a, with grouping using the *'+'* symbol on an arrow and even in an ellipse: *"lumière salon + télé"*. 2 (2,0) participants used multi-target arrows, as shown in Figure 5.5 with the arrow starting at *"envoyerFilm"* from the phone that splits up into two target arrows. The first one points to the TV with *"film"* written on it, while the second one targets the lights marked with *"appel de fct"* (function call), to express that these actions happen at the same time. Only 1 (1,0) participant used the same numbers next to actions to indicate that they happen at the same time.

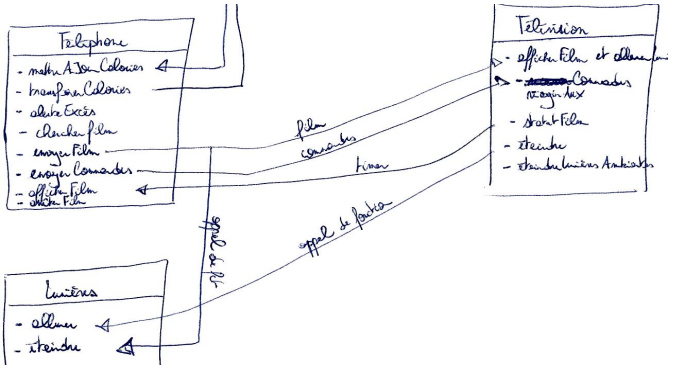
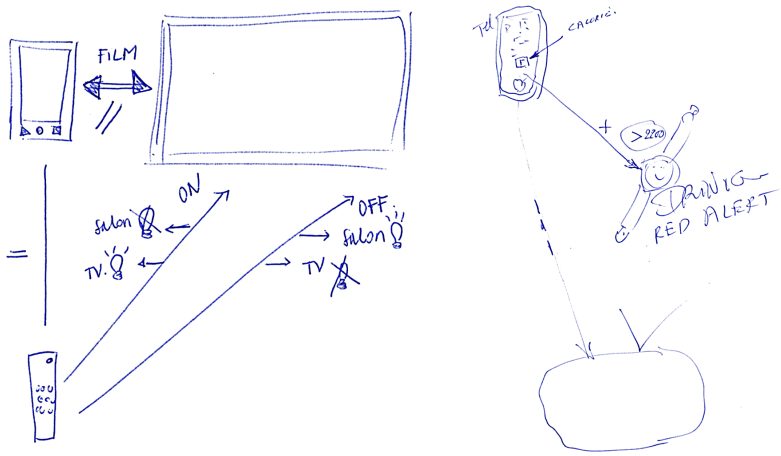


Figure 5.5: Drawing example of a multi-target arrow expressing concurrent actions by participant P9

In order to demonstrate sequential actions, 9 (8,1) participants noted down the actions from left to right with an arrow in between, one of them did not use arrows in between but just indicated at the top of their drawing that time was going from left to right. Further, 2 (1,1) participants indicated time going from top to bottom, one of them using a sequence diagram as illustrated in Figure 5.4. Last but not least, 5 (2,3) participants used numbers to indicate the order of actions. Half of the participants (15 (5,10)) did not clearly indicate sequential actions.

5.4.4 Multiple Instances of the Same Data

At one point in the scenario, a movie is shown on the TV and on Alex' smartphone. These two instances of the movie are supposed to be synchronised. 3 (0,3) participants made this clear by adding a double-sided arrow, as shown in Figure 5.6a. 6 (3,3) participants used some keywords to show that the movie is duplicated on the phone, including "copy", "cast split", "duplicate" or "||". A few participants (7 (3,4)) drew or wrote the same thing on or next to both devices as shown in Figure 5.3a with the same check-list drawing on the laptop and tablet. Further, 1 (1,0) participant just wrote down the option that the film can be shown on both TV and smartphone. 15 (8,7) participants did not clearly indicate multiple instances of the same data.



(a) Showing multiple instances that are synchronised by participant P5

(b) Location awareness by participant P3

Figure 5.6: Drawings examples of multiple instances and location awareness

Single Device Interactions

Some interactions in the scenario just happened on one device, such as the phone becoming a remote controller and then a movie viewer, or the smartwatch monitoring

Alex while running. In order to illustrate these kind of actions, 8 (6,2) participants drew an arrow from one instance of a device to another. 3 (1,2) participants drew an arrow going from the device to some text describing what is happening on this device, while one computer scientist drew an arrow originating and ending at the same device, as illustrated in Figure 5.4. 7 (2,5) participants drew an arrow from the device to Alex, as shown in Figure 5.7, where the different interactions that are possible with the watch are written on the arrow originating from the watch to Alex. In this case the watch is currently tracking Alex' GPS location on her running track, but can also give Alex directives on the rest of the track and when to start sprinting (for the last 50 meters). Lastly, 9 (4,5) participants just wrote the actions of the device below the device.

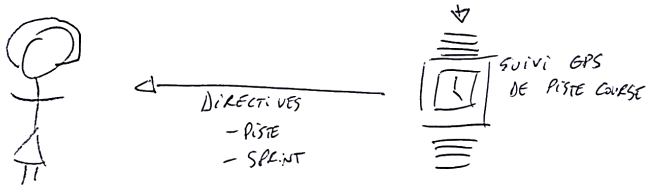
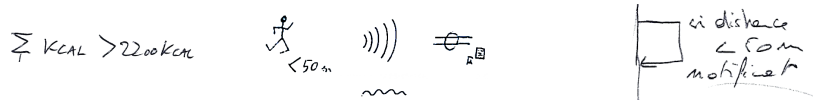


Figure 5.7: Drawing example of arrow to Alex to express what the watch can show, by participant P11

5.4.5 Conditional Statements

Throughout the scenario, a couple of conditional statements were mentioned. Examples of conditions were, *"Since Alex exceeded the 2200 kcal today, she receives a notification on her smartwatch warning her about this excess"* or *"When Alex has 50 meters left to run, the smartwatch sends a vibration to motivate her to sprint the last few meters"*. Such conditions have been noted down by 4 (3,1) participants by using the "IF" keyword or "SI" in French, as shown in Figure 5.8c. 15 (7,8) other participants noted the conditions without the use of these keywords, for example, by writing ">2200 kcal" on or in between arrows as shown in Figure 5.8a. A last group just drew the conditions as shown in Figure 5.8b, where the first symbol indicates that there is less than 50m left, the second symbol shows that then a "*vibration*" is sent and the last symbol shows the smartwatch receiving the notification.



(a) Participant P12 (b) Participant P26 (c) Participant P11

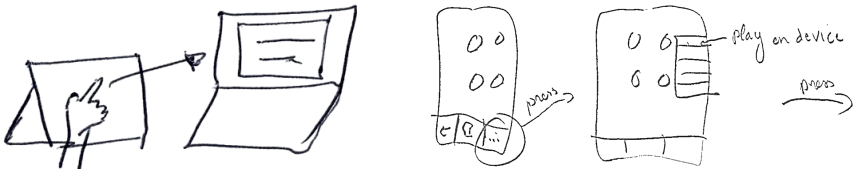
Figure 5.8: Drawing example of conditions

5.4.6 Location

The scenario took place at different locations, at the university in a classroom, at home in the living room and kitchen, and outside in a park. 7 (1,6) participants wrote down the location names where the interaction took place, even though it was not playing a direct role in the interaction that was happening in the given scenario. Likewise, in most of these drawings the written location did not matter, only one of them mentioned that the location played a role for the interactions. Rather than writing the location down, 3 (1,2) participants mentioned some form of location awareness, for example by drawing a sensor and explaining that a certain interaction happens when this sensor detects Alex leaving the room. Participant P3, for instance, mentioned that when Alex was at a certain distance from her TV, her smartphone would show the movie instead of just serve as a remote controller. This distance between the phone and the TV is depicted in Figure 5.6b by the dashes on the arrow between these two devices.

5.4.7 Presence of Actors

The drawings of 16 (6,10) participants included the presence of Alex, the main character of the scenario. She was either drawn by the participant as seen in Figure 5.7 or her name was written in textual form. Some participants (2 (2,0)) only drew Alex' hand as shown in Figure 5.9a.



(a) Swiping by participant P13 (b) Pressing buttons by participant P3

Figure 5.9: Drawing example of interface interaction

5.4.8 Actor's Interactions

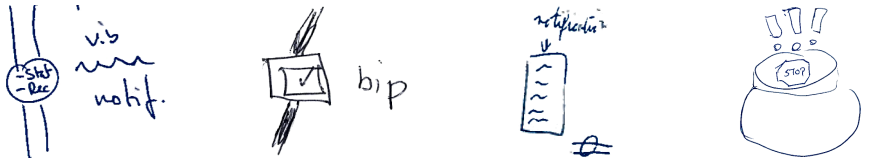
Interactions between Alex and her devices, such as "swipe", "touch" or "press", were noted down or drawn by many participants, as depicted in Figure 5.9b. 19 (9,10) specified these kinds of interactions by either writing them next to Alex, on an arrow or by drawing Alex performing the action. 6 (3,3) of them represented the interactions graphically only, as shown in Figure 5.9a, where Alex' finger is swiping her tablet. Another 2 (0,2) participants only drew an arrow from Alex to a device without specifying the type of interaction.

5.4.9 Representation of Devices

Participants represented devices and smart appliances either graphically or in textual form. 25 (12,13) participants used a realistic graphical representation of the devices as shown in Figure 5.9a. 4 (2,2) participants wrote the device names into squared boxes, as illustrated in Figure 5.5. Further, 7 (3,4) participants just wrote the name of the devices, as shown in Figure 5.4 and one person wrote the device names into ellipses, as depicted in Figure 5.2a. Some participants were not consistent in the representation of devices, sometimes representing them in textual form and sometimes graphically. Participant P15 even mixed the use of squares and ellipses to represent devices, as shown in Figure 5.2a. When devices were represented graphically, participants often (15 (8,7)) drew certain UI elements such as buttons for triggering actions on the devices as well, as we have seen in Figure 5.9b. This is surprising given that the scenario never mentioned buttons and the questions asked to the participants focussed on the interaction between devices rather than on the UI of an individual device.

5.4.10 Use of Symbols and Keywords

It is interesting to analyse the symbols and keywords used by the participants, since only 6 (3,3) participants did not use symbols other than the devices and arrows. The Wi-Fi and Bluetooth symbol were sometimes used to indicate whether the connection had to be made via Wi-Fi or Bluetooth. The moon and sun were used by 2 (0,2) participants to indicate day and night. Notifications were often shown by drawing a vibration symbol next to or on a device with the corresponding message usually shown on the device but also symbolised in different ways as highlighted in Figure 5.10.



(a) Participant P25 (b) Participant P13 (c) Participant P26 (d) Participant P28

Figure 5.10: Drawing examples of notifications

5.4.11 Informative Interview

In addition to the drawings, participants of our elicitation study were also interviewed. In this section we shortly summarise the answers of our semi-structured interview that took place at the end of the study. The interview was structured as follows. First participants were asked two questions related to the part of the study where they

had to draw cross-device and IoT interactions, which gave us further indications on whether they drew based on previous experience and whether they had any difficulties. The next two questions of the interview were asked in order to get insights on whether or not participants would be interested in an authoring tool allowing them to create cross-device and IoT applications. The participants were briefly explained what kind of authoring tool it would be (i.e. being based on their drawings and mental models). Then they were asked if they would use such an authoring tool and if they had enough customisation possibilities with the current applications on the market. As final question we also asked participants whether they had any comments about the study in general. A summary of the interview questions is shown below:

- Question 1: How did you come up with such a drawing?
- Question 2: Did you have any difficulties during the study? If yes, which ones?
- Question 3: Do you think you would use an application proposing IoT and cross-device interaction that you can create by using the components that you drew? Why, Why not?
- Question 4: Do you think you have enough control to customise your interfaces with existing applications?
- Question 5: Do you have any comments?

The analysis of these questions is discussed in this section. We will mainly focus on question 3 and 4, as the other questions were meant to be primarily used for the analysis of the participants' drawings. Therefore we shortly summarise the answers to the first question as follows: 10 (7,3) participants explicitly said they drew their drawing based on what they know about technology or based on their background from their university studies. 8 (4,4) said that they started by first representing all objects present in the given scenario first to get an overview and 3 (2,1) other participants mentioned that the scenario reflected chronological events and therefore drew a time line type of diagram. The remaining 9 (2,7) participants simply told us they drew what seemed more easy for them, either a schema when they did not want to draw or just the opposite.

For the second question related to difficulties during the study, 10 (4,6) participants mentioned no difficulties in general. Other participants had some minor difficulties, 3 (3,0) participants said that they had difficulties knowing which level of details was required, 4 (3,1) expressed having complications for the representation of *connectivity* in a graphical way and 2 (1,1) participants were first confused due to their focus on the graphical user interface of the devices rather than on interaction between devices.

In total 18 (12,6) participants were enthusiastic about having an authoring tool where they could manage interaction across smart devices and objects using the abstractions that they drew. Participants mentioned thinking about a drag-and-drop interface where they could connect the components using arrows. One technical participant mentioned that they were currently using the IFTTT interface but were lacking a more graphical UI. Another technical participant also said that it would be nice to group all technologies. A non-technical one mentioned that they could already imagine customising their own smart environment a bit like in "*construction mode*" in the Sims game. 2 (1,1) participants also mentioned they would only use such an authoring tool if it is safe and does not contain any bugs.

12 (3,9) participants were not interested in an authoring tool, the main reason being that they do not feel to have the need for it. 7 non-technical participants mentioned to not being a "*fan*" of such technologies, when speaking about IoT. From these 7 people, one even mentioned being scared of such applications that are "*doing everything for you*" and one said they are not ready to depend on applications: "*I don't think I'm ready to just rely on apps that run my life, I would feel like I'm in jail*". The other 5 out of these 7 participants also mentioned either that they have no interest in technology (one mentioned not even owning a smartphone), could not use technology or that they want to stay in control and do not want to become lazy and dependent on technology. The 2 remaining non-technical participants, simply said that they were not interested having such a tool for themselves but found the concept interesting. From the 3 technical participants, one mentioned not having enough devices or smart objects for needing such a tool and the others expressed that they would rather program it themselves, one explicitly saying that they would trust it more if making it themselves. One of these 3 participants further mentioned a preference for textual rather than graphical user interfaces, as they would be faster. A bar chart summarising the answers of question 3 and 4 is shown in Figure 5.11.

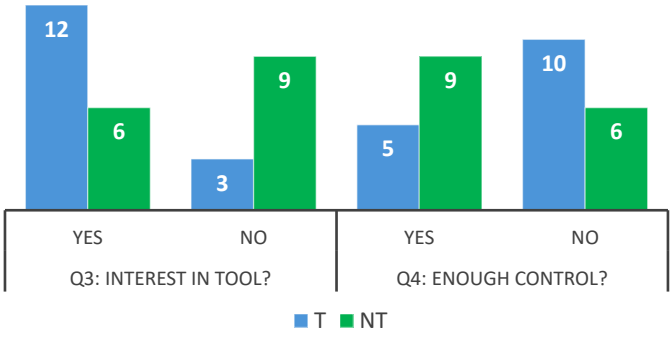


Figure 5.11: Summary of interview question 3 and 4

Regarding question 4, 16 (10,6) participants said they did not have enough control to customise their interfaces and gave some examples. One participant mentioned having problems when controlling the Netflix application using the Google Home voice interface, which does not allow users to specify which user profile should play which TV show. Another participant mentioned having YouTube recommendations for her husband instead of for herself. Two participants also focussed on the fact that they do not have enough control about their privacy, not knowing what data the applications were using. One also gave an example about the 'Apple TV' application UI that cannot be personalised. On the other hand, the remaining 14 (5,9) participants mentioned having enough control for customisation of their interfaces; one non-technical participant even mentioned having too much control. 6 (2,4) participants said that they did not use many applications so they never experienced anything lacking with the ones they were using. One technical participant expressed that whenever something was lacking in one application, they just tried finding another one. Lastly, one non-technical participant said they managed to get the control that they wanted over their devices by combining multiple applications.

In summary, as we can see from Figure 5.11, the answers of questions 3 and 4 are quite mixed, even between technical and non-technical users. We noticed that many of the participants that were not interested in an authoring tool for the control and creation of cross-device and IoT interactions, were often simply not interested in this kind of technologies and therefore also answered having enough control over their user interfaces for question 4. Out of the 30 participants, 22 had opposite answers for question 3 and 4, which means that whenever they had an interest in the authoring tool they also said they did not have enough customisation control, or whenever they were not interested in the tool they also expressed having enough control over their current applications.

As last note, we would like to point out that even though we never mentioned anything about security and privacy of cross-device and IoT interactions, 6 (2,4) participants raised concerns about these issues during our interviews. 4 (0,4) participants also mentioned bugs and update issues in general with applications they used, which could have influenced their current use of applications and were reluctant in trusting applications due to these known problems.

5.4.12 Concluding Remarks

While we primarily performed a qualitative study, we still checked for any correlation and differences between technical and non-technical participants per category identified in Section 5.4 and summarised in Figure 5.12. There is not enough data in every subcategory to run statistically significant tests, however already with little data we

can see that there is a fairly high correlation between both groups of participants' drawings in each category except for the sequential actions which seem to be mainly depicted by technical users. The Pearson correlation coefficients varied between 0.65 and 0.95 with 0.12 as lowest value for sequential actions. Finally, on a side note, while the *comparison of drawings* phase allowed participants to improve their drawings, only minor changes where made during this part of the study. Sometimes some arrows or a symbol or two were added. Therefore we did not go into a detailed analysis of this phase of the study.

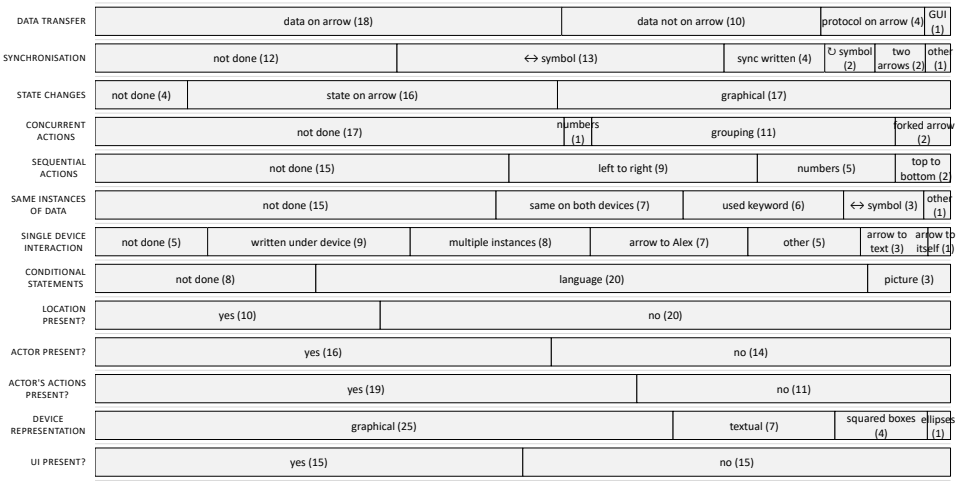


Figure 5.12: Overview of study results

In the next section, we introduce design guidelines based on our study findings in order to answer RQ3.2. We further check existing work in the XDI and IoT research domains introduced in Chapter 2 against our design guidelines in Section 5.6 and finish by discussing the results of our study questionnaire. Note that since the drawings of technical and non-technical participants had a fairly high correlation, we therefore do not distinguish between the two groups in the remainder of this chapter.

5.5 Design Guidelines

Based on our study findings and the investigation of related work, we defined a number of initial design guidelines for cross-device and IoT end-user authoring tools. While these guidelines serve developers to create end-user authoring tools based on what end users prefer, understand and expect when dealing with XDI and IoT interactions, they might be extended and further refined over time.

5.5.1 G1: Use Pipeline Metaphor to Represent Interactions

We recommend the use of the pipeline or graph metaphor to represent interactions in an end-user authoring tool, given the fact that all but one of the study participants used arrows to represent interactions between smart devices and *things* (see Section 5.4.1). The pipeline metaphor graphically represents applications as directed graphs where nodes correspond to elementary services with interconnecting links (i.e. pipelines) [67]. To further facilitate the use of the pipeline metaphor, compatible inputs and outputs should use the same colour as done in [66], in order to prevent end users from linking incompatible devices and services. Finally, popup windows can be used to gather data about the configuration of an interaction as done in the E-Wired prototype [72].

5.5.2 G2: Use Different Arrow Types for Different Interaction Types

Related to the previous guideline, we advocate assigning a specific arrow type to a certain type of interaction. Uni-directional interactions should be represented by regular arrows, while synchronisation between multiple smart technologies should be represented by double-sided arrows. Optionally, a difference could also be made between user-initiated (e.g. button press) and contextual interactions (e.g. time). This guideline stems from the fact that of the 18 participants who represented synchronisation, 13 used double-sided arrows (see Section 5.4.1). Further, by using different arrow types, users could identify the different interaction types present in the authoring environment at first sight. Participant P28, who drew different types of arrows, also mentioned that they could have used these different arrows to represent various types of interaction, which would have improved their drawing. Lastly, in order to differentiate user-initiated and contextual interactions, one could use regular arrows for user-initiated interactions and dashed arrows for contextual interactions, given that our participants used these dashed arrows second most after the regular ones.

5.5.3 G3: Provide a Realistic Graphical Device Representation

An authoring tool should provide a realistic graphical representation of the available smart devices and *things*, as done by 25 participants of our study (see Section 5.4.9). Moreover, during our interviews, some participants who did not draw the devices mentioned that they would have done so if they had better drawing skills. Ideally, a user should be able to choose these graphical representations in order to easily recognise their smart technologies. Note that this kind of representation offers the possibility to also show graphical user interface components on the screen of the graphical device representations, as done by many participants (15).

5.5.4 G4: Provide a Graphical Representation of Users

An authoring tool should integrate the graphical representation of users, either as individuals or groups of users. Ideally the action that the user is performing to trigger an interaction, such as pressing a button, should be expressed as well. This representation of users should also be used to show contextual interactions involving the user, such as *"If Alex is at home turn on the Wi-Fi on her phone"*. This guideline is derived from the fact that, even though our scenario did not include specific user-dependent interactions, Alex the main actor of our scenario was still depicted by 16 participants (see Section 5.4.7). Further Alex' actions were visualised by 19 participants (e.g. drawing Alex' finger swiping an interface), as described in Section 5.4.8.

5.5.5 G5: Represent Sequential Interactions from Left to Right and Group Concurrent Interactions

We recommend the use of an implicit timeline by representing sequential interactions graphically from left to right. For more complex interactions, optional numbering could also be used to avoid confusion. We further recommend grouping to represent concurrent triggers and actions of an interaction. This can, for example, be done either by grouping the actions on one arrow when possible or by representing the elements involved in the actions below each other (optionally framed). Next to the fact that timelines are widely used for organising information in chronological order [199], of the 15 participants who clearly showed sequential interactions, 9 depicted these interactions from left to right and 5 used numbering (see Section 5.4.3). Further, participants who depicted concurrent interactions grouped these interactions either on the same arrow, underneath each other, using numbers or using other symbols (e.g. '+'). Finally, this grouping is also in line with the *similarity* and *proximity* Gestalt principles [115].

5.5.6 G6: Provide Textual as well as Graphical Representations for Conditional Statements

The representation of conditional statements in an authoring tool should be both textual as well as graphical. Regarding the graphical representation, conditional statements should be represented via graphical elements accompanied by extra textual information, such as the condition of a rule. This stems from the fact that most study participants (20) mixed both representations in their drawings, by, for example, writing down the condition in textual form (e.g. *"sum of calories > 2200 kcal"*) on arrows, in between arrows or on devices (see Section 5.4.5). Further, as suggested by Dey et al. [74], we recommend making these conditional statements available in textual form for better comprehension. In order to represent conditional statements

in textual form, *IF-THEN* statements could be used. While only 4 participants used the *"IF"* keyword to show conditional statements, it has been shown by Ur et al. [213] that If This Then That (IFTTT)[1] can quickly be learned by inexperienced users to create programs containing multiple triggers or actions. In addition, we have seen from related work in Chapter 2 that the rules metaphor was used by many end-user authoring tools to represent conditional statements, such as in [74, 83, 92, 93]. Moreover, guidelines for rule composition have been provided by Desolda et al. [72] with the *Rule_5W* model. Finally, note that end users should also be able to freely switch between those two representations in the authoring tool.

5.5.7 G7: Support UI Design

An authoring tool should offer users the possibility to create their own user interface such as in [92, 147], given that half of our study participants represented concrete UI elements such as buttons (see Section 5.4.9). This allows for more customisation and provides the possibility to create UI-triggered interactions as often depicted by our participants.

5.5.8 G8: Use of Symbols and Annotations

The authoring environment should include symbols as well as support for annotations. The symbols and annotations should not always have to be linked to functionality, but might rather serve as a way for end users to better understand and remember what is represented in the authoring space. We derived this guideline based on the fact that only 6 participants did not use symbols other than devices and arrows (see Section 5.4.10). Further, a few participants mentioned that they liked having icons and symbols to represent all kinds of interactions between devices. For instance, Participant P21, mentioned that they preferred using icons or animations instead of text to show what was happening in the scenario.

5.6 Checking Related Work Against Guidelines

In this section we compare related end-user authoring tools in the domains of cross-device interaction and IoT with the presented design guidelines and investigate what they might imply for existing as well as future end-user authoring tools. We start by shortly introducing our selected related work from the background chapter including the more prominent XD and IoT end-user authoring tools and then check for their compliance with our guidelines as summarised in Table 5.2. All the described cross-

[1] `https://ifttt.com`

device authoring tools are shown first in alphabetical order, separated by a double line from all the IoT authoring tools which are also sorted in alphabetical order.

Systems	G1	G2	G3	G4	G5	G6	G7	G8
ACCORD [183]	○	◌	●	●	◖	◖	◌	◖
Direwolf [121]	○	◌	○	○	◌	◌	◖	◖
Ghiani et al. [92]	○	◌	●	●	◖	◖	?	◖
Interplay [149]	○	◌	○	○	◌	◌	◌	○
Jelly [147]	○	◌	○	◌	◌	◌	●	◖
Platform Composition [171]	◖	◌	◖	○	◌	◌	◌	◖
SmartComposition [123]	○	◌	○	◌	◌	◌	◖	◖
XDBrowser 2.0 [154]	○	◌	●	◌	◌	◌	◖	○
a CAPpella [73]	○	◌	●	○	●	◌	◌	○
AppsGate [60]	○	◌	○	○	◖	◖	◌	○
Atooma[1]	○	◌	●	○	◖	◖	◌	◖
CMT [212]	◖	◌	○	○	●	◖	◌	○
Direwolf 3.0.0 [120]	○	◌	?	?	?	?	?	?
E-Wired [72]	●	○	○	?	●	◖	◌	○
EPIDOSITE [125]	○	◌	○	○	○	○	◌	○
HomeRules [190]	○	◌	●	?	●	◖	◌	◖
iCAP [74]	○	◌	●	●	◖	◖	◌	◖
IFTTT[2]	○	◌	●	○	○	◖	◖	◖
ImAtHome [83]	○	◌	●	◖	○	◖	◌	◖
Keep Doing It [68]	○	◌	●	○	◖	●	◌	◖
Puzzle [66]	○	◌	○	?	●	◖	◌	○
SmartFit [17]	○	◌	○	○	◖	○	◌	○
T4Tags 2.0 [19]	○	◌	●	?	●	◖	◌	◖
TARE [93]	○	◌	○	○	○	○	?	○
Tasker[3]	○	◌	●	○	◖	◖	●	◖
TouchCompozr [124]	○	◌	?	?	○	◖	◌	○
Versatile [103]	◖	◌	○	◌	○	○	◌	○
ViSiT [7]	○	◌	○	◌	●	◖	◌	◖
Zipato[4]	○	◌	●	?	◖	◖	◌	◖

Legend
○ Not fulfilling guideline
◖ Partially fulfilling guideline
● Completely fulfilling guideline
◌ Functionality not present
? Not specified

Table 5.2: Compliance of XD and IoT authoring tools with guidelines G1–G8

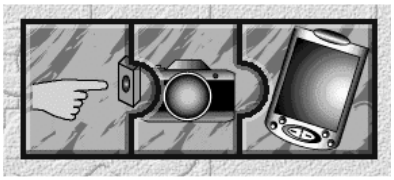
[1] https://resonance-ai.com/about.html
[2] https://ifttt.com
[3] https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm&hl=en
[4] https://www.zipato.com

5.6.1 Authoring of Cross-Device Applications

As we have seen in Chapter 2, many web-based tools such as XDBrowser [155, 154], SmartComposition [123] and DireWolf [121], have emerged over the years. Further, after analysing the metaphors that these tools are using, we have seen that none of them use the pipeline metaphor, meaning that none of them is compliant with guideline G1. XDBrowser 2.0 which allows users to re-author exisitng web pages by distributing them across several devices using a selection tool, uses a metaphor close to the *copy/paste* metaphor. SmartComposition [123] and DireWolf [121] use the *drag-and-drop* metaphor to distribute widgets across devices. Furthermore, looking at the other presented solutions, we notice that Jelly [147] also uses the *copy/paste* metaphor to copy part of the UI from one device to another and thus does not satisfy G1 either. However, Jelly provides more freedom in the interface design by offering a design environment where UIs can be designed for multiple platforms in parallel and therefore it complies to G7. While these tools all integrate the concepts of multiple devices, only XDBrowser 2.0 shows these devices graphically by the use of icons and is therefore fully compliant with guideline G3. Since SmartComposition, DireWolf, XDBrowser 2.0 and Jelly do not include the notion of user, time and conditional statements, guidelines G4 to G6 are represented by dashed circles in Table 5.2. As mentioned above, Jelly fully complies to G7, compared to SmartComposition, DireWolf and XDBrowser 2.0 which all only partially satisfy G7, since the UI design is limited to widgets and tiles where these widgets can be placed, which reduces the flexibility in the UI creation process for the end user. Except for XDBrowser 2.0, the three other solutions include some pre-defined symbols and annotation possibility. However, these symbols and annotations are always linked to some functionality and therefore do not fully comply to guideline G8.

Next, the authoring environment by Ghiani et al. [92] allowing users to create contextual, adaptation and distribution *rules*, does also not use the pipeline metaphor. The authors presented two versions of their tool, one for advanced users and one for domain experts or end users. The latter will be used to compare their tool to our guidelines. While rules are primarily represented textually in the main authoring environment, the second version of their tool provides a more graphical but less expressive view for creating rules. Nonetheless, this graphical view does not offer a textual representation of rules, making guideline G6 only partially fulfilled. Whereas the tool groups the triggers and actions in separate frames, the sequence of actions is not represented from left to right, making guideline G5 also partially fulfilled. It is unclear whether a simplified version of the UI creation part is offered as well, and the conformance with guideline G7 is therefore unspecified.

(a) Overview of the task composition screen of Interplay [149]



(b) Overview of interaction representation in ACCORD [183]

Figure 5.13: Cross-device solutions

Interplay by Messer et al. [149] is one of the simplest systems we have seen in related work, allowing users to issue commands through pseudo-English sentences to control connected devices in their home. Since most of the interface is in textual form only (see Figure 5.13a), and the concepts of time and conditions are not present, all our guidelines are either unsatisfied or the functionality of the guideline is not present.

Still not using the pipeline metaphor, the ACCORD editor by Humble et al. [106, 183] uses the jigsaw metaphor, to enable users to configure their ubiquitous computer environments and thus does not comply with G1. In contrast to the pipeline metaphor suggested in guideline G1, the jigsaw metaphor's expressiveness is limited by the number of sides of a puzzle piece. As shown in Figure 5.13b, in this case puzzle pieces can only have one input and one output. The interaction shown in this figure represents a push-button (the doorbell) being used to signal the webcam to take a picture and display it on the portable display. ACCORD provides a good visual overview of the interactions to the end users by showing devices and user actions, such as a "*finger pressing a button*". Guidelines G3 and G4 are thus fulfilled. Further, time is also shown graphically from left to right by left-to-right couplings of puzzle pieces. However the grouping of concurrent interactions is not supported, therefore guideline G5 is only partially fulfilled. While some textual explanation is provided below each puzzle piece in the menu, the connected pieces' configuration is only visible graphically, which makes it difficult to recall how the pieces are configured, meaning that guideline G6 is only partially fulfilled. Since ACCORD does not provide a way to design UIs, G7 is marked with a dashed circle.

Coming closer to the pipeline metaphor, the Platform Composition by Pering et al. [171] uses the *join-the-dots* metaphor, where connections between a device and a service is shown by drawing a line between them. However, these lines are not directed and thus do not show the direction of the interaction flow, which is

understandable since the lines in this case represent resource sharing. Therefore, we consider G1 partially satisfied. The tool provides a graphical overview of all available devices and their services and outlines the entire system state. However, since devices are represented as circles with their services graphically represented by icons, guideline G3 is not fully addressed. Further, users are only represented textually, making guideline G4 unfulfilled. Guidelines G5 to G7 are represented by dashed circles since the notion of time, conditional statements and UI creation are not present. Similar to Ghiani et al.'s tool [92] and ACCORD [106, 183] that we described earlier, Platform Composition includes some icons, but these are tied to some functionality, which only partially satisfies guideline G8.

Concerning guideline G2, which refer to using different arrow types for different types of interaction, it has been marked for all cross-device systems by a dashed circle, since this guideline can only be fulfilled if guideline G1 is fully satisfied, which is not the case for all these systems.

5.6.2 Authoring of Internet of Things Applications

From our background research in Chapter 2, we found that many commercial solutions allowing users to configure their smart environments where using *Event-Condition-Action (ECA)* rules as metaphor. In their original mobile user interface, the well-known IFTTT shows these rules half textually and half graphically as depicted in Figure 5.14. However, this interface changed end of 2016 when IFTTT introduced *applets* showing the rules in a more textual form with some icon next to it, making guideline G6 partially fulfilled since a visual representation of the rules is missing.

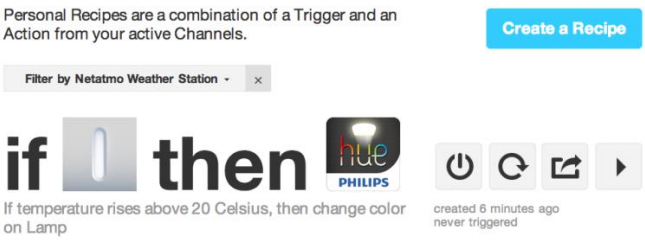
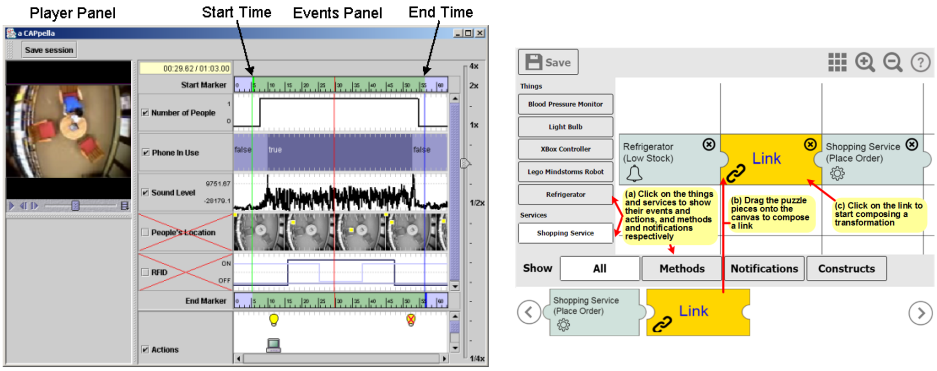


Figure 5.14: Older mobile IFTTT interface

Since device icons are shown on the bottom right of an applet, we consider guideline G3 fulfilled. While users are not graphically represented, the action of pressing a button is illustrated by an icon, therefore guideline G4 is marked as partially fulfilled. Given that time is not represented graphically from left to right, and there is no grouping of concurrent actions, as there can only be one trigger and one action, IFTTT does not fulfil guideline G5.

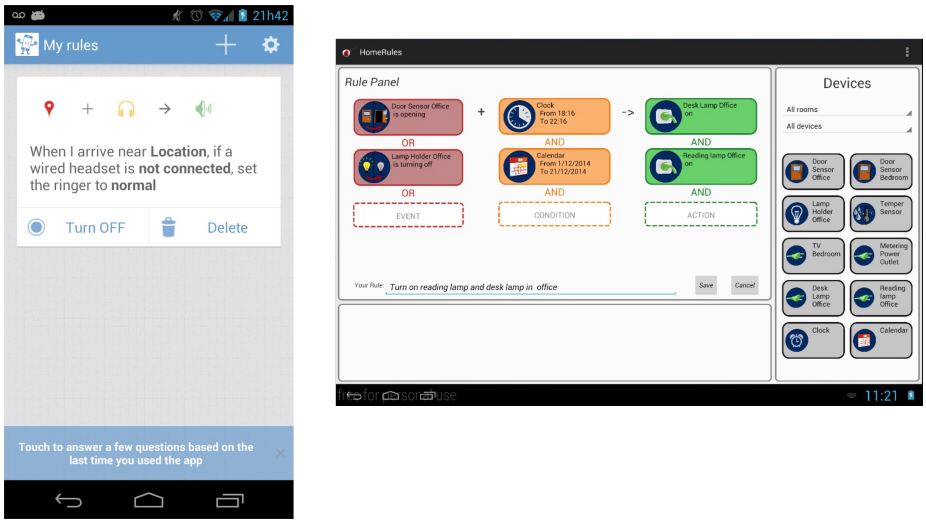


(a) Overview of the a CAPpella UI [73] (b) ViSiT's interaction representation [7]

Figure 5.15: IoT solutions

The majority of other solutions that we described in our background chapter, do support multiple triggers and actions. However, only a few of them, including a CAPpella [73], Puzzle [66] and ViSiT [7], depict the sequence of an interaction (trigger-action) as described in guideline G5 by grouping concurrent actions and showing sequential actions graphically from left to right. Figure 5.15a shows the a CAPpella user interface with concurrent actions grouped under each other in the *actions* area, in this case the icons mean that the light and the computer turns on at the same time. Actions happening later in time are placed more to the right. Figure 5.15b illustrates an interaction created using ViSiT, where a *Link* wires the *LowStock* event of the refrigerator to the *PlaceOrder* Shopping service method. This interaction is depicted from left to right. Further by selecting the *Link* puzzle piece this interaction can be detailed using *if, then, else* puzzle pieces to define the value of certain variables. When multiple variables are required, they are grouped below each other. While the *if, then, else* puzzle pieces end up being placed below each other due to place constraints, they are still meant to be interpreted from left to right, which is why we still indicated that ViSiT satisfies guideline G5.

Next to the grouping of events and actions below each other, it has been done by using the '+' symbol in Atooma, and Keep Doing It [68], as shown in Figure 5.16a. HomeRules [190] also uses the '+' symbol but to group events together with conditions, further grouping to represent interactions happening at the same time is done by placing all elements below each other, as outlined in Figure 5.16b. iCAP [74] on the other hand uses frames to group concurrent triggers and actions. The frame containing the triggers is placed on top of the one containing the actions, making guideline G5 only partially fulfilled as sequential interaction is not shown from left to right, but from top to bottom.



(a) Keep Doing It rule representation [68]

(b) HomeRules rule representation [190]

Figure 5.16: IoT solutions

We have seen that many IoT authoring tools are using the rule metaphor to represent conditional statements. Compared to IFTTT, tools often offer a more graphical representation using text and icons, such as in Atooma, HomeRules [190], ImAtHome [83], Keep Doing It [68] and Tasker. Examples of such mixed representations are shown in Figure 5.16. However, except for Keep Doing It, there is no support to see the rule either textually or graphically, which makes those tools not entirely compliant to guideline G6. Nevertheless, since all the above mentioned tools show devices in a graphical form, they all do comply to guideline G3. Further, only ImAtHome partially satisfies G4, as the user is only represented in an action, which is *coming and leaving home*, but the concept of users represented graphically on their own is missing. The other tools except HomeRules, for which no information about the representation of users was found, do neither represent users nor user actions in a graphical way.

While most IoT tools do not offer support for UI creation, IFTTT proposes a button widget that can be linked to some functionality and thus has a limited fulfilment of guideline G7. Tasker goes a step further and supports the UI creation for popup screens on mobile devices. The rest of the above mentioned tools do not provide any UI design functionality. From the non-commercial tools discussed in Chapter 2, TARE [93] is the only one supporting UI modifications and distribution, however it is unknown whether or not the tool supports UI design, therefore we marked guideline G7 as not specified. Since the authoring environment of TARE is textual rather than graphical, it does not comply to our other guidelines. The same goes for EPIDOSITE [125] which uses a textual script to show automated interactions on

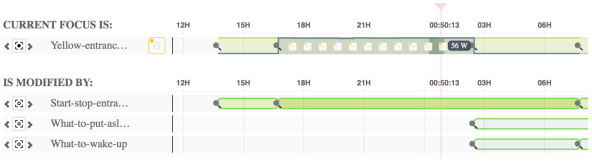
a smartphone. The end-user development interface by Kubitza and Schmidt [124] called TouchCompozr is also mainly textual with fields that can be entered trough physical demonstration for if-then-else rules, which are presented from top to bottom and thus does not comply with G5. The only graphical part of the interface is an icon showing a button press and since not much information is available about this tool, G3 and G4 are left as unspecified.

A more graphical and visual representation of the rules is presented by Bellucci et al. [19] in T4Tags, where users define trigger-actions rules by going through radial menu user interface. The chosen triggers are grouped on the left while the actions are placed on the right, hereby complying to guideline G5. Most of the user interface of T4Tags is using icons to represent devices, and device actions or states. It is however not clear how users are represented in the system during rule creation. Therefore, we marked guideline G3 as satisfied but left guideline G4 as unknown. When users create a rule in T4Tags they can assign this rule a picture to describe the goal of the rule as well as a textual description of the rule. Since the textual description is not necessarily a textual counterpart of the graphical representation of the rule, we indicate guideline G6 as partially fulfilled. Further, the pipeline metaphor is not used and UI design is not supported, so guidelines G1, G2 and G7 are either considered as not fulfilled or not supported. Since the tool uses many symbols but no free-form annotations, guideline G8 is partially fulfilled.

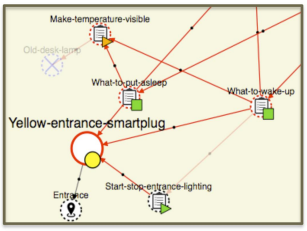
Using the *timeline* metaphor, a CAPpella [73] uses programming-by-demonstration but in contrast to TouchCompozr and EPIDOSITE, it provides a more graphical UI, as already shown in Figure 5.15a. Since a CAPpella shows a graphical representation of devices it fulfils guideline G3, but it does not fulfil guideline G4 since users are not graphically represented in the UI. Further, given that a CAPpella is based on behaviour recognition and not rules, the functionality for guideline G6 is not present. From the same first author of a CAPpella, iCAP [74] is again a rule-based system, but in contrast to previous systems, rules are represented entirely graphically and thus complies with guidelines G3 and G4. While this visual UI has been proven simple and intuitive for end users by the authors' study, it lacks some textual counterparts and therefore does not fully satisfy guideline G6.

Going for a few different metaphors, AppsGate [60] uses mainly *rules* for allowing users to control their smart homes using a pseudo-natural language but it also allows users to monitor the state of the home via *timelines* and see the relationships between devices through a *dependency graph*, as shown in Figure 5.17. Figure 5.17a shows the history of when the yellow entrance smart plug has been triggered by a program in the late evening. In this case the progam "Start-stop-entrance-light" is changing the state of the plug at the indicated time (red line), and Figure 5.17b shows the dependency graph focussing on the *Yellow-entrance-smartplug* which is located at the

entrance and is involved in multiple programs indicated by dashed circles and a play and stop symbol to represent whether the program is currently running or not. More details about these different visualisations can be found in [60].



(a) AppsGate timeline interface



(b) AppsGate dependency graph representation [60]

Figure 5.17: AppsGate user interfaces [60]

Since the timelines and graph view cannot be modified and thus have an impact on the created program or rules, it does not satisfy our guidelines G5 and G1. In the rule authoring environment, grouping is supported but interactions are read from top to bottom making guideline G5 partially fulfilled. While devices are represented as circles in the dependency graph they are not represented in a realistic graphical way, making guideline G3 unfulfilled. Guideline G4 is not satisfied as well since users are only represented textually. Concerning guideline G6, we marked it as partially compliant because there is no graphical representation of a rule on their own but rather representations of when they are triggered or running (see Figure 5.17).

Still using *rules* as a metaphor, the Context Modelling Toolkit (CMT) [212] provides a drag-and-drop environment to create *IF-THEN* rules by using the AND template in the main rule creation interface. While the main rule creation view does not make use of the pipeline metaphor, the CMT template authoring view allowing expert users to define new templates does use a metaphor similar to the *pipeline* metaphor. Figure 5.18 shows how the *"sleeping"* template is created by using this view. The different components of the template are linked to each other using undirected arrows, therefore we consider guideline G1 as partially fulfilled for CMT. Further, we indicated guideline G2 by a dashed circle since CMT does not provide directed links. Most of the UI of CMT is textual, making guidelines G2, G3 and G8 unfulfilled. CMT groups triggers and actions with the triggers always illustrated on the left and actions on the right, which is compliant to guideline G5. The toolkit shows the rules by arranging different blocks of text in a template block, but does not provide a textual counterpart. Guideline G6 is thus only partially satisfied. UI design is not supported and is therefore marked with a dashed circle.

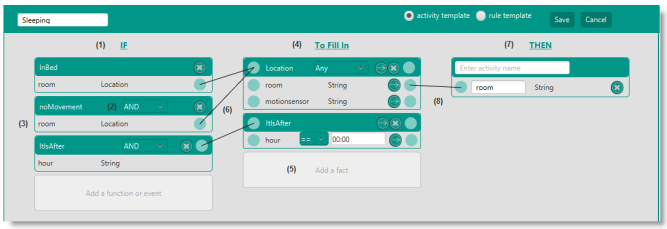
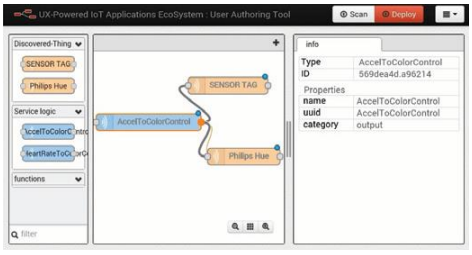
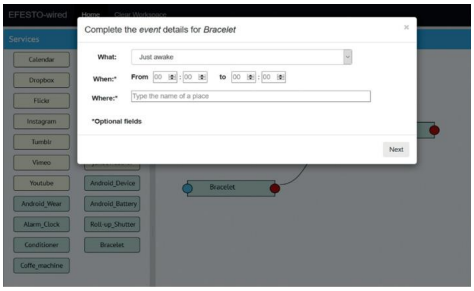


Figure 5.18: CMT template authoring view depicting the creation of the *"sleeping"* template by filling in the "room" and "hour" parameters using undirected links [212]

While most of the previously described systems use rules to express conditional statements, these tools do not make use of the pipeline metaphor to show interaction across devices, making them not compliant to our guideline G1 and thus marking guideline G2 with a dashed circle. In contrast, the authoring environment called Versatile, of Heo et al.'s IoT Mashup Application Platform (IoT-MAP) [103] does use the pipeline metaphor as main metaphor to allow users to compose IoT interactions, as shown in Figure 5.19a. The composition UI is based on Node-RED[1]. However, since there are no arrows to indicate the direction of the data flow, guideline G1 is only partially fulfilled. Guideline G2 is thus also indicated by a dashed circle. Further, Versatile's UI does not include a graphical representation of devices nor symbols, making guidelines G3 and G8 unsatisfied. The tool does not include the concept of users and UI design functionality, which results in guidelines G4 and G7 being represented with dashed circles in our table. Concurrent actions in Versatile are not grouped and sequential interaction is not shown from left to right which is not compliant with guideline G5. Finally, since there is no graphical and textual representation of the interactions, guideline G6 is unsatisfied as well.



(a) Versatile user interface [103]



(b) E-Wired popup interface [72]

Figure 5.19: IoT solutions

[1] `https://nodered.org`

The only prototype from all the discussed solutions in Chapter 2 that fully supports guideline G1 is the E-Wired authoring environment presented by Desolda et al. [72]. In order to gather information about the interaction, popup windows are used, as shown in Figure 5.19b. The arrows of this prototype represent *cause-effect* relationships and the authors do not sub-divide arrows in multiple categories to represent different kinds of interactions, making guideline G2 unfulfilled. Given that E-Wired does not include a lot of graphical elements, devices are not represented in a realistic graphical manner and symbols or icons are not present, therefore guidelines G3 and G8 are unsatisfied. Since there is no mention of a representation of the user in the tool by the authors, we marked guideline G4 as unknown. Concerning the representation of interactions, they can be grouped and represented from left to right, thereby complying to guideline G5. Further, guideline G6 is not entirely supported since conditional statements are not both graphically and textually represented.

As discussed in our background chapter, some tools such as Puzzle [66] and the Zipato Rule Creator, are using the jigsaw puzzle metaphor, which is not compliant to G1 and marks G2 as functionality that is not present. While Puzzle does not include a realistic graphical representation of devices, Zipato does, the latter thus satisfying guideline G3. It is unclear whether Puzzle has a graphical representation of users, we therefore marked guideline G4 as unspecified. Zipato's only representation of a user is a user icon, but it is unclear if users can be involved in an interaction, which is why we also indicated guideline G4 as unspecified. Puzzle fully supports guideline G5 since it allows grouping and shows the interactions from left to right. Zipato, however, shows the interaction flow from top to bottom, but supports grouping and therefore partially complies to guideline G5. Both tools do not include a textual counterpart to their graphical representation of the interaction, making guideline G6 only partially completed. Zipato does provide a textual explanation of the rule but it is up to the user to give this description and therefore does not correspond to a textual counterpart.

Lastly, we discussed the component-based web mashup approach by Koren and Klamma [120], called DireWolf 3.0.0 that integrates Web of Things (WoT) devices including their own UI components which should be configurable and draggable in the DireWolf UI view. Although the system sounds promising, little information is available on the status and use of this prototype.

As a final note, similar to the tools introduced in the previous section, symbols or icons are used by many tools but never to freely annotate the authoring environment and therefore guideline *G8* is only partially fulfilled for these tools.

5.6.3 Concluding Analysis

While some guidelines are more present in related work than others, it is quite striking that almost none of the presented authoring solutions supports our first guideline G1 promoting the use of the pipeline metaphor for the representation of interactions. This might be due to the fact that some studies have shown that this metaphor requires more cognitive effort [65, 72]. However, as illustrated by our study results, it seems to be the first thing popping into people's mind when speaking about interaction across devices. The use of colour coding as explained in guideline G1 in combination with the visual representation of devices (G3) might further reduce the cognitive effort. None of the existing authoring tools supports guideline G2, mainly because it is closely related to G1. The E-Wired prototype supports the pipeline metaphor but does not provide different arrows for different kinds of interaction. Guideline G3 is the most supported guideline by the presented tools. While devices are often realistically graphically represented, this is not the case for users, which are fully or partially represented graphically only by 4 of the authoring tools. When the concept of time is present in authoring tools, they are often at least partially compliant with guideline G5. Most tools that do not fulfil this guideline at all, either are not graphical such as EPIDOSITE [125], or tools which only support single trigger and action rules, like IFTTT and TouchCompozr [124]. Guideline G6 is almost always partially fulfilled when the functionality is supported by the tool and only 4 solutions do not fulfil this guideline at all. The reason why guideline G6 is only entirely satisfied by one solution, is because tools often offer a mix of textual and graphical representations of rules but never a way to switch between them, which is quite important since a visual representation does not include details about the rule that might be needed when reconfiguring a rule. The functionality to allow end users to create and customise their own UI (G7) is often missing in existing authoring tools, and if present it is frequently limited to widgets. Finally, guideline G8 is at least partially fulfilled by most systems. However, for none of the tools symbols and annotations can be used to simply add extra information to the authoring environment without affecting any interactions.

On a sidenote, as mentioned earlier our guidelines might still need to be refined over time. It could, for example, be interesting to perform a study with people with a different culturel background, which could lead to an extension of the guidelines. Such culture-dependent guidelines exist for user interface design in general [141], but have not been discussed much in the related work that we have investigated.

5.7 Cross-device and IoT Knowledge Analysis

We end this chapter with an analysis of people's knowledge about cross-device interaction and the Internet of Things. In order to get a better idea of the amount of smart technologies people have, we also asked them which smart devices and *things* they own. This data has been collected by asking participants of our elicitation study as well as other volunteers to fill in a questionnaire, which can be found in Appendix A.2. The questionnaire involved questions about the person's background, the smart technologies they own, their knowledge about technology, cross-device interaction and the Internet of Things. Further participants were asked about their way of transferring a picture between devices (e.g. between a phone and a computer). Note that, in the original questionnaire the last question (regarding the elicitation study) has been removed from the questionnaire for people who did not attend this study. In this section we discuss the results of the questionnaire.

In total we collected 70 filled in questionnaires of volunteers with different backgrounds. We found participants amongst friends, family and university students and staff. 35 participants had some technical background, a majority of them a computer science background, while others had different engineering sciences backgrounds. The remaining 35 participants had more diverse backgrounds not related to computer science or engineering. The questionnaire has been filled in by 39 males and 31 females.

The term Internet of Things was quite well known, as 59 participants marked that they had already heard this term. In contrast, participants were less familiar with the term cross-device interaction, as only 44 people had heard about this term. All participants however performed some cross-device interactions, as they all responded to the question related to how they usually transfer pictures from one device to another. Still a large number of people do not know this term. When looking at how people transfer pictures across devices, we noticed that the most used methods are by using an USB stick and via email, which are relatively ancient methods and are not very fast. In Figure 5.20 we illustrate the top five most popular methods for transferring pictures written down by participants. The third most popular method was by making use of an USB cable, which is also an old method. Google Photos and Google Drive have been written down by 23 participants, as people often wrote down only one of those two methods except for 3 participants that wrote both methods down. This is approximately one third of the people who filled in the survey and we notice that the majority of people still prefer to rely on older already known methods rather than using newer and more efficient ones. Relying on well-known interaction styles instead of using more effective techniques has been already identified in other studies [36, 114, 174] and represents the effects of legacy bias. Some people also "misuse" certain services to transfer their picture across devices, such as sending

the pictures to themselves using Facebook, which is done by 10 participants. Some other transferring methods included but were not limited to the use of Bluetooth (9), Dropbox (8) and WeTransfer (5). Note, that the average number of transfer methods given by participants was 2.1 methods.

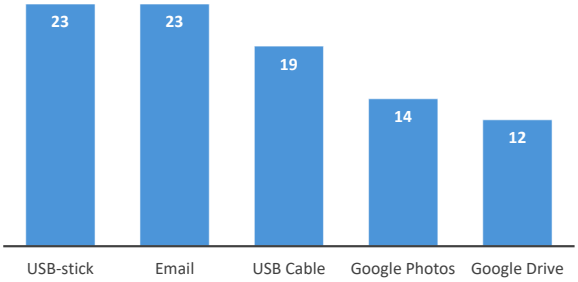
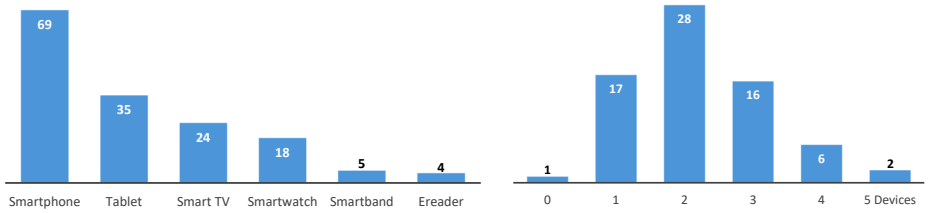


Figure 5.20: Transfer methods used for transferring pictures across devices

The average number of smart devices per person derived from our questionnaire was 2.2 devices on average in general and very similar for male and female, with 2.3 devices on average for males and 2.1 for females. These numbers are a bit lower than the numbers presented by Google in 2017, who calculated an average of 2.9 devices per person[1]. The reason behind this difference could be the kind of devices that are taken into account as "smart devices". In the questionnaire by Google when asking people which devices they use, they take into consideration the following devices: mobile phones, smartphones, computers, tablets, internet-enabled TVs, MP3 players, handheld gaming devices and eReaders. In contrast, we considered smartphones, smartwatchs, smartbands, tablets, smart TVs and eReaders as smart devices. We summarise the list of smart devices owned by our participants in Figure 5.21a and show the number of smart devices per person in Figure 5.21b. Thereby, we notice that only one participant did not own a smartphone and that half of our participants own a tablet. Smart TVs are getting popular as well, with nearly half of our participants possessing one. The eReader was the least popular one. However this could be due to the fact that people might not think of an eReader as a smart device. From Figure 5.21b we can derive that 52 participants own more than one device, this number would have certainly been higher if we would have counted a computer into our smart devices as well.

From the people who filled in our questionnaire, 28 had some kind of IoT device at home. We distinguished 11 different IoT devices, namely smart speakers, Bluetooth speakers, smart scales, cameras, Chromecasts[2], smart vacuum cleaners, smart lawn-mowers, smart thermostats and smart light bulbs. We hesitated to keep Bluetooth

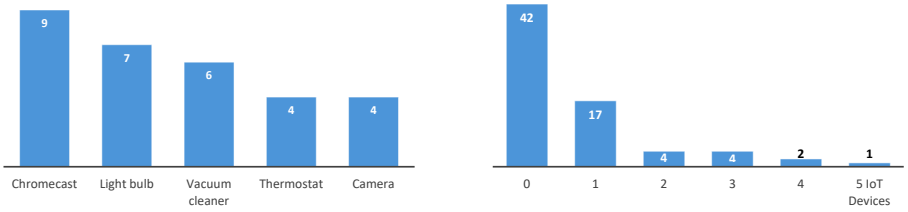
[1] https://www.thinkwithgoogle.com/intl/en-gb/advertising-channels/mobile/consumer-barometer-study-2017-year-mobile-majority/
[2] https://store.google.com/product/chromecast



(a) Smart devices owned by people (b) Number of smart devices per person

Figure 5.21: Smart devices analysis

speakers as IoT device, but since these devices could also come with some voice control, such as the Sonos speakers[1], we kept them in our list. In Figure 5.22a we show the top five most popular types of IoT devices, which includes the Chromecast, smart light bulbs, vacuum cleaners, thermostats and cameras. Figure 5.22b depicts the number of IoT device types per person. We clearly see that many of our participants do not own any IoT devices. Note that we speak about the number of IoT device types per person, not counting the number of instances per device type, given that users often have multiple instances of certain device types (e.g. smart light bulbs).



(a) Top five most popular IoT devices (b) Number of IoT device types per person

Figure 5.22: IoT devices analysis

We conclude this analysis that clearly shows that we live in a multi-device world. Many people nowadays possess at least two smart devices (74%). Further, while many of our participants do not yet own any IoT devices (60%), we believe that the number of people with IoT devices will grow as predicted by multiple studies, such as Deloitte's study [182] and a study by Ericsson [46] which was presented in Chapter 1. It is evident that people will soon need effective and unified ways to communicate with all these smart technologies in an intuitive manner as we already argued in our problem statement and research questions.

[1] https://www.sonos.com

Chapter 6

End-User Authoring Tool

Good UI design gives users a comprehensible sense of power that consistently helps them feel in control.

Jim Nielsen

In this chapter we present the result of the investigation and work done in the previous chapters by describing our end-user authoring tool prototype, called eSPACE (end-user Smart PIACE), for the design and development of cross-device and IoT applications. We choose the name *eSPACE* also as a reference to 'space' in terms of *userspace* in English or in French called *espace utilisateur*, which refers to the part of a system memory that is reserved for user processes, where user mode applications run as opposed to the kernel space, which is the part of memory where the kernel, being the core of the OS, runs, where users are not allowed to interfere[1]. This can also be seen as user mode versus supervisor mode, explained in layman's term.

This interlude aside, let us come back on topic, as everything that we have introduced in the previous chapters finally comes together in order to create our EUD authoring tool. We combined our knowledge of related work, the requirements derived from this related work and from our use case scenario, our reference framework and conceptual model together with the design guidelines that we have defined in the previous chapter, in order to design the eSPACE authoring tool presented in this chapter. The development of eSPACE is part of the answer to our last research question. We complete the answer to RQ4 by describing and demonstrating the functionality of the eSPACE authoring tool and performing an initial evaluation of the tool, which is

[1] `https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode`

detailed in the next chapter. Note that the creation of the eSPACE authoring tool also serves as a way to validate the different research artefacts that resulted from our previous research questions.

In this chapter we first present the different views of the frontend of our eSPACE authoring tool. After reviewing the front-end user interfaces, we highlight the architecture of the tool which uses the server part that has been introduced at the end of Chapter 4 and the new frontend. We further describe the implementation of the eSPACE authoring tool and finish with a section demonstrating the functionality provided by our proof-of-concept eSPACE prototype with a number of examples.

6.1 eSPACE Authoring Tool

We designed the eSPACE prototype based on our elicitation study, the presented design guidelines and the existing body of related work. eSPACE provides visual tools allowing end users to author their own XD and IoT applications. The eSPACE client runs in a web browser and thus requires no prior installation. We structured the authoring tool into four views, the *home*, *UI design*, *interaction* and *rules* view. The *home* view provides an overview of all the user-defined applications, rules, devices and other users, while the *UI design* view allows users to design user interfaces. The other two views serve for defining the XDI and IoT interaction. As suggested by Dey et al. [74] and guideline G6, our authoring environment supports both a visual and textual specification of rules or interactions. The more graphical one is presented in the *interaction* view while the more textual one is managed by the *rules* view. Lastly, the user-defined applications can be seen in the *app* view, which will retrieve the application model and interpret it in order to generate the final user interface(s). The five views together with their functionality are explained in more details in the following subsections.

6.1.1 Home View

The *home* view can be seen as a dashboard-like overview as also used in other authoring tools [60, 83, 93]. In addition, during our elicitation study, participants who own smart devices also expressed the need for a way to group their applications rather than having to switch between them to use different smarthome appliances. Participants also often started to draw an overview of the connected devices, but then continued with more specific drawings.

The *home* view is illustrated in Figure 6.1 and groups the user-defined applications, rules and devices as well as other users in different `dropdowns` (e.g. applications or rules dropdown). Applications can be created by pressing the `Add App...` button,

which will prompt the user to enter a name and description of the application. The user is then redirected to the *interaction* view where they can start designing the app. By selecting an application in the applications dropdown, users can access their user-defined applications which will be opened in another browser tab, called the *app* view. Since an application might consist of multiple FUIs for different devices, a browser tab will be opened for showing each of these UIs. Note that, in the future, an improved version of the tool could deploy the FUIs directly to the involved devices. For instance, the *grocery list* application (described in Chapter 3) consists of two FUIs, one on the smartphone and one on the fridge. With the current version of the authoring tool these two FUIs will open in two browser tabs of the device that is running the eSPACE authoring environment. In the future, it should be possible to show these FUIs directly on the smartphone and fridge. Any existing application can be modified by pressing its `Edit` button in the applications dropdown. The *interaction* view of the corresponding application will then be opened.

If a user wants to create a rule, they can press the `Add Rule...` button. A popup will then ask the user to select to which of the existing applications the new rule will belong to, or whether the rule will be part of a new application. After that, the *rules* view will be opened, where the user can start creating a new rule. Existing rules can be modified simply by pressing the `Edit` button of a rule in the rules dropdown, which will open the selected rule in the *rules* view.

Next, the devices of a user can be seen in the devices dropdown. When pressing a device's `Edit` button, a popup screen will appear where users can change the graphical representation of a device. While we added the `Add Device...` button, we did not implement the functionality to actually add new devices using the GUI, as it falls beyond the scope of this dissertation. Our authoring tool is meant as a prototype to see whether users could create interactions across smart devices and appliances by using it, and therefore does not focus on how to add new devices to the system. However, note that adding devices to the system can easily be done by developers using the RESTful API described in our implementation section. Further notice that in the authoring tool we use the term "device" to refer to smart devices as well as *things*, this term will be more understandable for end users than the term "smart technologies".

Lastly, a user can access their friends' profiles by using the users dropdown. Similar to the `Add Device...` button, the functionality behind the `Add User...` button is not yet implemented. On the other hand, the application sharing functionality is provided by pressing the `share` button, which will prompt the user with a popup asking which application the user wants to share with the selected user. It should be noted that the *home* view is accessible from all other authoring views by pressing the *app* icon in the upper left corner of any page.

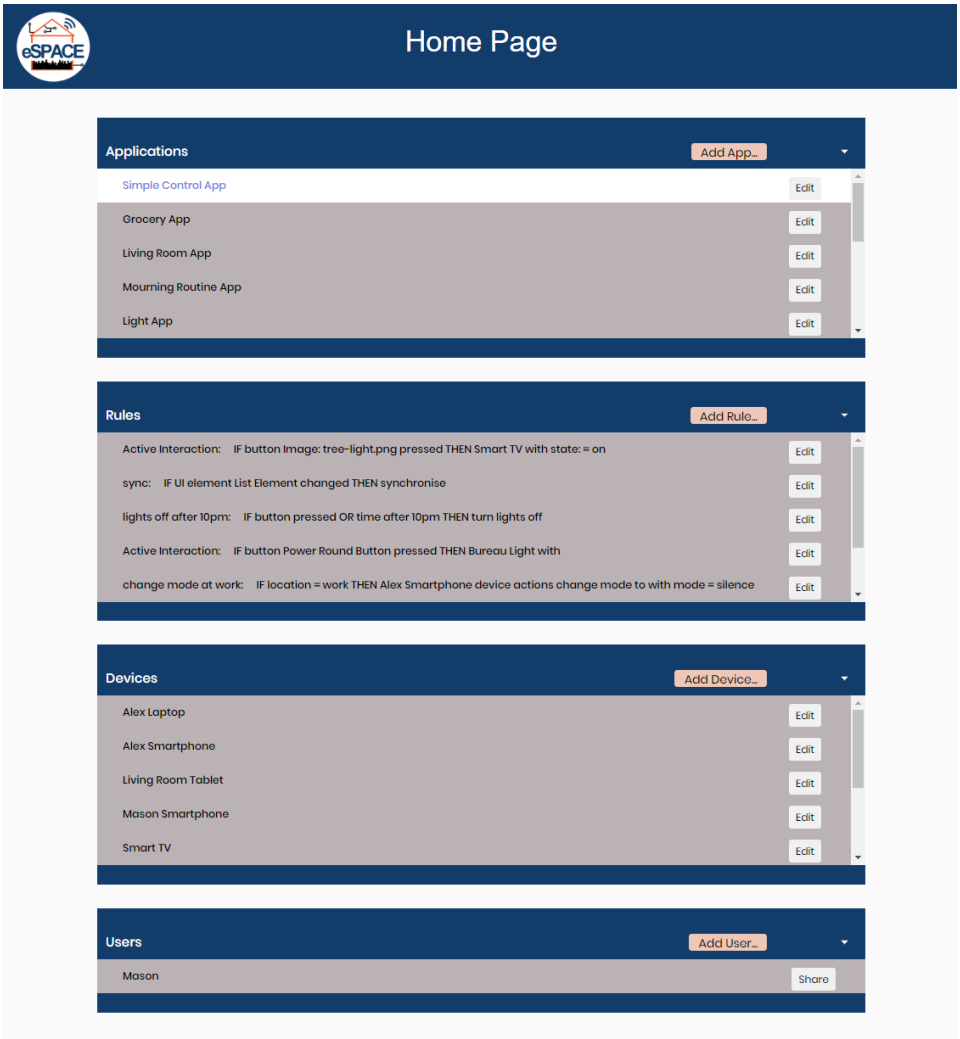


Figure 6.1: *Home* view with user cursor hovering over the *Simple Control App*

6.1.2 UI Design View

In the *UI design* view users can design simple graphical user interfaces for their applications. The design environment is depicted in Figure 6.2 with UI elements in the sidebar on the left that can be dragged and dropped on the device screen on the right. The toolbar on top of the design space allows for easy copy/paste, undo/redo, deletion and grouping of UI elements. It further provides a preview of the UI in a new web page when pressing the `Preview UI` button. In order to design a user interface, the user must first choose for which device they are creating this UI, as

seen in GUMMY [148]. The device screen, represented by a `window` element, will be illustrated according to the size of the screen of the chosen device. Note that even though the UI will be tailored for this specific device, it can still be viewed on other compatible devices. This typical drag-and-drop design environment has also been seen in the authoring tool of Ghiani et al. [92] and in Jelly [147]. While the *UI design* view allows end users to design a personalised user interface for a certain device, it only creates the *view*, in the sense that there is no functionality linked to any of the UI elements. The functionality of a UI element can be added in the *interaction* view, where users can define any interaction that is, for example, triggered via UI events (e.g. button press). Allowing end users to create their own UIs makes sense since many participants of our elicitation study drew UI elements even though none where mentioned in the given scenario that they received. Some participants further mentioned that they needed some button to press in order to trigger interaction because they wanted "*to stay in control*".

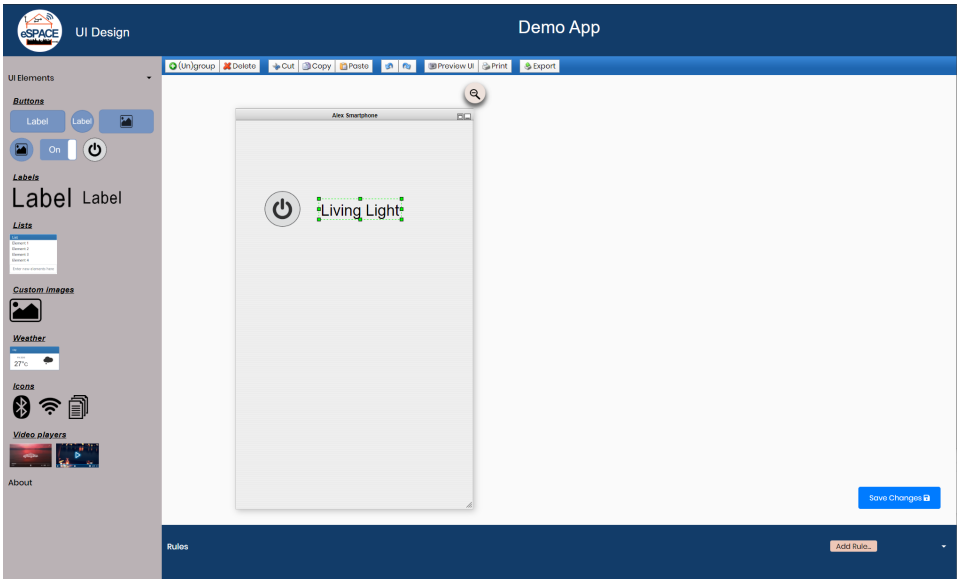


Figure 6.2: *UI design* view

6.1.3 Interaction View

The *interaction* view shown in Figure 6.3 allows end users to graphically represent interactions across devices. On the left hand side of the view, a panel is shown containing the elements that users can drag-and-drop to the authoring space on the right. The elements are grouped into five categories. The first category contains the `Devices` containing the smart devices as well as smart appliances of the user. The second category contains `Services` such as a *weather forecast* service. The next

one groups the contextual elements, comprising user, time and location. A fourth category consists of the different types of arrows to create different types of interactions between devices. Regular arrows are used to represent all actions happening between devices, except for synchronisation which is represented via double-sided arrows. Dashed arrows are further used for contextual interactions, which always have one of the contextual elements as a source. The last category includes `Symbols` and annotations. They do not offer any extra functionality, but can help end users to better remember their defined interactions by for example adding some annotations, as illustrated on the double-sided arrow.

Devices and users are represented graphically using icons. The device icons can be changed in the *home* view as explained before. Devices that are placed in the authoring space are visualised as shown in Figure 6.3. The magnifying glass in the top right corner of each device allows users to switch between this view and the *UI design* view, where they can design the UI for the selected device. When selecting an arrow in the side panel, elements that can be connected in the authoring space will be surrounded by green dots and the ones that cannot be connected by this type of arrow will be surrounded by red dots. After selecting a source element by clicking on a green dot, users can join the dots by selecting another green dot which surrounds the target device, as illustrated in Figure 6.16a. An arrow will then appear between both elements and depending on the type of arrow, a popup window (inspired by the work of Desolda et al. [72]) will ask users to fill in parameters such as the data to be synchronised or the time when the synchronisation should take place.

An example interaction could be *"when home, synchronise my pictures between my phone and my computer"*, as shown in Figure 6.3. In the authoring space depicted in this figure we see a `time` contextual element that is linked to the `smartphone` device by using a dashed arrow which represents contextual interaction. The `smartphone` is further linked to the `laptop` with a double-sided arrow that represents the synchronisation between the two devices. All interactions shown in the authoring space are also described in terms of trigger-action rules and shown in the bottom container of this view. In Figure 6.3 the rules container describes our example rule as "*When Home sync photos: IF location is Home THEN Alex Smartphone synchronises with Alex Laptop the files: C:\fakepath\pictures*". The first part before the "IF" is the *name* of the rule given by the end users when creating this rule. When clicking on a rule or on the `Add Rule...` button, users will be taken to the *rules* view which can be used to create or modify a rule. Rules defined in this view are graphically represented in the *interaction* view. The two views offer a consistent graphical and textual representation of interaction rules. Rules created in the *rules* view will be shown graphically in the *interaction* view with the elements of a rule shown from left to right.

Figure 6.3: *Interaction* view

6.1.4 Rules View

Interactions can be defined using the *rules* view as well. The way to define interaction rules in this view is depicted in Figure 6.4. The *rules* view offers a more textual representation where interaction is defined in the form of trigger-action rules.

The tile-based view illustrated in Figure 6.4 has been inspired by Ghiani et al.'s Trigger-Action Rule Editor (TARE) [93] that uses tiles to group elements belonging to the same category. When selecting a tile, the options related to the selected tile will appear in the dropdowns on the IF or THEN-side. For instance, when selecting the `Location` tile, the different location options will appear in the dropdowns on the IF-side, as shown in Figure 6.4. An overview of the current rule defined in the dropdown is shown in a sentence above the tiles. The sentence is generated based on the selected content of the dropdowns, once the user selects another option in a dropdown this sentence will be updated. We named the tiles according to the *Rule_5W* model [72]. The 5 "Ws" stand for *Which* services are involved in the rule, *Who* triggers the events and actions, *When* are they triggered and *Where*. The last "W" stands for *Why*, which is used to provide a short description explaining the behaviour of the rule. Rather than using the 5Ws as such, we renamed them as follows in order to allow for a better understanding of the rule composition by end users: `Devices` (which), `Users` (who), `Time` (when), `Location` (where) and `Description` (why). Note that `Devices` in this view refers to services and devices as done by Wisner and Kalofonos [222] who believe that end users think more easily in terms of devices. Additionally, we added saved triggers and actions to promote reuse, as done by Trullemans and Signer [212]. Similar to the IFTTT recipes, we used `IF` <`trigger_expression`> `THEN` <`action_expression`> but also allow for complex expressions using the boolean AND and OR operators. Future versions of the tool could also include the NOT operator. Users can add multiple triggers or actions to the rule by pressing the + symbol and remove them by pressing the – symbol. While users could add an unlimited amount of triggers and actions using the + symbol, our current version of the tool only supports up to two triggers and two actions.

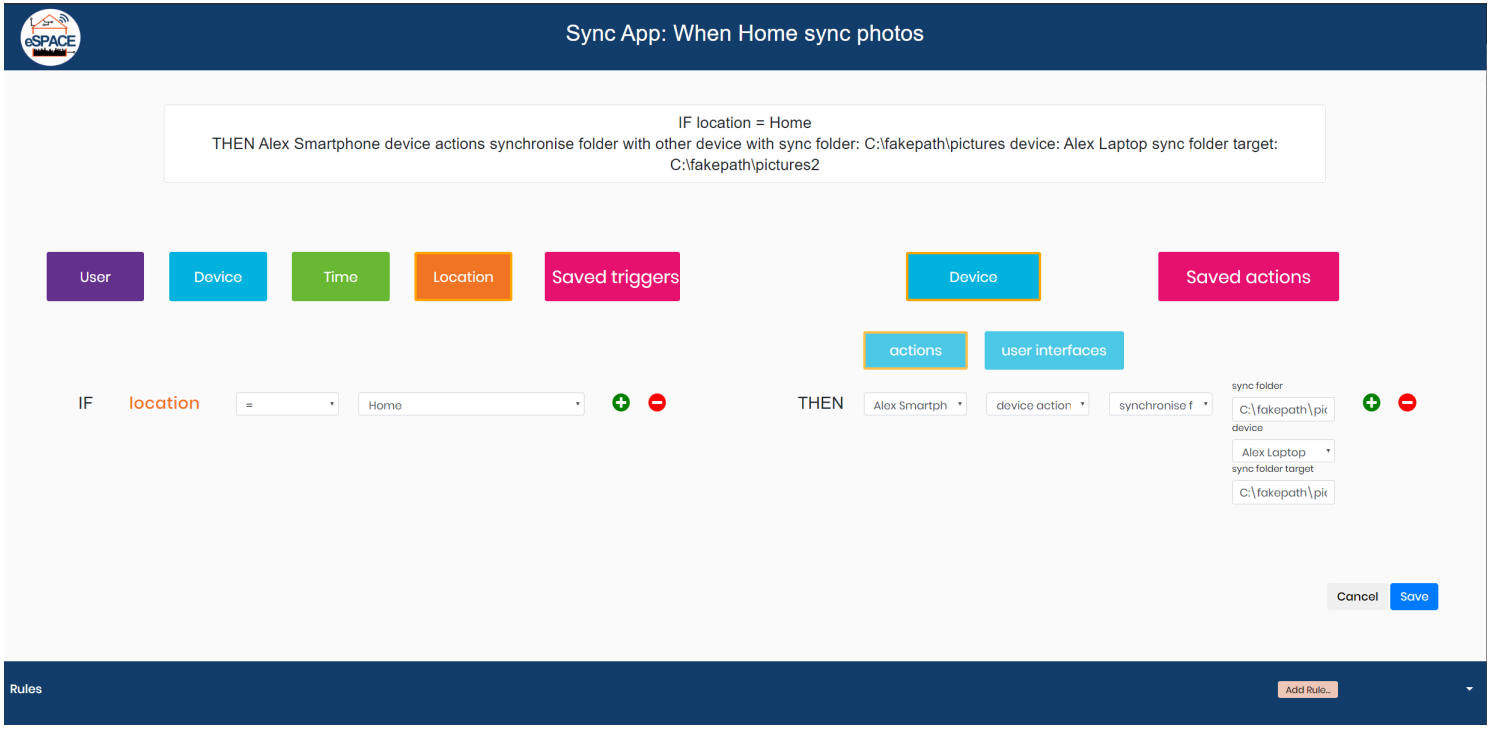


Figure 6.4: Rules view

6.1.5 App View

In this view, a user-defined final user interface will be interpreted and shown to the users. Users are directed to this view by selecting an application from the applications dropdown in the *home* view which will open a tab for each FUI involved in the selected application, as explained earlier. In Section 6.3 we explain how exactly a FUI model is interpreted by this view. An example of a simple user-defined application to control lights is shown in Figure 6.5.

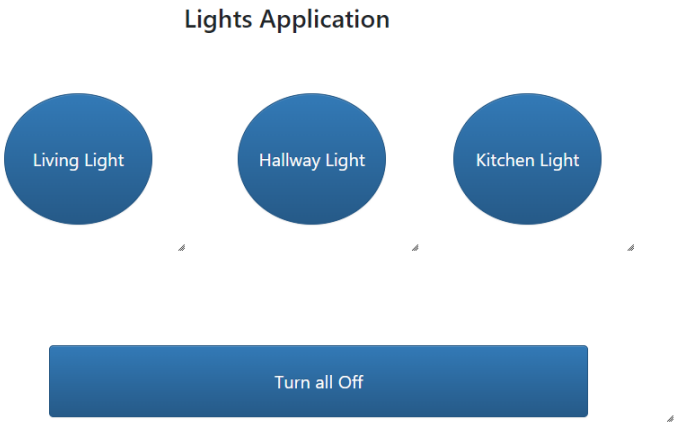


Figure 6.5: Simple lights application

By longpressing a UI element in this view, a popup window will appear, allowing users to distribute the selected element to another device on which the eSPACE authoring tool is currently running. Once the target device selected, the UI element will be added to the application that is currently running on this device. On the server side, this is done by adding the UI element to the FUI of the running application on the target device. Therefore, in order to see the distributed UI element on the target device, the user must refresh the application page. Ideally, in a future version of the tool a popup should appear asking the user of the target device whether they want to accept this distributed UI element. More detailed explanation about the implementation of such UI distribution is provided in Section 6.3.

6.2 Architecture

In this section we present the general architecture of the eSPACE authoring tool, which follows a client-server architecture and is highlighted in Figure 6.6. To summarise the figure, we see the three main components represented by the *eSPACE Authoring* (client), the *RSL link Server* and the *eSPACE Application* (client). The RSL link Server, that has been introduced in Chapter 4, serves as backend, stores

all application data and provides a way to communicate with this data through a RESTful API. The eSPACE authoring component is one of the front-end interfaces providing the authoring environment for end users and containing some authoring logic. Lastly, the eSPACE application represents the applications created by the end users via the eSPACE authoring environment. More specifically, the application logic and application view corresponds to what the user has defined as interaction and as user interface, respectively. Since all data is kept in the RSL library, both the eSPACE authoring as well as the eSPACE applications get access to the data via the API mentioned earlier. A more detailed explanation of these three components is provided in the next section.

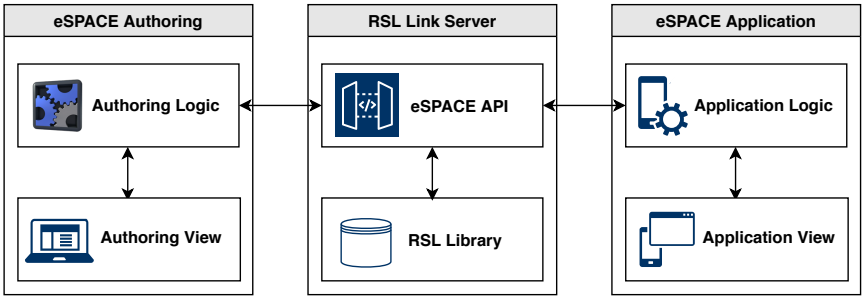


Figure 6.6: eSPACE authoring tool architecture with RSL-based link Server backend

6.3 Implementation

In order to fulfil the portability requirement (R6) we implemented the eSPACE authoring tool using web technologies, including JavaScript, HTML5, CSS3 and some additional JavaScript libraries (e.g. canvasutilities[1] and the mxGraph diagramming library[2]). The resulting applications created by the end users are also in the form of websites, which can be opened on any device supporting a web browser. In this section we provide more details on how our end-user authoring tool has been implemented.

6.3.1 eSPACE Authoring Views

In the eSPACE authoring environment end users can design their own cross-device and IoT applications by using the different views described in Section 6.1. Each view has their corresponding HTML page, an individual stylesheet, in the format of an external CSS file, for defining the look and feel of the page, and some JavaScript libraries for providing the functionality of the page. In addition, Bootstrap[3] and CSS Grid Layout[4]

[1] `http://dbp-consulting.com/scripts/canvasutilities.js`
[2] `https://jgraph.github.io/mxgraph`
[3] `https://getbootstrap.com`
[4] `https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout`

are used to manage the layout and structure of the pages. Retrieving data from the server is done by using the jQuery API[1] for sending requests to the server and parsing the incoming data. In order to orchestrate all incoming data, we further use JavaScript Promises[2]. A simple example of the use of promises is depicted in Listing 6.1, which illustrates some of the promises required for filling the dropdowns of the *home* view. The current version of the eSPACE authoring tool does not provide a login screen and for now, we always have user *Alex* logged in with id `userID`. Therefore, to fill the dropdowns with all the applications and rules this user has access to as well as the devices the user owns, we make a request to the RSL link Server to only get all their accessible entities. Note that the interaction rules defined in the authoring tool are stored as components of a `DComp` with a trigger and action component in our underlying model. This is why line 9 in our listing depicts a post request to get all entities of type `DCompResource` of the user with id `userID`.

```
1  let promiseApps = new Promise( (resolve) => {
2    //get applications of user
3    $.post(url, {query: JSON.stringify({"command":"
         getAccessibleEntitiesByType", "parameters":{"id": userID, "type": "
         ApplicationResource"} } )}, function (data, status)  {
4      resolve(JSON.parse(data).result);
5    });
6  });
7  let promiseRules = new Promise( (resolve) => {
8    //get rules of user
9    $.post(url, {query: JSON.stringify({"command":"
         getAccessibleEntitiesByType", "parameters":{"id": userID, "type": "
         DCompResource"} } )}, function (data, status)  {
10     resolve(JSON.parse(data).result);
11   });
12 });
13 let promiseDevices = new Promise( (resolve) => {
14   //get devices of user
15   $.post(url, {query: JSON.stringify({"command":"
         getAccessibleEntitiesByType", "parameters":{"id": userID, "type": "
         DeviceResource"} } )}, function (data, status)  {
16     resolve(JSON.parse(data).result);
17   });
18 });
19 ...
20 Promise.all([promiseApps, promiseDevices, promiseRules, promiseUsers]).
       then(values => {
21   ...
22   //fill dropdowns
23   fillContainers(document.getElementById("appsSubmenu"), applications, "
         interaction-design.html", "app");
24   fillContainers(document.getElementById("devicesSubmenu"), devices, "
         devices.html", "device");
25   fillContainers(document.getElementById("rulesSubmenu"), rules, "rules.
         html", "rule");
26   fillContainers(document.getElementById("usersSubmenu"), users, "users.
         html", "user");
27 });
```

Listing 6.1: JavaScript Promises example of home view

[1] https://api.jquery.com
[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

In the remainder of this section, we will shortly mention some implementation details for each of our more complex views. We therefore do not describe the *home* view, since this view did not require additional JavaScript libraries nor particularly challenging coding.

UI Design View

The *UI design* view has been implemented using the mxGraph JavaScript library, which is a diagramming library using SVG and HTML for rendering. A popular production grade example that has been implemented using this library is draw.io[1]. While we could reuse most of the toolbar's original functionality (see top of Figure 6.2), we had to define custom UI elements for filling in the left sidebar since pre-defined elements mostly included geometrical shapes (e.g. rectangles and circles). Listing 6.2 shows how a part of the UI elements have been defined using mxGraph.

```
1  function CustomStyles() {
2      this.labelButton = 'laBtn;perimeter=ellipsePerimeter;shape=label;
           rounded=1;align=center;verticalLabelPosition=mxConstants.
           ALIGN_MIDDLE;verticalAlign=middle;spacingTop=4;labelColor=#fff;
           fillColor=#337ab7;strokeColor=#2e6da4;spacing=0;spacingLeft=0;
           spacingTop=0;fontColor=#fff;fontSize=16;fontStyle=0';
3      this.label = 'label;perimeter=rectanglePerimeter;shape=label;align=
           center;fontSize=50;autosize=1';
4      this.image = 'image;perimeter=rectanglePerimeter;shape=image;
           ImageAlign=center;image=img/img.png';
5    ...
6  }
7  ...
8  // adding UIes to sidebar
9  Sidebar.prototype.init = function() {
10   this.addSection("Buttons");
11     this.addVertex('images/UIe/labelButton.png', 100, 40, this.
           customstyles.labelButton, 'Button with label', this.customlabels.
           UIe1);
12     ...
13     this.addSection("Labels");
14     this.addVertex('images/UIe/labelBlack.png', 100, 40, this.customstyles
           .label, 'Text field', this.customlabels.UIe5);
15   ...
16     this.addSection("Custom images");
17     this.addVertex('images/UIe/img.png', 100, 40, this.customstyles.image,
            'Image', this.customlabels.UIe7);
18 }
```

Listing 6.2: JavaScript mxGraph example for the creation of custom UI elements

When the `Save Changes` button is pressed in this view, each element on the drawing canvas—which represents a device's screen—will be saved as a UI element in the RSL link Server. Further these UIes will be added to the FUI of the device for which the UI is being designed using a DComp as shown in Figure 6.9.

[1] `https://www.draw.io`

Interaction View

For the implementation of the *interaction* view, the `connectingLine.js`[1] jQuery plugin, has been used and modified. The plugin is used for the *joint-the-dots* functionality when linking elements on the authoring canvas. As explained earlier, when an interaction arrow is selected in the sidebar, the elements on the authoring canvas that can be linked using this arrow will be surrounded by green dots while elements that cannot, will be surrounded by red dots. By selecting a green dot surrounding an element, this element will be set as source of the interaction and the elements which can be used as target will be surrounded by green dots. When the user selects the dot surrounding the target element, the interaction arrow will appear between the two elements. In order to draw the different types of arrows between elements on the authoring canvas, we combined both this `connectingLine.js` jQuery plugin with the `canvasutilities.js`[2] JavaScript library.

When two canvas elements are linked using an interaction arrow, a popup window appears asking users for information about the interaction they want to define between these two elements. Figure 6.7 depicts the popup window that will appear when a regular interaction arrow is used to link a smart TV and a smart light bulb. The

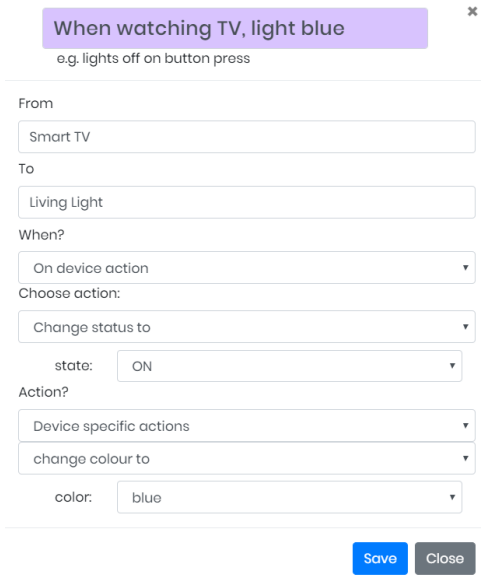


Figure 6.7: Example popup when connecting the smart TV to the living room smart light bulb via an interaction arrow

[1] `https://github.com/gunjankothari/jquery.connectingLine`
[2] `http://dbp-consulting.com/scripts/canvasutilities.js`

`From` and `To` form elements are pre-filled with the name of the source and target devices. The `When` dropdown is meant to ask users when they want a certain action to take place, this corresponds to the trigger of the interaction. The dropdown will give the following choices to the user: `On device action` or `On button click`[1]. Depending on which option is selected the next dropdown will either show the possible device actions and the parameters needed for these actions or the UIes available on the source device. Note that, to make UIes available, one needs to create a UI for the source device in the *UI design* view, otherwise this dropdown will be empty. The actions of a device correspond to the `ACs` which were defined in Chapter 4. All devices and services have their own actions that they can perform, with different parameters to be filled in. Figure 6.7 shows that the user selected the `On device action` as trigger, with the action being `Change status to` with parameter `status ON`. After selecting the trigger, users can define the action by choosing either an action of the target device or an action of a third-party service. If the user chooses the service, the actions available for this service will be shown to the user in another dropdown menu and when pressing the `save` button, a new arrow will appear between the target device and the selected service. In our example popup, the user selected a device-specific action, which is the `change colour to` with the parameter `colour: blue`. Lastly, as can be seen in Figure 6.7, users can give a name to the interaction (in the top part of the popup). If no name is provided, it will simply get the default name depending on the kind of interaction, such as "contextual interaction" for interactions involving a contextual element. Note that the trigger and action must be compatible. Therefore whenever a trigger is selected, only compatible actions are shown to the user. The compatibility check is done on the server side by matching the input and output slots, as explained in the following sections.

When the `Save Changes` button is pressed, the different elements on the authoring canvas are saved into the `canvaselements` and `canvaslines` properties of the application for which the interactions are designed. Further, for each screen device, a `FUI` is linked to the application if a UI has been designed for this screen device by the user. `UIes` are saved in a `DComp` which is attached to the `FUI`. Each interaction is saved into a `DComp` that consist of a trigger and action component. In Figure 6.8 we illustrate how our example popup data will be saved into the database, we left out some metadata as well as the ACs' parameters for clarity since these were already illustrated in the Chapter 4 (see Figure 4.18). As users might forget to save before going to the *UI design* view by pressing a magnifying icon next to a device, we also save all interaction to the database before redirecting users to the *UI design* view.

[1] This is the only implemented UI action, as it is the most used one

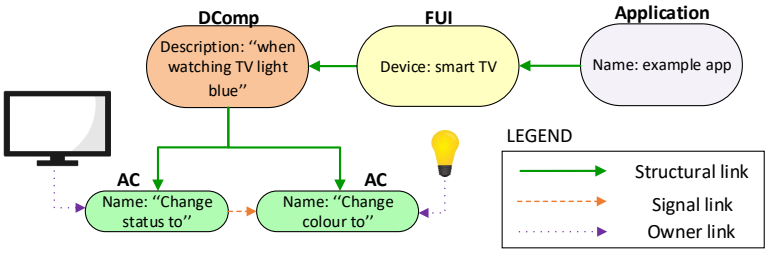


Figure 6.8: Example interaction saved as model into the RSL link Server (simplified)

Rules View

The *rules* view does not make use of any additional third-party JavaScript libraries. The view is mainly implemented by using Bootstrap for the layout of the different tiles and the dropdowns that have to be filled in to create a rule. One of the most challenging parts of the implementation of this view is filling in the different dropdowns when a user wants to view or modify an existing rule. In contrast to the *interaction* view, interaction rules in the *rules* view can be modified and can contain multiple triggers and actions. The current implementation can recognise two triggers and two actions. However, in anticipation of an extension of our tool, we already allow users to add more than two triggers or actions using our *rules* view. These additional triggers or actions will be ignored when saving the rule in our current version of the authoring tool. Adding the possibility for rules with more than two triggers or actions is not only challenging to show in the *interaction* view but also represents challenges for the interpretation and execution of these rules in the *app* view. Given that Dey et al.'s study [74] showed that users usually do not define very complex interactions, we did not deem it necessary to implement this functionality in our initial proof-of-concept prototype.

As previously mentioned, a preview of the rule is shown on top of the tiles which is generated by parsing the content of the dropdowns. Note that this preview "IF-THEN sentence" could be made more readable in future versions of the authoring tool. Finally, when a user saves their rule, the same *methods* are used than for saving the interactions in the *interaction* view. All interactions that are either defined by using an arrow in the *interaction* view or by using IF-THEN sentences in this *rules* view, are saved as components of `DComps` of a `FUI` belonging to a specific application. The user-defined applications that have been built according to the eSPACE reference framework and by using the concepts introduced in the eSPACE conceptual model can then be viewed in the *app* view. This view will interpret a FUI model in order to build the user-defined application as explained in Section 6.3.3.

6.3.2 RSL Link Server

The RSL link Server and library has been introduced at the end of Chapter 4. There, we explained how the different components of our eSPACE model and framework are stored as RSL entities via the RSL library. We further described the creation of an eSPACE microservice providing specific functionality to create, modify and manage these different model components and that is intended to be used by the authoring tool. The microservice's functionality is accessible through the RESTful API of the RSL link Server. An example of some specific functions are `updateFUIofDComp()`, `getParasofAC()` or `setOwner()`. As mentioned before, such functions allow the authoring tool to make less requests to the server when requiring complex information. Next to functions for managing and accessing the data needed by the authoring environments, we also added some functions related to UI distribution among screen devices which are currently represented by different *app* view browser tabs. Given that an *app* view tab displays a certain `FUI` of an application, whenever this view is opened, we add the `FUI` to a list on the server. This list will contain all FUIs that are currently opened, whenever an *app* view tab is closed, the corresponding FUI will be removed from this list. We called the list of running FUIs the `subscribers` list. When a user wants to distribute a UI element to another device, a request is made to the server to return this list. The entire UI distribution process is explained in more details in the *UI Distribution* subsection of Section 6.3.3.

6.3.3 eSPACE User-defined Application

An eSPACE application is an application created by the end user with the help of our authoring tool. Once a user wants to test or deploy their application, the authoring tool will create the user-defined application based on the underlying application model resulting from the users' UI design and interaction design in the authoring environment. The generated application is a web application that is optimised for the device for which it has been designed for by the user, but can of course be opened on any other device supporting a web browser. Device optimisation is currently limited to UIs made for a specific device screen size. In this section we first explain how a user-defined application is created from an application model. Then we explain how UI distribution can be performed in an *app* view at the implementation level.

App Creation Process

As explained before, we follow the layers that we defined in the eSPACE reference framework to structure a user-defined application, therefore an application consists of several FUIs which in turn contain DComp elements that are made up of at least one UIe or AC. An *app* view corresponds to the `FUI` of an application model.

Therefore, whenever a user-defined application is opened via the *home* view, our eSPACE authoring tool will open as many *app* view tabs as there are FUIs in this user-defined application. The *app* view will be given the `fuiID` and `appID` in order to retrieve and interpret the FUI model of the corresponding application. This is done by the *application logic* component shown in Figure 6.6 which interprets the FUI model in order to build the view and interaction of the *app* view. Throughout this section we will use the example FUI model illustrated in Figure 6.9 to explain how such a model is interpreted in the *app* view.

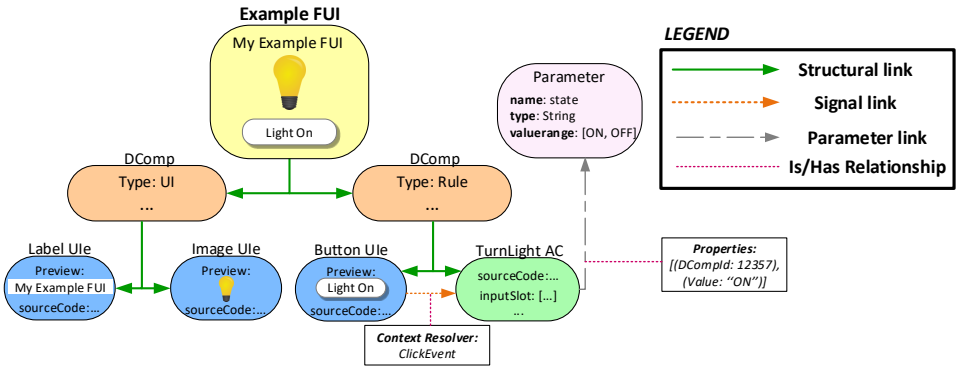
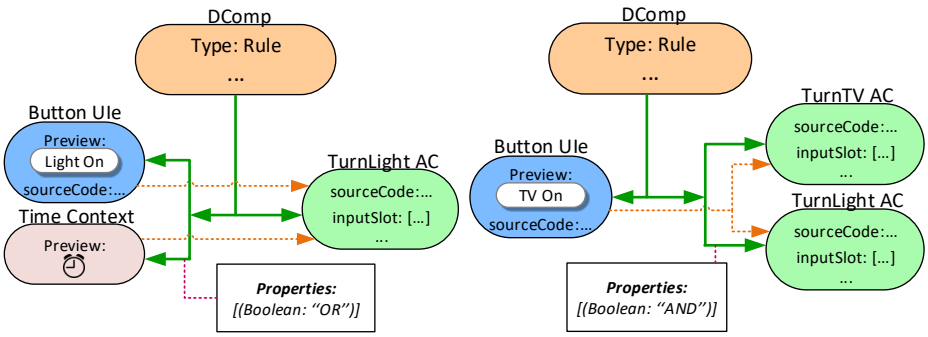


Figure 6.9: Example FUI model

We illustrate the flowchart of the model interpretation process in Figure 6.11. The *app* view will start by requesting the `DComps` of the given fuiID from the server. A DComp can either contain a number of UIes that are not tied to any functionality or it can consist of the trigger(s) and action(s) of an interaction rule. We differentiate the DComps of the first category as DComps of type `UI` and the second category as DComps of type `rule`, as depicted in Figure 6.9. For each DComp of type `UI` we simply place all its UIes on the HTML page. This is done using the UIe's source code. All UIes and ACs have a property `sourceCode` which contains a JavaScript class with either a function `displayUI()` for UIes or a function `execute()` for ACs. In Figure 6.9 we left out the `Parameter` links of the UIes for clarity reasons, but left the one for the `TurnLight` AC as a reminder of how parameters and their values are modelled. Both the `Label` and `Button` UIe, for example, have the parameter `label` with the value "My Example FUI" and "Light On" respectively.

For DComps of the type `Rule`, we first verify whether the rule has multiple triggers or actions. This type of DComps always have a structural link with two targets, the first target points to the trigger(s) while the second one points to the action(s) of a rule. If the first target points to a structural link, it means the rule has multiple targets, as shown in Figure 6.10a. If the second target points to a structural link, then the rule has multiple actions, as illustrated in Figure 6.10b. We then need to get the target

of these structural links to retrieve the actual triggers and actions, once this is done, we push all triggers and actions into two separate array, the `triggersArray` and the `actionsArray`.



(a) DComp with multiple triggers (b) DComp with multiple actions

Figure 6.10: DComp of type `Rule` simplified model examples

We then iterate over the `actionArray` in order to retrieve the JavaScript class of each AC, add the class to the HTML page and pushing it with the correct parameters in the `acCodeArray`, which will contain all AC classes after this iteration. Note that the current version of the authoring tool does not support the `NOT` boolean operator for combining different triggers or actions but only the `OR` and `AND` operators are supported. Next, we iterate over the `triggersArray`, depending on the type of trigger, a different trigger function will be pushed into the `triggerCodeArray`. With our current implementation we keep our contextual information on the client side, which means that the *app* view verifies whether a certain context is true or not. Whenever this `app` view is closed the rule will become inactive. In order for rules to stay activated we would need to verify contextual information on the server side, possibly by using a rule engine.

After filling in the `acCodeArray` and `triggerCodeArray`, we verify whether the interaction rule contains multiple triggers. If this is not the case we simply combine the trigger code with the execution of the ACs in the `acCodeArray` by verifying the context resolver on the signal link. Listing 6.3 illustrates how the `DComp` with type `Rule` of our example FUI model would be interpreted. Since the trigger is a button UIe's click event, we link a click listener to the `button` UIe using its id[1] and call the `execute()` function of all ACs. In our example, this code snipped would only call the `execute()` function of the `TurnLight` AC.

[1]UIes placed in the HTML page will get the same id as their database id

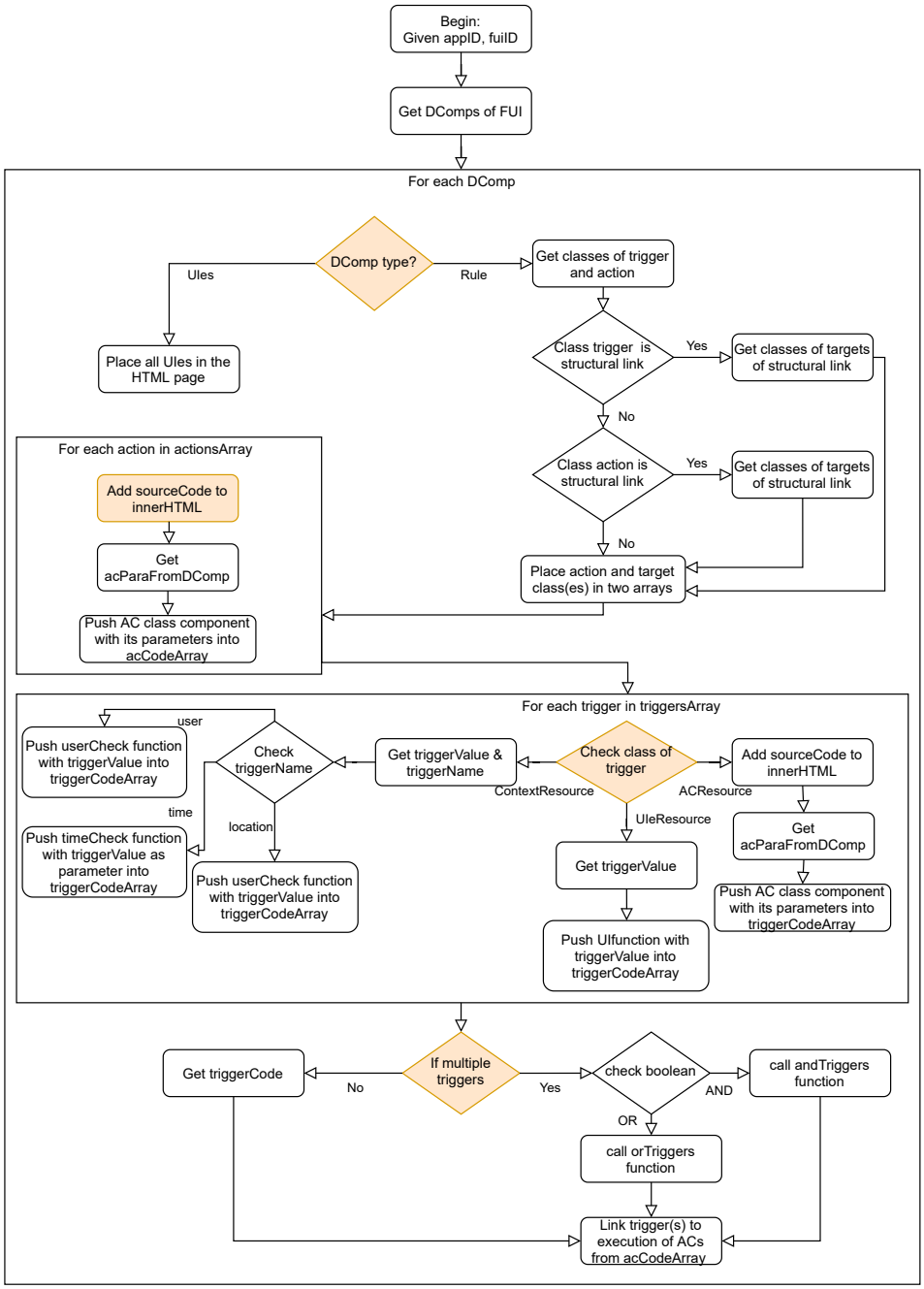


Figure 6.11: Flowchart of the *app* view implementation with the starting block of each iteration marked with an orange background for clarity

```
$("#" + trigger[i].id).click(function () {
  //executes all ACs
  for (let j = 0; j < acCodeArray.length; j++) {
    let paras = acCodeArray[j].paras;
    acCodeArray[j].execute(paras);
  }
});
```

Listing 6.3: JavaScript code snippet for executing actions on button click

If the interaction rule contains multiple triggers, we verify whether these are linked with an `OR` operator or *AND* operator. Depending on this information different code will be produced for the triggers. For an `OR` operator different signal links will be present as shown in Figure 6.10a. We will then follow each of them and end up with similar code as for single target interaction rules. Lastly, if the triggers are combined with the *AND* boolean, nested code will be created based on the different triggers. More explanation about signal links is given in the next subsection.

Functionality Linking Process

In this subsection we shortly explain how we use `Signal Links` to trigger certain actions depending on a specific event. The event could be a contextual event depending on time, location or user, but it could also be a user-initiated event such as a button click. We further also explain the compatibility check that is performed before linking a trigger and an action with a signal link. We based ourselves on the *signal and slot* architecture to verify whether a trigger and an action can be linked. The principle of this architecture is to link a signal with a compatible slot. A signal is emitted on state change of an object and a slot is the function that is called whenever the signal is received. As an example we take a button UI element as trigger of the action "*turn light on*", as illustrated in Figure 6.12. A button UIe can emit multiple signals including a signal as result of a ClickEvent, LongPressEvent, or some other button related events[1]. The `execute()` function of an `AC` represents the slot. In order to link a signal with a slot, the signature of the signal must match the one of the receiving slot. Whenever a user selects a trigger in a popup of the *interaction* view or in a dropdown of the *rules* view, a request is sent to the server to look up the compatible ACs. These ACs will appear as possible options that a user can select as action of an interaction rule. In order to find these compatible ACs, the server will verify which signals can be sent by the trigger and verify which ACs can "accept" these signal events. An AC that accepts a signal from an `Event` can be linked to any other events, since it is the most generic event. However, an AC that accepts a signal only from a `showEvent` can only be linked to a component emitting a signal from a `showEvent`. Each AC has a property `inputSlot` containing a list of events

[1]Only ClickEvents are supported in current version of our authoring tool

that can be accepted by this AC. Next, an AC also has a property `outputSignal` containing a list of all signals that this AC can emit. For example, the `turnLight` AC emits a *lightOn* and *LightOff* signal. These two same properties are also present in each `UIe` and `ContextResource`. Figure 6.12 shows that the `ClickEvent` signal is linked to the `turnLight` AC slot. In the previous subsection, we have seen that this means that a listener will be created with as function to be executed when the `click` event is triggered, the `execute()` function of the `turnLight` AC, by using the code snippet shown in Listing 6.3.

Further, an interaction rule might contain multiple triggers or actions. If multiple triggers are combined using an `OR` boolean, it will treat the interaction rule as multiple separate interaction rules, which means that for each trigger a signal link will be created with as source that trigger and as target the action of the interaction rule. An example of such an interaction rule is depicted in Figure 6.10a. On the other hand if the `AND` boolean is used to combine the multiple triggers, only one signal link is created. The signal link will have as sources the triggers and as target the action of the interaction rule. This signal link will further contain multiple context resolvers—one per trigger—to indicate which signal event is selected for each trigger. An interaction rule can also have multiple actions, if this is the case a signal link is created with as source the trigger and as targets the actions, as shown in Figure 6.10b.

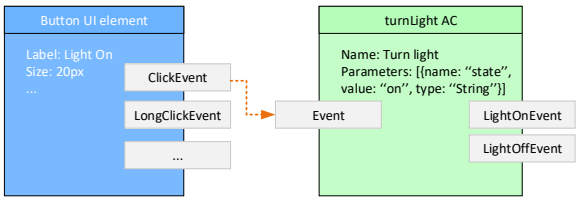


Figure 6.12: Linking of a `button` UI element and `turnLight` AC based on signal and slot architecture

UI Distribution Support

As a last part of the eSPACE user-defined application functionality, we would like to discuss how UI distribution is supported. We briefly mentioned in Section 6.3.2 that the RSL link Server is keeping track of every *app* view that is opened and manages a list of all running FUIs called `subscribers`. In the *app* view, all UIes contained in a DComp with the distributable property set on *true* can be distributed to another *app* view. Note that the distributable property of all DComps are set on *true* by default in our current prototype. In future versions of our tool end users should be given the possibility to decide whether or not a UIe can be distributed while creating their user interfaces. In order to distribute a UI element from the *app* view, the user has to perform a *longpress* on this element. When the user performs this action, they

will be prompted with the names of the available devices together with the names of their opened applications (i.e. *app* views) to which the UI element can be distributed. These device names are derived from the list of FUIs that are kept on the server. Concretely, the distribution of a UI element happens as described by the sequence diagram shown in Figure 6.13. When the *app* view detects a longpress, it will send a request to the server, asking which devices are currently using the eSPACE *app* view. The list is shown to the user in a popup, where users can select the target device and application to which they want to distribute the chosen UI element. Notice that a device can have multiple FUIs or thus *app* views opened at the same time. Once a target device and FUI have been selected, the *app* view will send a request to the server containing the `ids` of the UIe that needs to be distributed, the FUI of the *app* view and the target device with its target FUI. The server will then add the UI element to the FUI that is currently running on the target device, by first verifying to which *DComp* it belongs to on the source device and add this *DComp* to the target device's FUI using a `structural link`. By doing so the UIe also maintains its functionality on the target device. This means that if the UIe is a button that can be used to turn on the light, it will still have this ability once distributed to the target device. In order to see the new UIe on the target device the user will have to reload the *app* view on the target device.

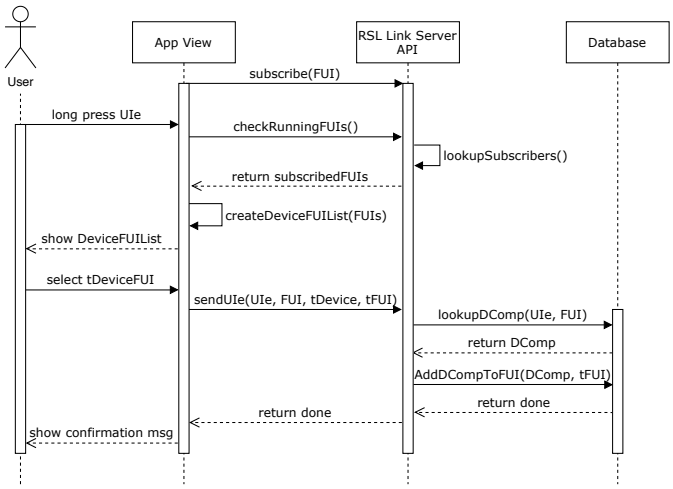


Figure 6.13: Sequence diagram representing the UI distribution

Note that an improved version of the eSPACE authoring tool should use an event system for the UI distribution by, for example, using WebSockets to push information from the server to the clients, hereby avoiding a page refresh to see the changes. Further, a popup should also first ask the user of the target device if they want to accept the distributed UIe before automatically adding the element to their FUI.

6.4 Design Discussion

In this section we describe how our eSPACE authoring tool fulfils all the design guidelines introduced in Chapter 5. We therefore briefly explain how each guideline is addressed by using some examples.

G1: Use Pipeline Metaphor to Represent Interactions This guideline is fulfilled in the *interaction* view, where users can connect smart devices, IoT appliances, context elements and services using arrows. When selecting an arrow, the elements that could serve as source of that arrow are surrounded by green dots while the ones that are incompatible will be surrounded by red dots. The same is true when selecting a target element. Finally, as done in the E-Wired prototype [72], we use popup windows to gather more data about the interaction.

G2: Use Different Arrow Types for Different Interaction Types Still in the *interaction* view, we followed this guideline by presenting different arrow types for different interaction types. Regular arrows are used for interaction between all kinds of devices and services. Dashed arrows are used for contextual interaction, when linking a contextual element to any device or service. Hereby, we also fulfil the optional differentiation between user-initiated triggers (regular arrows) and contextual triggers (dashed arrows). Lastly, double-sided arrows represent synchronisation between data or UI elements of two smart devices, as recommended in this guideline and illustrated in Figure 6.3.

G3: Provide a Realistic Graphical Device Representation We followed this guideline by representing devices by using realistic icons for the default device representation in the *interaction* view. As explained before, this representation can be changed in the *home* view. A more abstract representation of a device screen is shown in the *UI design* view, which could be more realistic by adding a more device-like frame instead of a simple `window` element (see "Alex Smartphone" screen representation in Figure 6.2).

G4: Provide a Graphical Representation of Users All users are represented through a *user* symbol allowing users or groups of users to be part of a contextual interaction or rule. Accordingly, we fulfill this guideline as well. Although more graphical user interaction representations, such as having an icon showing a user pressing a button, could be added in the future depending on the outcome of a usability study. These kind of icons have, for example, been added in ACCORD [183].

G5: Represent Sequential Interactions from Left to Right and Group Concurrent Interactions When creating a textual rule of an interaction in the *rules* view, the corresponding interaction will be created in the *interaction* view, in order to get a more visual representation. The trigger elements of the interaction are then always represented on the left while action elements are displayed on the right. If multiple triggers are defined they will be shown under each other in the *interaction* view. The same is true for an interaction rule containing multiple actions, as shown in Figure 6.14. However, when users are creating their interactions in the *interaction* view, they could place the trigger and action element anywhere on the canvas. We do not enforce the placement of triggers on the left-hand side and actions on the right-hand side, which might be needed in the future to avoid confusion when, for example, sharing interaction rules with other users.

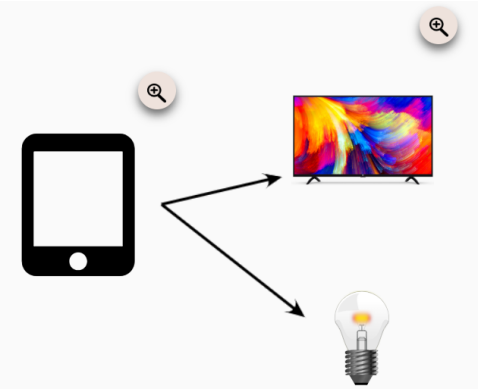


Figure 6.14: Example interaction of a tablet which triggers the TV to turn on and the light to turn off

G6: Provide Textual as well as Graphical Representations for Conditional Statements By creating the *interaction* and the *rules* view, we fulfilled this design guideline. As the *interaction* view shows a graphical representation of interaction rule or conditional statements using the pipeline metaphor, while the *rules* view provides the textual representation. Note that another textual representation of the rules can be found in the *home* view in the `rules` dropdown, and in the *UI design*, *interaction* and *rules* view in the bottom container.

G7: Support UI Design Through the *UI design* view end users can create their own user interfaces for a certain target device. Further the UI elements that are used in these interfaces can be used as trigger of an action, as explained in the previous sections. We therefore fully satisfy this guideline.

G8: Use of Symbols and Annotations Using the `symbols` dropdown in the sidebar of the *interaction* view, users can drag and drop different symbols or labels that will serve as extra annotations in the authoring space. As specified in this guideline, these symbols and annotations will not influence the interactions defined in the authoring environment but rather serve as "notes" for the end users to better remember the interaction that they defined. Further studies of the authoring tools could point out recurrent symbols that could then be added automatically when defining a certain interaction rule.

Our initial eSPACE end-user authoring prototype for cross-device and Internet of Things applications is thus fully compliant with the proposed design guidelines G1–G8. While an initial study is presented in Chapter 7 additional user studies will have to be conducted in order to assess the usability of the end-user authoring tool.

6.5 Discussion of the Functionality

Next to design guidelines, we discuss how our tool supports the requirements that have been presented in Chapter 2, Chapter 3 and Chapter 4. A summary of the requirements and their fulfilment by our eSPACE model and tool is presented in Table 6.1. In this section we further explain how each requirement is fully or partially fulfilled.

Requirement 1 (R1). Provide an overview of the smart technologies, environments and applications The overview of all smart devices, rules and applications can be found in the *home* view. Note that, the smart environment itself could have been represented in a more graphical manner as done in Platform Composition [172], we just represented it in the form of different dropdown lists.

Requirement 2 (R2). Interaction support The interaction across all types of devices is the main focus of the eSPACE authoring tool. Different types of interactions can be defined, such as contextual interaction (dashed arrow), synchronisation interaction (double-sided arrow) and user-initiated interaction (regular arrow).

Requirement 2.1 (R2.1). Support for interaction across multiple smart technologies By using the *interaction* and *rules* view, users can define interactions across devices and *things* either through the use of arrows or through the use of IF-THEN statements.

Requirements	eSPACE Model	eSPACE Tool
Requirement 1 (R1). Provide an overview of the smart technologies, environments and applications	●	●
Requirement 2 (R2). Interaction support		
Requirement 2.1 (R2.1). Support for interaction across multiple smart technologies	●	●
Requirement 2.2 (R2.2). Support for creation, customisation and distribution of cross-device and IoT user interfaces	●	●
Requirement 2.3 (R2.3). Offer fine granularity UI distribution	●	●
Requirement 3 (R3). Shareability		
Requirement 3.1 (R3.1). Support for sharing and integration of apps in a central smart apps repository	●	○
Requirement 3.2 (R3.2). Enable sharing of applications, user interfaces or parts of a user interface with specific users	●	◖
Requirement 4 (R4). Extensibility		
Requirement 4.1 (R4.1). Offer extensibility at the level of communication protocols, devices and user interfaces	●	◖
Requirement 4.2 (R4.2). Enable the integration of third-party applications	●	●
Requirement 4.3 (R4.3). Offer extensibility of adaptive behaviour and distribution configurations	●	●
Requirement 5 (R5). Reuseability		
Requirement 5.1 (R5.1). Support for reuse and combination of existing user interfaces	●	◖
Requirement 5.2 (R5.2). Support for reuse and combination of existing functionality	●	◖
Requirement 6 (R6). Portability		
Requirement 6.1 (R6.1). Offer platform independence	●	◖
Requirement 6.2 (R6.2). Support for context awareness	●	◖
Requirement 7 (R7). Support for end-user development	●	◖

Legend
○ Not fulfilling requirement ◖ Partially fulfilling requirement ● Completely fulfilling requirement

Table 6.1: Requirements addressed by the eSPACE model and authoring tool

Requirement 2.2 (R2.2). Support for creation, customisation and distribution of cross-device and IoT user interfaces The creation of user interfaces is possible in the *UI design* view, where users can design and customise their own UIs by dragging and dropping different UI elements to the authoring canvas. With our current prototype, the distribution of user interfaces can be done only at runtime by longpressing a UI element in the *app* view and selecting the device to which it has to be distributed. Note that we do not support distribution of UI elements across different RSL link servers but only across applications hosted on the same server.

Requirement 2.3 (R2.3). Offer fine granularity UI distribution Since all UI elements can be distributed in the *app* view, we support distribution at a fine level of granularity. The distribution is triggered once the user has longpressed a UI element and selected a target device. The target device must have the *app* view opened in its browser and will show the distributed UI element after a page refresh, as explained in Section 6.3.3. It should be noted that in the current version of our authoring tool it is up to the developer to decide whether or not a UIe can be distributed by using the *distributable* property of a DComp. In future versions we should enable end users to decide which UIe can be distributed and which ones cannot. Further, other distribution properties

could also be exploited such as defining whether a UI element is splittable and how or when it should be split or adapted to the target device to which it is distributed.

Requirement 3 (R3). Shareability Sharing is the main weakness of our eSPACE prototype. While this requirement is supported on the server side, it has not yet been fully implemented in the client side authoring environment.

Requirement 3.1 (R3.1). Support for sharing and integration of apps in a central smart apps repository A central apps repository is located on the server and contains all user-defined applications. However, the functionality to share applications by making them public using the authoring environment is not present.

Requirement 3.2 (R3.2). Enable sharing of applications, user interfaces or parts of a user interface with specific users By using the *home* view, users can share an application with another user by selecting the `share` option next to the corresponding user. This will add the application to the `shared entities` of the chosen user. However, we do not implement the sharing of a specific UI or parts of a user interface in our eSPACE GUI. Further, we did not yet provide a login page for users to switch accounts and currently a certain user is logged in by default.

Requirement 4 (R4). Extensibility This requirement is fully supported for developers and almost fully supported for end users as explained below.

Requirement 4.1 (R4.1). Offer extensibility at the level of communication protocols, devices and user interfaces This requirement is mainly fulfilled on the server side where developers can add new communication protocols, devices and UI elements. In the *UI design* view users can add their own images as well. These can be used as extra icons in a UI. As mentioned before, devices cannot be added by end users for now because "adding new devices to the tool" was not the main focus of this dissertation, which focusses more on the interaction part.

Requirement 4.2 (R4.2). Enable the integration of third-party applications Developers can add `ACs` to the pool of active components on the server. These `ACs` can be an action that is performed by an external service or third-party application. The weather service, for example, includes weather-related functionality in the form of `ACs`, which can be used by the user when defining an interaction in the *interaction* or *rules* view.

Requirement 4.3 (R4.3). Offer extensibility of adaptive behaviour and distribution configurations Adaptive behaviour can be added through contextual rules or interactions. The current adaptations are quite limited but can be extended over time by, for example, using the users' `preferences` to change the font size or colour of a UIe according to their needs. Note that the concept of preferences is present on the server side but currently not used on the client side. Our initial prototype supports adaptations such as "*when Lucy uses a button then action A is triggered, but when Lucas uses the same button action B is triggered instead*". Actions can also be triggered depending on the environment, for example, "if outside then set brightness of smartphone screen to 90%". Finally, actions can be triggered in function of time. In order to address the extensibility of the distribution configuration we allow users to distribute UI elements at runtime, as explained in R2.3.

Requirement 5 (R5). Reuseability In order to fulfil this requirement, we provide a clear separation between UI elements and their tasks or functionality. The view of an application's user interface is done in the *UI design* view where users choose which UI elements will be part of their UI. After that, users can define functionality for a UI element in the *interaction* or *rules* view.

Requirement 5.1 (R5.1). Support for reuse and combination of existing user interfaces All components of the sidebar in the *UI design* view can be reused for multiple applications. While the grouping of UI elements is possible to move or copy/paste multiple elements at once, we plan to provide users with the possibility of grouping UI elements and saving them into the sidebar for later reuse of these elements in other applications. Therefore, we see this requirement only as partially fulfilled.

Requirement 5.2 (R5.2). Support for reuse and combination of existing functionality Functionality is represented by active components in our model and server, but is presented to the end users either as option in a popup window when creating an interaction in the *interaction* view or as option in the different dropdowns in the *rules* view. The same functionality can be reused and combined with other functionality multiple times using rules or interaction arrows. However, only the *rules* view provides easy reuse of existing interactions by providing a button to access the `saved triggers` and one to access the `saved actions`. Note that for now, all triggers and actions of created rules are accessible by pressing one of these buttons, allowing users to select the desired trigger or action in a dropdown. In the future, it might be better to let users decide whether to save a certain trigger or action. Given that this easy reuse is not possible in the *interaction* view, we consider this requirement as partially fulfilled.

Requirement 6 (R6). Portability The general portability requirement is supported since the eSPACE authoring tool generates web application that can be opened on any devices that supports a web browser.

Requirement 6.1 (R6.1). Offer platform independence Given that a user-defined application can be used through a simple web browser, we allow a large range of smart devices to run the users' applications. However, while all UIs can run on any device's web browser, some UIs might become unusable due to, for example, buttons that are too small because the UI has been originally designed for a tablet but has been opened on a smartwatch. Users could create multiple versions of the same UI using the *UI design* view, but since our tool does not detect on which device the *app* view is opened the user still have to manually select the right *tab* in the web browser. Therefore, we marked this requirement as partially fulfilled. By detecting the device one could also automatically adapt the UI depending on the device's characteristics on which it is opened, as explained in our next requirement.

Requirement 6.2 (R6.2). Support for context awareness Interactions can be defined using a contextual element, representing either a user, location or time, as trigger. Depending on this trigger, some user-defined action will then take place. Such interactions can be defined in the *interaction* or *rules* view. The detection of a specific context of use is only done when an *app* view of an application containing a contextual trigger is opened. Since our context detection is done at the client side, whenever the *app* view is closed, the contextual rule will be deactivated. As mentioned before, future versions of our tool should perform the context detection on the server side. Further, the tool should also be able to adapt a UI depending on the device on which the UI is opened. Since our tool does not yet detect the type of device, this adaptation is not yet possible. Additionally this device detection could also be used to only open the *app* view created for this specific device, or let users choose how to open and adapt a UI originally made for another device.

Requirement 7 (R7). Support for end-user development The eSPACE authoring tool provides multiple abstractions, namely the pipeline and rules metaphors, in order to hide the technical details from end users. Accordingly, end users do not require any technical skills to use our tool. As will be demonstrated in Section 6.6, none of the functionality has to be performed using any programming code or complex representations as used in some other model-based approaches. However, given that the support for shareability, extensibility and reusability could still be improved for the end users, we marked this requirements as partially satisfied.

6.5.1 Limitations

We finish this discussion section with a short summary of the limitations of our initial eSPACE authoring prototype. As explained in this discussion, our eSPACE model has been designed to be reusable, flexible and extensible. While we designed our eSPACE authoring tool with these same criteria in mind, we did not yet implement all the rich functionality described in Chapter 4. We therefore shortly describe which elements of the model we did not yet use and how they could be used in a future version of our eSPACE authoring tool.

Resources

In our current prototype we used all resources of our domain-specific conceptual model, with as exception the `Layout` resource. By using the `Layout` resource one could store more complex UI designs and thus allow end users to create a UI using a pre-defined layout, such as a `List` layout or `Grid` layout.

Links

We used all link types introduced by our domain-specific extension of the RSL meta-model. However, we did not use the original `Navigational Links`. Thereby, we limited our eSPACE prototype to only single page applications, as navigation between different views is not yet integrated. `Navigational Links` could play an important role when adapting or distributing UIs from a large screen device to a device with less screen space, where the UI might need to be split over multiple views that need navigation between them. An example is the *leaving home* application on Lucy's phone that requires only one view compared to Lucas' version on the smartwatch which requires three views.

Selectors

Our current prototype does not make use of RSL `Selectors`. It could be interesting to create and integrate an image or video selector so that end users could allocate actions to only part of an image or video. By linking an image selector to the top of an image one could define an interaction rule with as trigger "clicked at the top of the image". This way, a user could define an interaction triggering a different action depending on where the user clicks on an image.

Users

As mentioned in requirements R3 and R4.3, user management is the main weakness of our initial prototype. We do make use of the accessibility and sharing relationship, but

we do not use it to its full potential. Future versions of our authoring tool should also allow sharing at a finer granularity, meaning not only sharing of an entire application but also of specific parts of an application, such as a certain FUI. Further, a public repository where users could make their entities (e.g. UI designs, FUIs, DComps) accessible to an online community would greatly increase the reusability. Therefore, the `Group` component of our model, which is currently not applied, should be used. Next, we also do not yet take advantage of the `Preferences` of a user, which could be used to tailor the user-defined application or some of the authoring views depending on these preferences. One could for example use a larger fontsize for users with a poor eyesight or adapt the interface according to their favourite colour scheme.

6.6 Use Case Demonstration

We finish this chapter by presenting some use cases, which demonstrate how to use the tool in a stepwise manner. In this section, we also show that our database, based on our conceptual model, can be populated with concrete data in the way as described in Chapter 4, and that it can be used by the authoring tool following the UI development process presented by our reference framework. As explained earlier in this chapter, each final user interface that has been built using links of the conceptual model in the way described by the reference framework is transformed into executable code by the authoring tool and will be shown in the *app* view. Note that the use cases presented in this section are based on our use case scenario presented in Chapter 3 and the smart technology we had at our disposal.

Smart Environment

In this section we shortly explain all smart technologies present in the smart environment of our use cases. We differentiate between the following screen devices: `Alex Smartphone`, `Living Room Tablet`, `Mason Smartphone`, `Alex Laptop`, `Smart TV` and `Smart Fridge`. For these devices a user can create a user interface. The remaining smart technologies that can be controlled include three Philips Hue smart light bulbs[1] used for the Living Room Light, the Bed Room Light and the Desk Light, one WiFi light bulb[2] serving as the Cat Room Light, a decorative `Light Tree` and a `Water Boiler`. Note that the last two objects can be controlled by using TP-Link Smart Plugs[3] which are used to connect these objects to the power supply.

[1] `https://www2.meethue.com`
[2] `https://www.amazon.fr/gp/product/B013DJYYYW`
[3] `https://www.tp-link.com/fr-be/home-networking/smart-plug/hs100`

In order to control these smart technologies, we implemented a number of ACs. For controlling the smart TV, ACs have been created using the Pylips Python API[1]. Further we used the TP-Link SmartPlug API wrapper[2] for implementing ACs that can control the two smart plugs and thereby control the decorative light tree and water boiler. The ACs to control the lights make use of the official Philips Hue API[3] for the Philips Hue light bulbs and the led_flux Python API[4] for controlling the WiFi light bulb in the cat room.

Regarding contextual interaction rules, we used HTML Geolocation[5] for defining multiple locations based on coordinate ranges. We kept it simple by only providing `home` and `not home` as locations. Next, we use the JavaScript `setInterval()` function for implementing time constraints. When an interaction rule includes a user constraint, meaning that an action could, for example, only be triggered by a specific user, we simply verify which user is logged in. We are aware that the implementation of contextual rules should still be improved, as already discussed in Section 6.3.3.

Further, we implemented two ACs for synchronisation, one for the synchronisation of the files of a folder and another one for UIe synchronisation. For synchronising folders we used the Windows command line `xcopy` command. Note that we limited this synchronisation to a certain directory, which means that one can only synchronise folders within this directory. We choose to do this to avoid overwriting folders on the computer by making mistakes in our rules. For keeping UIes on different devices (i.e. *app* view tabs) synchronised, we used the PubNub JavaScript library[6].

Lastly, we included an AC that allows users to see the weather of a certain city. The AC has been added to illustrate the functionality of third-party services, which in this case is the weather forecast. The implementation of this AC uses the OpenWeather API[7] to retrieve the weather of a given city.

Simple Controller App

We start by demonstrating how to create a simple device controller application. The application is meant to be used on the `Living Room Tablet` and should be able to control the TV, the ceiling light of the living room and the decorative light tree next to the TV. In order to do this, user `Alex` will create a user interface consisting of three buttons, one to control each of these smart technologies.

[1] `https://github.com/eslavnov/pylips`
[2] `https://github.com/vrachieru/tplink-smartplug-api`
[3] `https://developers.meethue.com/develop/hue-api/lights-api`
[4] `https://github.com/Danielhiversen/flux_led`
[5] `https://www.w3schools.com/html/html5_geolocation.asp`
[6] `https://www.pubnub.com/docs/web-javascript/pubnub-javascript-sdk`
[7] `https://openweathermap.org/api`

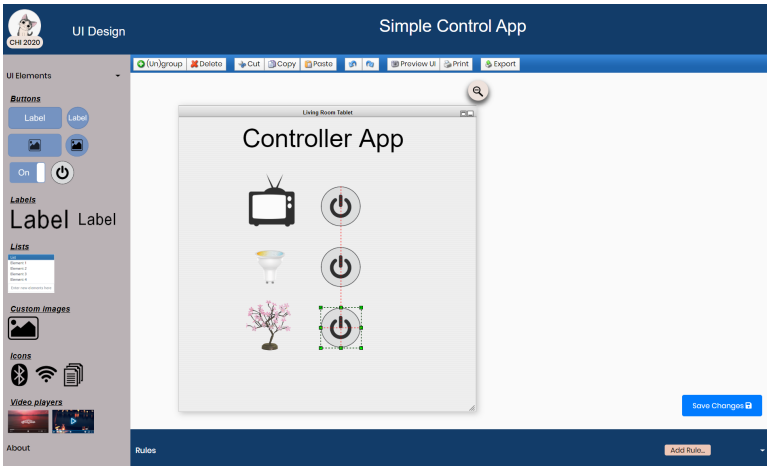
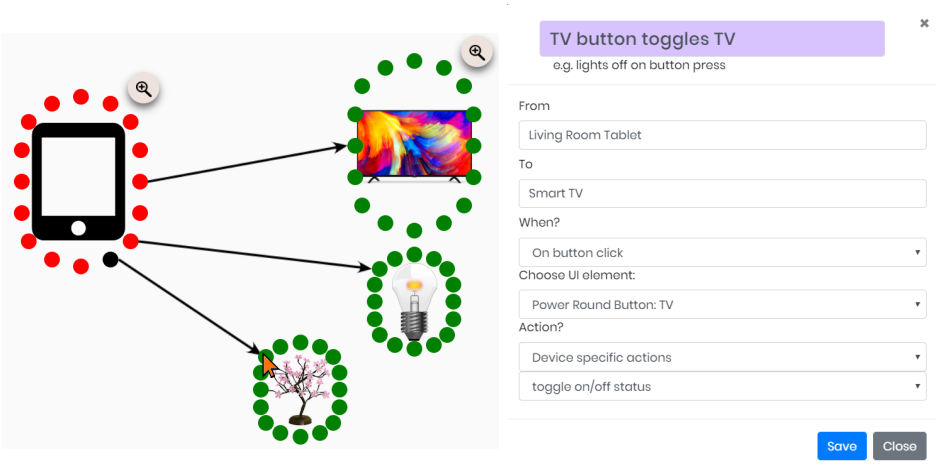


Figure 6.15: Example user interface for the `Simple Control App`

Alex starts in the *home* view, where she presses the `Add App...` button, fills in the name of the application (`Simple Control App`) in a popup window and gets redirected to the *interaction* view. In this view she starts by dragging and dropping the `Living Room Tablet` element on the authoring canvas and then clicks the magnifying glass of this tablet element, in order to go to the *UI design* view where she can start designing the tablet UI. An example UI for the `Simple Control App` is shown in Figure 6.15. Once the UI design is saved, she goes back to the *interaction* view to add functionality to the created buttons, by pressing the magnifying button.



(a) Closeup of the process of connecting a tablet element with the light tree (cursor is hovering the green dot)

(b) Popup for defining the interaction between the `Living Room Tablet` and `Smart TV`

Figure 6.16: Creating the `Simple Control App` interactions

In the *interaction* view she places the `Smart TV`, `Living Smart Light` and `Light Tree` elements on the authoring canvas where she then links the tablet with each of these smart technologies. Figure 6.16a shows how Alex is going to connect the tablet element with the light tree (cursor is hovering the green dot). Once the green dot of the light tree selected, a popup window will appear. When all information in the popup is filled in and the *save* button is pressed, all dots will disappear.

Instead of linking the different smart technologies in the *interaction* view, Alex could also use the *rules* view to define the interaction. Once finishing the UI design, she could press the `Add Rule...` button in the bottom container of the view, which will redirect Alex to the *rules* view. Figure 6.17 shows how to allow the smart TV to be turned on and off using one of the power buttons created in the *UI design* view. As a comparison, the popup window for defining the same interaction is shown in Figure 6.16b. While the name of the rule can be given directly in the popup window in the *interaction* view, in the *rules* view the name can only be given after saving the rule.

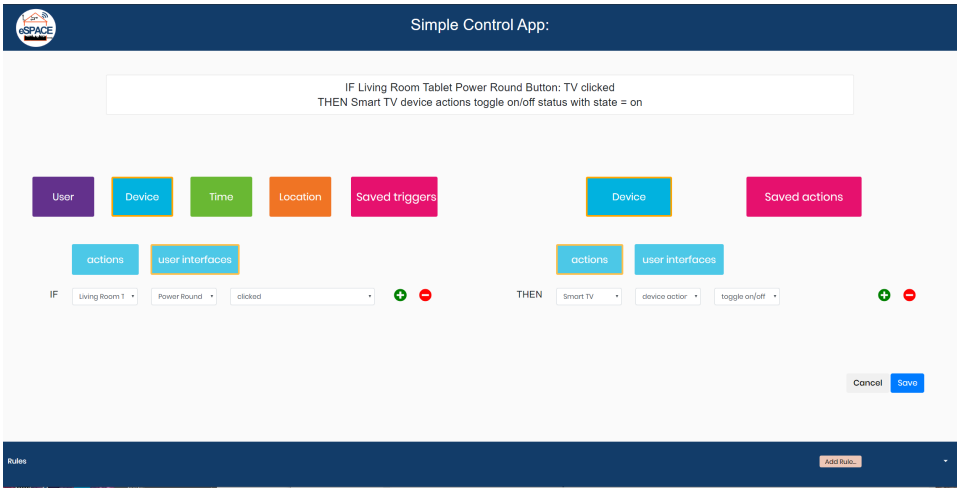
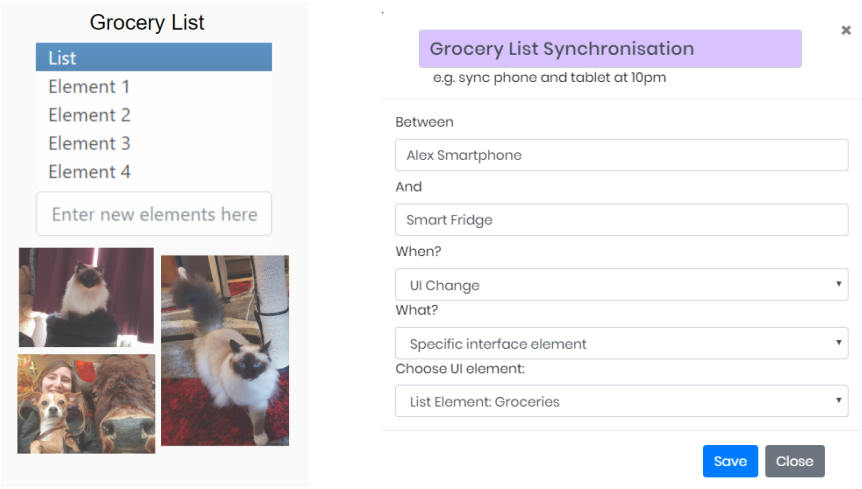


Figure 6.17: Interaction rule for controlling the smart TV on button press

Synchronisation of Grocery List

The `Grocery List` use case application is based on the *grocery list* application from Chapter 3. The application's UI is distributed across two devices, the smartphone and the fridge. The smartphone UI simply contains a list of groceries, while the fridge UI depicts this same list together with some pictures. With this application we show the use of the double-sided interaction arrow representing synchronisation.



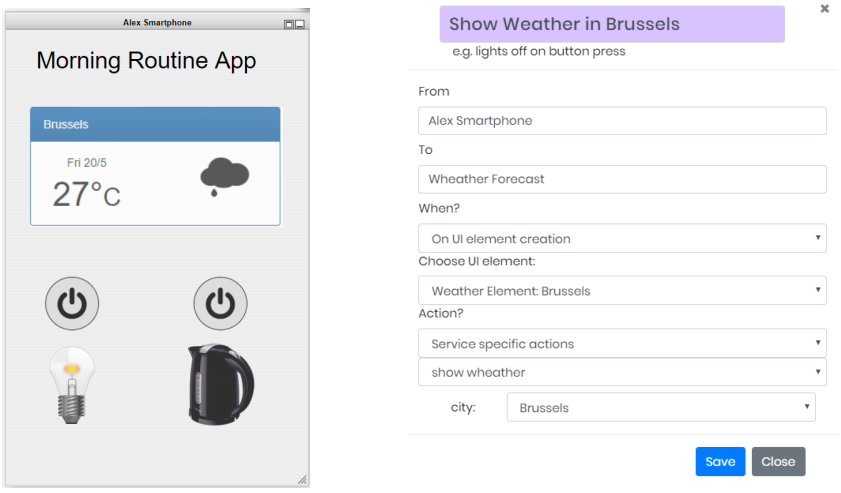
(a) Preview of the Fridge UI (b) Popup for defining the synchronisation

Figure 6.18: Creating the `Grocery List App` interactions

Similar to the creation of the previous application, Alex clicks the `Add App` button in the *home* view, fills in the name of the app, then adds the smartphone element to the authoring canvas in the *interaction* view, presses the magnifying button and creates the UI for the smartphone. After that, she also creates the UI for the fridge (see Figure 6.18a), saves this UI design and comes back to the *interaction* view where she connects the smartphone element to the fridge element on the authoring canvas using a double-sided arrow. A popup window then appears, where Alex fills in the information regarding the grocery list UI which should be synchronised between the two devices, as illustrated in Figure 6.18b. Note that in the current version of our authoring tool we assume that the selected UI element that should be synchronised is present in both UI designs of the smartphone and fridge, which could cause problems if this is not the case.

Morning Routine

The `Morning Routine` use case application is a simplified version of the *morning routine* application described in Chapter 3. The application consists of a smartphone UI showing the weather in Brussels and allowing to control the bedroom light and water boiler. Further the application also triggers the light of the bedroom and the water boiler to turn on at 8 a.m. With this application we illustrate a more complex application which uses user-initiated interactions as well as a contextual interaction involving multiple actions that should be triggered at the same time given a certain situation. We further also included our third-party service, being the weather forecast.



(a) Morning Routine App UI design

(b) Popup for defining when to show the weather forecast

Figure 6.19: Defining the `Morning Routine App` interactions

Again Alex, uses the *home* view to create an application. Once in the *interaction* view Alex drops the smartphone element on the authoring canvas and presses the magnifying glass. In the *UI design* view she designs the desired UI for her application. Figure 6.19a illustrates an example UI design. After saving the design and going back to the *interaction* view, Alex drags and drops the weather forecast service to the authoring canvas next to the smartphone element. After that, she uses a regular arrow to connect the smartphone to the weather forecast service and fills in the additional information in the popup window, which is depicted in Figure 7.3b. Then Alex adds the functionality for the buttons of her interface in the same way as already explained in the previous use cases such that the buttons can be used to turn on or off the bedroom light and water boiler.

Finally, in order to add the contextual interaction she can use the dashed arrow and link a `Time` contextual element to the bedroom light element and then add another dashed arrow between this `Time` contextual element and the water boiler element. By using the *interaction* view Alex would have to fill in two different popups since defining multiple actions are not supported in this view. Therefore, a faster way to define this interaction is by using the *rules* view where the rule can be defined as "*IF time is 8 a.m., THEN turn on the bedroom light AND turn on the water boiler*", as shown in Figure 6.20.

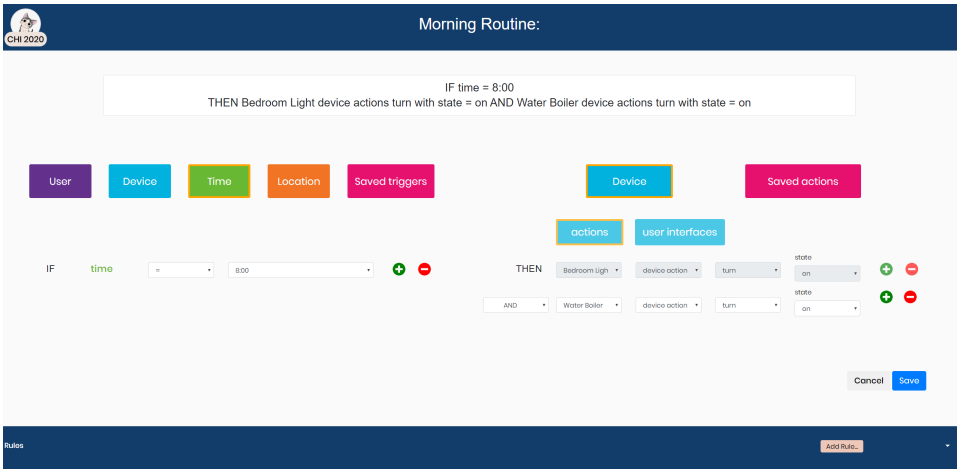


Figure 6.20: Interaction rule to turn on bedroom light and water boiler at 8 a.m.

With these three applications we demonstrated the diverse functionality provided by our unified XD and IoT end-user authoring tool. In order to verify whether end users can build such applications using the eSPACE authoring tool we performed an initial user evaluation, which is described in the next chapter.

Chapter 7

Evaluation

Any darn fool can make something complex; it takes a genius to make something simple.

Albert Einstein

In this chapter we present an initial evaluation of the eSPACE proof-of-concept end-user authoring tool. While our first user study focussed on finding the right abstractions and design guidelines for an end-user authoring tool allowing the creation of cross-device and IoT applications, this second study will give us insights on the end users' first impressions concerning our authoring tool. The study will also indicate whether or not end users can create cross-device and IoT applications by using eSPACE. Therefore, we asked eight volunteers without programming skills to design three applications using eSPACE, examined how well they performed and captured their first impressions. In order to capture their first thoughts about our tool we used the Microsoft Desirability Toolkit or Microsoft Reaction Cards Method as explained later in this chapter. In addition, we let participants fill in a series of questions to get an idea of the current user satisfaction and to ask participants whether they preferred the *interaction* or *rules* eSPACE authoring view. Based on our observations and the results of the post-study questionnaires, we end this chapter with a summary as well as some design implications and future work.

7.1 Setup

In this user study we used the eSPACE authoring tool with the same smart environment as the one described in Section 6.6 of our previous chapter. The smart environment was set up at our apartment, where users were supposed to perform the user study. Due to exceptional circumstances[1], we had to perform the study remotely, which more than likely had an impact on the users' experience. We discuss this in more details later in Section 7.5.

7.2 Participants

Eight participants (5 females) aged 22 to 56 years (M=32.3, SD=10.2) were recruited from our entourage. Since our target group are people without programming knowledge, we chose candidates with limited to no programming skills. The only requirement for each participant was to possess a personal computer at home with a decent Internet connection. Half of the participants were native French-speaking people while the other half were native Dutch-speaking people.

7.3 Protocol

Participants were asked to perform a series of tasks by using the eSPACE authoring tool. Since eSPACE is running as localhost on our computer and set up to communicate with our smart home environment, in order to let candidates interact with the tool, we gave them remote access and control over our computer using TeamViewer[2]. After completing the tasks with eSPACE, we concluded with a questionnaire and a short interview.

eSPACE Presentation (15–20 min)

Each participant received a PDF document explaining our smart home setup, the available devices and different views of the eSPACE authoring tool (see Appendix B.1). Note that this document has been translated into French to make it more understandable for participants with limited English skills. Participants also received a short demonstration video[3] that explained how to use eSPACE through the creation of a small example application. The document and video were available at all times during the study.

[1] Covid-19 social distancing measures
[2] `https://www.teamviewer.com`
[3] `https://www.youtube.com/watch?v=NnQ9auXKj68`

Tasks Execution (40–50 min)

Participants were asked to play the role of *Alex* who lives in the smart home described in the PDF document. Alex wishes to make her smart technologies communicate with each other using a couple of applications. Participants were therefore asked to create such applications using eSPACE. We chose the same three applications as described in Section 6.6 of the previous chapter:

1. The Simple Controller Application
2. The Grocery List Application
3. The Morning Routine Application

These applications were chosen since they cover most of the functionality offered by our authoring tool, ranging from simple buttons to control smart appliances to synchronisation of UI elements and the use of a contextual element (time) as well as an external service (weather forecast).

After reading the PDF document with the description of each application and after having watched the tutorial video, participants were given access to the eSPACE authoring tool via the web browser. They were told to inform the study supervisor when something was unclear and to not hesitate to ask questions when they were stuck. The study supervisor followed the actions of the participants during the whole study and noted down when they had to intervene, help or clarify certain things. Lastly, for every finished application, participants were told to try the application in order to see whether it had the expected behaviour.

Questionnaire and Interview (15 min)

In order to capture the first impressions of the participants and the desirability of our authoring tool, we used the Microsoft Reaction Card method [20] which consists of giving participants a set of words from which they have to select the ones that best describe the tool they just used. The word set contains negative, neutral and positive words. While the original set contains 118 cards, it is recommended to shorten this list depending on which words are more relevant for the tool being evaluated. We chose to keep 40 words that are shown in Appendix B.2.1. Participants could select between 5 to 10 of these words and classify them either as *highly relevant* or just *relevant* with a maximum of 5 words per category. Note that the set of words has been randomised for each participant. Next, participants had to fill in the Post-Study System Usability Questionnaire (PSSUQ)[1] in order to have an indication of the system usefulness, information quality and interface quality of eSPACE. This questionnaire is shown in

[1] `https://uiuxtrend.com/pssuq-post-study-system-usability-questionnaire/`

Appendix B.2.2 and consists of 16 questions to be answered via a 7-point Likert scale and an n/a option. Since questions 7 and 8 are not applicable to our authoring tool, given that we neither provide error messages nor allow users to easily recover from mistakes, we marked them as n/a for all participants. Participants concluded the study by a last questionnaire asking them how easy it was to perform each task, which view they preferred and whether they had any comments or suggestions to improve the tool. This questionnaire is available in Appendix B.2.3. Finally, we performed a short interview asking participants about the reason why they selected specific words from the reaction cards and why they classified them in a certain category.

7.4 Results

In this section we first go through the words selected by the participants to best describe their experience with the eSPACE authoring tool. After that, we go through the results of our questionnaires and finish by describing our study observations.

7.4.1 Microsoft Reaction Cards

Figure 7.1 depicts a wordcloud which represents the selection of reaction cards chosen by the participants during the last phase of our study. The bigger a word the more often the word has been selected by our participants. Further, a word classified as *highly relevant* has been given a larger weight than a word that has been placed in the *relevant* category. Concretely, a word in the *relevant* category has a weight of 1, while a word in category *highly relevant* has a weight of 1.5. During the interview with each participant, we asked them why they selected these words to describe their experience with eSPACE. In this section we discuss the reason behind a majority of the words represented in the wordcloud.

Figure 7.1: Wordcloud representing how participants described eSPACE

The word that was chosen the most is *Organised*. 6 participants mentioned that they selected organised because the tool classified information in multiple categories that were well organised in the views. Thereby, they were referring to the categories in the left sidebar in the *interaction* view and the different options in the *rules* view. One participant said "*with everything per theme, we can find what we are looking for*".

The second most selected word is *Useful*, which has been chosen by 5 participants. Some participant mentioned it would be handy to have such a tool given that we are becoming more *"lazy"*. Others said the tool would be useful to control their smart devices. Next, 4 participants placed the word *Effective* in the *highly relevant* category, mainly because the tool was working and participants could control the smart devices as they wanted with the applications they created. A participant further mentioned: "*the tool is doing what it should do.*" The term *Easy to use* has also been picked by 4 participants. This word was often placed in the *relevant* category, the 3 participants that did so explained that they did not put it into the *highly relevant* category because they first had to take some time to get use to the tool, but "*once you get it, it is easy to use*". The participant that placed *Easy to use* in the *highly relevant* category mentioned that they placed it there because "*one video of 5 minutes is enough to start using the tool*".

3 participants selected the word *Flexible* and explained that the tool offered quite some flexibility in terms of what they want or could do with it. Participant P3 expressed that "*the tool offered a multitude of possibilities*". The word *Attractive* has also been chosen by 3 participants. A participant that chose this word mentioned that the tool made them want to create more connections between devices and another one said that the study made them want to use the authoring tool more.

Participants P2 and P6 selected the word *Confusing*, because they found some of the menu options not easy to understand and thus confusing. They were both referring to certain dropdown options in the popup menu of the *interaction* view and the ones in the *rules* view. For the same reasons they both also chose the word *Complex*. Participant P6 further selected *Difficult*, as they had difficulties using the tool in the beginning, but mentioned that "*after a while it was ok*". The word *Stressful* has also been selected by participant P6 because in the beginning they felt a bit lost. A possible factor that may have played a role is that this participant performed the study at a later timeslot than originally planned, because they had a busy day at work, which could have impacted their level of stress.

Efficient has been picked by 2 participants. One of them said "*it works well and it goes to the point*", when speaking about the authoring tool. A couple of participants selected the word *creative* because you could create your own applications with the tool and be creative. Further, 2 participants chose the word *Exciting*, as they both

found eSPACE fun to use. The word *Advanced* was also selected by 2 participants saying that the tool provided functionality they were not used to, therefore they perceived the tool as quite advanced. *Inspiring* has been indicated by a participant because it gave them some ideas of interactions that could be done with their smart devices. *Satisfying* was noted down by another participant because they had a feeling of satisfaction when creating their own applications. Participant P8 selected the word *Simplistic*, considering that the tool was "*relatively simple*", they further mentioned that there were "*not too many bells and whistles*", which was seen as a good thing by the participant. The word *Unpredictable* has also been chosen by participant P8, just because of the bugs that they encountered. This participant, for example, found a bug which made the *rules* view not able to save a certain rule.

The word *Time-Consuming* has been selected by one participant that mentioned that it still took quite some time in the end: "*one hour to create three applications*". Another participant did not interpret the word well and said that they chose this word because they could quickly make use of the tool. This participant also selected the word *Approachable* which resembles more to what they meant, as they mentioned believing that "*everyone could easily master the authoring tool*" as reason for selecting *Approachable*.

Note that we did not translate these words to French for the French-speaking participants, because we did not want to risk having different meanings for some words when translating them. Therefore, we told participants to ask us when a word was unclear for them. However, the lack of translation led to some words that were interpreted differently. For example the word *Consistent* was chosen by two participants who interpreted it more as "*consistant*" in French and explained that the authoring tool was offering a lot of possibility and that there were lots of things to do with it.

7.4.2 Questionnaire

The first part of the questionnaire consisted of the post-study system usability questionnaire (PSSUQ). Although this user study has been conducted in order to examine how users will behave and interact with our tool as well as see whether they can create applications with it, we still gave them this questionnaire to have an indication of the current usability of our tool. However, since participants are all acquaintances, we will handle the results with a grain of salt. The PSSUQ gives an overall score corresponding to the user's perceived satisfaction of the tool, but it can be further broken down into three subcategories, being the system usefulness (SYSUSE) (average scores of questions 1 to 6), the information quality (INFOQUAL) (average scores of questions 7 to 12) and the interface quality (INTERQUAL) (average scores of questions 13 to 15). Figure 7.2 shows the means per question and the means per subcategories according the responses of our participants. The lower the value the better.

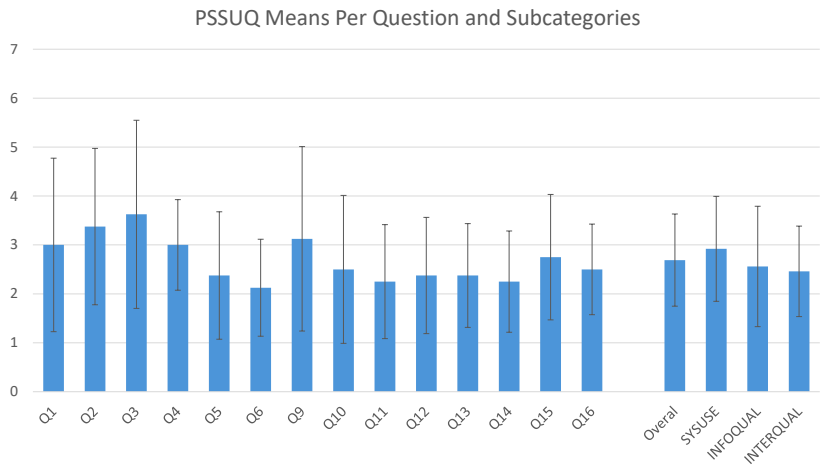


Figure 7.2: Results of the post-study system usability questionnaire

Based on Figure 7.2, we see that the mean for each question is between 2 and 4, which implies that participants answers to the questions tend to be on the positive side, as the 4-score represents "neutral". Question 6 that asked participants whether they believe they could be productive quickly by using our tool, has the best score, which is 2.13 on the 7-point Likert scale. The next best rated questions are question 11 and question 14 with an average score of 2.25. Question 11 was related to whether the information received by the participant was effective in helping them completing their tasks. Therefore, we believe that the demo video and tutorial document served their purpose well. Question 14 asked participants if they liked using the interface of our authoring tool. Since this question has the second best rating, we can estimate that the interface of our authoring tool was pleasant to use for our participants. The question that has the worst score, is question 3 with a score of 3.63 and is related to the time for completing the tasks. More concretely, participants had to indicate whether they were able to accomplish the tasks quickly using our tool. Some participants mentioned that it still took almost an hour to create the three applications and therefore gave a higher score to this question. Future studies could be performed to compare how quickly users can create these applications using another authoring tool. The next worst score of 3.38 was attributed to question 2, which asked participants whether the tool was simple to use. Participants who indicated a score of 3 or more, expressed having difficulties at the beginning of the study and needed some time to get use to the eSPACE authoring environment, after a while they mentioned it was getting easier. These difficulties in the beginning of the study were mostly the reason why participants tended to give a higher score to question 2.

Overall our authoring tool scored quite well, with a score of 2.69. The authoring tool scores best on the interface quality, followed by the information quality and as lowest the system usefulness. As mentioned above, participants liked the interface of our tool which is most likely why this subcategory scored best. By removing two questions related to information quality (question 7 and question 8) as they were not applicable to our tool, the authoring tool still performed well on this category mainly thanks to the demo video and tutorial document. The system usefulness has the lowest score mainly because of the questions 2 and 3 discussed earlier.

We further interpret our results by comparing the means per category to the norm defined by Sauro and Lewis [195] based on 21 studies and 210 participants. By looking at Table 7.1 we see that our means are similar to the ones of Sauro and Lewis, which means that the user satisfaction of eSPACE is quite good. Of course, these results are merely a first indication of the user satisfaction as we had a low number of participants and participants were taken from people that we know, which introduces a bias. The low number of participants is also reflected in the error bars shown in Figure 7.2 representing the relatively large standard deviation, indicating that the individual responses strongly deviate from the mean values. However, it still gives us some confidence that the initial version of our authoring tool is already on the right track for further improvements.

	Sauro and Lewis [195]	Our study
SYSUSE	2.80	2.91
INFOQUAL	3.02	2.56
INTERQUAL	2.49	2.46
Overall	2.82	2.69

Table 7.1: Comparing results with Sauro and Lewis [195].

The second part of the questionnaire consisted of rating the difficulty of the creation of each of the three applications. While the *Simple Controller* application is the most simple application of the three, not all participants marked it as easy. The 3 participants who marked the app as *neutral* or *difficult* said that the reason behind this was that they were not yet used to the eSPACE authoring tool. Therefore participant P2 indicated the two first apps as *neutral* and the last one as *easy*, since by then they were more accustomed to the tool. Participant P6 followed a similar train of thoughts by noting the first app as *difficult*, the second one as neutral and the last one as *easy*. Lastly, participant P7 marked the first app as *difficult* and the remaining ones as *easy*. The 5 other participants expressed that they had some difficulties in the beginning but that when reflecting back on it, the app was quite easy to create and therefore indicated the first app as *easy*. 2 participants mentioned that the *Morning Routine* was the most difficult one because it also involved new

concepts such as the *time* and the use of the *weather forecast* service. Therefore they indicated this app as *neutral* or *difficult*. The remaining participants marked the *Grocery List* and *Morning Routine* as *easy* and one even indicated *very easy*. An overview of the responses of the participants is shown in Table 7.2.

Participant	Simple Controller	Grocery List	Morning Routine
P1	Easy	Easy	Easy
P2	Neutral	Neutral	Easy
P3	Easy	Easy	Very easy
P4	Easy	Easy	Neutral
P5	Easy	Easy	Difficult
P6	Difficult	Neutral	Easy
P7	Difficult	Easy	Easy
P8	Easy	Easy	Easy

Table 7.2: Results of the degree of difficulty for each application per participant

In the next part of the questionnaire, we simply asked participants whether they preferred the *interaction* or *rules* view. The possible answers to this question were: "interaction view", "rules view", "both equally preferred" or "both equally disliked". Half of our participants mentioned that they preferred the *interaction* view, while the other half said they preferred both views equally. During our observation we did notice participants having more difficulties with the *rules* view, however some of them still chose both views as equally preferred. A more detailed discussion regarding our observations is presented in the next section.

The last part of the questionnaire included two open questions. The first one asked participants if they had any suggestions to improve our authoring tool, while the second one just invited participants to give some general comments. 2 participants requested help menus to clarify some of the dropdown choices in the *interaction* view popups and in the *rules* view. One of these participants also suggested a "game-like" walkthrough of the authoring tool for people that are using the tool for the first time. Participant P8 proposed interesting suggestions, namely to add "auto save" functionality, some zoom-in and zoom-out effect when pressing the magnifying icons in the *interaction* and *UI design* views as well as adding a button to switch between the *rules* and *interaction* view. These were some good suggestions, since we indeed do not provide a way to go back to the *interaction* view once in the *rules* view. The auto save functionality is for now only present when pressing the magnifying button in the *interaction* view. In a first instance we did not add this functionality due to some of the bugs we had. For example, if the user misses the target dot of the target device when they create an interaction with an arrow in the *interaction* view, the arrow will remain on the canvas with no target element and cannot be deleted. As a solution we simply refresh the page, which would not be possible if the arrow was saved automatically. Finally some general comments were *"easily learnable and*

has multiple applications"; *"useful application in daily life, the management of devices and connection between them is simple and effective"*; *"looking forward to using the app when finished"*. Based on the collected data of these questionnaires we provide a short summary of the results in Section 7.5 and present design implications and future work in Section 7.6.

7.4.3 Observations and Discussion

While participants were performing their tasks, we observed certain recurrent behaviours and difficulties which will be discussed in this section. When creating the first application, we asked participants to create one interaction using the *interaction* view and one with the *rules* view. After that, they could chose which view they wanted to use. However, for their first interaction we let them choose which view to use. If they used the *interaction* view, we asked them to use the *rules* view for defining the second interaction, and vice versa. Only one participant started with the *rules* view first.

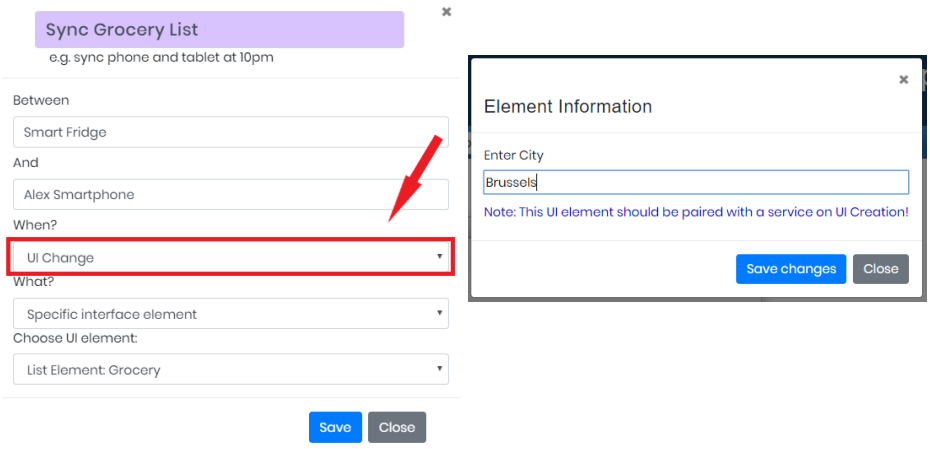
The first notable difficulty which we observed is the selection of the trigger in the *rules* view. 4 participants indicated as trigger *User = Alex*. These participants said they selected user Alex because she is performing the action or because the action was supposed to be performed by her smartphone. While selecting user Alex as trigger is not wrong in itself, selecting user Alex as *only* trigger is wrong, since the trigger in the first application is *on button click*. One participant further mentioned that they thought they had to go through all different categories of the "IF-side". Some participants also had the tendency to fill in the action in the trigger part and were then stuck when they had to fill in the action part (THEN-side).

In the *interaction* view, participants quickly understood how to use the arrows to create an interaction between two devices. However almost all participants had troubles when selecting the target device. They needed to drop the arrow on a green dot, but on their first try dropped it somewhere next to this dot, with the consequence that the tool did not register the target device. Some participants said that the application was lagging, which could be the cause. Another participant said they also had a small screen making it difficult to read the text and drop the arrows on the dots. In order to make the interface more readable for this participant, we used the scaling option in Chrome, which solved the unreadable text issue.

Regarding the popup window in the *interaction* view, 2 participants mentioned that the `when` dropdown, was not always clear. For example, the `on button click` was more interpreted as button of the phone rather than a button of the user interface. When creating the interactions for the *Simple Controller* application, many participants chose the action `turn` with parameter `on` for the action part, instead of toggle,

resulting in making a button that could only turn on a device. A participant mentioned that they would have chosen toggle if the `turn on` option was not pre-selected. The eSPACE authoring tool always selects the first action available of the device in the dropdown, by clicking this dropdown participants can select the desired actions, however they often omitted this action when seeing that the selected option was `turn on`.

Almost all participant did not know which trigger to select for the synchronisation of the grocery list in the second application they had to create. Our tool provided the `UI Change` option in the *interaction* view and the `On Change` option in the *rules* view. We therefore should find a more appropriate label or manner to indicate the correct trigger of UI synchronisation. A participant suggested that "always" would be better in the `when` dropdown of the popup window in the *interaction* view instead of the `UI Change` as shown in Figure 7.3a. Further studies should be performed in order to investigate better menu options.



(a) Popup window for synchronisation in the *interaction* view

(b) Popup window when dropping the weather UI element on the canvas in the *UI design* view

Figure 7.3: Popup windows

All participants had difficulties with the *weather* service of the last application. Some said it would be more logical connecting the weather service to the phone, while others found it logical to connect the phone to the weather service. When participants used the *rules* view they again had the tendency to fill in the "show weather" action in the "IF-side" instead of in the "THEN-side". We anticipated the difficulty of defining the trigger for the *weather* service and therefore had added a note for the users when they placed the *weather* UI element on the phone's user interface which is shown in

Figure 7.3b. However, most of our participants did not see this note, or needed a reminder after asking what the trigger should be. One participant explicitly said they could not imagine what "on UI creation" meant because these are not terms they are used to work with. A participant suggested "on app startup" as alternative, but further investigation should be done in order to find a label that is more straightforward and understood by people.

Similar to our first user study, participants had the tendency to first place all the devices on the canvas before starting to design a UI or to define an interaction rule. When participants needed a contextual element, such as time for the last application, this element was usually added afterwards, when deciding to create the interaction related to time.

For defining the interaction involving time, namely turning on the water boiler and light at 8 a.m., 4 participants chose to use the *rules* view. 3 of them created one interaction rule with two actions while the remaining participant created two different interaction rules. 4 participants used the *interaction* view, 3 of them easily figured out that the contextual arrow had to be used while the remaining participant needed some guidance and explanation on what a context represents.

Symbols and annotations in the *interaction* view were noticed by 3 participants, one used an image icon that they placed on the smartphone element on the canvas. In our introduction video and tutorial document we did not introduce the sidebar elements in details which could be the reason why participants did not pay attention to this part of the sidebar. In future user studies, the sidebar functionality in the *interaction* view should be better presented to the participants.

We end this observation section with some final minor remarks. Only one participant added a rule via the home screen, all participants did it via the `Add Rule...` button in the *interaction* view. One participant used this button while in the *UI design* view. Many participants clicked at least once the application in the *applications* dropdown of the *home* view they originally meant to edit. With this action they opened the application instead of editing it. Further, it was unclear for participants that when opening an app, the authoring tool opened new tabs corresponding to the devices on which the user interfaces of the application are running. This might be because the TeamViewer option bar at the top of the screen was hiding the tabs for some participants, as shown in Figure 7.4. All participants watched the demo video twice, the second time mostly to find a specific action, some followed along, trying to reproduce the actions from the video. For most participants it was hard to juggle between the document describing the applications they had to create and the authoring tool, which led to some annoyance. Participant P2 placed the document on the left and the view of the tool on the right of the screen, but thereby made

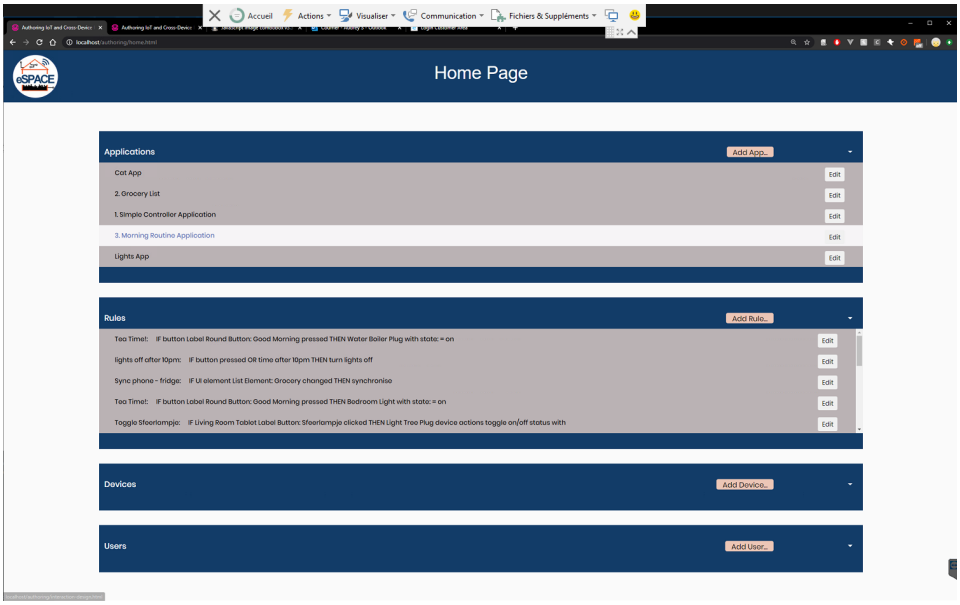


Figure 7.4: TeamViewer screenshot of a participants' view of the *home* view

the *rules* view not very usable, as this view is better employed in full screen. Some participants made a screenshot of the description of the apps they had to create and could then go back to this description using the screenshot more easily than switching between the document and the authoring tool in TeamViewer. Printouts of this document would have given participants a better experience. Last but not least, as our authoring tool is still in the prototype phase, we still had some bugs and missing functionality. Some participants encountered more bugs than others, which has played a role in the participant's experience with the tool as well.

7.5 Summary

While our study environment was not ideal—given that people had to cope with some lags, small screen sizes and sometimes even a connection interruption (2 participants)—we could still gain some insights into peoples' behaviour with our eSPACE authoring tool and get valuable feedback. In this section we summarise the results of our evaluation.

Overall, our authoring tool received a lot of positive feedback. The most recurring words of our wordcloud were positive, such as *organised*, *useful* and *easy to use*. The more negative words, such as *confusing* and *unpredictable*, were often chosen because of the bugs rather than a bad design. The main criticism concerning the design was

more about some word choices in the dropdowns such as *UI change* and *UI creation*, which was unclear for participants, since indeed those terms were too technical for them to understand.

The results of the PSSUQ should be taken lightly, given the small number of participant, the large error bars and the fact that participants were all acquaintances which may introduce some bias. We did not perform this user study to test the usability of our tool, but still asked participants to quickly fill in this questionnaire to have an indication of which subcategory and questions would be rated better than others. We have seen that all three categories, of system usefulness, information quality and interface quality scored between 2.46 and 2.91, which according to the norm are really good scores. We have seen with this questionnaire, that many participants believed they could become quickly productive with our tool.

While the last application that participants had to create was the most difficult one, it was only perceived as difficult or neutral by two participants, which means that using new functionality such as contextual interaction and interaction involving a service, was not perceived as difficult by many participants even though some had requested some guidance. This also confirms the statement from the questionnaire we mentioned above, about participants believing they can become productive quickly with the tool. It further is also in line with the results of the question about whether or not the tool was easy to learn.

Half of the participants preferred both views equally, while the other half preferred the *interaction* view. This is in accordance with our first study, which showed that when thinking about interaction across devices, people think about arrows, which were more natural to use than the IF-THEN statements in the *rules* view. A participant further mentioned that connecting devices with arrows was more intuitive and that they did not have the logical mindset to directly understand the *rules* view, but probably could get there with some time and practice. Nevertheless, this participant still noted that they equally liked both views afterwards. Therefore, we think both views could not only be seen as equivalents but more as complementary. Since the *interaction* view does not yet offer the functionality to edit a rule, participants used the *rules* view to adapt wrong interaction rules. This behaviour could become common practice as modifying a rule seemed more easy for participants to grasp in the *rules* view than creating a rule from scratch using this view.

Finally, our observations gave us insights about the behaviour of participants with our tool. We noticed difficulties with the *rules* views as well as difficulties with terminology of certain dropdowns. It was very surprising that all participants could not drop the arrow on the green dot of the target device, as explained before this could be due to lags and small screens but further investigation should be performed in order to see

whether there is a real issue with the size of the dots surrounding the target device. Throughout the study participants sometimes discovered some new bugs, which are mostly fixed by now. However, a thorough examination should still be performed in order to eliminate all bugs once and for all.

7.6 Design Implications and Future Work

While some of the feedback we received on the design of our tool might need further investigation, in this section we start by discussing the feedback that can already be applied. Overall, there are no big design changes required which is good news and means that our design guidelines certainly helped for designing something intuitive for end users.

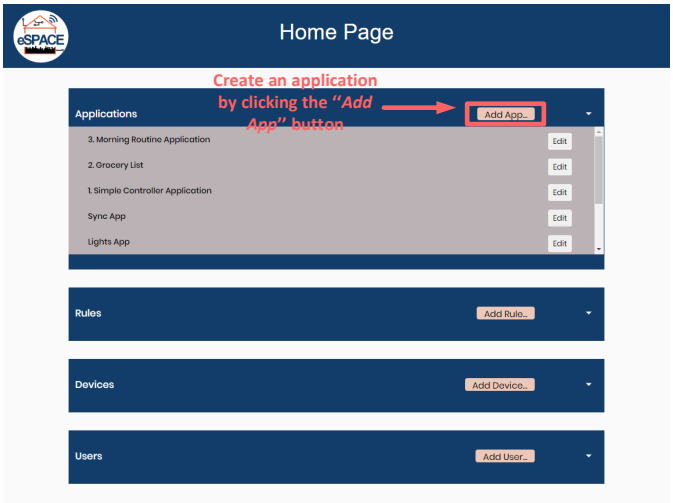


Figure 7.5: *Home* view with a walkthrough layer showing end users how to build a demo application

We first discuss how to improve the demonstration and explanation of the authoring tools' functionality. Next to the demonstration video, there should be the option to follow a walkthrough tutorial of the authoring tool. Thereby, we would provide end users with an on-tool explanation of the functionality of the tool. Users would then follow a step-by-step tutorial to build their first demo application. Figure 7.5 illustrates how users could follow the steps to create a demo app and Figure 7.6 shows a possible way of explaining the different functionality of the tool while following these steps. This could provide a better hands-on experience and further lower the learning curve as well as build more confidence. Such a learning by example is popular in games. According to Schell [196], males prefer a trial-and-error approach, while females tend

to prefer clear step-by-step tutorials so that when attempting a task they know what they are supposed to do.

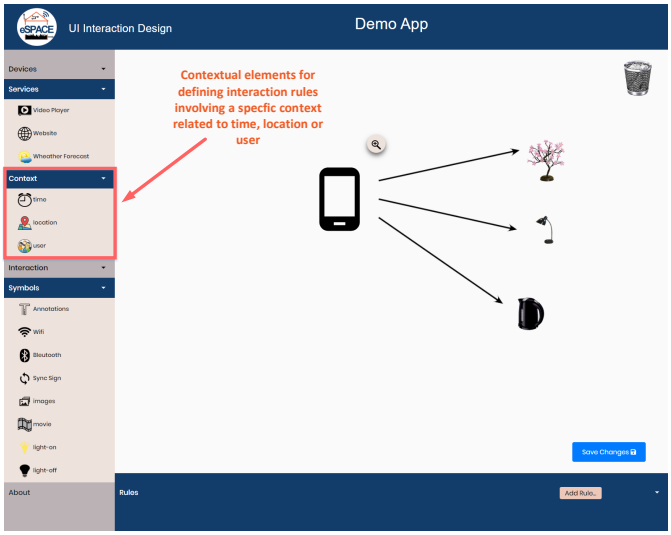


Figure 7.6: *Interaction* view with a walkthrough layer showing the functionality of the contextual elements in the sidebar

In order to further provide an explanation about the tools' functionality, some help icons should be provided next to the dropdown options in our different views, mainly in the *interaction* and *rules* view. When hovering such a help icon (⃝), users would be prompted with a short explanatory text in a speech bubble above the icon.

Next, we consider making the `Edit` buttons in the *home* view wider since participants were often clicking the application instead of the edit button and mentioned that this button was quite far away from the name of the application. An additional solution is adding an `Open` button next to the `Edit` button, which also could further indicate the presence of the editing and opening options. In general we plan to make the labels and buttons of the authoring tool a bit larger, as it seems to be borderline too small for many users. Some labels will also be renamed, for instance the `on UI creation` option should be renamed to "on app startup", however, as mentioned before some studies are necessary in order to find the most appropriate labels for defining certain interaction rules.

In terms of functionality, apart from the missing functionality described in Section 6.5 of the previous chapter, we shortly mention some lacking functionality experienced by the participants of our study and which should be added before performing further studies. As explained earlier, participants had some trouble when selecting the target dot in the *interaction* view, resulting in the creation of an interaction arrow without

target that could not easily be removed by the participant. Therefore, the *interaction* view should provide a way for participants to either delete arrows or undo their last actions as provided in the *UI Design* view. Ideally the tool would of course provide both options. Further, some feedback should be given to the user when hovering on a dot, which will also help them to know when they correctly selected a target dot. Such feedback could be given by the cursor that could show a different symbol depending on the user's action. For example, when dragging the arrow, the cursor would be a *closed hand* icon and when hovering a target dot it would transform into an *open hand* icon. Additionally the target dot could take another colour once hovered, to indicate that the user can drop the arrow on colour change. Related to the *interaction* view, one could make clear that when pressing the `magnifying glass` icon next to a device, we actually "zoom into" the device to create a UI for this device by adding a zoom-in effect, as often done in Prezi[1] presentations for instance. Another indication could be including a preview of the UI on the devices in the *interaction* view, which was our original idea but could not be explored due to lack of time. This idea is illustrated in Figure 7.7 and is based on Shneiderman's mantra: "*overview first, zoom and filter, then details-on-demand*" [199], where an overview is given in the *interaction* view and details are provided in the *UI design* view.

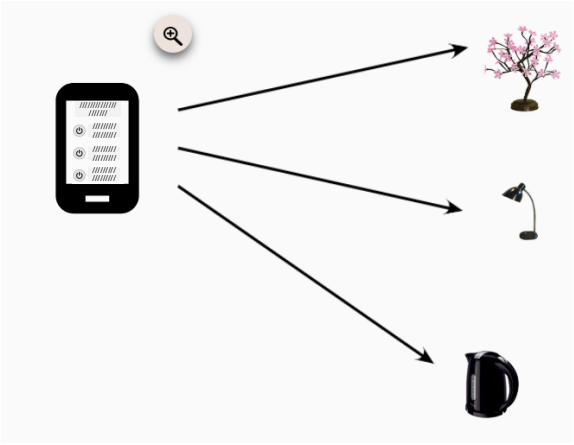
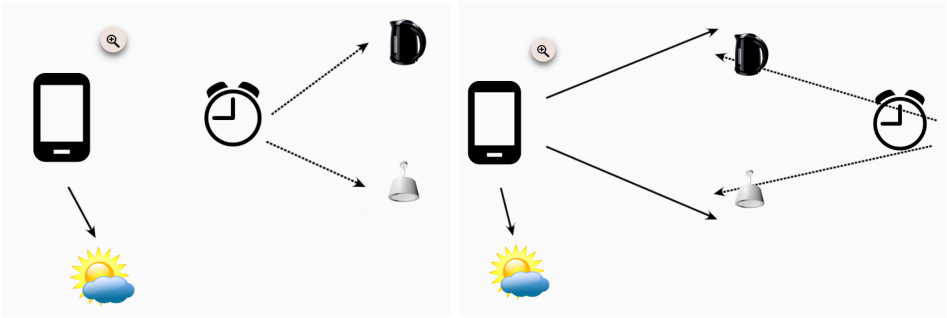


Figure 7.7: *Interaction* view with a preview of the smartphone's UI created in the *UI design* view

The toolbar provided in the *UI Design* view allowing to delete, undo, copy and preview the user interface was not used a lot by participants. Since keyboard shortcuts were also working in this view, such as `Ctrl+C` to copy or `Del` for deleting, participants principally used those. Only one participant used the copy option for copying buttons, but this caused some issues afterwards when they had to select a button for a specific

[1] `https://prezi.com`

interaction. Since the user copied the buttons they all had the same name and were thus undistinguishable in the dropdown menu. In order to avoid such a situation, we should automatically add a number after the *name* of a button when the user copies this button and also allow users to see the name of the buttons when hovering over them. Lastly, for now a rule is only active when the application containing this rule is opened in a browser tab. We already explained in the previous chapter that in future versions of the tool rules should run on the server-side and remain active even when the application is not opened. Therefore, we should also provide a way to activate or deactivate certain rules, by, for example, adding a toggle button next to each rule in the *home* view. The toggle button can then be set on active or inactive to show the state of the rule. A participant even asked the question how to deactivate a rule, explaining that when waking up earlier than 8 a.m., they would not want the light and water boiler to turn on at that time anymore. This functionality should certainly be added in a next version of our tool.



(a) *Interaction* view showing time-related interaction as well as interaction involving the weather forecast

(b) *Interaction* view showing all interaction of the *Morning Routine* application with the time contextual element moved to the right

Figure 7.8: *Interaction* view of participant P8 for the *Morning Routine* application

We end this section, on a last remark we received from a participant. Since when linking devices in the *interaction* view we create a link between two dots, when moving the elements on the canvas the arrow might end up traversing the different canvas elements. This is illustrated in Figure 7.8, where originally the elements were linked as in Figure 7.8a and the participant wanted to move the time element as shown in Figure 7.8b when adding the two extra arrows. A possible solution would be to link the arrows to the center of the canvas elements instead of to the dots. However, we chose the dots so that interactions with the same trigger could be grouped on the same dot, which could become an issue if the arrows only take into account their source and target element. Note that, participants all showed interactions from left to right, meaning that the trigger element was always located left of the action element on the canvas, sometimes the actions was also shown under the trigger when other

elements were already placed on the right. Only participant P8 changed the location of the trigger element and placed it to the right of the actions, as we have seen in Figure 7.8b. Future versions of the tool should still enforce the positioning of triggers on the left and actions on the right, but it is interesting to see that even without enforcing it, participants already applied this guideline, which is in line with the results of our elicitation study. Grouping in the *interaction* view was not performed by the participants, only grouping of actions in the *rules* view has been used by 3 participants. In order to enforce grouping in the *interaction* view, the tool could detect when a user selects the same trigger in the popup menu and automatically assign the same dots for the same triggers.

Chapter 8

Conclusions and Future Work

The best way to predict the future
is to invent it.

Alan Kay

With this dissertation we attempt to improve the end users' control over their smart devices and IoT appliances by empowering them with a solution allowing the management of all their smart technologies in one place, rather than having this control fragmented over different applications. As we have seen in the introduction, cross-device interactions are already an inherent part of our daily life, but the interaction possibilities are still limited. Moreover, people tend to use older technologies with which they are more familiar, such as using emails or USB sticks, as we have seen from the results of our questionnaire. It gets even more complicated with the rise of new IoT devices, which all come with their dedicated applications. Solutions have been introduced in the domain of cross-device and IoT research, for improving such XDI and IoT interactions. However they mostly represent closed solutions with limited capabilities to adapt to evolving user needs. Therefore, we turned towards end-user development solutions in these two domains, which provide end users with means to develop their own XD or IoT applications that can then be adapted to their changing needs and are thereby more futureproof. While some XDI and many IoT EUD tools exist, they still presented some limitations as illustrated in Chapter 2. Further, they all support different devices and functionality that is expressed using different metaphors that are sometimes difficult to understand for end users. This is also a reason why we wanted to get a better understanding of the end users' mental models when confronted with cross-device and IoT interactions. We achieved this understanding through an elicitation study where we derived a set of design guidelines mapping the

users' mental models, which served as a base to construct our end-user authoring solution. The road that led us towards this first end-user authoring solution prototype is summarised in this chapter by going through the research questions presented in Chapter 1 and providing concrete answers to them. This will give us the opportunity to summarise our achievements presented throughout this dissertation as well. Next, we continue with a critical discussion reflecting upon these achievements and some limitations. After that, we provide conclusions and directions for future work.

8.1 Summary

In order to tackle the problems introduced in Chapter 1, including the fragmented control, lack of uniformity and limited functionality as well as the lack of consensus on guidelines for a XD and IoT end-user authoring tool, we presented our main research question with its subquestions. We now provide a summary of our work that has been described in this dissertation, by answering the original research questions. For each question we discuss our findings and the corresponding contributions.

Research Question 1 *What are the main requirements for the end-user authoring of unified cross-device and IoT user interfaces?*

In order to answer this question, we analysed related work in the domain of end-user development in cross-device research and the Internet of Things, which led to a set of initial requirements. Next, we came up with a use case scenario involving XDI and IoT interactions in order to find out whether additional requirements were necessary to support all the interactions described in the scenario. As a part of the next research question we investigated model-based solutions, which allowed us to further finetune our requirements. After this broad analysis of the functionality and requirements described in related work, we unified those requirements to come up with our list of final requirements shown in Table 8.1.

These requirements reflect all important concepts retrieved from related work and from the presented use case scenario. For example, the first requirement R1, reflects the importance to provide an *overview* of all smart technologies available in the smart environment to end users. As we have seen, this is currently not the case for existing IoT applications, given that they are often limited to the integration of devices of the same brands. Consequently, the users end up with multiple applications to control all their devices and therefore lack an overview of their smart environment, as also stated by some of our study participants.

Next to an overview of the availability of smart technologies, users should also be able to easily interact with all their devices and perform cross-device interactions. In

Requirement 1 (R1)	Provide an **overview** of the smart technologies, environments and applications
Requirement 2 (R2)	**Interaction** support
Requirement 2.1 (R2.1)	Support for interaction across multiple smart technologies
Requirement 2.2 (R2.2)	Support for creation, customisation and distribution of cross-device and IoT user interfaces
Requirement 2.3 (R2.3)	Offer fine granularity UI distribution
Requirement 3 (R3)	**Shareability**
Requirement 3.1 (R3.1)	Support for sharing and integration of apps in a central smart apps repository
Requirement 3.2 (R3.2)	Enable sharing of applications, user interfaces or parts of a user interface with specific users
Requirement 4 (R4)	**Extensibility**
Requirement 4.1 (R4.1)	Offer extensibility at the level of communication protocols, devices and user interfaces
Requirement 4.2 (R4.2)	Enable the integration of third-party applications
Requirement 4.3 (R4.3)	Offer extensibility of adaptive behaviour and distribution configurations
Requirement 5 (R5)	**Reuseability**
Requirement 5.1 (R5.1)	Support for reuse and combination of existing user interfaces
Requirement 5.2 (R5.2)	Support for reuse and combination of existing functionality
Requirement 6 (R6)	**Portability**
Requirement 6.1 (R6.1)	Offer platform independence
Requirement 6.2 (R6.2)	Support for context awareness
Requirement 7 (R7)	Support for **end-user development**

Table 8.1: Requirements for XDI and IoT end-user authoring tools

the introduction of this dissertation we presented the results of Google and Microsoft studies showing that many people switch from one device to another to perform certain daily tasks. Sometimes, this is done to continue the same activity on another device, requiring them to refind the same information on the other device. This activity could be made less complicated by using some cross-device interaction technique that would allow users to easily transfer data from one device to another. When asking how people usually perform a picture transfer between two devices, they still often use a USB stick, as shown in the results of the questionnaire described in Chapter 5. Therefore, we refer to *interaction support* as a second important requirement (R2), for improving a user's control over their smart technologies.

In addition, the concept of *reusability* is relevant as well, since it prevents people from having to start from scratch by reusing the same components in similar or different ways. The reusability of rules has been on the requirements list of recent systems presented in related work, such as TARE [93] and CMT [212] and is used as an evaluation criteria by Akiki et.al. [6] for adaptive model-driven UI development

systems. The need for reusability is not only reflected through R5, since *sharing* (R3) can also be seen as a form of reusability of components created by other users.

As the number of devices and IoT appliances grow over time, it is also important to make a solution futureproof by supporting *extensibility* on multiple levels. One can, for example, increase the extensibility by making it easy to add new communication protocols, devices or user interface elements. Further, extensibility can also be shown through easy integration of third-party applications and services. It is essential to provide a system that is open for future changes, allowing the system to evolve not only together with technology, but also with user needs, as reflected through R4 and its subrequirements.

Naturally, a crucial element for a solution supporting the creation of cross-device and IoT application is *portability*, meaning that the tool as well as the applications should be accessible on many platforms and devices. Additionally, applications should be able to adapt to the device on which they are used and ideally adapt to the context of use as well, demanded by requirement R6.

Finally, the most important criteria for such an end-user authoring solution is making all these highlighted concepts available to end users, which is reflected by requirement R7, where we stress the importance of appropriate metaphors to present technical details to end users.

Research Question 2 *What are the necessary concepts and methods to address the requirements resulting from answering RQ1?*

We answer this question in Chapter 4, where we decided to follow a model-based approach to facilitate the development process of user-defined XD and IoT applications [110]. It further also helps structuring our approach, promotes reusability, flexibility as well as extensibility and can give us an idea on how to integrate the identified requirements of RQ1 at an early stage of the development cycle. By analysing existing model-based and model-driven solutions, we decided to integrate the adaptive aspect of a user interface as well. Since many current systems often try to adapt to the users' characteristics and needs, we deemed it necessary to also add this adaptivity option to the user-defined applications. Note that, in addition, the need for adaptation was already present in R6.2.

After the analysis of related work, we presented the design of the eSPACE reference framework and conceptual model for the creation of XD and IoT applications, that supports all requirements resulting from RQ1. The reference framework has been inspired by the CAMELEON reference framework (CRF) [40] and structures the UI development process into multiple levels of abstractions. In contrast to the CRF, we do not introduce different models for each abstraction layer, but rather provide a

single model combining the elements of each layer by using structural links. Thereby, we avoid the need for complex model transformations along the different steps of the process. Only one transformation is needed to transform the model into executable code for the user-defined applications.

Building upon the components introduced in the reference framework, we presented the eSPACE conceptual model that includes all components of cross-device and IoT solutions. The model is based on the RSL hypermedia metamodel [203] and allows the modelling of XD and IoT applications using the domain-specific extensions of the resource, selector and link concepts provided by the metamodel. In Chapter 4 we gave some examples of how such applications can be modelled using these different concepts. Lastly, the model is designed such that it fully supports all requirements depicted in Table 8.1, as described at the end of the model chapter.

Research Question 3 *Which metaphors or abstractions should be used on top of our conceptual foundation to allow end users to visualise and create their unified cross-device and IoT interactions?*

While for the previous research question we mainly focussed on the data level, for this question we focus on the visualisation level. Many end-user authoring tools have been proposed for IoT systems, some for cross-device interaction and DUI solutions. However, the proposed solutions support different functionality using different methods and metaphors which is often confusing for users. Further, some solutions are limited in terms of supported devices or present closed systems which cannot be extended. The different metaphors used by related authoring solutions are used to hide the complexity behind the creation of XD or IoT applications to the end users, but are generally based on the developers' choices. Afterwards, the authoring tool is often evaluated by end users through a usability study. To the best of our knowledge, only two of the presented authoring tools in Chapter 2 were based on a preliminary investigation of the end users' mental models regarding XDI or IoT interactions. The design of these tools is mainly based on related work and individual design choices. One of the two exceptions was found in Dey et al.'s work [74], who did perform interviews to gain an understanding about the conceptual models of users regarding context awareness in a smart home environment. While participants of this study described application scenarios in terms of if-then rules, it is still hard to find the best way to show these rules in a graphical user interface. Therefore, we performed an elicitation study in order to find how people would visualise XDI and IoT interactions so that we could base our authoring solution on what people already have in mind when thinking about such interactions. From this elicitation study we inferred design guidelines for the development of an end-user authoring tool allowing the design of XD and IoT applications.

Research Question 4 *How can we design a unified cross-device and IoT EUD authoring tool given the requirements from RQ1, the conceptual foundations from RQ2 and the guidelines including the appropriate metaphors found in RQ3?*

For the last objective of this dissertation we designed and developed the eSPACE end-user authoring tool, which allows end users to create XD and IoT applications. Therefore, we based ourselves on the artefacts that resulted from the previous research questions. In a first step we took into account our conceptual model that resulted from RQ2. As we chose the RSL-based link Server—a model-driven information system—as backend for the authoring tool, we loaded the model into the server to be able to store all concepts introduced by our model. Then, by following the UI development process described by our reference framework, we structured and linked the concepts of our model so that it would form final user interfaces, which could be converted into executable code for an application supporting cross-device and IoT interactions. Note that, by using the conceptual model and reference framework in our authoring tool, we also validate both of these artefacts. Next, we designed the frontend of the authoring tool according to the design guidelines originating from RQ3. As explained in Chapter 6, we chose web technologies, such as JavaScript and HTML5, to implement this tool so that it would be compatible with a large number of devices. The resulting user-defined applications created with the authoring tool are web applications as well. Along the development cycle of the eSPACE authoring tool, we incorporated a significant amount of requirements which were presented in RQ1. Finally, as a first form of validation of our design guidelines and tool, we performed an initial study with 8 participants having a non-computer science background. The study revealed that end users could use our tool to create IoT and XD applications after reading a small tutorial document and watching a short demonstration video. Most participants described the eSPACE authoring tool as organised, useful, effective and easy to use. Overall our tool scored well on user satisfaction. While these first results are highly encouraging we do take them lightly given that our study was done with a low number of participants and that participants were taken from our entourage. Our observations during this study were very instructive, as we have seen which parts of the authoring tool should be improved, being mainly some better word choices in certain dropdowns. Further, participants' feedback also pointed out that more explanation of all functionality provided by our tool is required as well as some help menus. In Chapter 7 we described in more details how this initial user study helps us to line out future directions for our authoring tool.

8.2 Discussion and Limitations

In the previous section we summarised how our research trajectory led to our proof-of-concept prototype of the eSPACE end-user authoring tool for the design of XD and IoT applications. In this section we discuss the limitations of the research artefacts presented in the previous section.

While we designed the eSPACE tool based on our design guidelines and used the concepts defined by our reference framework and conceptual model, eSPACE did not yet reach its full potential, since not all requirements presented in Table 8.1 are met. eSPACE therefore does not use all concepts defined in our model, such as the user management. As explained in Chapter 6, we did take into account the fact that a user is logged in and assigned all created applications and their components to the logged in user, but we do neither provide a complete login nor advanced sharing mechanism. Consequently, we did not yet validate this specific part of our conceptual model. However, in contrast, our reference framework has been fully validated by structuring the user-defined UIs as described by this framework and generating working applications from it using the eSPACE authoring tool. Note that our model and framework are meant to be applicable for any type of cross-device and IoT application. While we validated the model and framework with our authoring tool by populating it with elements for smart home applications, in theory it could be populated with any kind of devices and *things*. Given that we are not familiar with other domains such as eHealth or smart cities, it could still be that some minor changes are required to best fulfil the requirements of these other domains.

The design guidelines resulting from our elicitation study, have been tested in a first evaluation of our eSPACE authoring prototype. A more thorough assessment is still required in order to possibly adapt and improve these design guidelines. Note that, some of these design guidelines might seem trivial, yet to the best of our knowledge no design guidelines for these specific kinds of authoring tools have been presented in existing literature. Some requirements were presented by Ghiani et al. [93], which we took into account for the requirements defined in RQ1 and the design of our *rules* view. Dey et al. [74] mentioned the need for a graphical as well as textual interface, which we took into account in the design guidelines. Nonetheless, these authors did not provide a set of guidelines to be followed by developers of XD and IoT EUD authoring tools. A set of guidelines has been provided by Russis and Corno [190], but it has been defined from related work focussing only on IoT in a specific smart home context and the authors did not investigate related work in XDI authoring tools. Further, do note that our guidelines can still evolve based on the outcome of potential future user studies and the impact of cultural characteristics that could be integrated over time, as discussed in Chapter 5.

The eSPACE authoring tool is a first prototype and can definitely be improved by supporting the remaining requirements, taking into account the results of our initial user study and by performing extra user evaluations. Adding new ACs and UIes would also add more functionality to the tool, as for now it only includes basic UI elements and functionality. The adaptability of UI elements to the context of use is, for example, very limited and could be significantly improved. In order to conclude this section we compare our eSPACE prototype with the ISO product quality model shown in Figure 8.1. The components surrounded by an orange outline are the ones we focussed on for our initial eSPACE prototype.

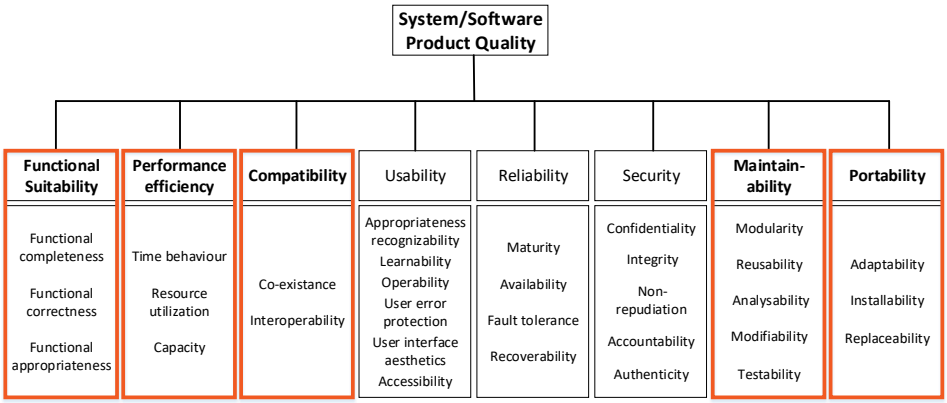


Figure 8.1: ISO/IEC 25010:2011 product quality model

We focussed on providing end users with the functionality of creating XD and IoT applications, which is accomplished in our tool by using the different views introduced in Chapter 6. However, as explained above not all our requirements were satisfied. While implementing our different components we also paid attention to performance. By using Roels' implementation of the RSL link Server [184]—which has improved performance compared to previous implementations—as well as limiting the number of API calls in the front-end authoring tool, we guarantee a good resource utilisation. By integrating different devices and smart objects into our authoring tool, we demonstrated high compatibility. We included smart plugs, smart light bulbs of different brands and a smart TV, as described in our use cases at the end of Chapter 6. Interoperability has been shown by making these smart technologies interact with each other using the pipeline or rules metaphor in our multiple authoring environments. We did not focus on the usability yet, although we performed an initial study to get a general idea of the desirability, usefulness and interface quality of our initial eSPACE prototype. We still plan to conduct usability studies in the future in order to improve the tool's usability. The eSPACE authoring tool is a first prototype and therefore still contains some bugs and is not fault tolerant. The reliability is thus something to improve by making eSPACE more robust. Security in the cross-device

and IoT research could be a topic for an entire PhD dissertation. In our research and during the development of our tool we did not take security into account very much as it was beyond the scope of our research topic. We simply make use of the user accessibility properties of RSL, so that a user can only get access to their own devices and applications. By using a model-based approach, we provide a highly modular and reusable backend, as components can be reused and linked together using different structures. The frontend could easily be replaced or adapted without requiring to change anything to the model on the server side. Finally, as required by R6, we implemented the authoring tool using web technologies for a better portability.

8.3 Conclusion

This dissertation aims at empowering the end users to better manage their smart technologies and make these technologies communicate with each other through the use of an end-user authoring solution for the creation of XD and IoT applications. Therefore, we first investigated existing related work in order to find the requirements for building such a solution. Next, we came up with a use case scenario, that resulted in additional requirements. As we chose a model-based approach for facilitating the development process of user-defined applications, we introduced the eSPACE reference framework and conceptual model based on the established requirements. On the one side, the reference framework provided a way to structure this UI development process, while on the other side, the conceptual model further elaborate on the concepts presented in the framework and allows a way to store all this information using the RSL library. Through the use of the RSL link Server API all information stored in this library can be accessed and modified. The reusability, flexibility, extensibility, sharing and adaptiveness to the context of use of the model and framework has been demonstrated in Chapter 4 by some use cases. In addition, effectiveness of both the model and framework has been demonstrated by integrating them into our eSPACE authoring tool. In contrast to existing EUD authoring tools in the domain of XDI and IoT, we first deepened our analysis of the users' mental models concerning XDI and IoT interaction, in order to better integrate their needs into our authoring tool's user interface. Based on the design guidelines that resulted from an elicitation study aiming at finding these mental models, we designed our authoring tool. The eSPACE authoring tool supports most of the presented requirements as well, allowing end users to create their own XD and IoT applications by using different abstractions or metaphors. An initial study aiming at assessing the desirability and user satisfaction of our tool revealed that end users can use eSPACE to create IoT and XD applications. The positive results showed that the eSPACE authoring tool is on the right track, but of course still requires further improvements and additional user studies. We discuss these future improvements in the next section. Finally, we hope

that our research artefacts can still grow over time and that they will inspire existing research in the domain of EUD of cross-device interaction and IoT applications, and consequently help end users to have more control over their smart environments.

8.4 Future Work

Although this dissertation presents in-depth work on informing the design of an end-user authoring tool that enables end users gaining a better control over their smart environments through the design of their own XD and IoT applications, given the limited time frame for this research, we prioritised certain research artefacts and see a window of opportunities for future work.

As mentioned in Section 8.2, our eSPACE authoring tool does not yet support all the requirements defined in Table 8.1. Given the results of our initial study, we can consider that the tool is a decent first prototype in terms of overall design, but still needs some improvements for the formulation of certain functionality described in the dropdowns of the popup windows in the *interaction* and *rules* view. While in the first place, the next step would be to improve the tool based on the participants feedback and results of the study as well as *integrating the remaining requirements*, another important next phase is evaluating the tool with end users that already own a number of smart devices and letting them use the tool for a certain period of time in their homes. Such an in-situ evaluation would give us more valuable feedback and insights on how to further improve the tool in the future. Depending on the results of such studies, our defined requirements and design guidelines might be refined and evolve over time. Note that, the current version of eSPACE works with one instance of the RSL link Server, in order to develop such high scale studies we will probably require to have multiple instances of the server running at peoples' home and be able to communicate with each other.

Since we want our authoring tool to evolve over time, we need to *guide developers* on how to extend the set of existing functionality (ACs), user interface elements, devices and *things*. Therefore we plan to provide a manual explaining the tools required for the integration of new entities into the RSL information storage and authoring tool. It could be interesting to let these developers populate our tool with smart technology for other domains, such as retail or eHealth. By doing so, one might still find some missing functionality or concepts we did not think about.

A next step could also be the *integration of different modalities* either to control the authoring process itself or as extra functionality in the resulting user-defined applications. Modalities such as speech [136] and gestures [22, 207] have already been integrated in some of the model-based solutions presented in this dissertation

from which we can draw inspiration from. As explained in Chapter 4, one could use ACs to implement gesture interaction as shown with the `touch-and-throw` gesture. Adding new modalities can be very challenging, particularly in terms of recognition. Research has to be done in order to identify the best tools that can be used that are reliable and have decent recognition rates. Further, our design guidelines will need to be extended for supporting these new modalities.

When mentioning the recognition of patterns of voice and gestures, one might think of artificial intelligence (AI) and machine learning algorithms and techniques. Going beyond the use of AI for improving pattern recognition of different modalities, we foresee the *integration of artificial intelligence* into our authoring tool in order to help end users create better UIs and applications. Since end users do normally not have the same skills as professional designers, we should support them during the UI design process. Therefore, in future work, we could investigate how to recommend alternative and potentially better designs to end users, by extending the eSPACE end-user authoring tool. These alternative design recommendations could be optimised by applying some AI techniques and algorithms based on existing well-established user interface and interaction design guidelines. The resulting human-AI interaction [10] during the UI design process might lead to better user-defined user interfaces. However, the main challenge is to find a way to provide end users with necessary feedback in order that they are able to develop usable and aesthetically pleasing user interfaces. There is also an opportunity to not only support end users in the graphical design of their UIs but to provide them suggestions on the interaction level. The use of human-AI interaction in end-user UI development tools definitely introduces some challenges, but also offers new opportunities for enhanced designs by end users. Thorough investigations are necessary in order to fully capture the potential of human-AI interaction in end-user UI development by performing extensive usability and user experience studies. Additionally such AI assistance could be helpful if we would like to add new types of interfaces that users may not be familiar with. Given that foldable displays start to emerge on recent smartphones, our authoring tool could foresee the creation of interfaces for foldable screens. One could even integrate shape changing interfaces [153], introducing quite some challenges as new kinds of UI properties would have to be integrated.

Finally, as we mentioned earlier in this section, user testing and evaluation will be required for each of these future extensions for our end-user authoring tool in order to guarantee a good user experience. We would like to end this dissertation with a final quote by Douglas Adams:

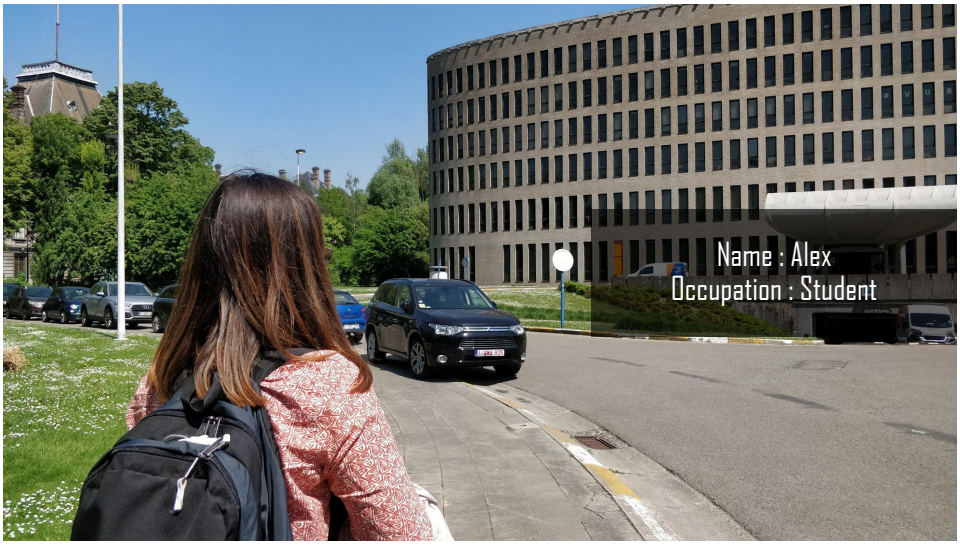
"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools." — Douglas Adams

Appendix A

Elicitation Study

A.1 Scenario

The scenario involving cross-device and IoT interactions has been presented to the study participants in the form of PowerPoint slides and are presented in this section.



Alex has a smartwatch and a smart scale to keep track of her physical activities and weight

Back home, Alex transfers her notes from her tablet to her laptop and continues working on her laptop for a while…

A few hours later, Alex' boyfriend comes home and the couple decides to watch a movie together in the living room

Therefore, Alex takes her phone, browses for a movie on her phone and displays the movie on the TV.

When doing this, the following actions take place:

- The movie shows on the TV
- The smartphone becomes a remote control for the TV
- The light in the living room turns off
- The ambient lights at the back of the TV turn on

After a while, Alex wants to prepare some popcorn and get some drinks in the kitchen

Instead of pausing the movie, she copies the movie to her smartphone to continue watching while making the popcorn (while her boyfriend continues watching the movie on the TV)

The following actions take place:

- The movie is shown on the TV but is now also shown on Alex' smartphone

While preparing the popcorn and drinks, Alex adds the calories of the popcorn and her drink to her calorie tracker application on her phone

The following actions take place:

- The calories are updated on the calorie tracker application
- The data from the calorie tracker application synchronises with her fitness application on her smartwatch
- Since she exceeded the 2 200kcal today, Alex receives a notification on her smartwatch warning her about this excess

When Alex comes back from the kitchen, she stops the movie on her phone and continues watching it on the TV

The following actions take place:

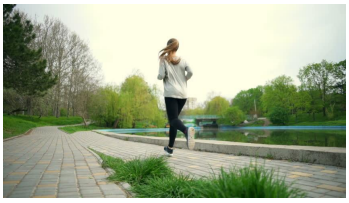
- The movie is no longer shown on Alex' smartphone
- Alex' smartphone becomes a remote control for the TV again

After the movie finished, they turn off the TV and go to bed

When turning off the TV, the following actions take place:

- The TV turns off
- The ambient lights at the back of the TV turn off
- The light in the living room turns on

The next morning, Alex goes running in a park nearby



While running, her heartrate and pace is monitored by her smartwatch. Her smartwatch defined a running track depending on the calories she needs to burn.

The following actions take place:

- Since Alex went over her daily calorie intake yesterday evening, the running track defined by the smartwatch is longer than usual
- When Alex has 50 meters left to run, the smartwatch sends a vibration to motivate her to sprint the last few meters
- If she broke her speed record, she receives a notification at the end of the running session

Alex designed these applications to help her facilitate her daily life routine.

- How would you graphically represent the functionality and interactions between the different components in this scenario?
  For example, how would you draw that the interaction with Alex' phone triggers the TV to turn on?
- How would you show that one output of a device is used as input for another device. For instance, on page 5 the amount of calories kept by the calorie tracker application on Alex' smartphone (output) is used as input by the smartwatch to notify Alex about an excess.
- Go through the slides one by one and try to describe the interactions present on each slide in a single graphical drawing.
- Try to be precise and add as many details as possible.

A.2 Post-Survey Questionnaire

The following custom questionnaire was used to gain insight into the knowledge of people concerning the terms *cross-device interaction* and *Internet of Things*. It further gave us an idea of the number of smart devices and *things* people own, as discussed in Chapter 5.

Post survey Questionnaire

Age: O Under 18 years
O 19-25 years
O 26-40 years
O 41-55 years
O Over 55 years

Gender: _____ Education: _____

Highest educational degree obtained or pursuing: _____

Position: _____

Did you ever hear about the term "Internet of Things" (IoT)? _____

If yes, are you familiar with IoT technology? _____

If yes, do you have any IoT devices at home (if so, which ones)?_____

How good are you in using technology?

Very bad (1)	(2)	Neutral (3)	(4)	Very good (5)

Did you ever hear about the term "cross-device interaction"? _____

If yes, how often do you perform cross-device interaction? _____

If yes, how are you performing cross-device interaction? _____

How do you usually transfer a picture from your phone to your computer or other devices? (e.g. email, USB stick, specific application, etc.)

_____

Do you own some smart devices (e.g. smartphone, tablet, smartwatch, etc.)? If yes, which one(s):

_____

How comfortable or easy was this survey for you?

Very difficult (1)	Quite difficult (2)	Neutral (3)	Easy (4)	Very easy (5)

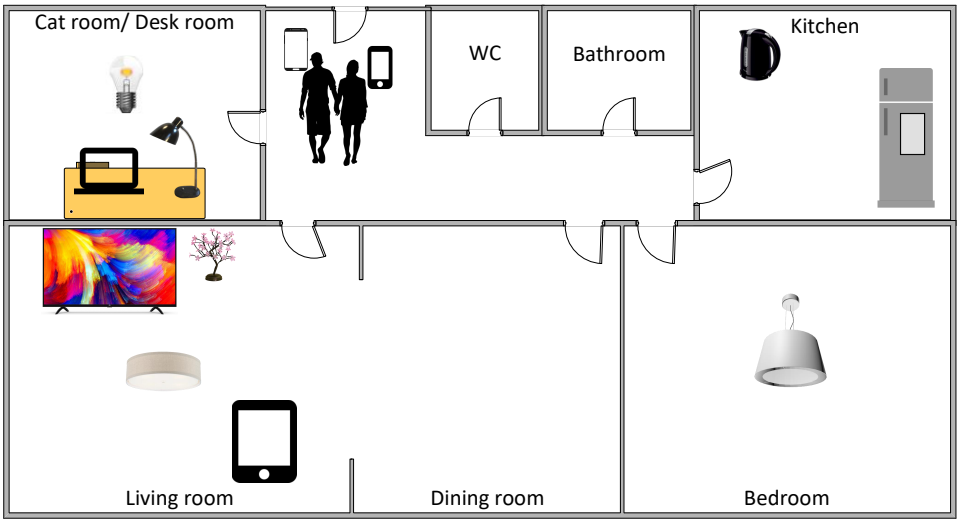
THANKS FOR YOUR PARTICIPATION!

Appendix B

Evalutation of eSPACE

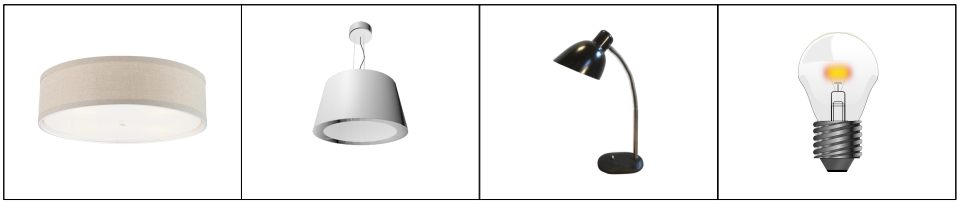
B.1 Tutorial Document

During this user study you will be playing the role of Alex a young student that wants to create some interaction across her smart technologies comprising some smart devices as well as Internet of Things devices. Below you can see a picture of her smart home:

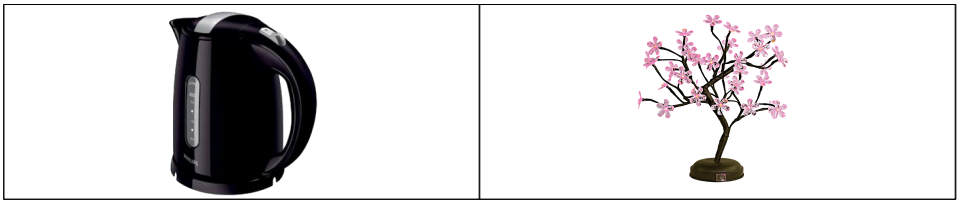


The smart home comprises the following Internet of Things devices:

– 4 smart light bulbs for the Living Room Light, Bed Room Light, Desk Light and Cat Room Light

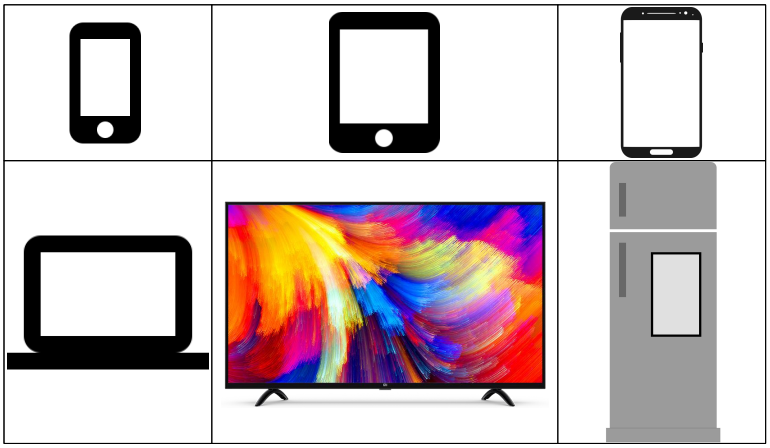


– 2 smart power plugs to control the Water Boiler and the Decorative Light Tree (situated next to your TV)

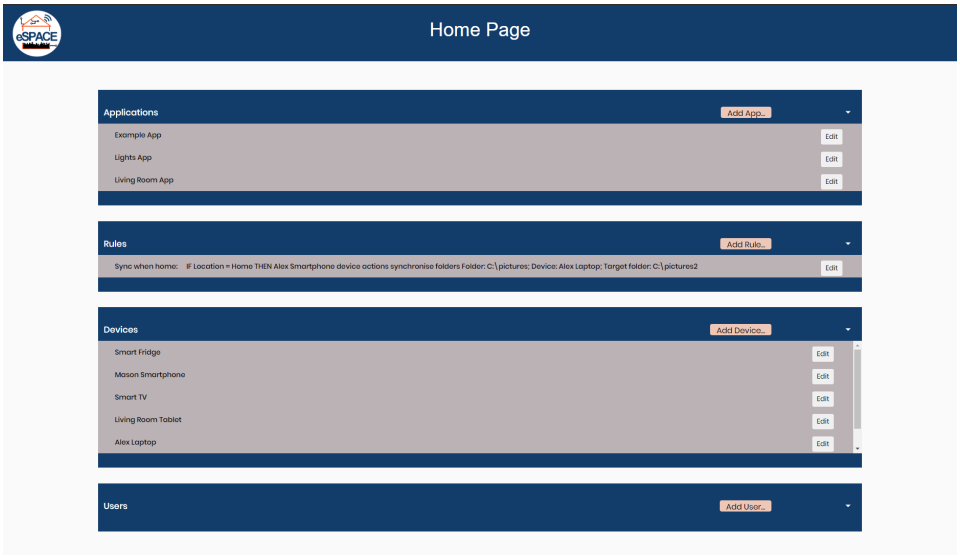


Further you also have the following screen devices for which graphical user interfaces can be created:

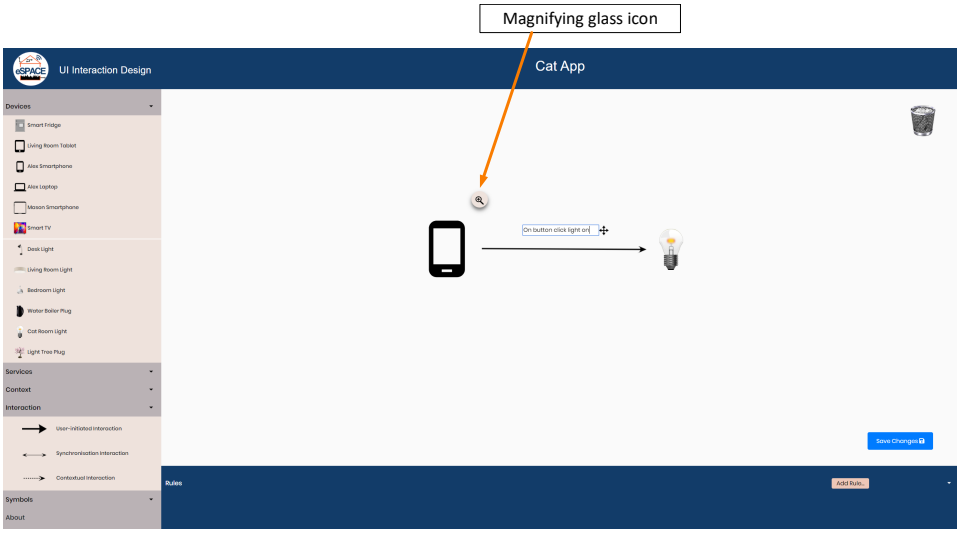
– Alex Smartphone, Living Room Tablet, Mason Smartphone, Alex Laptop, Smart TV and Smart Fridge



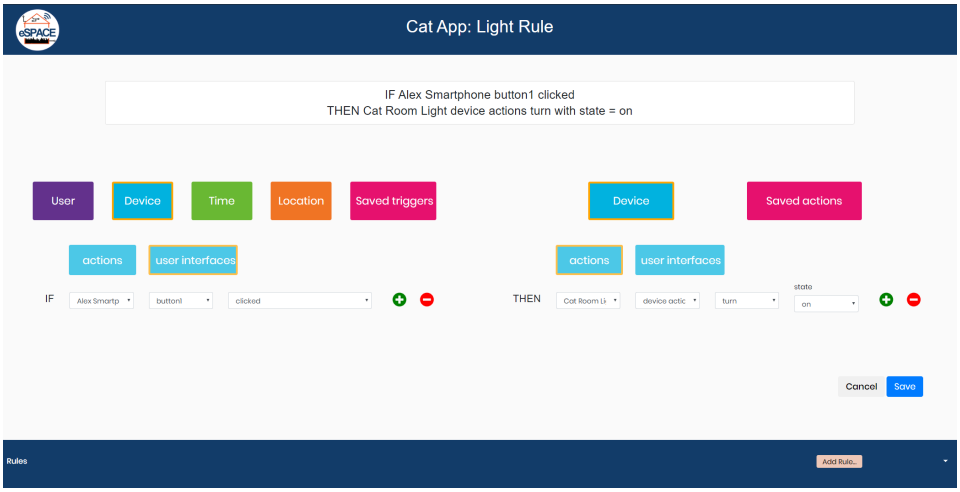
In order to interact with some of these devices, you want to make three different applications. Your task is to create each application by using the eSPACE authoring tool. This tool allows to define interaction across your devices and contains 4 different views: The **Home View** provides an overview of all your created applications and rules as well as your devices, as shown in the screenshot below:



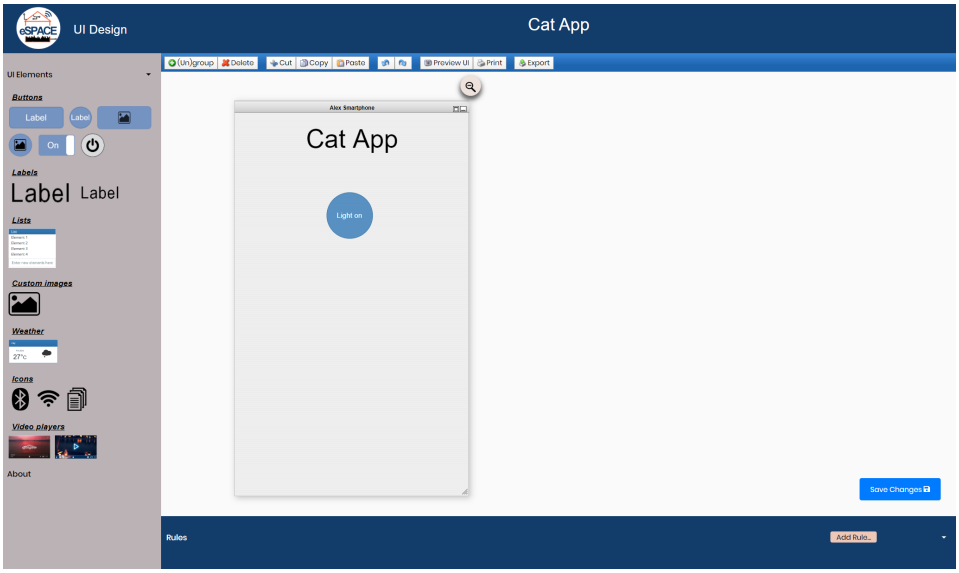
The **Interaction View** shows you a graphical representation of your interaction rules. An interaction rule defines how devices are supposed to interact with each other. The left sidebar contains elements that can be dragged and dropped to the canvas. The interaction shown on the screenshot below indicates that when a button is pressed on Alex' smartphone the cat light will turn on.



The interaction rules can either be defined using the Interaction View or by using the **Rules View** shown in the next screenshot:



On this screenshot the same interaction rule is created but using an IF-THEN statement. In order to create a graphical user interface for a specific device, the **UI Design View** can be used, which can be accessed by pressing the magnifying glass icon in the Interaction View. This is needed to create a button on Alex' smartphone before defining the above interaction rules. A screenshot of the UI Design View is shown on the next page.



In order to know more about how to use this tool you can now watch the demo video of the tool. The demo video will show you how to create an application called

"Evening Routine" app which contains a user interface for Alex' smartphone with a button to turn on the TV and the light tree. After watching the video you can start by creating the first application which description is given below:

1. The Simple Controller Application

   This application is meant to be used on the **Living Room Tablet** and should be able to control the **smart TV**, the **ceiling light of the living room** and the **decorative light tree** next to the TV. In order to control these three devices, you want to create a user interface (UI) made of three buttons, one to control each of these smart technologies.

2. The Grocery List Application

   This application contains **two user interfaces**, one for the **smartphone** and one for the **smart fridge**. The smartphone UI simply contains a **list of groceries**, while the fridge UI depicts a **list of groceries** and **some family pictures**. The list of groceries of the smartphone is synchronised with the one on the smart fridge. Which means that when something changes on the grocery list of the smartphone, the same changes will be done in the list on the smart fridge and vice versa.

3. The Morning Routine Application

   The application consists of a **smartphone UI** showing the **weather** in *Brussels* and allowing to control the **bedroom light** and **water boiler**. Further the application also triggers the light of the bedroom and the water boiler to turn on at **8 a.m**.

If there are any questions, please feel free to ask your study supervisor.

B.2 Post-Survey Questionnaires

B.2.1 Microsoft Reaction Cards

Part 1 on 3

Read over the following list of words. Considering the tool you have just used, choose those words that best describe your experience with it. You can choose between 5 to 10 words. Classify those into two categories: Highly Revelant and Relevent. (Max. 5 words per category)

Irrelevant	Stressful	Inspiring	Approachable
Predictable	Confusing	Overwhelming	Simplistic
Organized	Annoying	Unattractive	Difficult
Innovative	Familiar	Consistent	Relevant
Effective	Straight Forward	Advanced	Inconsistent
Trustworthy	Efficient	Undesirable	Satisfying
Flexible	Exciting	Useful	Dull
Complex	Attractive	Easy to use	Desirable
Engaging	Time-consuming	Unpredictable	Comfortable
Helpful	Reliable	Creative	Ineffective

Highly relevant	Relevant

B.2.2 Post-Study System Usability Questionnaire

Part 2 on 3

On a scale between Strongly Agree to Strongly Disagree, please rate the following statements:

	Strongly Agree						Strongly Disagree	
	1	2	3	4	5	6	7	N.A.
1. Overall, I am satisfied with how easy it is to use this system.								
2. It was simple to use this system.								
3. I was able to complete the tasks and scenarios quickly using this system.								
4. I felt comfortable using this system.								
5. It was easy to learn to use this system.								
6. I believe I could become productive quickly using this system.								
9. The information (such as online help, on-screen messages, and other documentation) provided with this system was clear.								
10. It was easy to find the information I needed.								
11. The information was effective in helping me complete the tasks and scenarios.								
12. The organization of information on the system screens was clear.								
13. The interface of this system was pleasant.								
14. I liked using the interface of this system.								
15. This system has all the functions and capabilities I expect it to have.								
16. Overall, I am satisfied with this system.								

B.2.3 Informative Questionnaire

Part 3 on 3

Please fill in the last series of questions as best as you can (select only one option):

	Very Easy	Easy	Neutral	Difficult	Very Difficult
1. How easy or difficult was it to create the Simple Controller Application?					
2. How easy or difficult was it to create the Grocery List Application?					
3. How easy or difficult was it to create the Morning Routine Application?					

4. Which view did you prefer for defining interaction rules?	Rules View	Interaction View	Both Equally Prefferred	Both Equally Disliked

5. Any suggestions to improve the tool?	

6. Comments?	

References

[1] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, 1999. "*UIML: An Appliance-Independent XML User Interface Language*". Computer Networks, 31(11-16):1695–1708, 1999. https://doi.org/10.1016/S1389-1286(99)00044-4.

[2] H. Ajam and M. Mu, June 2017. "*A Middleware to Enable Immersive Multi-Device Online TV Experience*". Proceedings of TVX 2017, International Conference on Interactive Experiences for TV and Online Video, pages 27–32, Hilversum, Netherlands. https://doi.org/10.1145/3084289.3089919.

[3] P. A. Akiki, A. K. Bandara, and Y. Yu, June 2012. "*Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications*". Proceedings of ICEIS 2012, 14th International Conference on Enterprise Information Systems, pages 72–77, Wroclaw, Poland. SciTePress. https://doi.org/10.5220/0003975800720077.

[4] P. A. Akiki, A. K. Bandara, and Y. Yu, June 2013. "*Cedar Studio: an IDE Supporting Adaptive Model-Driven User Interfaces for Enterprise Applications*". Proceedings of EICS 2013, Symposium on Engineering Interactive Computing Systems, pages 139–144, London, United Kingdom. https://doi.org/10.1145/2494603.2480332.

[5] P. A. Akiki, A. K. Bandara, and Y. Yu, June 2013. "*RBUIS: Simplifying Enterprise Application User Interfaces Through Engineering Role-Based Adaptive Behavior*". Proceedings of EICS 2013, Symposium on Engineering Interactive Computing Systems, pages 3–12, London, United Kingdom. https://doi.org/10.1145/2494603.2480297.

[6] P. A. Akiki, A. K. Bandara, and Y. Yu, 2014. "*Adaptive Model-Driven User Interface Development Systems*". ACM Computer Surveys, 47(1):9:1–9:33, 2014. https://doi.org/10.1145/2597999.

[7] P. A. Akiki, A. K. Bandara, and Y. Yu, 2017. "*Visual Simple Transformations: Empowering End-Users to Wire Internet of Things Objects*". ACM Transactions on Computer-Human Interaction, 24(2):10:1–10:43, 2017. https://doi.org/10.1145/3057857.

[8] A. I. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, 2015. "*Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications*". IEEE Communications Surveys and Tutorials, 17(4):2347–2376, 2015. https://doi.org/10.1109/COMST.2015.2444095.

[9] A. I. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, 2015. "*Toward Better Horizontal Integration Among IoT Services*". IEEE Communications Magazine, 53(9):72–79, 2015. https://doi.org/10.1109/MCOM.2015.7263375.

[10] S. Amershi, D. S. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. T. Iqbal, P. N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz, May 2019. "*Guidelines for Human-AI Interaction*". Proceedings of CHI 2019, Conference on Human Factors in Computing Systems, page 3, Glasgow, Scotland, UK. https://doi.org/10.1145/3290605.3300233.

[11] M. Ammar, G. Russello, and B. Crispo, 2018. "*Internet of Things: A Survey on The Security of IoT Frameworks*". Journal of Information Security and Applications, 38:8–27, 2018. https://doi.org/10.1016/j.jisa.2017.11.002.

[12] C. Anderson, January 2012. Maker: The New Industrial Revolution. Crown Business, January 2012.

[13] K. Ashton, 2009. "*That 'Internet of Things' Thing*". RFID Journal, 22(7):97–114, 2009.

[14] L. Balme, A. Demeure, N. Barralon, J. Coutaz, and G. Calvary, November 2004. "*CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces*". Proceedings of EUSAI 2004, Second European Symposium on Ambient Intelligence, pages 291–302, Eindhoven, The Netherlands. https://doi.org/10.1007/978-3-540-30473-9_28.

[15] H. Balzert, F. Hofmann, V. Kruschinski, and C. Niemann, June 1996. "*The JANUS Application Development Environment - Generating More than the User Interface*". Proceedings of CADUI 1996, International Workshop on Computer-Aided Design of User Interfaces, pages 183–208, Namur, Belgium.

[16] J. Bardram, S. Gueddana, S. Houben, and S. Nielsen, May 2012. "*ReticularSpaces: Activity-based Computing Support for Physically Distributed and Collaborative Smart Spaces*". Proceedings of CHI 2012, Conference on Human Factors in Computing Systems, pages 2845–2854, Austin, Texas, USA. https://doi.org/10.1145/2207676.2208689.

[17] B. R. Barricelli and S. Valtolina, 2017. "*A Visual Language and Interactive System for End-User Development of Internet of Things Ecosystems*". Journal of Visual Languages and Computing, 40:1–19, 2017. https://doi.org/10.1016/j.jvlc.2017.01.004.

[18] A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. Van Kranenburg, S. Lange, and S. Meissner, 2013. Enabling Things to Talk: Designing IoT Solutions with the IoT Architectural Reference Model. Springer Berlin Heidelberg, 2013. https://doi.org/10.1007/978-3-642-40403-0.

[19] A. Bellucci, A. Vianello, Y. Florack, L. Micallef, and G. Jacucci, 2019. "*Augmenting objects at home through programmable sensor tokens: A design journey*". International Journal of Human-Computer Studies, 122:211–231, 2019. https://doi.org/10.1016/j.ijhcs.2018.09.002.

[20] J. Benedek and T. Miner, July 2002. "*Measuring Desirability: New Methods for Evaluating Desirability In a Usability Lab Setting*". Proceedings of UPA 2002, European Conference on Usability Professionals Association, volume 2003, page 57, Orlanda, Florida, USA.

[21] Y. V. Berghe, 2004. Etude et Implémentation d'un Générateur d'Interfaces Vectorielles à Partir d'un Langage de Description d'Interfaces Utilisateur. Master's thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

[22] F. Beuvens and J. Vanderdonckt, May 2012. "*UsiGesture: An Environment for Integrating Pen-Based Interaction in User Interface Development*". Proceedings of RCIS 2012, 6th International Conference on Research Challenges in Information Science, pages 1–12, Valencia, Spain. https://doi.org/10.1109/RCIS.2012.6240449.

[23] J. T. Biehl, W. T. Baker, B. P. Bailey, D. S. Tan, K. M. Inkpen, and M. Czerwinski, April 2008. "*IMPROMPTU: A New Interaction Framework for Supporting Collaboration in Multiple Display Environments and Its Field Evaluation for Co-located Software Development*". Proceedings of CHI 2008, Conference on Human Factors in Computing Systems, pages 939–948, Florence, Italy. https://doi.org/10.1145/1357054.1357200.

[24] M. Blumendorf, 2009. Multimodal Interaction in Smart Environments: a Model-based Runtime System for Ubiquitous User Interfaces. Ph.D. thesis, Berlin Institute of Technology. https://doi.org/10.14279/depositonce-2229.

[25] M. Blumendorf and S. Albayrak, July 2009. "*Towards a Framework for the Development of Adaptive Multimodal User Interfaces for Ambient Assisted Living Environments*". Proceedings of UAHCI 2009, 5th International Conference on Universal Access in Human-Computer Interaction, pages 150–159, San Diego, California, USA. https://doi.org/10.1007/978-3-642-02710-9_18.

[26] M. Blumendorf, S. Feuerstack, and S. Albayrak, October 2006. "*Event-Based Synchronization of Model-based Multimodal User Interfaces*". Proceedings of MDDAUI 2006, Workshop on Model-Driven Development of Advanced User Interfaces, Genova, Italy.

[27] M. Blumendorf, S. Feuerstack, and S. Albayrak, November 2007. "*Multimodal User Interaction in Smart Environments: Delivering Distributed User Interfaces*". Proceedings of AmI 2007, Workshop on Constructing Ambient Intelligence, pages 113–120, Darmstadt, Germany. https://doi.org/10.1007/978-3-540-85379-4_14.

[28] M. Blumendorf, S. Feuerstack, and S. Albayrak, Janary 2008. "*Multimodal Smart Home User Interfaces*". Proceedings of IUI4AAL 2008, Workshop on Intelligent User Interfaces for Ambient Assisted Living, Maspalomas, Grand Canaria, Spain.

[29] M. Blumendorf, S. Feuerstack, and S. Albayrak, May 2008. "*Multimodal User Interfaces for Smart Environments: The Multi-Access Service Platform*". Proceedings of AVI 2008, Conference on Advanced Visual Interfaces, pages 478–479, Napoli, Italy. https://doi.org/10.1145/1385569.1385665.

[30] L. Borman, 1996. "*SIGCHI: The early years*". ACM SIGCHI Bulletin, 28(1):4–6, 1996. https://doi.org/10.1145/249170.249172.

[31] A. Bragdon, R. DeLine, K. Hinckley, and M. R. Morris, November 2011. "*Code Space: Touch + Air Gesture Hybrid Interactions for Supporting Developer Meetings*". Proceedings of ITS 2011, International Conference on Interactive Tabletops and Surfaces, pages 212–221, Kobe, Japan. https://doi.org/10.1145/2076354.2076393.

[32] M. Brambilla and P. Fraternali, 2018. "*The Interaction Flow Modeling Language (IFML), Version 1.0. Technical report*". http://www.ifml.org/. Accessed: 2018-03-13.

[33] M. Brambilla, E. Umuhoza, and R. Acerbis, 2017. "*Model-Driven Development of User Interfaces for IoT Systems via Domain-Specific Components and Patterns*". Journal of Internet Services and Applications, 8(1):14:1–14:21, 2017. https://doi.org/10.1186/s13174-017-0064-1.

[34] P. Brizzi, D. Conzon, H. Khaleel, R. Tomasi, C. Pastrone, M. A. Spirito, M. Knechtel, F. Pramudianto, and P. A. Cultrona, September 2013. "*Bringing the Internet of Things Along the Manufacturing Line: A Case Study in Controlling Industrial Robot and Monitoring Energy Consumption Remotely*". Proceedings of ETFA 2013, 18th Conference on Emerging Technologies & Factory Automation, pages 1–8, Cagliari, Italy. https://doi.org/10.1109/ETFA.2013.6647947.

[35] F. Brudy, C. Holz, R. Rädle, C. Wu, S. Houben, C. N. Klokmose, and N. Marquardt, May 2019. "*Cross-Device Taxonomy: Survey, Opportunities and Challenges of Interactions Spanning Across Multiple Devices*". Proceedings of CHI 2019, Conference on Human Factors in Computing Systems, page 562, Glasgow, Scotland, UK. https://doi.org/10.1145/3290605.3300792.

[36] F. Brudy, S. Houben, N. Marquardt, and Y. Rogers, November 2016. "*CurationSpace: Cross-Device Content Curation Using Instrumental Interaction*". Proceedings of ISS 2016, Conference on Interactive Surfaces and Spaces, pages 159–168, Niagara Falls, Ontario, Canada. https://doi.org/10.1145/2992154.2992175.

[37] M. M. Burnett and C. Scaffidi, 2014. The Encyclopedia of Human-Computer Interaction, chapter 10. End-User Development. Interaction Design Foundation.

[38] F. Cabitza, D. Fogli, R. Lanzilotti, and A. Piccinno, 2017. "*Rule-Based Tools for the Configuration of Ambient Intelligence Systems: a Comparative User Study*". Multimedia Tools and Applications, 76(4):5221–5241, 2017. https://doi.org/10.1007/s11042-016-3511-2.

[39] A. Caione, A. Fiore, L. Mainetti, L. Manco, and R. Vergallo, 2017. "*WoX: Model-Driven Development of Web of Things Applications*". Managing the Web of Things: Linking the Real World to the Web, pages 357–387. Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-809764-9.00017-2.

[40] G. Calvary, J. Coutaz, L. Bouillon, M. Florins, O. Limbourg, L. Marucci, F. Paternò, C. Santoro, N. Souchon, D. Thevenin, and J. Vanderdonckt, September 2002. "*The CAMELEON Reference Framework*". CAMELEON Project Deliverable 1.1.

[41] G. Calvary, J. Coutaz, and D. Thevenin, May 2001. "*A Unifying Reference Framework for the Development of Plastic User Interfaces*". Proceedings of IFIP 2001, International Conference on Engineering for Human-Computer Interaction, pages 173–192, Toronto, Canada. https://doi.org/10.1007/3-540-45348-2_17.

[42] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, 2003. "*A Unifying Reference Framework for Multi-Target User Interfaces*". Interacting with Computers, 15(3):289–308, 2003. https://doi.org/10.1016/S0953-5438(03)00010-9.

[43] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, N. Souchon, L. Bouillon, M. Florins, and J. Vanderdonckt, July 2002. "*Plasticity of User Interfaces: A Revised Reference Framework*". Proceedings of TAMODIA 2002, 1st International Workshop on Task Models and Diagrams for User Interface Design, pages 127–134, Bucharest, Romania.

[44] C. Cappiello, M. Matera, and M. Picozzi, 2015. "*A UI-Centric Approach for the End-User Development of Multidevice Mashups*". The journal Transactions on the Web, 9(3):11:1–11:40, 2015. https://doi.org/10.1145/2735632.

[45] C. Cappiello, M. Matera, M. Picozzi, G. Sprega, D. Barbagallo, and C. Francalanci, June 2011. "*DashMash: A Mashup Environment for End User Development*". Proceedings of ICWE 2011, International Conference on Web Engineering, pages 152–166, Paphos, Cyprus. https://doi.org/10.1007/978-3-642-22233-7_11.

[46] S. Carson, A. Furuskär, P. Jonsson, J. Kronander, P. Lindberg, R. Ludwig, K. Öhman, and J. S. Sehti, 2016. "*Ericsson Mobility Report: On The Pulse of the Networked Society*". Online.

[47] E. Cavalcante, M. P. Alves, T. Batista, F. C. Delicato, and P. F. Pires, May 2015. "*An Analysis of Reference Architectures for the Internet of Things*". Proceedings of CobRA 2015, 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures, pages 13–16, Montréal, Quebec, Canada. https://doi.org/10.1145/2755567.2755569.

[48] M. E. Cecchinato, A. Sellen, M. Shokouhi, and G. Smyth, May 2016. "*Finding Email in a Multi-Account, Multi-Device World*". Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, pages 1200–1210, San Jose, California, USA. https://doi.org/10.1145/2858036.2858473.

[49] M. Chechik, S. Nejati, and M. Sabetzadeh, 2012. "*A Relationship-Based Approach to Model Integration*". Innovations in Systems and Software Engineering, 8(1):3–18, 2012. https://doi.org/10.1007/s11334-011-0155-2.

[50] X. A. Chen and Y. Li, 2017. "*Improv: An Input Framework for Improvising Cross-Device Interaction by Demonstration*". ACM Transactions on Computer-Human Interaction, 24(2):15:1–15:21, 2017. https://doi.org/10.1145/3057862.

[51] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg, March 1994. "*An Open Agent Architecture*". Proceedings of AAAI 1994, Spring Symposium Series on Software Agents (AAAI Technical Report SS94-03), pages 1–8, Palo Alto, California, USA. ISBN 978-0-929280-59-2.

[52] K. Coninx, K. Luyten, C. Vandervelpen, J. V. den Bergh, and B. Creemers, September 2003. "*Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems*". Proceedings of Mobile HCI 2003, 5th International Symposium on Human-Computer Interaction with Mobile Devices and Services, pages 256–270, Udine, Italy. https://doi.org/10.1007/978-3-540-45233-1_19.

[53] D. Conzon, P. Brizzi, P. Kasinathan, C. Pastrone, F. Pramudianto, and P. A. Cultrona, February 2015. "*Industrial Application Development Exploiting IoT Vision and Model Driven Programming*". Proceedings of ICIN 2015, 18th International Conference on Intelligence in Next Generation Networks, pages 168–175, Paris, France. https://doi.org/10.1109/ICIN.2015.7073828.

[54] J. M. Corbin and A. Strauss, 1990. "*Grounded Theory Research: Procedures, Canons, and Evaluative Criteria*". Qualitative Sociology, 13(1):3–21, 1990. https://doi.org/10.1007/BF00988593.

[55] F. Corno, L. D. Russis, and A. M. Roffarello, May 2019. "*Empowering End Users in Debugging Trigger-Action Rules*". Proceedings of CHI 2019, Conference on Human Factors in Computing Systems, page 388, Glasgow, Scotland, UK. https://doi.org/10.1145/3290605.3300618.

[56] B. Costa, P. F. Pires, F. C. Delicato, W. Li, and A. Y. Zomaya, August 2016. "*Design and Analysis of IoT Applications: A Model-Driven Approach*". Proceedings of DASC/PiCom/DataCom/CyberSciTech 2016, 14th International Conference on Dependable, Autonomic and Secure Computing, 14th International Conference on Pervasive Intelligence and Computing, 2nd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress, pages 392–399, Auckland, New Zealand. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.81.

[57] M. F. Costabile, D. Fogli, P. Mussio, and A. Piccinno, 2007. "*Visual Interactive Systems for End-User Development: A Model-Based Design Methodology*". IEEE Transactions on Systems Man and Cybernetics - Part A, 37(6):1029–1046, 2007. https://doi.org/10.1109/TSMCA.2007.904776.

[58] M. F. Costabile, P. Mussio, L. Parasiliti Provenza, and A. Piccinno, May 2008. "*End Users as Unwitting Software Developers*". Proceedings of WEUSE 2008, International Workshop on End-User Software Engineering, pages 6–10, Leipzig, Germany. https://doi.org/10.1145/1370847.1370849.

[59] J. Coutaz, June 2010. "*User Interface Plasticity: Model Driven Engineering to the Limit!*" Proceedings of EICS 2010, Symposium on Engineering Interactive Computing System, pages 1–8, Berlin, Germany. https://doi.org/10.1145/1822018.1822019.

[60] J. Coutaz and J. L. Crowley, 2016. "*A First-Person Experience with End-User Development for Smart Homes*". IEEE Pervasive Computing, 15(2):26–39, 2016. https://doi.org/10.1109/MPRV.2016.24.

[61] J. Coutaz and J. L. Crowley, 2018. "*AppsGate, un Écosystème Domestique Programmable: "Vivre avec" Comme Retour d'xpérience*". Journal d'Interaction Personne-Système, Association Francophone d'Interaction Homme-Machine (AFIHM), 7(1 (1)):1–35, 2018.

[62] A. Coyette, S. Faulkner, M. Kolp, Q. Limbourg, and J. Vanderdonckt, November 2004. "*SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML*". Proceedings of TAMODIA 2004, 3rth International Conference on Task Models and Diagrams for User Interface Design, pages 75–82, Prague, Czech Republic. https://doi.org/10.1145/1045446.1045461.

[63] A. Coyette and J. Vanderdonckt, September 2005. "*A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces*". Proceedings of INTERACT 2005, International Conference on Human-Computer Interaction, pages 550–564, Rome, Italy. https://doi.org/10.1007/11555261_45.

[64] P. P. da Silva, June 2000. "*User Interface Declarative Models and Development Environments: A Survey*". Proceedings of DSV-IS 2000, 7th International Workshop on Interactive Systems: Design, Specification, and Verification, pages 207–226, Limerick, Ireland. https://doi.org/10.1007/3-540-44675-3_13.

[65] Y. Dahl and R. M. Svendsen, July 2011. "*End-User Composition Interfaces for Smart Environments: A Preliminary Study of Usability Factors*". Proceedings of DUXU 2011, Conference on Design, User Experience, and Usability, pages 118–127, Orlando, Florida, USA. https://doi.org/10.1007/978-3-642-21708-1_14.

[66] J. Danado and F. Paternò, 2014. "*Puzzle: A Mobile Application Development Environment Using a Jigsaw Metaphor*". Journal of Visual Languages and Computing, 25(4):297–315, 2014. https://doi.org/10.1016/j.jvlc.2014.03.005.

[67] O. Davidyuk, I. S. Milara, E. Gilman, and J. Riekki, 2015. "*An Overview of Interactive Application Composition Approaches*". Open Computer Science, 5(1), 2015. https://doi.org/10.1515/comp-2015-0007.

[68] R. de A. Maues and S. D. J. Barbosa, August 2013. "*Keep Doing What I Just Did: Automating Smartphones by Demonstration*". Proceedings of MobileHCI 2013, 15th International Conference on Human-Computer Interaction with Mobile Devices and Services, pages 295–303, Munich, Germany. https://doi.org/10.1145/2493190.2493216.

[69] D. Dearman and J. S. Pierce, April 2008. "*It's On My Other Computer!: Computing with Multiple Devices*". Proceedings of CHI 2008, Conference on Human Factors in Computing Systems, pages 767–776, Florence, Italy. https://doi.org/10.1145/1357054.1357177.

[70] A. Demeure, J. Sottet, G. Calvary, J. Coutaz, V. Ganneau, and J. Vanderdonckt, March 2008. "*The 4C Reference Model for Distributed User Interfaces*". Proceedings of ICAS 2008, 4th International Conference on Autonomic and Autonomous Systems, pages 61–69, Gosier, Guadeloupe. https://doi.org/10.1109/ICAS.2008.34.

[71] G. Desolda, C. Ardito, and M. Matera, June 2015. "*EFESTO: A Platform for the End-User Development of Interactive Workspaces for Data Exploration*". Proceedings of RMC 2015, International Rapid Mashup Challenge, pages 63–81, Rotterdam, The Netherlands. https://doi.org/10.1007/978-3-319-28727-0_5.

[72] G. Desolda, C. Ardito, and M. Matera, 2017. "*Empowering End Users to Customize their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools*". ACM Transactions on Computer-Human Interaction, 24(2):12:1–12:52, 2017. https://doi.org/10.1145/3057859.

[73] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu, April 2004. "*a CAPpella: Programming by Demonstration of Context-Aware Applications*". Proceedings of CHI 2004, Conference on Human Factors in Computing Systems, pages 33–40, Vienna, Austria. https://doi.org/10.1145/985692.985697.

[74] A. K. Dey, T. Sohn, S. Streng, and J. Kodama, May 2006. "*iCAP: Interactive Prototyping of Context-Aware Applications*". Proceedings of PERVASIVE 2006, 4th International Conference on Pervasive Computing, pages 254–271, Dublin, Ireland. https://doi.org/10.1007/11748625_16.

[75] K. R. Dittrich, S. Gatziu, and A. Geppert, September 1995. "*The Active Database Management System Manifesto: A Rulebase of ADBMS Features*". Proceedings of RIDS 1995, Second International Workshop on Rules in Database Systems, pages 3–20, Glyfada, Athens, Greece. https://doi.org/10.1007/3-540-60365-4_116.

[76] N. Elmqvist, 2011. "*Distributed User Interfaces: State of the Art*". Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, pages 1–12. https://doi.org/10.1007/978-1-4471-2271-5_1.

[77] Engelbart, Douglas C, December 1968. "*The Mother Of All Demos*". Fall Joint Computer Conference, volume 9, San Francisco, California, USA.

[78] K. Everitt, C. Shen, K. Ryall, and C. Forlines, January 2006. "*MultiSpace: Enabling Electronic Document Micro-Mobility in Table-centric, Multi-Device Environments*". Proceedings of TableTop 2006, 1st International Workshop on Horizontal Interactive Human-Computer Systems, pages 27–34, Adelaide, Australia. https://doi.org/10.1109/TABLETOP.2006.23.

[79] Federico Ciccozzi and Romina Spalazzese, October 2016. "*MDE4IoT: Supporting the Internet of Things with Model-Driven Engineering*". Proceedings of IDC 2016, 10th International Symposium on Intelligent and Distributed Computing, pages 67–76, Paris, France. https://doi.org/10.1007/978-3-319-48829-5_7.

[80] S. Feuerstack, M. Blumendorf, V. Schwartze, and S. Albayrak, May 2008. "*Model-based Layout Generation*". Proceedings of AVI 2008, Conference on Advanced Visual Interfaces, pages 217–224, Napoli, Italy. https://doi.org/10.1145/1385569.1385605.

[81] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev, 2004. "*Meta-Design: a Manifesto For End-User Development*". Communications of the ACM, 47(9):33–37, 2004. https://doi.org/10.1145/1015864.1015884.

[82] F. Fleurey, B. Morin, A. Solberg, and O. Barais, October 2011. "*MDE to Manage Communications with and between Resource-Constrained Systems*". Proceedings of MODELS 2011, 14th International Conference on Model Driven Engineering Languages and Systems, pages 349–363, Wellington, New Zealand. https://doi.org/10.1007/978-3-642-24485-8_25.

[83] D. Fogli, M. Peroni, and C. Stefini, 2017. "*ImAtHome: Making Trigger-Action Programming Easy and Fun*". Journal of Visual Languages and Computing, 42:60–75, 2017. https://doi.org/10.1016/j.jvlc.2017.08.003.

[84] J. M. C. Fonseca, 2010. "*Model-based UI XG Final Report*". https://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/. Accessed: 2018-02-10.

[85] D. S. Fors, B. Magnusson, S. G. Robertz, G. Hedin, and E. Nilsson-Nyman, July 2009. "*Ad-hoc Composition of Pervasive Services in the PalCom Architecture*". Proceedings of ICPS 2009, International Conference on Pervasive Services, pages 83–92, London, United Kingdom. https://doi.org/10.1145/1568199.1568213.

[86] P. Fremantle, 2015. "*A Reference Architecture for the Internet of Things*". WSO2 White paper, 2015.

[87] L. Frosini and F. Paternò, June 2014. "*User Interface Distribution in Multi-Device and Multi-User Environments with Dynamically Migrating Engines*". Proceedings of EICS 2014, Symposium on Engineering Interactive Computing Systems, pages 55–64, Rome, Italy. https://doi.org/10.1145/2607023.2607032.

[88] J. G. Garcia, J. Vanderdonckt, and J. M. G. Calleros, May 2008. "*FlowiXML: a Step Towards Designing Workflow Management Systems*". International Journal of Web Engineering and Technology, 4(2):163–182, May 2008. https://doi.org/10.1504/IJWET.2008.018096.

[89] A. García Frey, E. Céret, S. Dupuy-Chessa, G. Calvary, and Y. Gabillon, June 2012. "*UsiComp: an Extensible Model-Driven Composer*". Proceedings of EICS 2012, Symposium on Engineering Interactive Computing Systems, pages 263–268, Copenhagen, Denmark. https://doi.org/10.1145/2305484.2305528.

[90] J.-L. Gassée, 2014. "*Internet of Things: The "Basket of Remotes" Problem*". https://mondaynote.com/internet-of-things-the-basket-of-remotes-problem-f80922a91a0f. Accessed: 2020-02-26.

[91] L. D. Geronimo, M. Husmann, A. Patel, C. Tuerk, and M. C. Norrie, June 2016. "*CTAT: Tilt-and-Tap Across Devices*". Proceedings of ICWE 2016, 16th International Conference on Web Engineering, pages 96–113, Lugano, Switzerland. https://doi.org/10.1007/978-3-319-38791-8_6.

[92] G. Ghiani, M. Manca, and F. Paternò, November - December 2015. "*Authoring Context-dependent Cross-device User Interfaces based on Trigger/Action Rules*". Proceedings of MUM 2015, 14th International Conference on Mobile and Ubiquitous Multimedia, pages 313–322, Linz, Austria. https://doi.org/10.1145/2836041.2836073.

[93] G. Ghiani, M. Manca, F. Paternò, and C. Santoro, 2017. "*Personalization of Context-Dependent Applications Through Trigger-Action Rules*". ACM Transactions on Computer-Human Interaction, 24(2):14:1–14:33, 2017. https://doi.org/10.1145/3057861.

[94] G. Ghiani, F. Paternò, and C. Santoro, May 2012. "*Push and Pull of Web User Interfaces in Multi-Device Environments*". Proceedings of AVI 2012, International Working Conference on Advanced Visual Interfaces, pages 10–17, Capri Island, Naples, Italy. https://doi.org/10.1145/2254556.2254563.

[95] G. Ghiani, F. Paternò, L. D. Spano, and G. Pintori, 2016. "*An Environment for End-User Development of Web Mashups*". International Journal of Human-Computer Studies, 87:38–64, 2016. https://doi.org/10.1016/j.ijhcs.2015.10.008.

[96] A. Giri, S. Dutta, S. Neogy, K. P. Dahal, and Z. Pervez, October 2017. "*Internet of Things (IoT): a Survey on Architecture, Enabling Technologies, Applications and Challenges*". Proceedings of IML 2017, 1st International Conference on Internet of Things and Machine Learning, pages 7:1–7:12, Liverpool, United Kingdom. https://doi.org/10.1145/3109761.3109768.

[97] T. Griffiths, P. J. Barclay, N. W. Paton, J. McKirdy, J. B. Kennedy, P. D. Gray, R. Cooper, C. A. Goble, and P. P. da Silva, 2001. "*Teallach: a Model-Based User Interface Development Environment for Object Databases*". Interacting with Computers, 14(1):31–68, 2001. https://doi.org/10.1016/S0953-5438(01)00042-X.

[98] J. Grudin, April 1990. "*The Computer Reaches Out: The Historical Continuity of Interface Design*". Proceedings of CHI 1990, Conference on Human Factors in Computing Systems, pages 261–268, Seattle, Washington, USA. https://doi.org/10.1145/97243.97284.

[99] D. Guinard and V. Trifa, April 2009. "*Towards the Web of Things: Web Mashups for Embedded Devices*". Proceedings of MEM 2009, Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (International World Wide Web Conferences), Madrid, Spain.

[100] D. Guinard and V. Trifa, 2016. Building the Web of Things: With Examples in Node.Js and Raspberry Pi. USA. Manning Publications Co., 1st Edition, 2016. ISBN 1617292680.

[101] T. Halpin and T. Morgan, 2010. Information Modeling and Relational Databases. Morgan Kaufmann, 2010. https://doi.org/10.1016/B978-0-12-373568-3.X5001-2.

[102] R. Han, V. Perret, and M. Naghshineh, December 2000. "*WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing*". Proceedings of CSCW 2000, Conference on Computer Supported Cooperative Work, pages 221–230, Philadelphia, Pennsylvania, USA. https://doi.org/10.1145/358916.358993.

[103] S. Heo, S. Woo, J. Im, and D. Kim, October 2015. "*IoT-MAP: IoT Mashup Application Platform for the Flexible IoT Ecosystem*". Proceedings on IOT 2015, 5th International Conference on the Internet of Things, pages 163–170, Seoul, South Korea. https://doi.org/10.1109/IOT.2015.7356561.

[104] A. R. Hevner, S. T. March, J. Park, and S. Ram, 2004. "*Design Science in Information Systems Research*". Management Information Systems Quarterly, 28(1):75–105, 2004.

[105] T. T. Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong, and W. Verplank, 1992. ACM SIGCHI Curricula for Human-Computer Interaction. ACM, 1992.

[106] J. Humble, A. Crabtree, T. Hemmings, K. Åkesson, B. Koleva, T. Rodden, and P. Hansson, October 2003. "*"Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments*". Proceedings of UbiComp 2003, 5th International Conference on Ubiquitous Computing, pages 256–263, Seattle, Washington, USA. https://doi.org/10.1007/978-3-540-39653-6_20.

[107] M. Husmann, M. Nebeling, S. Pongelli, and M. C. Norrie, oct 2014. "*MultiMasher: Providing Architectural Support and Visual Tools for Multi-device Mashups*". Proceedings of WISE 2014, 15th International Conference on Web Information Systems Engineering, pages 199–214, Thessaloniki, Greece. https://doi.org/10.1007/978-3-319-11746-1_15.

[108] M. Hussein, S. Li, and A. Radermacher, September 2017. "*Model-Driven Development of Adaptive IoT Systems*". Proceedings of MODELS 2017, 20th International Conference on Model Driven Engineering Languages and Systems, pages 17–23, Austin, Texas, USA.

[109] M. Hussein, R. Nouacer, and A. Radermacher, August - September 2016. "*A Model-Driven Approach for Validating Safe Adaptive Behaviors*". Proceedings of DSD 2016, Euromicro Conference on Digital System Design, pages 75–81, Limassol, Cyprus. https://doi.org/10.1109/DSD.2016.21.

[110] J. E. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, May 2011. "*Empirical Assessment of MDE in Industry*". Proceedings of ICSE 2011, 33rd International Conference on Software Engineering, pages 471–480, Waikiki, Honolulu , Hawaii, USA. https://doi.org/10.1145/1985793.1985858.

[111] G. Inc., August 2012. "*The New Multi-screen World: Understanding Cross-platform Consumer Behavior*". http://services.google.com/fh/files/misc/multiscreenworld_final.pdf. Accessed: 2019-03-13.

[112] G. Inc., September 2016. "*How People Use Their Devices - What Marketers Need to Know*". https://www.thinkwithgoogle.com/_qs/documents/276/twg-how-people-use-their-devices-2016.pdf. Accessed: 2019-03-13.

[113] G. Inc., 2017. "*The Connected Consumer Survey 2017*". https://www.consumerbarometer.com/en/graph-builder/?question=M3. Accessed: 2019-03-13.

[114] M. M. Jensen, R. Rädle, C. N. Klokmose, and S. Bødker, April 2018. "*Remediating a Design Tool: Implications of Digitizing Sticky Notes*". Proceedings of CHI 2018, Conference on Human Factors in Computing Systems, page 224, Montreal, Quebec, Canada. https://doi.org/10.1145/3173574.3173798.

[115] J. Johnson, 2010. Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules. San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 2010. ISBN 978-0-12-375030-3.

[116] B. Johnsson and B. Magnusson, 10 2017. "*Towards End-User Development of Graphical User Interfaces for Internet of Things*". Future Generation Computer Systems, 10 2017. https://doi.org/10.1016/j.future.2017.09.068.

[117] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, October 2006. "*ATL: A QVT-like Transformation Language*". Proceedings of OOPSLA 2006, Companion to the 21th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications, pages 719–720, Portland, Oregon, USA. https://doi.org/10.1145/1176617.1176691.

[118] W. Kang, K. Kapitanova, and S. H. Son, 2012. "*RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems*". IEEE Transactions on Industrial Informatics, 8(2):393–405, 2012. https://doi.org/10.1109/TII.2012.2183878.

[119] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran, January 2004. "*W3C Recommendation: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*". https://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/. Accessed: 2018-02-14.

[120] I. Koren and R. Klamma, June 2016. "*The Direwolf Inside You: End User Development for Heterogeneous Web of Things Appliances*". Proceedings of ICWE 2016, 16th International Conference on Web Engineering, pages 484–491, Lugano, Switzerland. https://doi.org/10.1007/978-3-319-38791-8_35.

[121] D. Kovachev, D. Renzel, P. Nicolaescu, and R. Klamma, July 2013. "*DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications*". Proceedings of ICWE 2013, 13th International Conference on Web Engineering, pages 99–113, Aalborg, Denmark. https://doi.org/10.1007/978-3-642-39200-9_10.

[122] D. Kovachev, D. Renzel, P. Nicolaescu, I. Koren, and R. Klamma, 2014. "*DireWolf Framework for Widget-based Distributed User Interfaces*". Journal of Web Engineering, 13(3&4):203–222, 2014.

[123] M. Krug, F. Wiedemann, and M. Gaedke, July 2014. "*SmartComposition: A Component-Based Approach for Creating Multi-screen Mashups*". Proceedings of ICWE 2014, 14th International Conference on Web Engineering, pages 236–253, Toulouse, France. https://doi.org/10.1007/978-3-319-08245-5_14.

[124] T. Kubitza and A. Schmidt, 2017. "*meSchup: A Platform for Programming Interconnected Smart Things*". IEEE Computer, 50(11):38–49, 2017. https://doi.org/10.1109/MC.2017.4041350.

[125] T. J. Li, Y. Li, F. Chen, and B. A. Myers, June 2017. "*Programming IoT Devices by Demonstration Using Mobile Apps*". Proceedings of IS-EUD, 6th International Symposium on End-User Development, pages 3–17, Eindhoven, The Netherlands. https://doi.org/10.1007/978-3-319-58735-6_1.

[126] H. Lieberman, F. Paternò, and V. Wulf, editors, 2006. End User Development: An Emerging Paradigm. Human-Computer Interaction Series. Springer, 2006. ISBN 978-1-4020-4220-1. https://doi.org/10.1007/1-4020-5386-X.

[127] Q. Limbourg and J. Vanderdonckt, July 2004. "*USIXML: A User Interface Description Language Supporting Multiple Levels of Independence*". Proceedings of ICWE 2004, 4th International Conference on Web Engineering, pages 325–338, Munich, Germany.

[128] J. Lin and J. A. Landay, April 2008. "*Employing Patterns and Layers for Early-stage Design and Prototyping of Cross-device User Interfaces*". Proceedings of CHI 2008, Conference on Human Factors in Computing Systems, pages 1313–1322, Florence, Italy. ISBN 978-1-60558-011-1. https://doi.org/10.1145/1357054.1357260.

[129] D. Locke. "*MQ Telemetry Transport (MQTT) V3.1 Protocol Specification*". https://www.ibm.com/developerworks/library/ws-mqtt/. Accessed: 2018-03-13.

[130] F. Lonczewski and S. Schreiber, June 1996. "*The FUSE-System: an Integrated User Interface Design Environment*". Proceedings of CADUI 1996, 2nd International Workshop on Computer-Aided Design of User Interfaces, pages 37–56, Namur, Belgium.

[131] G. Lucci and F. Paternò, September 2014. "*Understanding End-User Development of Context-Dependent Applications in Smartphones*". Proceedings of HCSE 2014, 5th International Conference on Human-Centered Software Engineering, pages 182–198, Paderborn, Germany. https://doi.org/10.1007/978-3-662-44811-3_11.

[132] G. Lucci and F. Paternò, May 2015. "*Analysing How Users Prefer to Model Contextual Event-Action Behaviours in Their Smartphones*". Proceedings of IS-EUD 2015, 5th International Symposium on End-User Development, pages 186–191, Madrid, Spain. https://doi.org/10.1007/978-3-319-18425-8_14.

[133] K. Luyten and K. Coninx, December 2005. "*Distributed User Interface Elements to Support Smart Interaction Spaces*". Proceedings of ISM 2005, 7th International Symposium on Multimedia, pages 277–286, Irvine, California, USA. https://doi.org/10.1109/ISM.2005.52.

[134] K. Luyten, J. V. den Bergh, C. Vandervelpen, and K. Coninx, 2006. "*Designing Distributed User Interfaces for Ambient Intelligent Environments using Models and Simulations*". Computers & Graphics, 30(5):702–713, 2006. https://doi.org/10.1016/j.cag.2006.07.004.

[135] L. Mainetti, L. Manco, L. Patrono, I. Sergi, and R. Vergallo, December 2015. "*Web of Topics: An IoT-Aware Model-Driven Designing Approach*". Proceedings of WF-IoT 2015, 2nd World Forum on Internet of Things, pages 46–51, Milan, Italy. https://doi.org/10.1109/WF-IoT.2015.7389025.

[136] M. Manca and F. Paternò, 2011. "*Extending MARIA to Support Distributed User Interfaces*". Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, pages 33–40. https://doi.org/10.1007/978-1-4471-2271-5_4.

[137] M. Manca and F. Paternò, June 2016. "*Customizable Dynamic User Interface Distribution*". Proceedings of EICS 2016, 8th Symposium on Engineering Interactive Computing Systems, pages 27–37, Brussels, Belgium. https://doi.org/10.1145/2933242.2933259.

[138] N. Marquardt, K. Hinckley, and S. Greenberg, October 2012. "*Cross-Device Interaction via Micromobility and F-formations*". Proceedings of UIST 2012, 25th Annual Symposium on User Interface Software and Technology, pages 13–22, Cambridge, Massachusetts, USA. https://doi.org/10.1145/2380116.2380121.

[139] G. Marvin, March 2013. "*Microsoft Study: Multi-Screen Behavior And What It Means For Marketers*". http://marketingland.com/microsoft-study-multi-screen-behavior-and-what-it-means-for-marketer-36456. Accessed: 2019-03-13.

[140] E. McAweeney, H. Zhang, and M. Nebeling, April 2018. "*User-Driven Design Principles for Gesture Representations*". Proceedings of CHI 2018, Conference on Human Factors in Computing Systems, page 547, Montreal, Quebec, Canada. https://doi.org/10.1145/3173574.3174121.

[141] D. D. McCracken, J. M. Spool, and R. J. Wolfe, 2003. User-Centered Web Site Development: A Human-Computer Interaction Approach. Pearson Education, 2003.

[142] G. Meixner, G. Calvary, and J. Coutaz, December 2013. "*Introduction to Model-based User Interfaces*". https://www.w3.org/2011/mbui/drafts/mbui-intro/. Accessed: 2018-03-8.

[143] G. Meixner, F. Paternò, and J. Vanderdonckt, 2011. "*Past, Present, and Future of Model-based User Interface Development*". Journal of Interactive Media, 10(3):2–11, 2011. https://doi.org/10.1524/icom.2011.0026.

[144] B. C. Mejias Candia, 2010. Beernet: A Relaxed Approach to the Design of Scalable Systems with Self-Managing Behaviour and Transactional Robust Storage. Ph.D. thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium.

[145] J. Melchior, 2016. A Model-based Approach for Dynamically Distributing Graphical User Interfaces Based on their Properties, Graphs, and Scenarios. Ph.D. thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium.

[146] J. Melchior, J. Vanderdonckt, and P. V. Roy, June 2011. "*A Model-based Approach for Distributed User Interfaces*". Proceedings of EICS 2011, 3rd Symposium on Engineering Interactive Computing System, pages 11–20, Pisa, Italy. https://doi.org/10.1145/1996461.1996488.

[147] J. Meskens, K. Luyten, and K. Coninx, May 2010. "*Jelly: a Multi-Device Design Environment for Managing Consistency Across Devices*". Proceedings of AVI 2010, International Conference on Advanced Visual Interfaces, pages 289–296, Roma, Italy. https://doi.org/10.1145/1842993.1843044.

[148] J. Meskens, J. Vermeulen, K. Luyten, and K. Coninx, May 2008. "*Gummy for Multi-Platform User Interface Designs: Shape Me, Multiply Me, Fix Me, Use Me*". Proceedings of AVI 2008, Working Conference on Advanced Visual Interfaces, pages 233–240, Napoli, Italy. https://doi.org/10.1145/1385569.1385607.

[149] A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K. H. Yi, March 2006. "*InterPlay: A Middleware for Seamless Device Integration and Task Orchestration in a Networked Home*". Proceedings of PerCom 2006, 4th International Conference on Pervasive Computing and Communications, pages 296–307, Pisa, Italy. https://doi.org/10.1109/PERCOM.2006.30.

[150] B. Michotte and J. Vanderdonckt, March 2008. "*GrafiXML, a Multi-target User Interface Builder Based on UsiXML*". Proceedings of ICAS, 4th International Conference on Autonomic and Autonomous Systems, pages 15–22, Gosier, Guadeloupe. https://doi.org/10.1109/ICAS.2008.29.

[151] G. Mori, F. Paternò, and C. Santoro, 2002. "*CTTE: Support for Developing and Analyzing Task Models for Interactive System Design*". IEEE Transactions on Software Engineering, 28(8), 2002. https://doi.org/10.1109/TSE.2002.1027801.

[152] K. Naito, 2017. "*A Survey on the Internet-of-Things: Standards, Challenges and Future Prospects*". Journal of Information Processing, 25:23–31, 2017. https://doi.org/10.2197/ipsjjip.25.23.

[153] K. Nakagaki, L. Vink, J. Counts, D. Windham, D. Leithinger, S. Follmer, and H. Ishii, May 2016. "*Materiable: Rendering Dynamic Material Properties in Response to Direct Physical Touch with Shape Changing Interfaces*". Proceedings of CHI 2016, Conference on Human Factors in Computing Systems, pages 2764–2772, San Jose, California, USA. https://doi.org/10.1145/2858036.2858104.

[154] M. Nebeling, May 2017. "*XDBrowser 2.0: Semi-Automatic Generation of Cross-Device Interfaces*". Proceedings of CHI 2017, Conference on Human Factors in Computing Systems, pages 4574–4584, Denver, Colorado, USA. https://doi.org/10.1145/3025453.3025547.

[155] M. Nebeling and A. K. Dey, May 2016. "*XDBrowser: User-Defined Cross-Device Web Page Designs*". Proceedings of CHI 2016, Conference on Human Factors in Computing Systems, pages 5494–5505, San Jose, California, USA. https://doi.org/10.1145/2858036.2858048.

[156] M. Nebeling, T. Mintsi, M. Husmann, and M. C. Norrie, April 2014. "*Interactive Development of Cross-Device User Interfaces*". Proceedings of CHI 2014, Conference on Human Factors in Computing Systems, pages 2793–2802, Toronto, Ontario, Canada. https://doi.org/10.1145/2556288.2556980.

[157] M. Nebeling, E. Teunissen, M. Husmann, and M. C. Norrie, June 2014. "*Michael Nebeling and Elena Teunissen and Maria Husmann and Moira C. Norrie*". Proceedings of EICS 2014, Symposium on Engineering Interactive Computing Systems, pages 65–74, Rome, Italy. https://doi.org/10.1145/2607023.2607024.

[158] T. H. Nelson, 1995. "*The Heart of Connection: Hypermedia Unified by Transclusion*". Communications of the ACM, 38(8):31–33, 1995. https://doi.org/10.1145/208344.208353.

[159] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, December 2015. "*FRASAD: A Framework for Model-Driven IoT Application Development*". Proceedings of WF-IoT 2015, 2nd World Forum on Internet of Things, pages 387–392, Milan, Italy. https://doi.org/10.1109/WF-IoT.2015.7389085.

[160] X. T. Nguyen, M. Zapf, and K. Geihs, December 2011. "*Model Driven Development for Data-Centric Sensor Network Applications*". Proceedings of MoMM 2011, 9th International Conference on Advances in Mobile Computing and Multimedia, pages 194–197, Ho Chi Minh City, Vietnam. https://doi.org/10.1145/2095697.2095733.

[161] R. Oppermann, 2017. Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software. Routledge, 2017. ISBN 9780805816556.

[162] P. Patel and D. Cassou, 2015. "*Enabling High-Level Application Development for the Internet of Things*". Journal of Systems and Software, 103:62–84, 2015. https://doi.org/10.1016/j.jss.2015.01.027.

[163] F. Paternò, 2013. "*End User Development: Survey of an Emerging Field for Empowering People*". ISRN Software Engineering, 2013, 2013. https://doi.org/10.1155/2013/532659.

[164] F. Paternò, C. Mancini, and S. Meniconi, July 1997. "*ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models*". Proceedings of INTERACT 1997, Interantional Conference on Human-Computer Interaction, pages 362–369, Sydney, Australia. https://doi.org/10.1007/978-0-387-35175-9_58.

[165] F. Paternò and C. Santoro, 2017. "*A Design Space for End User Development in the Time of the Internet of Things*". New Perspectives in End-User Development, pages 43–59. https://doi.org/10.1007/978-3-319-60291-2_3.

[166] F. Paternò and C. Santoro, 2019. "*End-User Development for Personalizing Applications, Things, and Robots*". International Journal of Human-Computer Studies, 131:120–130, 2019. https://doi.org/10.1016/j.ijhcs.2019.06.002.

[167] F. Paternò, C. Santoro, and L. D. Spano, 2009. "*MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments*". ACM Transactions on Computer-Human Interaction, 16(4), 2009. https://doi.org/10.1145/1614390.1614394.

[168] F. Paternò and G. Zichittella, October 2010. "*Desktop-to-Mobile Web Adaptation through Customizable Two-Dimensional Semantic Redesign*". Proceedings of HCSE 2010, 3rd International Conference on Human-Centred Software Engineering, pages 79–94, Reykjavik, Iceland. https://doi.org/10.1007/978-3-642-16488-0_7.

[169] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, 2007. "*A Design Science Research Methodology for Information Systems Research*". Journal of Management Information Systems, 24(3):45–77, 2007. https://doi.org/10.2753/MIS0742-1222240302.

[170] J. L. Pérez-Medina, S. Dupuy-Chessa, and A. Front, November 2007. "*A Survey of Model Driven Engineering Tools for User Interface Design*". Proceedings of TAMODIA 2007, 6th International Workshop on Task Models and Diagrams for User Interface Design, pages 84–97, Toulouse, France. https://doi.org/10.1007/978-3-540-77222-4_8.

[171] T. Pering, K. Lyons, R. Want, M. Murphy-Hoye, M. Baloga, P. Noll, J. Branc, and N. D. Benoist, September 2010. "*What Do You Bring To the Table?: Investigations of a Collaborative Workspace*". Proceedings of UbiComp 2010, 12th International Conference on Ubiquitous Computing, pages 183–192, Copenhagen, Denmark. https://doi.org/10.1145/1864349.1864389.

[172] T. Pering, R. Want, B. Rosario, S. Sud, and K. Lyons, May 2009. "*Enabling Pervasive Collaboration with Platform Composition*". Proceedings of PerCom 2009, 7th International Conference on Pervasive Computing, pages 184–201, Nara, Japan. https://doi.org/10.1007/978-3-642-01516-8_14.

[173] S. L. Peyton Jones, A. F. Blackwell, and M. M. Burnett, August 2003. "*A User-Centred Approach to Functions in Excel*". Proceedings of ICFP 2003, 8th International International Conference on Functional Programming, pages 165–176, Uppsala, Sweden. https://doi.org/10.1145/944705.944721.

[174] T. Plank, H. Jetter, R. Rädle, C. N. Klokmose, T. Luger, and H. Reiterer, May 2017. "*Is Two Enough?! Studying Benefits, Barriers, and Biases of Multi-Tablet Use for Collaborative Visualization*". Proceedings of CHI 2017, Conference on Human Factors in Computing Systems, pages 4548–4560, Denver, Colorado, USA. https://doi.org/10.1145/3025453.3025537.

[175] F. Pramudianto, I. R. Indra, and M. Jarke, June 2013. "*Model Driven Development for Internet of Things Application Prototyping*". Proceedings of SEKE 2013, 25th International Conference on Software Engineering and Knowledge Engineering, pages 703–708, Boston, Massachusetts, USA.

[176] A. R. Puerta and P. Szkeley, April 1994. "*Model-based Interface Development*". Proceedings of CHI 1994, Conference on Human Factors in Computing Systems, pages 389–390, Boston, Massachusetts, USA. ISBN 0-89791-651-4. https://doi.org/10.1145/259963.260519.

[177] W. D. Ra, 2011. "*Brave NUI World: Designing Natural User Interfaces for Touch and Gesture by Daniel Wigdor and Dennis Wixon*". ACM SIGSOFT Software Engineering Notes, 36(6):29–30, 2011. https://doi.org/10.1145/2047414.2047439.

[178] R. Rädle, H. Jetter, N. Marquardt, H. Reiterer, and Y. Rogers, November 2014. "*HuddleLamp: Spatially-Aware Mobile Displays for Ad-hoc Around-the-Table Collaboration*". Proceedings of ITS 2014, 9th International Conference on Interactive Tabletops and Surfaces, pages 45–54, Dresden, Germany. https://doi.org/10.1145/2669485.2669500.

[179] P. P. Ray, 2018. "*A Survey on Internet of Things Architectures*". Journal of King Saud University-Computer and Information Sciences, 30(3):291–319, 2018. https://doi.org/10.1016/j.jksuci.2016.10.003.

[180] E. S. Raymond, 2004. "*The art of unix usability*". Online: http://www.catb.org/~esr/writings/taouu/html/. Accessed on Januari 2020.

[181] A. Rieger, R. Cissée, S. Feuerstack, J. Wohltorf, and S. Albayrak, May 2005. "*An Agent-Based Architecture for Ubiquitous Multimodal User Interfaces*". Proceedings of AMT 2005, 3rd International Conference on Active Media Technology, pages 119–124, Kagawa, Japan. IEEE. https://doi.org/10.1109/AMT.2005.1505284.

[182] A. K. Rithu Thomas, Preetha Devan, 2018. "*The Internet of Things: A Technical Primer*". Online.

[183] T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Humble, K. Åkesson, and P. Hansson, May 2004. "*Configuring the Ubiquitous Home*". Proceedings of COOP 2004, 6th International Conference on the Design of Cooperative Systems, pages 227–242, Hyères Les Palmiers, France. ISBN 1-58603-422-7.

[184] R. Roels, 2019. MindXpres: Conceptual and Technical Foundations for Next Generation Presentation Solutions. Ph.D. thesis, Vrije Universiteit Brussel.

[185] R. Roels and B. Signer, 2019. "*A Conceptual Framework and Content Model for Next Generation Presentation Solutions*". Proceedings of the ACM on Human-Computer Interaction, 3(EICS):7:1–7:22, 2019. https://doi.org/10.1145/3331149.

[186] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, 2002. "*Gaia: A Middleware Platform for Active Spaces*". Mobile Computing and Communications Review, 6(4):65–67, 2002. https://doi.org/10.1145/643550.643558.

[187] D. Roscher, M. Blumendorf, and S. Albayrak, February 2009. "*A Meta User Interface to Control Multimodal Interaction in Smart Environments*". Proceedings of IUI 2009, 14th International Conference on Intelligent User Interfaces, pages 481–482, Sanibel Island, Florida, USA. https://doi.org/10.1145/1502650.1502725.

[188] D. Roscher, G. Lehmann, V. Schwartze, M. Blumendorf, and S. Albayrak, 2011. "*Dynamic Distribution and Layouting of Model-based User Interfaces in Smart Environments*". Model-Driven Development of Advanced User Interfaces, pages 171–197. Springer. https://doi.org/10.1007/978-3-642-14562-9_9.

[189] M. B. Rosson and J. M. Carroll, 2003. "*The Human-Computer Interaction Handbook*". chapter Scenario-based Design, pages 1032–1050. Hillsdale, New Jersey, USA. Lawrence Erlbaum Associates Inc. ISBN 0-8058-3838-4.

[190] L. D. Russis and F. Corno, April 2015. "*HomeRules: A Tangible End-User Programming Interface for Smart Homes*". Proceedings of CHI 2015, 33rd Annual Conference Extended Abstracts on Human Factors in Computing Systems, pages 2109–2114, Seoul, Republic of Korea. https://doi.org/10.1145/2702613.2732795.

[191] M. Salehie and L. Tahvildari, 2009. "*Self-Adaptive Software: Landscape and Research Challenges*". Transactions on Autonomous and Adaptive Systems, 4(2):14:1–14:42, 2009. https://doi.org/10.1145/1516533.1516538.

[192] A. Sanctorum and B. Signer, May 2019. "*A Unifying Reference Framework and Model for Adaptive Distributed Hybrid User Interfaces*". Proceedings of RCIS 2019, International Conference on Research Challenges in Information Science, pages 1–6, Brussels, Belgium. https://doi.org/10.1109/RCIS.2019.8877048.

[193] A. Sanctorum and B. Signer, 2019. "*Towards End-User Development of Distributed User Interfaces*". Universal Access in the Information Society, 18(4):785–799, 2019. https://doi.org/10.1007/s10209-017-0601-5.

[194] S. Santosa and D. Wigdor, September 2013. "*A Field Study of Multi-device Workflows in Distributed Workspaces*". Proceedings of UbiComp 2013, International Joint Conference on Pervasive and Ubiquitous Computing, pages 63–72, Zurich, Switzerland. https://doi.org/10.1145/2493432.2493476.

[195] J. Sauro and J. R. Lewis, 2016. Quantifying the User Experience: Practical Statistics for User Research. Morgan Kaufmann, 2016. ISBN 978-0-12-802308-2.

[196] J. Schell, 2008. The Art of Game Design: A Book of Lenses. San Francisco, California, USA. Morgan Kaufmann Publishers Inc., 2008. ISBN 978-0-12-369496-6.

[197] V. Schwartze, S. Feuerstack, and S. Albayrak, July 2009. "*Behavior-Sensitive User Interfaces for Smart Environments*". Proceedings of ICDHM, Second International Conference on Digital Human Modeling, pages 305–314, San Diego, California, USA. https://doi.org/10.1007/978-3-642-02809-0_33.

[198] C. Shen, K. Everitt, and K. Ryall, October 2003. "*UbiTable: Impromptu Face-to-Face Collaboration on Horizontal Interactive Surfaces*". Proceedings of UbiComp 2003, 5th International Conference on Ubiquitous Computing, pages 281–288, Seattle, Washington, USA. https://doi.org/10.1007/978-3-540-39653-6_22.

[199] B. Shneiderman, September 1996. "*The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*". Proceedings of VL 1996, Symposium on Visual Languages, pages 336–343, Boulder, Colorado, USA. https://doi.org/10.1109/VL.1996.545307.

[200] N. C. Shu, 1989. "*Visual Programming: Perspectives and Approaches*". IBM Systems Journal, 28(4):525–547, 1989.

[201] B. Signer, 2006. Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces. Ph.D. thesis, ETH Zurich.

[202] B. Signer, 2017. Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces. Books on Demand, 2017. ISBN 978-3837027136.

[203] B. Signer and M. C. Norrie, November 2007. "*As We May Link: A General Metamodel for Hypermedia Systems*". Proceedings of ER 2007, 26th International Conference on Conceptual Modeling, pages 359–374, Auckland, New Zealand. https://doi.org/10.1007/978-3-540-75563-0_25.

[204] B. Signer and M. C. Norrie, October 2008. "*A Framework for Developing Pervasive Cross-Media Applications based on Physical Hypermedia and Active Components*". Proceedings of ICPCA 2008, 3rd International Conference on Pervasive Computing and Applications, pages 564–569, Alexandria, Egypt. https://doi.org/10.1109/ICPCA.2008.4783676.

[205] B. Signer and M. C. Norrie, July 2009. "*Active Components as a Method for Coupling Data and Services: A Database-driven Application Development Process*". Proceedings of ICOODB 2009, Second International Conference on Object Databases, pages 59–76, Zurich, Switzerland. https://doi.org/10.1007/978-3-642-14681-7_4.

[206] F. M. Simarro and V. López-Jaquero, June 2006. "*IdealXml: An Interaction Design Tool*". Proceedings of CADUI 2006, Conference on Computer-Aided Design Of User Interfaces, pages 245–252, Bucharest, Romania.

[207] L. D. Spano, F. Paternò, and G. Fenu, June 2014. "*A Gestural Concrete User Interface in MARIA*". Proceedings of EICS 2014, Symposium on Engineering Interactive Computing Systems, pages 179–184, Rome, Italy. https://doi.org/10.1145/2607023.2610282.

[208] P. Szekely, June 1996. "*Retrospective and Challenges for Model-based Interface Development*". Proc. of DSV-IS 1996.

[209] P. A. Szekely, P. Luo, and R. Neches, April 1993. "*Beyond Interface Builders: Model-based Interface Tools*". Proceedings of INTERCHI 1993, Conference on Human-Factors in Computing Systems, pages 383–390, Amsterdam, The Netherlands.

[210] P. A. Szekely, P. N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher, August 1995. "*Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach*". Proceedings of the EHCI 1995, Conference on Engineering for Human-Computer Interaction, pages 120–150, Yellowstone Park, USA.

[211] R. Tesoriero and A. H. Altalhi, 2019. "*Model-based Development of Distributable User Interfaces*". Universal Access in the Information Society, 18(4):719–746, 2019. https://doi.org/10.1007/s10209-017-0600-6.

[212] S. Trullemans, L. V. Holsbeeke, and B. Signer, 2017. "*The Context Modelling Toolkit: A Unified Multi-layered Context Modelling Approach*". Proceedings of the ACM on Human-Computer Interaction, 1(EICS):8:1–8:16, 2017. https://doi.org/10.1145/3095810.

[213] B. Ur, E. McManus, M. P. Y. Ho, and M. L. Littman, April - May 2014. "*Practical Trigger-Action Programming in the Smart Home*". Proceedings of CHI 2014, Conference on Human Factors in Computing Systems, pages 803–812, Toronto, Ontario, Canada. https://doi.org/10.1145/2556288.2557420.

[214] J. Vanderdonckt, June 2005. "*A MDA-Compliant Environment for Developing User Interfaces of Information Systems*". Proccedings of CAiSE 2005, Conference on Advanced Information Systems Engineering, pages 16–31, Porto, Portugal. https://doi.org/10.1007/11431855_2.

[215] J. Vanderdonckt, September 2008. "*Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges*". Proceedings of ROCHI 2008, 5th Romanian Conference on Human-Computer Interaction, Iasi, Romania.

[216] J. Vanderdonckt and A. Coyette, October 2006. "*Vers un Prototypage Des Interfaces Graphiques Incluant Vraiment l'Utilisateur Final*". Proceedings of ERGO-IA 2006, 10ième Colloque Int. sur l'Ergonomie et l'Informatique Avancée, Biarritz, France.

[217] C. Vandervelpen and K. Coninx, October 2004. "*Towards Model-based Design Support for Distributed User Interfaces*". Proceedings of NordiCHI 2004, Third Nordic Conference on Human-Computer Interaction, pages 61–70, Tampere, Finland. https://doi.org/10.1145/1028014.1028023.

[218] C. Vandervelpen, G. Vanderhulst, K. Luyten, and K. Coninx, July 2005. "*Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments*". Proceedings of ICWE 2005, 5th International Conference on Web Engineering, pages 197–202, Sydney, Australia. https://doi.org/10.1007/11531371_28.

[219] A. Vogelsang and B. Signer, December 2005. "*The Lost Cosmonaut: An Interactive Narrative Environment on Basis of Digitally Enhanced Paper*". Proceedings of VS 2005, 3rd International Conference on Virtual Storytelling, pages 270–279, Strasbourg, France. https://doi.org/10.1007/11590361_31.

[220] F. Weingarten, M. Blumendorf, and S. Albayrak, August 2010. "*Towards Multimodal Interaction in Smart Home Environments: The Home Operating System*". Proceedings of DIS 2010, 8th Conference on Designing Interactive Systems, pages 430–433, Aarhus, Denmark. https://doi.org/10.1145/1858171.1858255.

[221] C. Wiecha, W. Bennett, S. Boies, J. Gould, and S. Greene, July 1990. "*ITS: A Tool for Rapidly Developing Interactive Applications*". ACM Transactions on Information Systems, 8(3):204–236, July 1990. ISSN 1046-8188. https://doi.org/10.1145/98188.98194.

[222] P. Wisner and D. N. Kalofonos, January 2007. "*A Framework for End-User Programming of Smart Homes Using Mobile Devices*". Proceedings of CCNC 2007, 4th Conference on Consumer Communications and Networking, pages 716–721, Las Vegas, Nevada, USA. IEEE. https://doi.org/10.1109/CCNC.2007.146.

[223] E. Yigitbas, S. Sauer, and G. Engels, June 2017. "*Adapt-UI: An IDE Supporting Model-Driven Development of Self-Adaptive UIs*". Proceedings of EICS 2017, Symposium on Engineering Interactive Computing Systems, pages 99–104, Lisbon, Portugal. https://doi.org/10.1145/3102113.3102144.

[224] E. Yigitbas, H. Stahl, S. Sauer, and G. Engels, July 2017. "*Self-adaptive UIs: Integrated Model-Driven Development of UIs and Their Adaptations*". Proceedings of ECMFA 2017, 13th European Conference on Modelling Foundations and Applications, pages 126–141, Marburg, Germany. https://doi.org/10.1007/978-3-319-61482-3_8.

Index