UInify: A Designer Studio for creating UI Mashups for Ambient Intelligence Environments

Alexandra Barka

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Professor *Constantine Stephanidis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

UInify: A Designer Studio for creating UI Mashups for Ambient Intelligence Environments

Thesis submitted by
Alexandra Barka
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Alexandra Barka

Committee approvals: _____
Constantine Stephanidis
Professor, Thesis Supervisor

_____
Panagiotis Tsakalidis
Professor, Committee Member

_____
Margherita Antona
Principal Researcher, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, November 2018

UInify: A Designer Studio for creating UI Mashups for Ambient Intelligence Environments

Abstract

Over the past decade, there is an accelerating interest in Ambient Intelligent (AmI) environments, that has resulted in an abundance of "smart" products, applications and research prototypes, which aim to revolutionize daily human life, boost productivity and enhance well-being. Given the vast number of currently available devices and services, it is not possible for developers to build all-inclusive applications, which will enable users to systematically monitor and control the variety of physical devices and services that they own.

Instead, users need to either rely on diverse applications from different vendors to accomplish their goals, or use integration platforms sacrificing functionality and rich interaction in favor of interoperability. With the emergence of "Internet of Things", several mature technologies are available nowadays to develop interactive-rich user interfaces. Various technologies have been established, that allow the composition of user interfaces (UIs) by making a "mashup" application, which is a web page, or applications in general, that combine data and services from existing interfaces, resulting in the production of functional and rich UIs.

This thesis proposes the UInify design studio, which empowers designers to compose flexible web applications in real-time. This framework binds, under a common roof, all the individual user interfaces that control and/or monitor hardware and software components of a smart environment (e.g. a smart home) by providing an intelligent user interface [99] developed as a web application mashup[103, 95]. UInify offers a set of tools allowing designers to combine multiple individual UIs together and introduces new rich user interface compositions. The end-users are provided with a single UI through which they can control the intelligent facilities of their surroundings, as well as tailor their surroundings according to their needs.

After reviewing the relevant motivational state-of-the-art, the high-level architecture and the implementation details are elaborated. Several use cases of composite applications with the proposed system will be described, such as an application for the Home TV [32] and an Interactive Kitchen Countertop.

Keywords: Ambient Intelligence, Ubiquitous Computing, Human-computer Interaction, UI Mashups, Smart Environments

UInify: Σχεδιαστική πλατφορμα για την δημιουργία σύνθετων διεπαφών σε περιβάλλοντα διάχυτης νοημοσύνης

Περίληψη

Κατά την τελευταία δεκαετία υπάρχει διευρυνόμενο ενδιαφέρον για περιβάλλοντα Διάχυτης Νοημοσύνης (ΔΝ) που έχει οδηγήσει στην παραγωγή πληθώρας «έξυπνων» προϊόντων, εφαρμογών, και ερευνητικών πρωτοτύπων, τα οποία στοχεύουν στην βελτίωση της καθημερινότητας των ανθρώπων, στην αύξηση της παραγωγικότητας και στην αύξηση της ευημερίας και της ποιότητας της ζωής. Δεδομένου του μεγάλου αριθμού των συσκευών και υπηρεσιών που παρέχονται για αυτά τα περιβάλλοντα, είναι αδύνατο για τους προγραμματιστές να δημιουργήσουν ολοκληρωμένες εφαρμογές που επιτρέπουν στους χρήστες να παρακολουθούν και να ελέγχουν συστηματικά την ποικιλία των συσκευών και των τεχνητών υπηρεσιών που κατέχουν.

Σήμερα οι χρήστες, είτε χρησιμοποιούν διαφορετικές εφαρμογές από διαφορετικούς προμηθευτές (με διαφορετική λειτουργικότητα και εμπειρία χρήσης η κάθε εφαρμογή) για να επιτύχουν τους στόχους τους, είτε βασίζονται σε ενοποιημένες πλατφόρμες που θυσιάζουν την λειτουργικότητα και την πλούσια αλληλεπίδραση της εφαρμογής στο βωμό της διαλειτουργικότητας. Παρόλα αυτά, η ανάπτυξη του 'Διαδίκτυο των Πραγμάτων' περιλαμβάνει πλέον αρκετά ώριμες τεχνολογίες για την ανάπτυξη πλουσίων διαδραστικών διεπαφών. Διάφορες τεχνολογίες που έχουν καθιερωθεί επιτρέπουν τη σύνθεση διεπαφών δημιουργώντας μια ιστοσελίδα ανάμειξης (mashup [103, 95]), οι οποίες είναι ιστοσελίδες ή εφαρμογές οι οποίες συνδυάζουν δεδομένα και υπηρεσίες από ήδη υπάρχουσες διεπαφές με αποτέλεσμα την παραγωγή μιας λειτουργικά πλούσιας εφαρμογής για τον τελικό χρήστη.

Η παρούσα εργασία προτείνει τη σχεδιαστική εφαρμογή UInify, η οποία δίνει τη δυνατότητα στους σχεδιαστές να συνθέτουν, σε πραγματικό χρόνο, ευέλικτες εφαρμογές διαδικτύου. Αυτή η τεχνολογική πλατφόρμα ενοποιεί, κάτω από κοινή στέγη, τις μεμονωμένες διεπαφές που ελέγχουν ή/και παρακολουθούν το υλικό (Hardware) και λογισμικό (Software) ενός έξυπνου χώρου (π.χ. έξυπνο σπίτι) με την παροχή μιας έξυπνης διεπαφής [99], με τη μορφή ιστοσελίδας ανάμειξης [103, 95]. Το UInify είναι μια τεχνολογική πλατφόρμα, η οποία συνεισφέρει με ένα σύνολο εργαλείων τα οποία επιτρέπουν στους σχεδιαστές έξυπνων χώρων να εισάγουν νέες πλούσιες συνθέσεις διεπαφών. Στους τελικούς χρήστες παρέχεται μια ενιαία διεπαφή μέσω της οποίας μπορούν να παρακολουθήσουν και να ελέγξουν τις ευφυείς εγκαταστάσεις τους ανάλογα με τις προσωπικές τους προτιμήσεις.

Στο πλαίσιο της εργασίας αυτής γίνεται μια ανασκόπηση της σχετικής βιβλιογραφίας και στη συνέχεια παρουσιάζεται η αρχιτεκτονική, σε υψηλό επίπεδο, καθώς και οι λεπτομέρειες της υλοποίησης του συστήματος. Στις τελευταίες ενότητες της εργασίας, περιγράφονται διάφορες περιπτώσεις χρήσης του προτεινόμενου συστήματος για την δημιουργία σύνθετων διεπαφών για ένα έξυπνο σπίτι, όπως για παράδειγμα μια πολυμεσική εφαρμογή για την έξυπνη τηλεόραση του σαλονιού [32] και μια διαδραστική εφαρμογή για τον πάγκο της κουζίνας.

Λέξεις κλειδιά: Διάχυτη νοημοσύνη, Διάχυτος Υπολογισμός, Αλληλεπίδραση ανθρώπου-υπολογιστή, Αναμίξεις διεπαφών, Έξυπνα περιβάλλοντα

στην οικογένεια μου και στην Ειρήνη

Contents

List of Tables

List of Figures

Abbreviations

AmI Ambient Intelligence

GUI Graphical User Interface

HCI Human-Computer Interaction

HTTP Hypertext Transfer Protocol

IFTTT If this Then that

IoT Internet of Things

IT Information Technology

VUI Voice User Interface

WoT Web Of Things

WWW World Wide Web

Chapter 1

Introduction

The advancements in Information Technology (IT) over the past three decades, have completely transformed interaction and have steadily increased the users expectations from technology due to their familiarity with computing devices. With the emergence of the Internet of Things (IoT) technologies (e.g. sensor networks, RFID, wireless communications) [61], the World Wide Web (WWW) has become mature enough to serve as a platform for creating rich interactive applications. On top of continuous technological advancements, the increasing interest in Ambient Intelligent (AmI) environments, highlighted the need for appropriate tools to encapsulate and control a wide range of connected devices and artificial services, offering smart behaviour by responding to events in such environments (e.g. turn on the lights when the user enters a room in an intelligent home) under a common application.

The success of Web 2.0 [80] and Web Of Things (WoT) [66, 65] allowed the seamless interconnection of real-world objects enhanced with computer technology to the existing web. Such advancements offer new directions for the development of intelligent environments in terms of web-based platforms (e.g. HomeWeb: An application framework for Web-based smart homes [72], meSchup: A Platform for Programming Interconnected Smart Things [75], IoT-Based Portable Smart Meeting Space [92] and more). These technological accomplishments (e.g. the use of client-side scripting languages, the advance of Web Services, the public APIs) shaped up a new concept, the *mashups* [103, 95]. Mashup web applications -and more specifically in the context of this work the **UI mashups** - are web applications that integrate interfaces at different levels of the application stack from different resources on the web.

To that end, this thesis proposes the **UInify design studio**, which empowers designers to *compose flexible rich user interfaces at real-time*. This framework binds under a common roof all the individual user interfaces that control and/or monitor the hardware and software components of an intelligent space by providing an intelligent user interface [99] as a web application mashup [103, 95].

Although this master mainly focuses on home automation paradigms, possible

applications in other intelligent environments (e.g smart classroom), with ultimate purpose to be useful in future smart cities, are also explored. Smart cities are a trending research topic, with multiple projects all over the world, which aim in three sections: (i) advanced communication technologies to support services for the administration of the city [111] (ii) Smart and Green Transportation [54] and (iii) Health-care [28] and Eco-living [110].

1.1 Overview - Setting the problem

An ambient intelligent environment is expected to have multiple display devices and sensors and to unobtrusively pay attention to users' in order to understand users' intentions and decide system's actions. A system with this kind of complexity needs a *controller* or an *assistant*, to aid the end user with daily tasks. Many major market players like Amazon and Google have released Voice User Interface (VUI) controllers. It is common these days that devices like Amazon Alexa [1], Google home [10] and Amazon Echo show [1] are installed in houses and offer voice UIs to assist their users (e.g. offer multimedia options). On the other hand, Apple has released the Home app [3] - a cross-platform iOS application- which controls the home accessories in a house. Despite the fact that these VUI controllers work out pretty well, user interfaces that convey visual information are also important. Additionally, the delivered application should not be restricted by vendors agreements (e.g. Apple is cooperating with 50 brands [3]).

Today, it is common that a user has at least one mobile device. Recently an explosive growth of mobile devices has been observed and expected to grow from an installed base of 15.4 billion devices in 2015 to 30.7 billion devices in 2020 and 75.4 billion in 2025 [81]. Furthermore, all these devices have different characteristics, that designers have to consider in the development of home automation applications. To that end, it is impossible for designers and developers to build all-inclusive applications, which will enable users to fully monitor and control the wide variety of physical devices and artificial services that they own.

The main challenges for home automation controllers will arise from Human-Computer Interaction (HCI) domain and refer to the application's usability. A usable [59, 77] interface has three characteristics, (i) it should be easy-to-use for the user, making her familiar with the UI, (ii) the users should achieve their objective easily by using the interface and (iii) it should be easy to remember in the subsequent visits by the users. It is important to create interactive applications through the iterative design process that HCI advises. Furthermore, the application should offer a complete, unified experience to the end-user. To that end, it is essential for designers to utilize User-Centered Design (UCD) approaches to define their functional requirements and to create successful user experiences [100].

The composite UI to be delivered, will be an intelligent User Interface (IUI) [99], which is a user interface that involves some aspect of computational intelligence. The created IUI will be the intersection of user preferences, sensor data, and

context from the intelligent space. IUIs arise two main challenges: (i) the *vast amount of information* produced, which are a combination of sensor data and context information and (ii) the *reasoning process*, where the system can make deductions about the user.

The core objective of this study is to develop a framework that aids the designers of intelligent spaces to deliver unified experiences to the end users. The contribution of this framework is a collection of tools, which enable designers to combine several individual UIs, and introduce new rich User Interface compositions. The end-users are provided with a user interface through which they can monitor and manage their intelligent facilities. This IUI [99] can evolve over time, by using mechanisms and computational techniques like *artificial intelligence*, *reasoning* and *machine learning*. These composite user interfaces (i) will create innovative and unique interactive experiences for end users and (ii) empower them to tailor their space according to their requirements.

1.2 Thesis structure

The rest of this master thesis is organized in eight (8) main sections as indicated in the table of contents:

- The second section will introduce the current state of the art in relevant topics that this thesis is based upon. It will first review the literature regarding ambient intelligence and its applications. Moreover, it will explore how these concepts apply in the context of home automation, including a brief historic reference of this concept. Then some technology advancements will be presented that lead to the concept of web application mashup. Finally, this thesis briefly discusses the system from Human-Computer Interaction (HCI) point of view.

- The third section will present a high-level architecture for UInify. It will report the functional requirements that were gathered for designing the system. Also, it will present the terminology (the vocabulary used in UInify), and finally an overall high level description of the functionality of the system.

- The fourth chapter will dive into the technical aspects of the platform. It will illustrate the conceptual model and the representation of the data that was adopted. Moreover, it will present the details of the implementation of the full stack application.

- The fifth chapter will describe of the interaction with the system, with a brief discussion of the components used and the design decisions made for the platform.

- The sixth section will present several use cases in which UInify can be utilised to create composite applications in an intelligent space. The first case will

present how UInify can be utilized to create an entertainment hub and the second a mobile phone launcher. Then, we briefly illustrate how the system can be used for two more complex scenarios, first is for a kitchen countertop and the second for an intelligent classroom.

- The seventh section will report the evaluation process, the scenarios utilised, as well as the findings that were obtained.
- Finally, the last chapter will summarize this work and discuss its future directions.

Chapter 2

Literature Review

Science fiction explored the idea of intelligent spaces decades ago when authors envisioned interactive homes and artificial intelligent agents capable of operating the household. Many films, books, and cartoon series have portrayed the home automation vision, with AI agents, robots, and fully automated houses. Futuristic domiciles examples have been explored by media in the cartoon series Futurama, where the writers envisioned the life in the 31st century, the Disney channel movie Smart house (1999) where an evil artificial agent control over the household, the movie The Hitchhiker's Guide to the Galaxy (2005), where the doors have emotions and express them when the people use them, the Iron Man trilogy movies with the artificial agent Jarvis and plenty other. This section, reviews topics of theoretical background research relevant to the present work, for the better understanding of the latest technologies and the deficiencies that this work tries to overcome. Next, it will explore the concept of web applications mashups, which is a major component of the implemented system. Finally, some secondary topics such as the Internet and the Web of things will be also briefly discussed to support the theoretical background.

The topics that will be investigated are:

- Ambient Intelligence, an overview of the concept, some definitions and the general features that encompass this topic
- Ambient Intelligence application in the context of home automation
- Home automation applications used to monitor and/or control the intelligent facilities overview
- Web application Mashups, a concept that allows to create new rich web application by utilising existing ones
- Internet and Web of things concepts that led to the introduction of mashup technology
- Human Computer interaction & User centered design

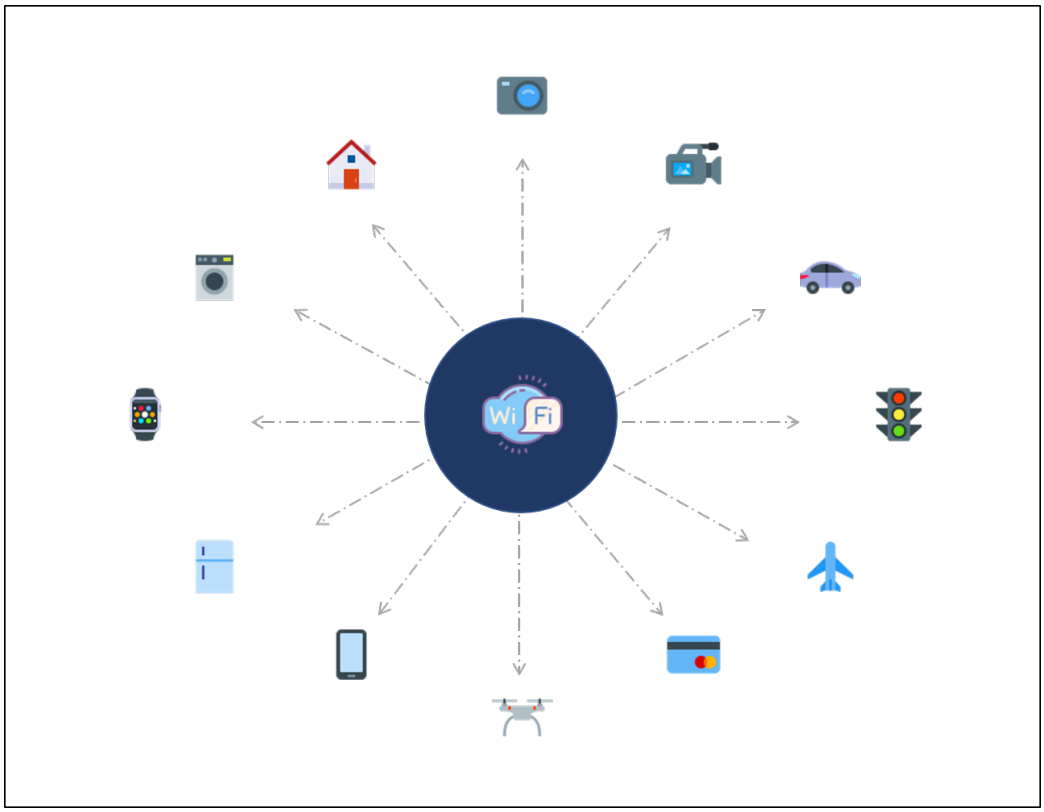

Figure 2.1: Computer power can be embedded in a variety of daily objects

2.1 Ambient Intelligence

2.1.1 AmI overview

Computer Science is a relatively new field of science that has progressed rapidly over the first decades of its existence. A rapid transformation[34] and continuous growth [34] has been observed from the starting point of digital machinery that performed simple calculations, to the point of microcomputers and their integration to everyday objects, that led to the introduction of several concepts including Ambient Intelligence (AmI).

The realization of AmI made possible for computer power to be embedded in objects (Figure 2.1) that are used daily like home appliances (e.g. programmable kitchenware or robotic hovering machines) [34], vehicles [70] or something as ordinary as a cup of coffee [60].

The evolution and the shift in the availability of computing power per person [46] are represented in Figure 2.2. Originally, the computers were expensive, with large volume and not particularly usable. A single computer unit would typically be utilized by many individuals. As the technology advanced and the market's prices dropped dramatically, the computers became more accessible to people.

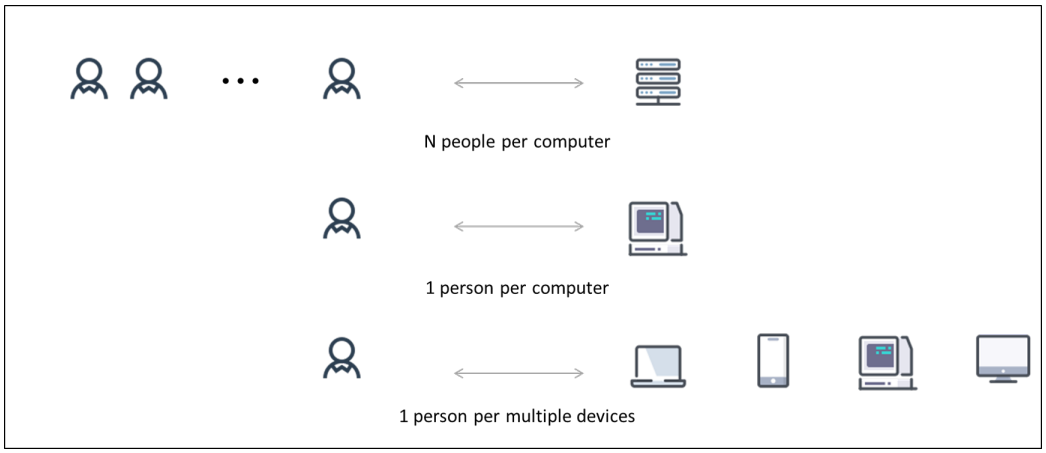

Figure 2.2: Change in user-computing ratio through time

Nowadays, one individual typically has access to multiple devices with computer power [46] (e.g. computers, laptops, mobile or tablets).

Computer science affects every aspect of human life making users' expectations grow due to their daily encounter with technological artifacts and innovative mobile applications[1]. The decreasing in size of microprocessors, the widespread availability of computer power and the successful user experience with the devices, led to the development and broad availability of digital resources [46], which facilitated the realization of AmI. AmI refers to electronic environments that they can sense and be responsive in the presence of humans, and combines multidisciplinary fields (Figure 2.3) to set new concepts for information processing [25].

The principal intention of AmI is to equip the environment with technology that operates *collectively*, using *context information* and *intelligence* which is hidden (Ubiquitous Computing) in connected devices. This aids to enhance the user experience through the assistance of natural and intuitive user interfaces [25]. Technology is integrated unobtrusively to common artifacts and utilizes (i) the **distributed information** and (ii) the **intelligence** in this interconnected and complex system. Sensors are providing valuable information, such as lighting conditions, temperature or vital signs (e.g. heart rate), with devices recording it in the environment. The intelligence is implemented by a special form of interaction and algorithms, that are able to identify humans in the environment, their behaviour, and act and react according to their needs.

2.1.2 AmI definition and directions

Even though the term "Ambient Intelligence" is commonly used, it exists in the bibliography only since 2001, when the term was used by the European Commission

[1]In the first quarter of 2018, Android users were able to choose between 3.8 million apps and Apple's App Store has about 2 million available apps [104]

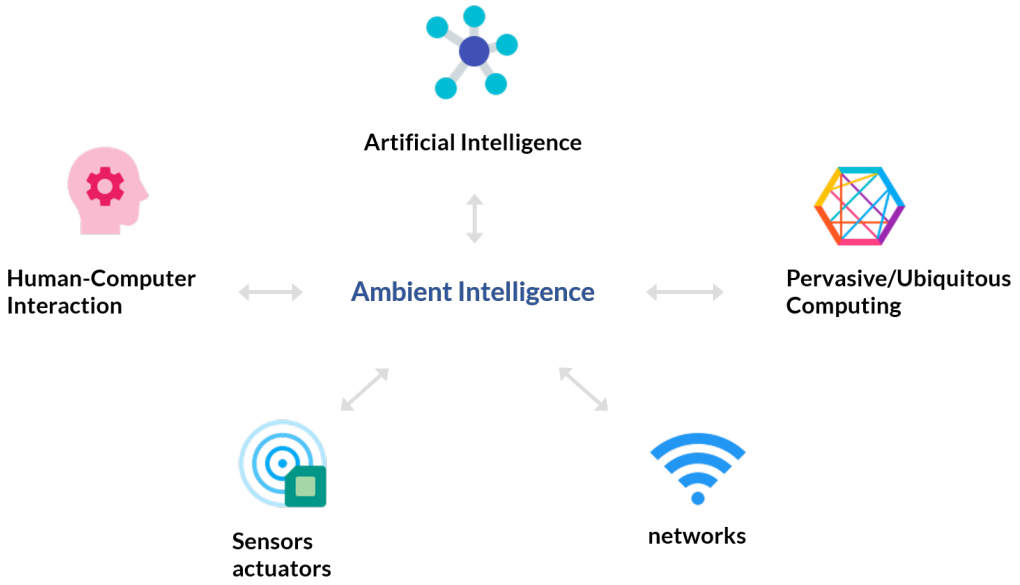

Figure 2.3: Interaction in between AmI and other fields [34]

[55] referring to a concept for the future of digital systems in the years 2010-2020, where users and devices will interact through interfaces in real-time and pro-actively [41]. Despite the widespread use of the term, a formal definition of AmI doesn't exist. Many researchers attempted to define the concept and some worth-mentioning definitions are presented in this section.

In European Commission's IST Advisory Group in 2001 [55], it is described as: "A potential future in which we will be surrounded by intelligent objects and in which the environment will recognize the presence of persons and will respond to it in an undetectable manner".Aarts et al. describes the concept as "A developing technology that will increasingly make our everyday environment sensitive and responsive to our presence." in [25].

Maeda et al. in their letter in the January 2006 issue of NTT technical review and in their revised paper [82] stated that Ambient Intelligence implies intelligence that is all around us. Cook et al. [46] report it as an emerging discipline that brings intelligence to our everyday environments and makes them sensitive to people. Ramos et al. [96] state that AmI deals with ubiquitous computing devices, allowing users to interact with their environment in an intelligent and unobtrusive way.

In [97] is reported that: "Ambient Intelligence (AmI) is a new research area for distributed, non-intrusive, and intelligent software systems both from the direction of how to build these systems as well as how to design the collaboration between ambient systems". Finally, Vasilakos and Pedrycz in [109], mention that "In an AmI environment people are surrounded with networks of embedded intelligent devices that can sense their state, anticipate, and perhaps adapt to their needs".

Several more definitions and explanations are available in the literature but

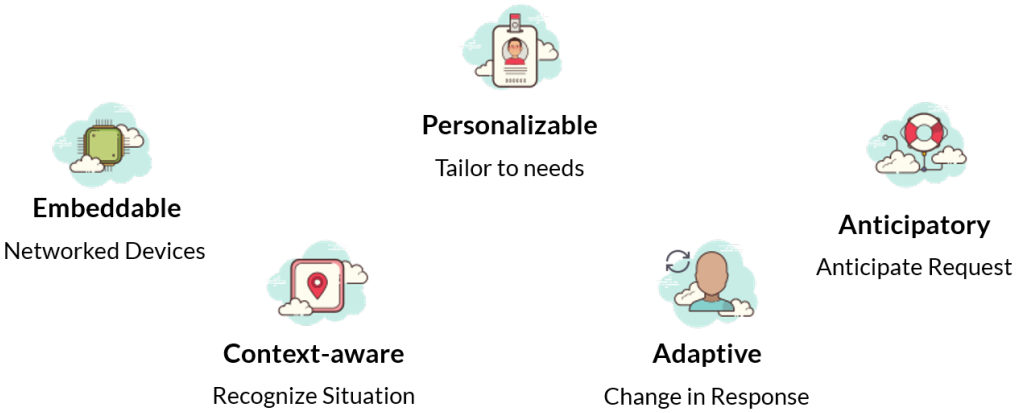

Figure 2.4: Different aspects of Ambient Intelligence (AmI)

despite the apparent differences, the idea is comparable. In general, an AmI environment is characterized by five primary features as depicted in Figure 2.4. It should build upon technological advancements as computer networks, sensors and sophisticated Machine Learning and artificial intelligence algorithms to enable the integration of distributing technology in a seamless manner to the environment. It should also be context-aware [101] and utilize the contextual information of the user, user location and situation identification. Another important aspect of such an environment is personalization, by customizing the systems' features on users and their behaviours. Furthermore, the concept of adaptation through learning users preference and acting accordingly as well as the concept of anticipation through artificial intelligence by predicting users intentions are really important to these systems [25].

2.2 AmI in Home Automation

Ambient Intelligence is a wide field with many applications. An example of an environment enriched with ubiquitous technologies is a "Smart Home". Several items in a typical house can be augmented with sensors (e.g. oven, fridge, bed) to gather useful information (sense) in order to process them and find an intelligent way (reason) for the system to act upon users' needs (act) (Figure 2.5).

In the next sections the home automation paradigm will be extensively explored, with main focus on systems that aim to control and monitor the environment [51]. These tasks (i.e. monitoring and controlling the users' surroundings) are challenging and without a suitable approach will cause frustration to users.

Figure 2.5: Ubiquitous computing in an Ambient Intelligence home

2.2.1 History of Home Automation Controllers

The idea of connected homes [63, 68, 38] was explored decades ago when authors envisioned interactive homes that will be able to operate the household digitally. Although the idea is relatively old, actual implementations mainly developed over the last few years due to the immense cost of creating such an interactive place. The first home automation system originated from the past 1966, with the ECHO IV [22, 6] made by Jim Sutherland. ECHO IV [6] was the first home automation controller, which if used correctly it could perform limited automation tricks, such as basic TV and stereo control, run some math and educational programs and have an alarm clock. The controller (see Figure 2.6) was so large that needed a lot of space and power to operate, as well as plenty of knowledge to use it.

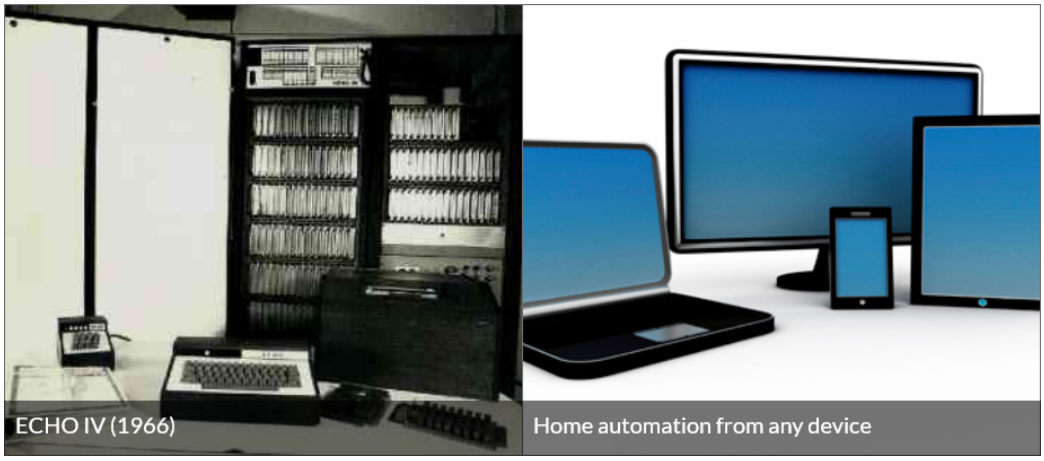

Figure 2.6: (a) ECHO IV home controller in 1966 (b) In the modern age, any display device can control and monitor an intelligent home

Home automation began to gain popularity in the early 2000s [69]. At this time, many attempts were made for implementing such an environment. Orange mobile network operator [69, 24] focused their research to comprehend the users' attitude towards these environments and the social factors that affect the technology use with their Orange-at-home (2001) project. MIT's House_n project (1999-2000) [23] is an architectural project that incorporated sensing components in the building

to implement a ubiquitous home which is responsive to its occupants and develop innovative user interface home automation applications.

The "Aware Home" (started in 1998) from Georgia Tech [27, 69] is a context-aware smart environment that assists people in their everyday activities. They developed several projects associated with the home, such as software that creates family photo albums, intercom systems, electronic tags for mislaid items (e.g. keys) and more. Finally, MavHome from the University of Texas [13] enhanced the idea of the home automation with concepts from artificial intelligence field by creating smart agents for home.

HomeLab (2004) [50] is a project where scenarios of Ambient Intelligence were implemented and tested. They mainly focused on creating user experiences using display technology and avoid to just provide huge amount information.

VERA Z-wave project (2008) [36] presented real-time feedback on power use and remote and automated control of four appliances through smart plugs. Additionally, four types of spaces were identified in homes; communication, work, private and public entertainment. These spaces, the placement of technology in them and the users' relationship with the technology are the main concepts of this analysis. Another project, the British Gas HIVE (2013) [37] provided remote control of heating and hot water. This system was having difficulties with boiler consistency so was only installed in eight homes.

Early research attempts towards implementing a smart home have reported the technical difficulties of implementing intelligent domestic environments [45]. In this home paradigm, the computer software plays the role of an intelligent agent that perceives the state of the physical environment and users and then takes actions to achieve specified goals. Current research in ambient intelligence aims to allow devices to interact with other devices and the networking infrastructure without human control. The intelligent home must also be imbued with an awareness of the resident context (location, preferences, activities), physical context (lighting, temperature, house design), and time context (hour of day, day of week, season, year) [45].

Kamilaris et al. [72] in their HomeWeb (2011), take a web-oriented approach in the intelligent home project. In their work, they developed a Web-based application framework, which supports concurrent interaction with multiple users. They use the concept of web and physical mashups [65], by exploiting real-world services offered by physical devices and combining them using the same tools and techniques of Web mashups [49].

Palanca et al. [91] introduced a system (2018) for goal-oriented, self-adaptive, smart home environment. In this approach, users are able to interact with the system by expressing their goals which are interpreted as a set of agent operations in a way that is transparent to the user. This is suitable for environments where ambient intelligence and home automation control are combined for the user's benefit.

Fogli et al. [58] in their systematic literature review of several tools supporting

End-User development configuration for Smart home proposed a conceptual framework for the design and continuous evolution of ambient intelligence environments. Most of the systems that are reviewed are based on rule-based paradigms, with visual interfaces for users to compose events, using structures like "if-condition(s)-then-action(s)" or "when-event(s)-then-action(s)".

Recent surveys confirm that home automation is an emerging field with a current revenue of 30 million that will outreach 70 million by 2020, without yet reaching its full potential [85], which triggers the extended research in order to revolutionize daily human life. Despite the extensive research, most of these projects lack of focus to the human factor. Most of them ignore that in the end humans will inhabit these intelligence surroundings, and only test how much technology a house can have.

2.2.2 Age of connected devices and home automation controllers

Over the past decade, many advances have been achieved regarding the Internet of Things (IoT) technology, which enables the implementation of actual ambient intelligent spaces. The interest in smart homes and home automation is shown in Figure 2.7. There is a growing trend towards Home automation queries [64] in Google information retrieval service (see Figure 2.7), that is the result of the rapid introduction of digital technology advancements as well as the plentiful embedded devices that are being released. These high-tech sensors can measure with high accuracy the environmental conditions, offering opportunities for intelligent and context-sensitive behaviour.

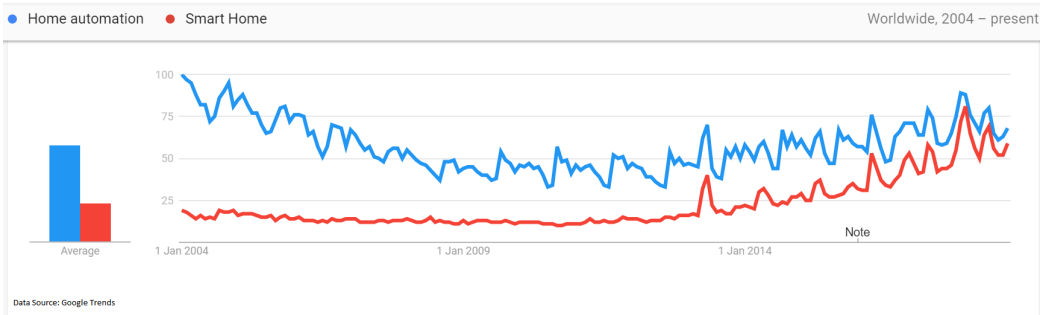

Figure 2.7: The worldwide interest over time for Home Automation and Smart Home topics since 1/1/2004 [64].

In 2010, Nest Labs was invented because of the frustration of their founder for the limited features in the thermostat devices available [16]. Smart things in 2012 raised $1.2 million on a Kickstarter, by promising to link any connected device within a house. They provided a mobile application that allows to operate and monitor the environment. This application works with If this Then that (IFTTT) recipes to trigger events. They also provided a web-based IDE for developers to create new rules [19].

By 2014, several vendors had produced next-generation connected devices with their individual APIs and mobile applications. Subsequently, a new problem appeared; the operation of different devices had the users to switch between multiple applications with different functionality among them. Users in order to use their smart devices had to download multiple applications from different vendors, with different "look and feel" and functionality, that make the experience rather difficult and confusing. In addition, the absence of standards on IoT devices, as well as the heterogeneity on their API protocols used and functionality made the cooperation of devices from diverse vendors impossible [88]. These difficulties emphasized the necessity for a unified application with a common presentation layer for every connected device and service.

One of the first unified commercial applications made by Ben Kaufman in 2014, is "Wink" [21]. Wink brought various smart items onto a small network in order to manage them under a common presentation layer. Since then many attempts have been performed for such an umbrella application both in the commercial and academic sector.

Samsung SmartThings [19], is an indicative commercial example, based on a smartphone application that is able to monitor and control connected devices in home. Gideon Smart home[9], is a mobile application that helps user to interact with and control multiple devices from multiple brands. It uses "Tricks", a set of IF/THEN rules to prevent damages, issues or helps user with daily life. They also provide a chat-bot to control vocally the application and the devices, locally and remote control of the home, as well as Device discovery. IFTTT [11] is a commercial application which aids connected devices to communicate with each other. It can connect with multiple vendors and gain insights from use and build services to control the home. Control4 [5] is a unified application to orchestrate the connected devices in order to communication with each other .

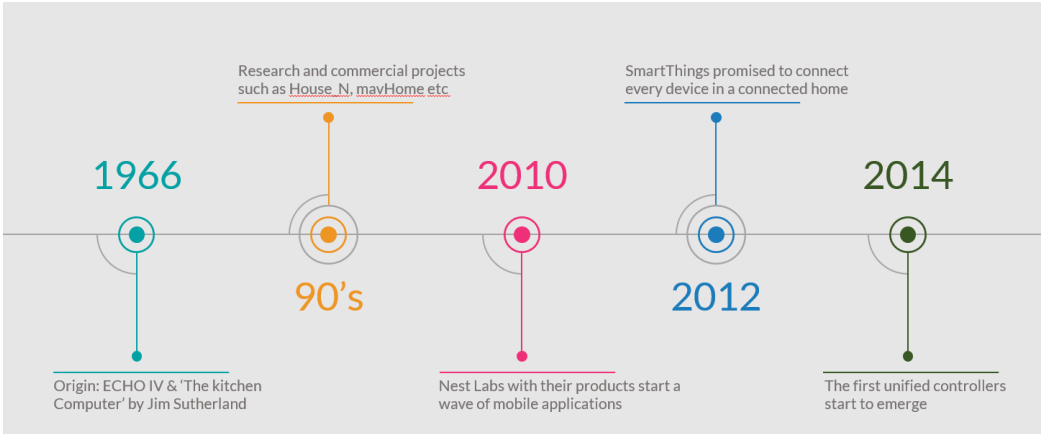

Figure 2.8: Home automation timeline

Similarly, in the academic sector, some projects are: SPOT [88] which is a

smartphone-based platform to tackle heterogeneity in smart home environments, HouseGenie [105] a smartphone application for Universal Monitoring and Controlling of Networked Devices in Smart Home, MeSchup [75] which is a mashup platform for programming interconnected smart things. A ZigBee [62] based home automation system and Wi-Fi network are integrated through a common home gateway.

In [76], a standalone, low cost smart home system is presented, which is based on an Android application, communicating with the micro-web server providing some complex functionality in the home. The authors in [94] developed a low cost home controller and monitoring system using an embedded micro-web server, with IP connectivity for accessing and controlling devices and appliances remotely using an Android application. All of the previous projects are implying the need for some unified system platforms.

Cloud-based and multi-vendor applications have also been released with the promise to connect any device (e.g. IFTTT [11], OpenHab [18], Universal Control Hub [113] amd An Integrated Cloud-Based Smart Home Management System with Community Hierarchy [78]), but users are still restricted by the appliances companies' partnerships and not sufficient support for programming heterogeneous devices.

Although research solutions allow more generic applications for the integration of the devices, they are restricted in auto-generated UIs, not responsive user interfaces for mobile applications which are not able to adjust their views into other display devices and limited applications that they can only be used as controllers or user interfaces that have not included user-centered design in their development cycle [78].

In summary, some of the problems that cause the home automation technology to be restricted to laboratory projects and narrow device connectivity are:

- The lack of flexibility of the user interface, meaning that existing systems use one modality (e.g. smartphone) to control and monitor the surroundings.
- The User interface heterogeneity, meaning that systems from various vendors provide different user interfaces, making it hard to organize them under a common application.
- The devices APIs' diversity narrows the interoperability among the devices. Different vendor devices may not cooperate well in an intelligent home environment.
- The cost to acquire a range of these devices.

The above limitations highlighted the need for the intelligent home approaches to direct into two parallel axes (i) to integrate these smart things under a common interface, and (ii) to make each connected device and service part of the existing web in order to offer flexibility for end-users.

2.3 Web Application Mashups

In this section, first the technology advancements in Internet of Things and Web Of Things will be discussed. These two topics will introduce the concept of Web Application Mashups, upon which the proposed system is based on. After a general overview of web application mashups, we will dive in UI mashups, which are mashups for the presentation layer of an application.

2.3.1 Internet of Things

Over the past years, the Internet has exponentially expanded to a global network serving billions of users [73]. The "Internet of Things" (IoT) [87, 56], describes the extension of Internet connectivity to everyday objects enriched with sensors. In other words, IoT transforms these objects into so called "smart", by utilizing the underlying technologies [29] (e.g. ubiquitous and pervasive computing, communication technologies, sensor networks, Internet protocols). A smart object represents the interface between the physical world and the digital world [57], using confluent interaction between users and technology.

Smart objects and their tasks establish domain-specific applications (vertical markets) while ubiquitous computing model application domain-independent services (horizontal markets). Figure 2.9 demonstrates how IoT associates every domain-specific application with domain-independent services, whereas in each domain sensors and actuators communicate immediately with each other [29].

The term "smart object" describes an embedded system, consisting of a **thing** (physical entity) and a **component** (the computer) that has two purposes: (i) the system processes the sensor data and (ii) establishes a wireless communication link to the Internet [73]. For example, an intelligent fridge (physical entity) keeps track of the availability and the expiration date of the food (Sensor data) as well as the nearest supermarkets (through internet and gps information) to place an order when a specified item is below a defined limit.

A study published in [20] tried to distinguish the most successful IoT segments for 2018. The majority of IoT projects are classified as Smart City (367 projects) due to current Smart City initiatives started by authorities around the world. The most popular application is "Smart Traffic", with projects like parking systems and traffic monitoring. The second trend in place is connected industry with 265 projects, including equipment monitoring in non-factory environments such as asset monitoring and remote control of connected types of machinery such as drills, or even entire mines and oil fields (e.g. Cisco's connected mining operations for Rio Tinto in Western Australia). The top ten applications in IoT are in figure 2.10.

Another successful IoT segment is Intelligent Home, with projects that have attempted to create new applications for intelligent building infrastructure over the years [61]. One requirement of intelligent home and ambient intelligence, is that the enhanced devices in the house, are connected to a network in order to

Figure 2.9: The overall picture of IoT (vertical and horizontal markets) [29]

reduce the cost and complexity of configuring and using the connected devices.

2.3.2 Towards the Web of Things era

In late 2004, the concept of Web 2.0 [80] began to gain popularity when O'Reilly Media and MediaLive hosted the first Web 2.0 conference. Web 2.0 encompasses user-generated content with user interaction and collaboration, ease of use and interoperability of the website for the end-users [66, 86]. Examples of Web 2.0 applications include blogs, wikis and popular systems such as Google Maps, FaceBook. The development of Web 2.0 gave the opportunity to use the web as an application-layer for the IoT devices, the Web of Things (Figure 2.11).

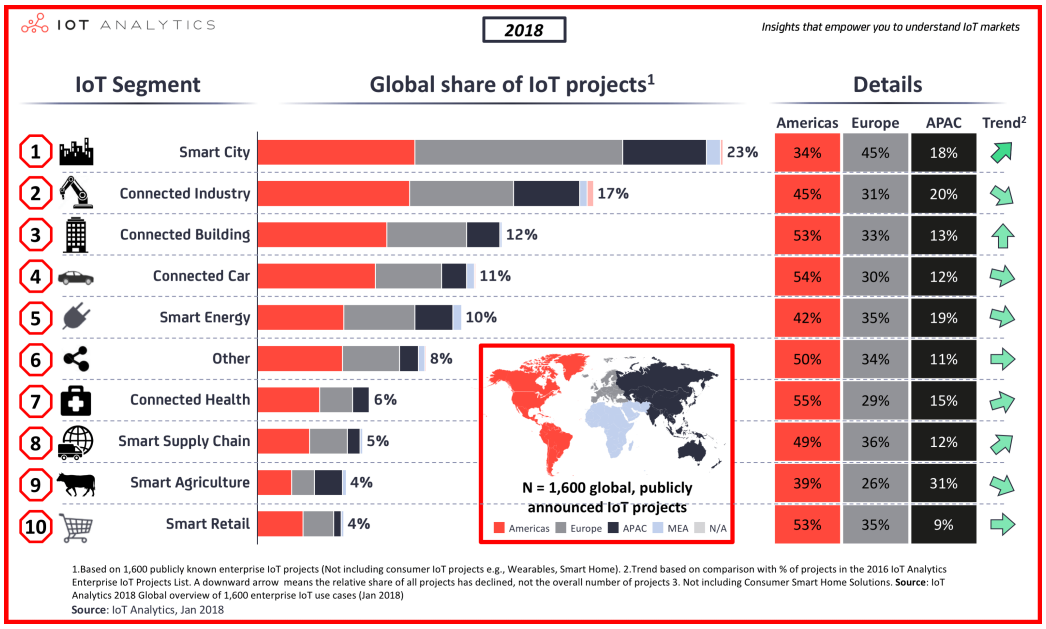

Figure 2.10: The ten most popular Internet of Things applications (2018 ranking [20])

This concept simplifies the creation of IoT applications and the seamless integration of connected devices to the Internet. The WoT architecture reuses successful existing web standards used in web for decades (e.g. REST, HTTP, JSON, WebSockets), to create networks of smart objects [67]. It took several years for this concept to be accepted in academic cycles and in 2010, the first Restful Architecture (see figure 2.12) for the Web of things was proposed.

Over the last few years, many applications have been developed using "Web of Things" approaches and integrating these concepts in the connected home paradigm. The main idea is that instead of using the Internet only as a transport protocol, it is proposed to use it as a platform, in order to make smart things a fundamental part of the Web [66, 65, 67]. "Smart Things" are everyday objects that can be enhanced digitally, such as embedded devices and sensors, electronic appliances or artificial services that the house can offer. Every connected smart thing has its individual user interface, set in a specific URI and all of the interfaces are composed in an integrated application are called a mashup [65, 72, 102, 71].

In terms of Web Services, a lot of research has been conducted in the area of Web of Things for the architecture that is most suitable in this kind of context. The direction recommended is using Rest architectural style recommended for Web 2.0 Mashup applications implemented by URIs in order to identify resources, HTTP as a transport protocol and standardized media types such as HTML for the interface implementation [66, 65, 67, 102].

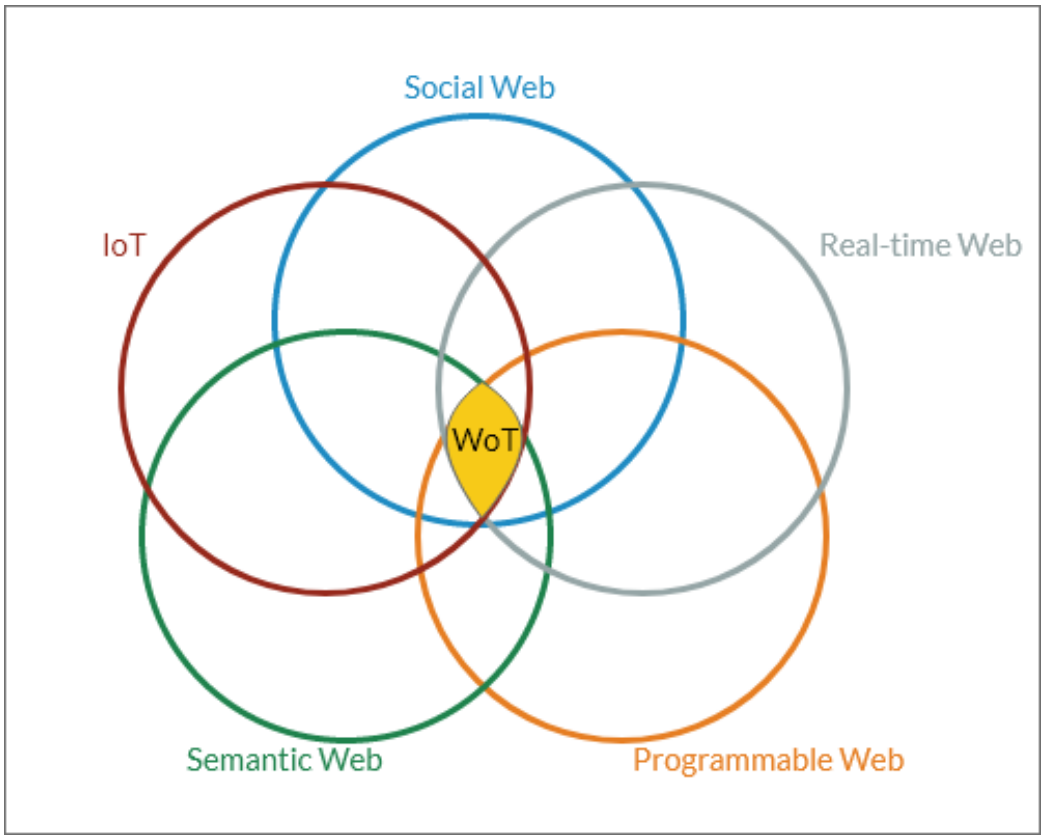

Figure 2.11: Popular web and technologies and well-known Web mechanisms, describe the Web of Things

2.3.3 UI Mashups for Home Automation.

2.3.3.1 Overview

An interesting topic that is connected with the accelerating development of Web of Things and Web 2.0 are the **web mashups** [49]. A *web mashup* is a web application that combines **data**, **logic** or/and **interfaces** from multiple sources. A typical definition of a Mashup by Daniel et al. [49] is: *"A mashup is a composite application developed starting from reusable data, application logic, and/or user interfaces typically, but not mandatorily, sourced from the Web."*

The typical architecture of a mashup consists of three layers [49, 102, 74]:

- Data Component: Monitors and handles the data that flow in an ambient environment. Data in this application are sent, stored or received using JSON format.
- Logic Component: the underlying functionality of the service using RESTful services.

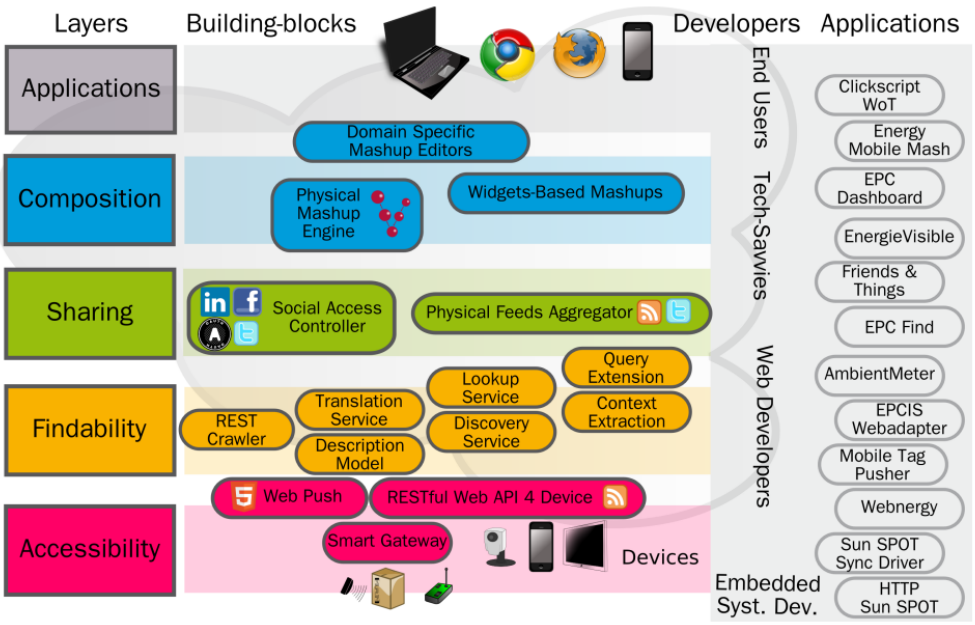

Figure 2.12: The four layers of the Web of Things architecture: Accessibility, Findability, Sharing, Composition.

- Presentation Component: Uses HTML, CSS and Javascript technologies, the developer can create a user interface created with user-centred approaches

For the ambient intelligent spaces, the term of physical mashups is used. Physical mashups are composite web application following the typical definition of mashups, which also involve smart things and virtual web services [65].

2.3.3.2 User Interface Mashups

One of the novelties of mashups is that they can integrate User Interfaces, sourced from the web to the presentation layer. This category of mashup applications is described as User Interface (UI) mashups [49]. They are able to combine various components at the presentation layer of the application stack while reusing data and synchronization elements from the involved of the UIs. The output of this integration is a newly published Web application. This approach is very beneficial when it is expensive to develop a new application, or the user interface is overly complicated. The resulting applications are mostly client-side applications since the business logic is concerning how to render and invoke the UI components. In Figure 2.13 is demonstrated the User Interface mashup model with inter-component communication and synchronization as described by Florian Daniel in [49].

This model include three (3) elements: (i) **Component Operations**, that

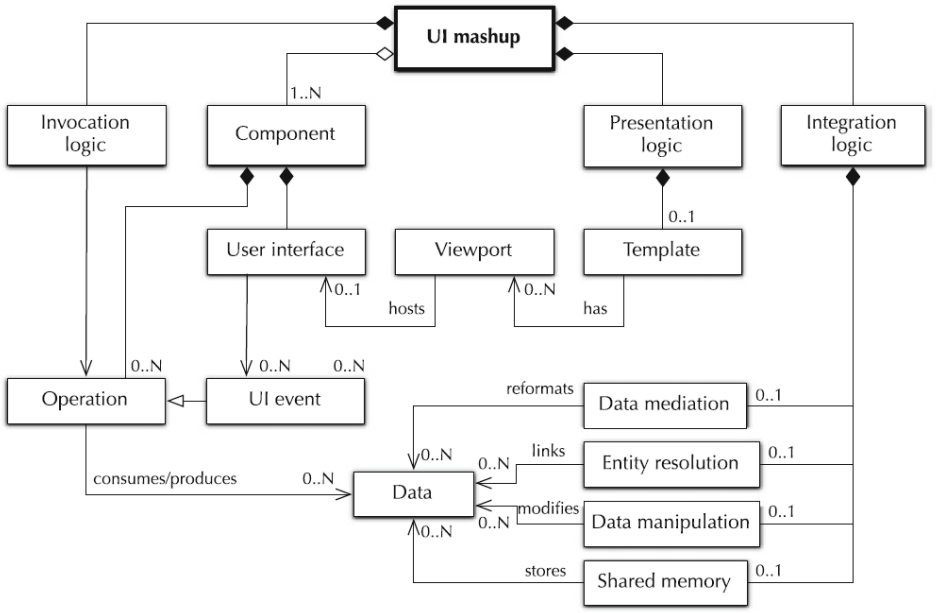

Figure 2.13: User Interface model with inter-component communication

allow to centrally monitor the components, (ii) **UI events**, that inform the other components about state changes and (iii) **Shared Memory**, that enables the exchange of data among the components.

A very simplistic form of UI mashups can be implemented with embedded third-party code (HTML snippets) or online resources within a basic HTML page. These mashups are called HTML UI mashups (see figure 2.14). This architecture consists of a HTML page, which may incorporate URI references to entire web pages, multimedia or other web components. Resources are embedded via iframes. Server-side technologies and widgets introduce new features to UI mashups. The UIs can be stored as autonomous widget UIs to a widget repository. Additionally, composite applications that can be synchronized and share data can be created with requests to a web server that stores the repository.

2.3.3.3 Mashups for home automation

Mashup web applications for home automation services started to emerge shortly after the release of the Web 2.0. A commercial Web of Things platform has been developed by EVRYTHNG software company [7, 33], which offers a digital ecosystem by providing each individual object with a unique active digital identity (ADI). The platform can connect consumer products and connected devices to the web with a real-time application, with cloud-to-cloud connectors. Their research includes three main projects. An IOT Smart Product Platform, where the Internet

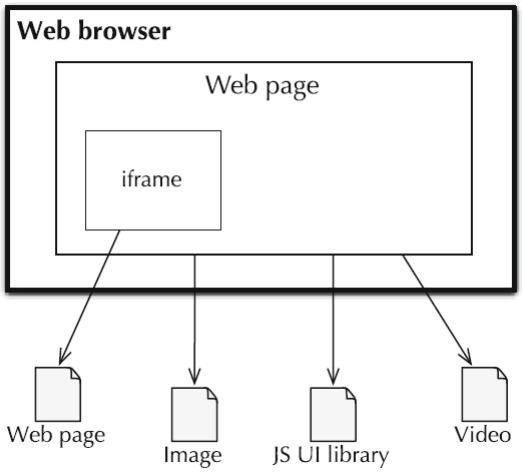

Figure 2.14: HTML UI mashup. A web page has embedded HTML snippets for displaying contents retrieved from endpoints.

of Things products are connected and managed in real-time. Reactor, an Enterprise Rules engine that makes real-time decisions and triggers scripts. And finally, Security and Access control for the smart objects, data and users and more.

An interesting example of a physical mashup home automation system is HomeWeb [72], a web-oriented application framework with the use of IPv6 and 6LoWPAN technologies. A prototype of a framework has also been introduced in [65], which uses the notions of WoT and mashup technologies. However, these projects are still under development trying to overcome client-server architecture and discovery of smart things problems.

All of these projects, besides their technical difficulties which are trying to be solved, are technology dependent without taking into consideration the user. They lack user-centered design, as well as the ability of personalization. Furthermore, the users will not only need all these artificial web services and applications but we have to consider an easy approach to even program their home tailored to their needs [108, 43].

2.3.4 The curious case of pluggable user interfaces

Despite the fast emerging IoT and WoT technologies, as well as fields such as Machine Learning and Artificial Intelligence, which are offering context-aware computing mechanisms, home automation is not fully integrated with everyday life. This is due to the complexity of the applications being developed and more specifically the complexity of their user interfaces, which makes the users frustrated and unsure of how to operate them.

In this ambient environment, numerous devices are working and cooperating, each one having its own functionality and an appropriate user interface. It would

be impossible for a single programmer to develop such an environment. A unified native application would be heavily weighted and impossible task even for a team of skilled programmers. Due to the complexity, the web-oriented approaches (e.g. mashups) along with pluggable interfaces, appear to be the correct direction in solving home automation problems [47].

Each one of the appliances' user interface must be plugged into the unified application. Also, these interfaces should be easy to use, so as to increase the usability in this complex environment. This kind of interfaces has been studied in [112], where the pluggable interfaces for the different underlying services were exposed by a Universal Control Hub [112], for a set of different display devices (i.e. controllers). In this work, the authors try to bridge the display devices (UI) and the individual sensors (functionality). This hub is being applied in the i2home project [30], where its main goal is to develop technology for the elderly and people with cognitive impairments.

2.4 When Web of Things met HCI

The aforementioned systems and research work made a significant contribution to the design and development of applications for the intelligent home. However, the existing products are mainly focused on controlling devices rather than monitoring and operating the whole home environment. Current technology concepts are mature enough for implementing ubiquitous systems, but it seems it is difficult to incorporate all these notions on a large scale, in order to make intelligent homes available for the end-users. Moreover, in such an intrusive environment, users must communicate their needs effectively for the technology to solve their difficulties. Aaarts et al. highlight the lack of user-centred design in ambient intelligence environments in [26], revealing that people need a evenhanded approach in which technology should assist people instead of monitoring their every move.

A user-centred design is a necessity for such a complex and interactive environment in order to provide the best User Experience with a simple, intuitive and consistent user interface. The users should be able to solve their own problems, with the application giving them the opportunity to contribute to their home software.

2.4.1 HCI in Mashups

As the web transforms to Web 2.0, the way users access content and interact with each other through web changes. This results to several issues [53, 93] which have been observed to the usability of the web sites.

The main challenge is how users interact with Web 2.0 applications. Nowadays the Internet is no longer used only as mean for viewing content, but as a platform where a user can interact with applications and services. Other challenge can be the unpredictable behavior of the user and how this affects the user interface [93].

The concepts of Mashup web applications and pluggable user interfaces, as well as concepts from human-computer interaction area such as user-centred design, inspired the idea for the proposed system. The UInify platform is a framework that contributes a set of tools, allowing designers to combine multiple individual UIs together and introduce new rich User Interface compositions. Moreover, end-users are provided with a single UI, through which they can monitor and control the intelligent facilities of their surroundings as well as empowering them to tailor the intelligent facilities according to their preference.

2.5 Discussion

Evidence from the present systematic review confirm that a complex AmI environment has various UI applications that need to cooperate and exchange data. Therefore, in order to aid the designers of smart environments in defining complex user interface, an approach that incorporates the state-of-the-art technology such as Web Of Things and UI mashups is required.

Based on our study, a platform with similar functionality as in UInify, which enables the entire design and deployment of a composite user interface, does not exist. In particular, this work proposes a HTML UI mashup that can embed third-party code (HTML snippets) of published web applications from the smart home within a basic HTML page. All these web applications are imported to the database by the AmI Solertis system [79] developed by FORTH-ICS. The third-party code may be part of pluggable user interface applications (e.g. Music, Movies), or widgets of utilities applications (e.g. AmI TV player, Recipe step-by-step guide in kitchen counter top). The proposed system aims to create compositions that are brand new applications that incorporate several UIs from the widget repository.

Chapter 3

System Requirements

This chapter provides the details regarding the technological components that comprise the UInify platform. This platform aims to support designers in the process of building complex user interfaces for their intelligent environments. In particular, UInify allows (i) the exploration of the available application and application components that are available in the intelligent ecosystem as well as (ii) the exploration and introduction of complex user interfaces, namely Compositions, through an intuitive editor and finally deploy them as a UI mashup [49].

3.1 The AmI Home environment

Generally, intelligent home residents are expected to manage multiple environments (e.g. home, office) that incorporate a variety of physical devices (e.g. Lamps, Thermostats, TV), and artificial services (e.g. Multimedia systems, Kitchen assistants). Each one of these ambient spaces requires to be monitored and controlled efficiently in a timely manner, regardless of the heterogeneity of its components. Furthermore, since a conventional user is not expected to have programming experience, the resulting user interface should be intuitive and simple to use. To achieve that, the system should follow well-established guidelines used in human-computer interaction. Furthermore, home surroundings need to observe the overall context of use (e.g. physical conditions, activity at hand) and should respond according to the sense-reason-act paradigm (see Figure 2.5); the triggering event could either be sensing data or user-oriented commands. To provide the best support to its users and automate various everyday tasks, an ubiquitous, multimodal and context-aware system is needed.

UInify facilitates the creation of composite applications for the Intelligent Home of FORTH-ICS. The web applications are exposed as services through the AmI Solertis platform [79], which is a system that offers tools, allowing management, programming, testing, and monitoring of all the individual artifacts of a Smart Environment. The ambient environment is equipped with a variety of display devices (e.g. smart phone, tablet, desktop or tangible large screens) that the user

could interact with the complex interfaces, that are introduced through UInify. All the web applications with Graphical User Interface (GUI) are provided by AmI Solertis, and they expose a set of metadata information, to facilitate the selection of the appropriate displaying device. The UInify framework can also assist users to create complex applications for other intelligent environments - beyond the intelligent home - that are actively under development in FORTH-ICS (i.e., Intelligent Classroom [31], Smart Greenhouse [39]).

In the rest of this chapter, various technical topics are presented concerning UInify platform. Firstly, the features and the functional requirements are presented. Then, the conceptual model to fulfill these requirements, that emerged from iterations is illustrated. Finally, the architecture as well as a brief discussion of the technologies used will be presented.

3.2 System requirements

UInify aims in supporting designers of smart environments that need to create an GUI application that will aggregate multiple user interfaces available in the environment. The main objective of the system is the orchestration of the applications by integrating them in a complex User Interface, namely Composition, that with the combination of a collection of rules is used to create adaptive, context-aware and multi-modal user interfaces. UInify is a web application that implements the above functionality and binds all the individual applications together. The basic functionality of the system includes:

- Application repository & details
- Composition repository & details
- Introduction of a new composition into the system

3.2.1 Features and Functional Requirements

This section describes the fundamental features and functional requirements [84, 83] that the UInify framework should meet. Functional requirements refer to a specific behavior and functionality that the UInify platform should have and this has been solicited through an iterative process, using various collection methods including brainstorming, personas and scenario building.

1. Login **FEATURE**

   Actors: All registered users

   Preconditions: Unauthenticated session

   Postconditions: The user is logged in the system in an authenticated session

Functional Requirements:

- The user is registered with a valid email and password
- The user provides credentials (email and password) in a form to login
- A valid authenticated session is created
- In case of an error an appropriate message is displayed

2. Logout **FEATURE**

   Actors: All logged in Users

   Preconditions: Authenticated session

   Postconditions: Unauthenticated session

   Functional Requirements:

   - The user logs out from the UInify system
   - The session stops to be valid

3. Register **FEATURE**

   Actors: All Users

   Preconditions: 1. Unauthenticated session 2. The user is not registered

   Postconditions: The user is registered to the system

   Functional Requirements:

   - The user must provide a valid name
   - The user must provide a valid and unique email
   - The user must provide a password
   - In case of an error an appropriate message is displayed

4. Dashboard **FEATURE**

   Actors: All registered users

   Preconditions: Authenticated session

   Postconditions: A dashboard with information about the system is displayed

   Functional Requirements:

   - Display statistics of UInify
   - Display the top ten recent compositions

- Display the top three recent applications
- Display the top three most used applications

5. Application Library **FEATURE**

   Actors: All Super Users / Designers

   Preconditions: Authenticated session

   Postconditions: List of available applications of the intelligent space

   Functional Requirements:

   - The application library will provide a list of the imported user interface applications of the intelligent space for every available screen device
   - Each application of the intelligent space should expose whether it is an application or an application component
   - The application Library list displays the number of available application components or the screenshots provided for the specific application component
   - The application library will provide sorting (alphabetically) and filtering capabilities
   - A search feature is provided in order to search for a specific UI application in the intelligent space

6. Application Details **FEATURE**

   Actors: All Super Users / Designers

   Preconditions: Authenticated session

   Postconditions: The details of the chosen application is displayed to the user

   Functional Requirements:

   - The name of the application
   - The creator of the application
   - An optional description of the application
   - The primary tag of the application as well as optional secondary tags
   - The list of the application components, if the item is an application, and the list of the available screenshots if the item is an application component

- The developer of the application should provide some information about the service such as name, description, primary room intended to use (e.g. the artificial service fridge has as primary room, the kitchen) as well as the capabilities or sub services, that the main service includes (e.g. the service 'Nutrition' can have subservices as 'Calories burned today', 'Weekly diet schedule', 'Notifications', 'Special recipes').

7. Composition List **FEATURE**

   Actors: All Super Users / Designers

   Preconditions: 1. Authenticated session 2. There is no similar/same Composition

   Postconditions: List of available compositions of the intelligent space

   Functional Requirements:

   - The composition library will provide a list of the created compositions of the intelligent space for every available screen device
   - Each composition of the intelligent space should provide the information about the room and device that it is intended to be displayed to
   - The composition Library list displays the number of available layouts for a composition
   - The details provide the option to create a new composition
   - The composition library will provide sorting (alphabetically) and filtering capabilities
   - A search feature is provided in order to search for a specific composition in the intelligent space

8. Composition Details **FEATURE**

   Actors: All Super Users / Designers

   Preconditions: Authenticated session

   Postconditions: The details of the chosen composition are displayed to the user

   Functional Requirements:

   - The name of the composition
   - The creator of the composition
   - An optional description of the composition
   - A primary tag of the composition as well as optional secondary tags
   - The list of the available layouts

- The option to create a new layout
- The option to delete the composition
- The details will provide sorting (alphabetically) and filtering capabilities
- A search feature is provided for searching a specific layout in the composition

9. Composition Creation **FEATURE**

Actors: All Super Users / Designers

Preconditions: 1. Authenticated session 2. There is no similar/same Composition

Postconditions: A new composition is created in the database

Functional Requirements:

- The name for the composition
- The intelligent space for the composition
- The room for the composition
- A primary tag and an optional secondary tag for the composition
- The display device for the composition

10. Intelligent Space List **FEATURE**

Actors: All Super Users / Designers

Preconditions: Authenticated session

Postconditions: List of the available intelligent space

Functional Requirements:

- The intelligent space library will provide a list of the available AmI environment space
- Each intelligent space entry should provide its type
- The intelligent space library list displays the number of the available rooms
- The intelligent space library will provide sorting (alphabetically) and filtering capabilities

- A search feature is provided in order to search for a specific intelligent space in the intelligent space

11. Intelligent Space Details **FEATURE**

Actors: All Super Users / Designers

Preconditions: Authenticated session

Postconditions: The details of the chosen application are displayed to the user

Functional Requirements:

- The details provide the name of the intelligent space
- The details provide the available rooms as well as the number of the compositions in that room

12. Search **FEATURE**

Actors: All Super Users / Designers

Preconditions: Authenticated session

Postconditions: System displays search result

Functional Requirements:

- The user should be able to search for the application or composition of her choice **Basic Search**
- The user types her query in a textbox
- The user clicks a search button or hits the enter key
- The system reads the query as a case insensitive substring
- The system matches the substring with the records in the database
- The system displays the corresponding in the form of a list or a table **Advanced Search**s
- The user can filter the results
- The user can sort the results

13. Profile **FEATURE**

Actors: All Super Users / Designers

Preconditions: 1. Authenticated session 2. Necessity to view 'Profile'

Postconditions: The user can view the profile

Functional Requirements:

- The information about the user is displayed
- A list of applications developed by the user are provided (if available)
- A list of compositions developed by the user are provided (if available)

14. AmIView editor **FEATURE**

Actors: All Super Users / Designers

Preconditions: Authenticated session

Postconditions: Create a template layout for the selected composition

Functional Requirements:

- The user is provided with an editor to create a layout
- The user can add rows to the layout and define their height
- The user can add columns to the layout and define their width
- The layout will be saved in the database after complete the layout
- The user can import web applications to the layout after completing the creation oftemplate

3.3 One UI to bind them all

Following these guidelines, the envisioned tool aims to serve as a unified system platform in an ambient environment. UInify adopts ideas from User Interface Mashup web technologies and User Interface (UI) composition [49, 48], to introduce a brand new unified application, that incorporates several UI components with a common presentation layer for all underlying devices and services. Our web-based approach simplifies the development of the individual applications, making them totally autonomous and easy to be integrated into the environment.

The designer in order to integrate an application in the intelligent environment has to design a UI mashup. Typically, those applications are composite Web applications, where their development requires both external resources (known as mashup components) and internal integration logic (or mashup logic) [49]. UInify application integrates two or more pluggable User Interface applications by enabling them to interconnect. The application orchestrates these different user interfaces and combines them into an integrated interface. The system's ability to control multiple connected devices offers the opportunity for combining several input devices and actuators in order to provide control over the overall home. That allows to the intelligent home to have several distinct software components

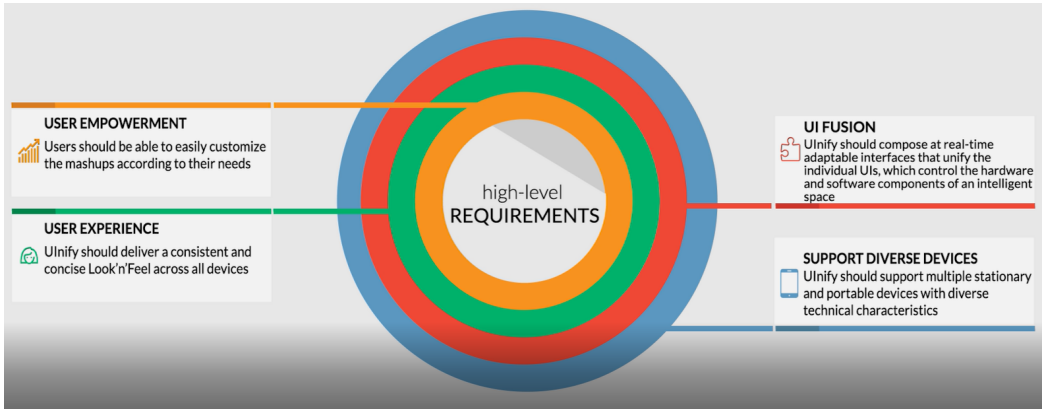

Figure 3.1: The high-level requirements of UInify

and connected devices interfaces developed by different experts, with the presentation layer being orchestrated under a common roof. Moreover, in future versions, UInify could easily allow the end-user to have more control of the environment by integrating simple programming tools such as Rule-based programming [42] and IFTTT and Graphical Programming [52], which can help inexperienced users to program their home like solving a jigsaw puzzle.

This approach has several benefits similar to the benefits of mashup tools [49] such as:

- End users can contribute to the home software by creating custom interfaces with several components provided, depending on their needs or create small applications with triggering action rules for their devices (IFTTT) [49].
- Developers don't need to start a unified application from scratch, but utilise the underlying functionality of the separate exposed web applications [49].
- The integration of end users and developers leads to more useful software products and user satisfaction [49, 48, 35].
- They reduce the cost of developing such an application, because of the involvement of the user in the development stage. This reduces the iterative experimentation and the end-user evaluation [49, 48, 35].

3.4 High-level scope of UInify

3.4.1 High-level requirements

Before the design and implementation details of UInify platforms are presented, it is important to have an overview of the high-level requirements of UInify platform (see Figure 3.1).

Support diverse devices

UInify platform should support multiple stationary and portable devices with diverse technical characteristics. These devices are named in the context of the system "Artifacts". This is essential in an ambient environment due to the multiple and diverse devices that the end user may own.

Fusion of individual User Interfaces

UInify platform should compose at real-time adaptable interfaces, namely Compositions, that unify the individual UIs, which control the hardware and software components of an intelligent space. The new complex user interfaces should be deployed "on the fly", and be ready for use after been created through UInify.

User experience

Uinify should deliver a consistent and concise Look'n'feel across all devices. This will partially be achieved by UInify through a consistent Look in the Launcher components. The rest of web applications should follow the style guides that are provided by the AmI environment to provide the best user experience. The inconsistent web applications should be rejected and never reach UInify.

User empowerment

The end users should be able to easily customize and create HTML UI mashups according to their needs. This is achieved through the AmiView editor, where the user can create the layout by placing the placeholders and then fill them with web applications that exist in the ecosystem. The end user should be able to complete this task by her own, without complex instructions or the support of a programmer.

3.4.2 High-level architecture

The high-level architecture of UInify framework has been driven by the aforementioned features and functional requirements. It consists of several interconnected components with specific role in the functionality of the platform as shown in the figure 3.4 below.

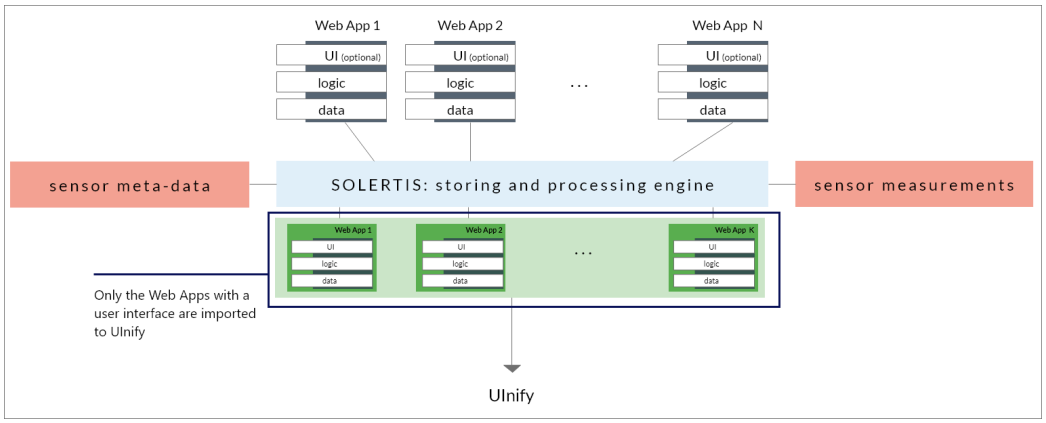

Figure 3.2: How UIs are imported to UInify

At first, all the web applications (external to UInify) that belong to the ambient environment, must pass through the AmI Solertis platform [79], where they are exposed in the environment as services. From all the services, only the web applications with graphical user interface are exposed in the UInify platform (see figure 3.3). From all the available services and web applications in the ambient ecosystem (WA_1, WA_2,...WA_N) only a subset of these (WA_1, WA_2,...WA_K)) are imported to UInify depending on whether they have or not a graphical representation. UInify when is notified about a new web application from AmI Solertis, stores a snapshot of it in the database with the appropriate attributes (as described in Table 4.2 in Section 4.1.2).

A reference to each imported web application is stored in a mongoDB database, to present it to the end user in the application repository. The published metadata of the web application will assist the correct representation of the data in the system and the proper functioning of the iframes that will form the overall UI mashup.

The main objective of the application is to create complex user interfaces (or Compositions) with the aid of two main components (see Figure 3.3). The first component is the **Layout builder**, where the end user can build the template layout for the HTML UI mashup. The layout builder is responsible to create a template for the device selected, with the appropriate dimensions for the screen. A set of rules establish that the user can create their design only in the workspace area. In future versions, with a set of rules, the system will aid the user and suggest her width and height according to web applications that exist in the ecosystem.

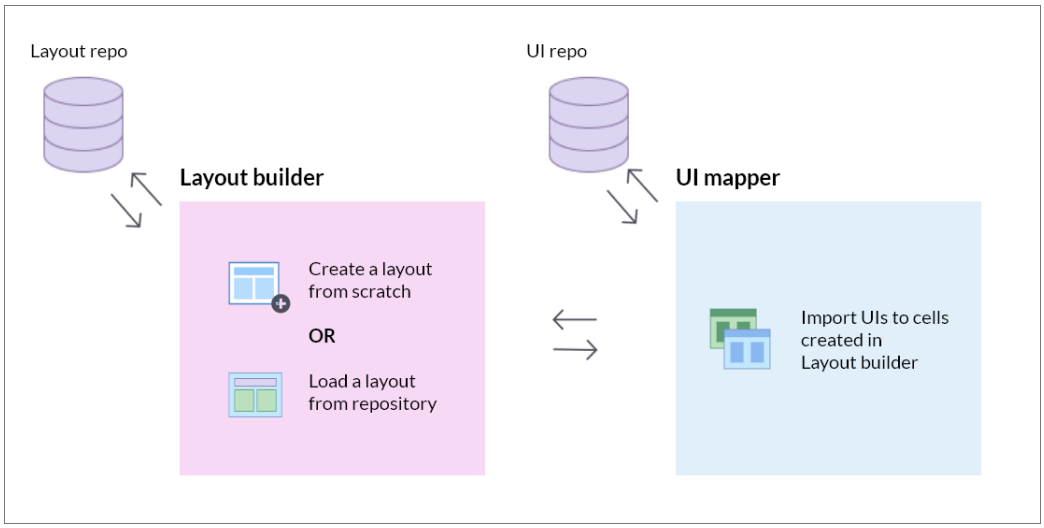

Figure 3.3: Layout builder and UI mapper

The second component is the **UI Mapper**, where the mapping of layout cell and web application occurs. In each cell that has been defined in the layout, a reference to the web application will complete the HTML UI mashups. When this composition is loaded to the screen, this cell will be inhabited by an HTML element "iframe", where it will point to the address that the web application runs. These two components are linked with each other for the correct designing and deployment of the composite user interface. UI Mapper can be extended with reasoning components in its future versions, where the system will suggest appropriate applications for the placeholders. It can utilize the artifacts display attributes as well as the context of the environment to effectively make recommendation of UIs.

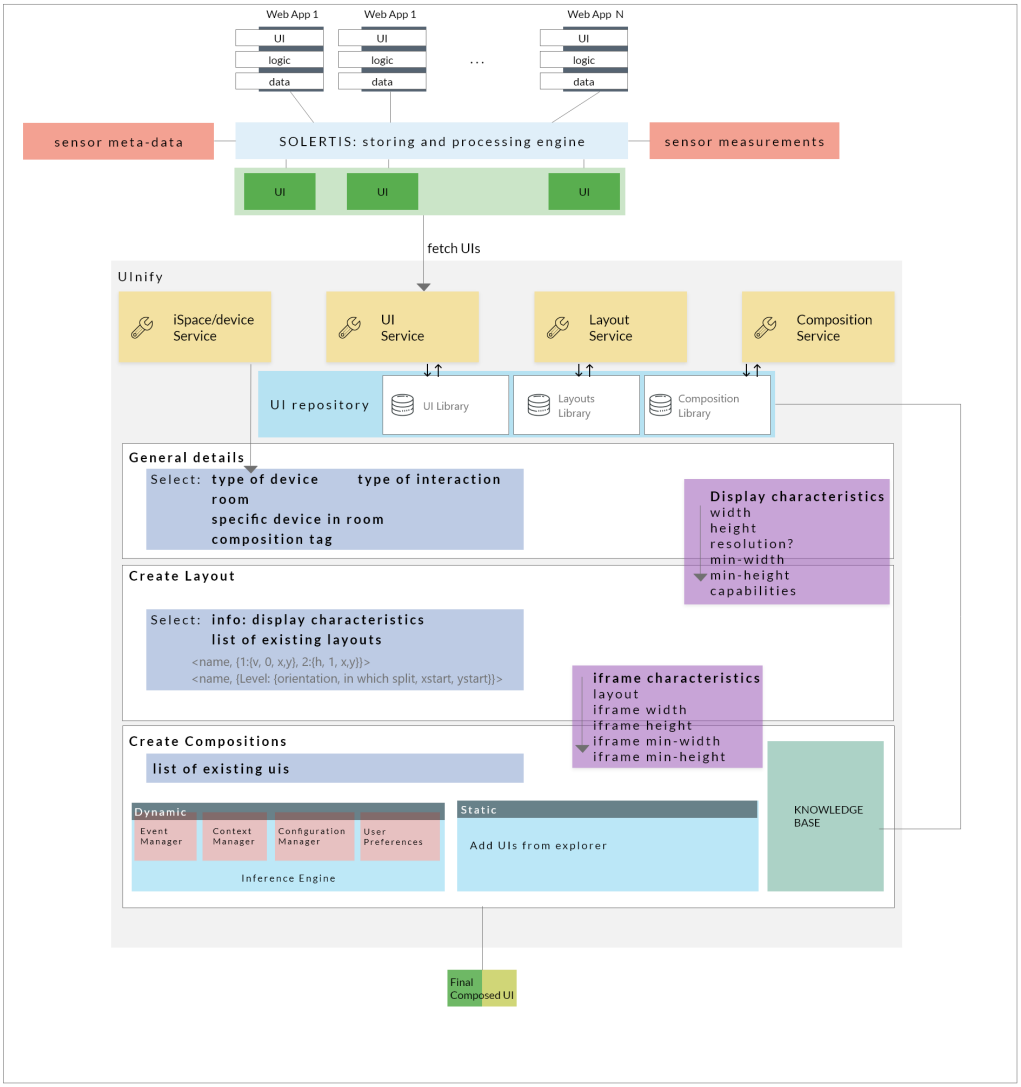

Figure 3.4: UInify overall architecture

3.5 UInify MEAN stack 2.0

UInify is using a MEAN stack 2.0 [14] architecture to develop the application. MEAN stack 2.0 is a term used for the applications that are developed with the classic MEAN stack [14] but they are using Angular 2+ for front-end development. MEAN is a free and open-source JavaScript software stack for building dynamic, fast and robust web applications. MEAN comprises of a set of four technologies stands for: (M) MongoDB [44, 107], the database that will be used , (E) Express.js [98, 8], (A) Angular 2+ [2], the client-side technology which is a JavaScript library to render the application, and finally (N) Node.js [106, 17], which is the Javascript server-side language used for the server logic.

The Overall architecture cycle in UInify platform is illustrated in 3.5

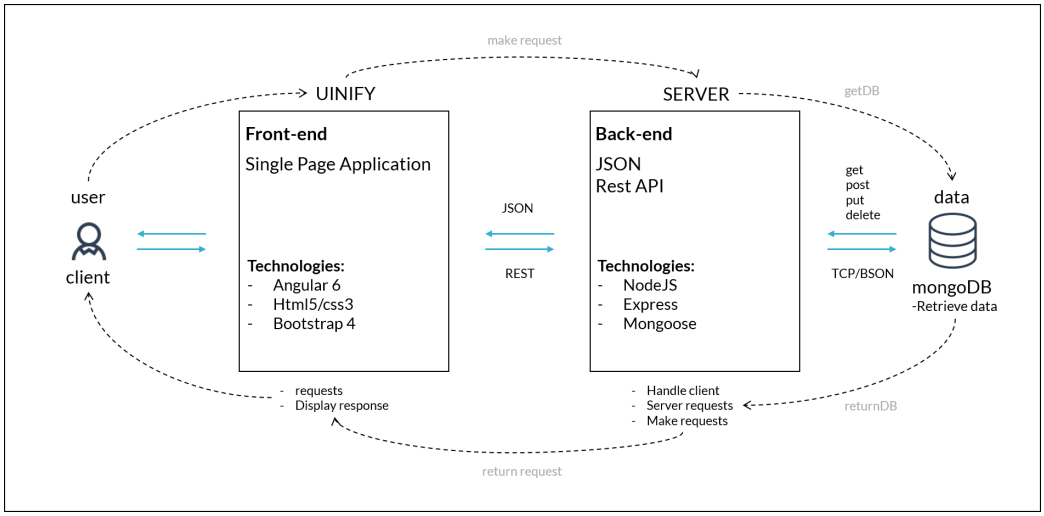

Figure 3.5: UInify MEAN stack 2.0 overall architecture

MongoDB

MongoDB [44, 107] is a free and open-source, cross-platform document-oriented database program. It is a NoSQL database program, meaning that MongoDB uses JSON documents with schemata. Also, since MongoDB supports data transfer through JSON format, data transfer from the web application is easy and economical. Moreover, JSON also allows easy client-server data transmission.

Express

Express [98, 8], is a web application framework for Node.js, released as a free and open-source software under the MIT License. It is designed for building web applications and APIs. It also provides routing, view rendering and more.

Node.js

Node.js [106, 17] is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript for server-side scripting—running scripts to produce dynamic web page content before the page is sent to the user's web browser. It is also responsible to send requests and receive responses, as well as interact with databases and files.

Angular 2+

Angular [2] is a JavaScript framework that makes it easy to build web single page applications (SPA). Angular combines declarative templates, dependency injection, end-to-end tooling, and integrated best practices to solve development challenges. It empowers developers to build applications that live on the web, mobile, or the desktop.

Single-page Application (SPA)

A single-page application (SPA) is a web application that interacts with the user by dynamically rewriting the current page, rather than loading entire new pages from a server. An Angular application only uses one route (index.html), which is served through Node.js and dynamically re-renders new context, without never requesting a second page.

UInify full stack architecture

As illustrated in 3.5, the end user interacts with the front-end components of UInify platform with a Single Page Application (SPA). This SPA behind the scenes, performs JSON requests to the backend server, through a REST API, to retrieve the appropriate information according to users requests. The server then, calls HTTP methods to retrieve the information from the MongoDB database. The database responds with the data in JSON format, through the server to the Single page application. The SPA is responsible for the display of the response to the end user.

In the next chapter, the front and back-end development process will be discussed extensively. In addition, details of the implementation will be briefly presented, as well as snippets of code, to make the development steps easy to understand.

3.6 Terminology in UInify

This section will explain the vocabulary used in UInify application. It will clarify the words adopted and in context are used.

Applications and Application Component

A major component of UInify studio is the Application Repository. The application repository is a library of the available Graphical User Interface (GUI) Applications that are available in the Intelligent spaces that are introduced in the environment.

Every Graphical User Interface (GUI) application is considered an autonomous AmI artifact provided by the technological Framework AmI Solertis [79]. The GUI applications are exposed as one Application or partial applications called Application components that can be used autonomously and in conjunction with other applications. The Application components must be part of a specific Application. Both types can be part of a mashup.

Artifacts

In UInify ecosystem every device that is installed in the intelligent environment and can display a graphical user interface, is considered an artifact. The artifacts have to provide their specifications to the system in order for correct displaying UIs and aid to the decision making process. In figure 3.6 are illustrated the artifact classes that was considered for UInify, as well as their specifications and the interaction they utilise.

Composition

In UInify context, compositions are the containers for creating a complex user interface, using one or more pluggable user interfaces. Compositions can have several amiViews, which are the template layouts created by the designer that host the interfaces.

AmiView

The user of the UInify system will have to either choose one of the existing or create a brand new amiView. The AmiViews are the template layouts where the designer have to make in order to import her application. Basically, they are the skeleton of the UI mashup.

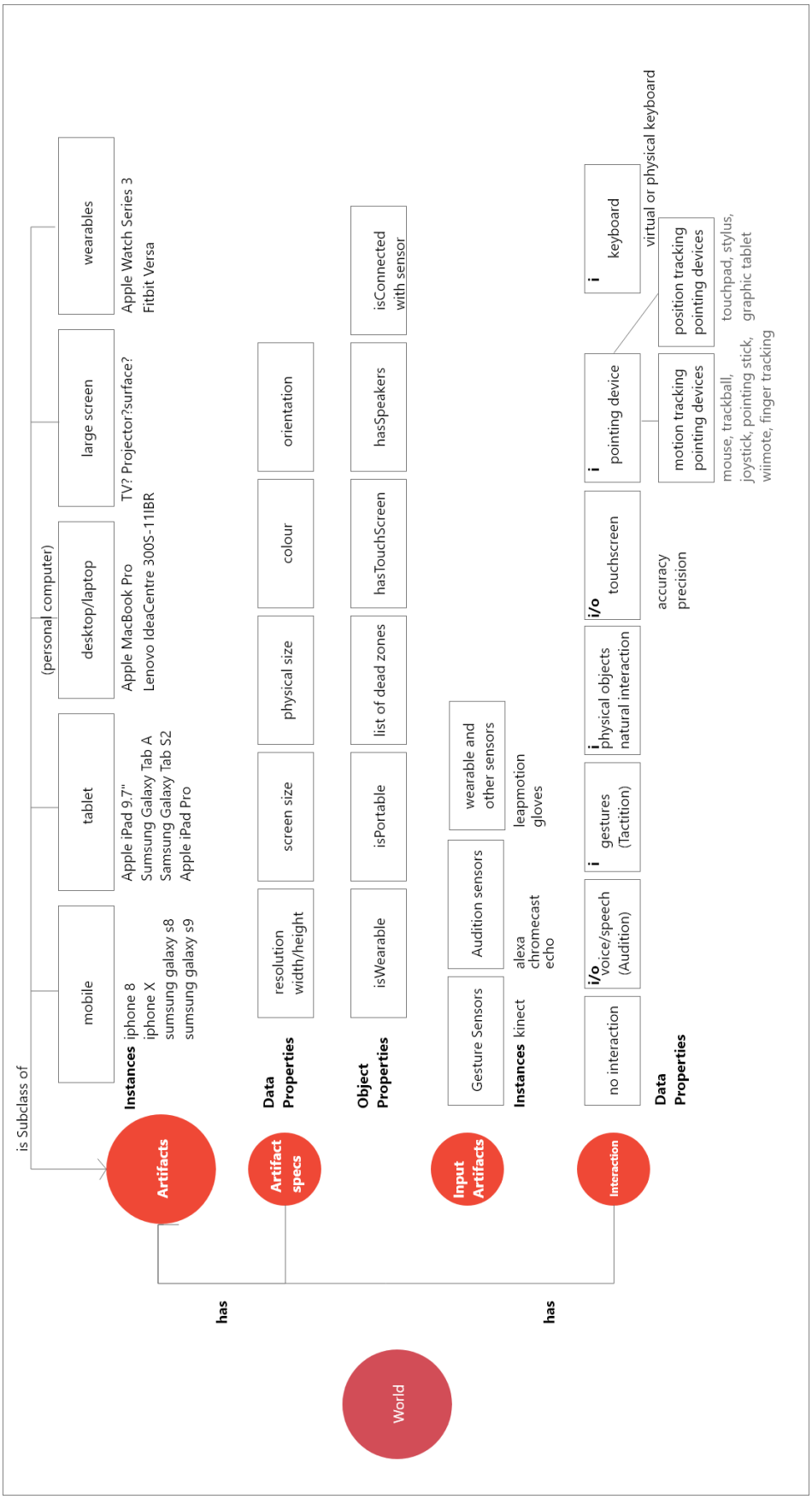

Figure 3.6: High-level definition of the artifacts in an intelligent environment

Chapter 4

System Modeling and Technological Infrastructure

4.1 Data Modeling

4.1.1 Overview

An important task for the development of UInify was to define a conceptual schema for the UInify platform. The data are stored in document [15] style (see Figure 4.1) and can be retrieved easily in JSON format. Each database contains collections which consecutively contain documents. The basic entities of UInify can be represented as collections (e.g. Composition, Application). Each document can be different with varying number of fields. The size and content of each document can be different from each other. The documents store the actual data of the application (e.g. Document for the complex user interface "Entertainment Hub" of the collection Compositions).

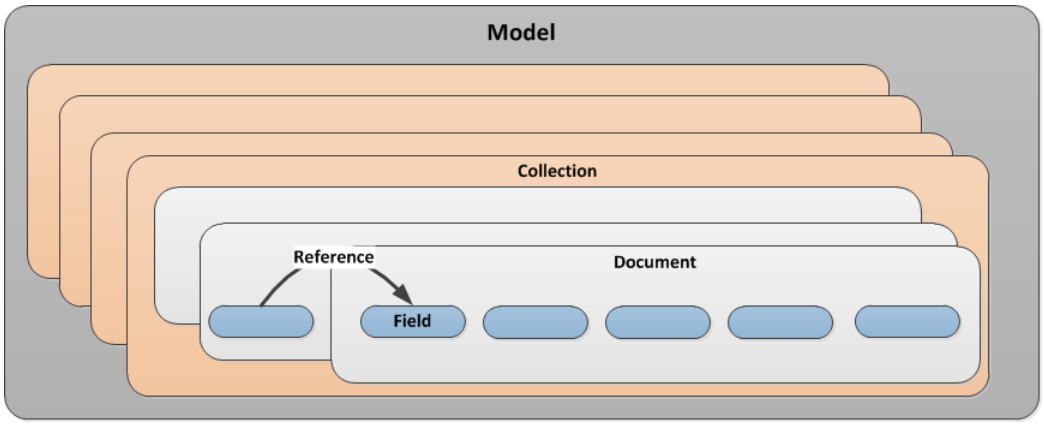

Figure 4.1: The document style mongoDB follows

UInify uses, MongoDB [44, 107] due the scalability and the freedom that gives

in modeling data and relationships between them. The schema does not need to have a defined structure beforehand and can be changed on the fly.

4.1.2 UInify entities

After the collection of functional and non functional requirements, the basic entities are extracted. Figure 4.2 shows the logical model in UML style of the database schema used.

Owner

Owner Entity describes the user of the system. She is the designer of the composite application (Composition) of the intelligent space. To use UInify platform, the owner must be registered in the system by providing a valid username, an e-mail and a password. The owner model fields are described in Table 4.1.

Field	Description
id: type: ObjectID	The unique ID of the Owner.
name: type: String, required	The user name of the Owner.
avatar: type: String	The avatar of the Owner.
email: type: String	The email of the Owner.
password: type: String	The password of the Owner (encrypted).
lastOnline: Date	When the Owner was last online.
appsCreated: type: [ObjectID]	The applications created by the Owner.
cmpCreated: type: [ObjectID]	The compositions created by the Owner.

Table 4.1: The Owner model

Application

The *Application* entity describes all the GUI components that are imported in UInify framework. All the imported applications are required to have a set of metadata in order to correctly display and eventually help in the decision-making process of the system. Each one of the applications must include some specification information to decide in which artifacts it can be displayed into (e.g. screen resolution, size), along with some additional information such as the orientation, the sensor requirement, the keyboard requirement.

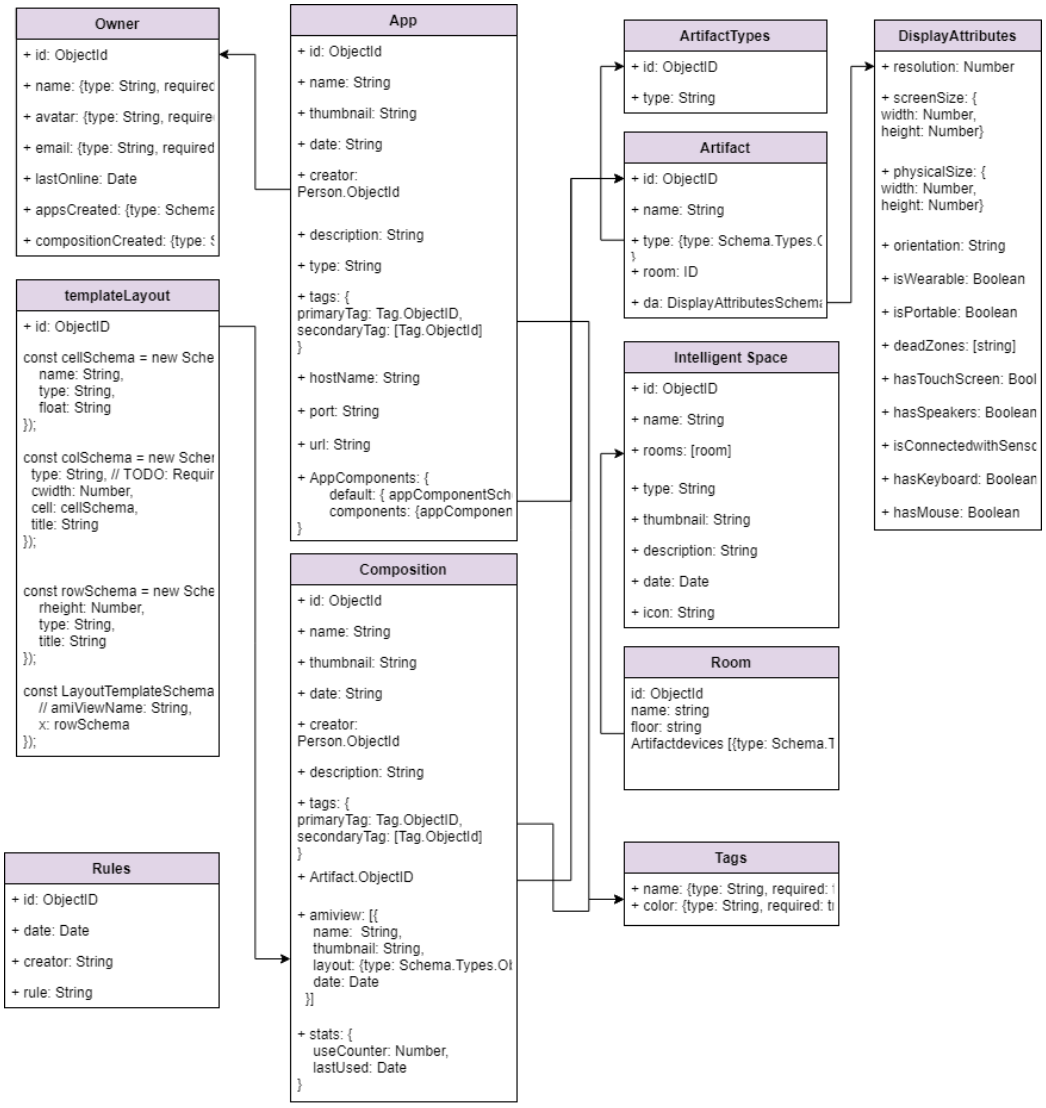

Figure 4.2: The conceptual data model of UInify platform

Field	Description
id: type: ObjectID	The unique ID of the Application.
name: type: String, required	The name of the Application.
dateCreated: type: Date	The data Application was created.
creator: type: Owner.ObjectID	The creator of the application.
thumbnail: type: String	The thumbnail of the application.
description: type: String	The description of the application.
tags: type: Tags.ObjectID	The primary and secondary tags of the application.
hostname: type: String	The hostname where the application runs.
port: type: String	The port where the application runs.
url: type: String	The url where the application runs under.
AppComponents: type: complex	The individual components the application exposes.

Table 4.2: Application model

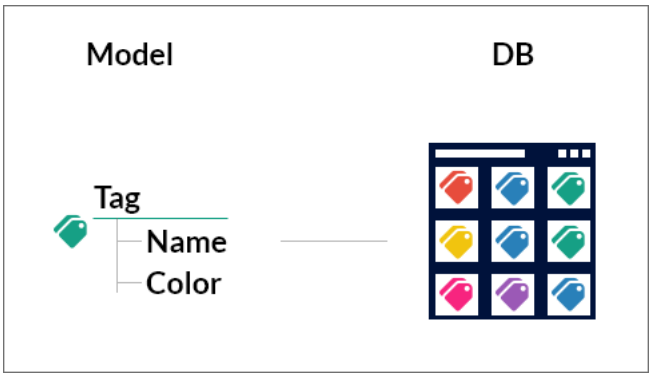

Figure 4.3: A tag as described in UInify

Tags

Additionally, the applications (and/or compositions) have tags. In information systems, a tag is a keyword that is assigned to a piece of information and helps to describe an item for searching and filtering purposes. In UInify tags are labels attached in the applications (and/or compositions) to describe the purpose and help to filter the applications. Each tag that is stored in the database has an id, a name, and a color that is used for representation reasons (see figure 4.3 and table 4.3).

Field	Description
id type: ObjectID	Unique identifier of the tag.
name type: String	The name of the tag.
color: type: String	The color (in hex) of the tag .

Table 4.3: Tag model

Figure 4.4: An application (or composition) tags representation

Moreover, every application (and/or composition) must be assigned with a primary tag (see table 4.4 and figure 4.4). Also, the end user can give some optional secondary tags (4.4), for a better description of the application (and/or composition).

Field	**Description**
primaryTag type: ObjectID	Primary tag assigned to the UI.
secondaryTags: type: [ObjectID]	The array of the secondary tags of the UI .

Table 4.4: Tags model

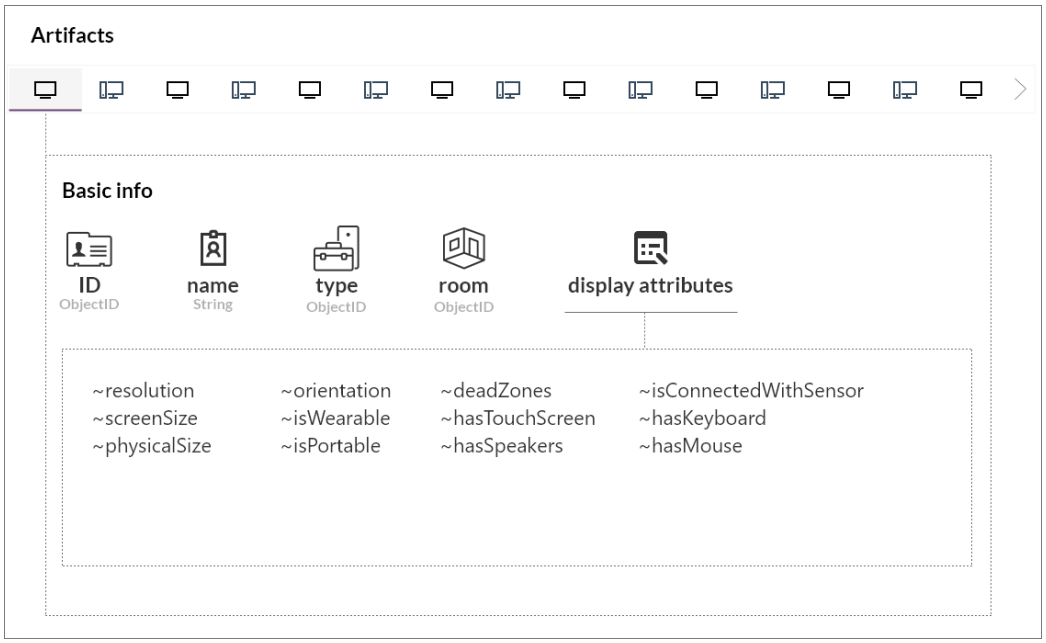

Figure 4.5: Representation of the artifact model

Artifact

The *Artifact* Entity describes the display devices (see figure 4.5) in the smart environment along with their system specifications. This entity will eventually assist in the decision making process, by indicating the applications that are eligible for the display device selected. The artifact model fields are described in Table 4.5. The specifications of display devices that were considered in this phase of the project, are displayed in Table 4.6.

Field	Description
id: type: ObjectID	The unique ID of the Artifact.
name: type: String, required	The name of the Artifact.
type: type: ArtifactTypes.ObjectID	The type of the Artifact.
room: type: Room.ObjectID	The room the Artifact is installed.
displayAttributes: (see Table 4.6)	The specifications of the Artifact.

Table 4.5: Artifact model

Field	Description
resolution: type: Number	The resolution of the Artifact.
screenSize: type: String, required	The screen size of the Artifact.
physicalSize: type: String	The physical size of the Artifact.
orientation: type: String	The orientation of the artifact in space.
isWearable: complex	Whether the artifact is wearable.
isPortable: complex	Whether the artifact is portable.
deadZones: complex	The UI zones that cannot accessed by a user.
hasTouchScreen: complex	Whether the artifact has a touch screen.
hasSpeakers: complex	Whether the artifact needs speakers.
isConnectedWithSensor: complex	Whether the artifact needs a sensor.
hasKeyboard: complex	Whether the artifact needs a keyboard.
hasMouse: complex	Whether the artifact needs a mouse.

Table 4.6: Display artifact requirements

Artifact Types

The *Artifact Types* is a sub entity that describes the types of the display devices exist in the smart ecosystem. The main displays devices that are described in UInify are: Mobile, Tablet, Desktop, and Large Screen. The types can be expanded in future versions of the system with smart watch display and a categorization of the Large Screen devices.

Field	Description
id: type: ObjectID	The unique ID of the artifact type.
type: type: String, required	The artifact type (e.g. Large Screen).

Table 4.7: The Artifact Type model

Intelligent Space & Room

The *intelligent space* and *room* describe the ambient environment and the specific parts that the display devices are installed into. One room can have multiple devices, but a specific device only dominates one room. These models are described in Table 4.8 and 4.9 respectively.

Field	Description
id: type: ObjectID	The unique ID of the Intelligent Space.
name: type: String, required	The name of the Intelligent Space.
type: type: String	The type of the Intelligent Space (e.g. Home).
thumbnail: type: String	The thumbnail of the Intelligent Space.
description: type: String	The description of the Intelligent Space.
date: type: Date	The date that the Intelligent Space was created.
rooms: type: [type: Room.ObjectID]	The rooms that the intelligent space has.

Table 4.8: The Intelligent Space model

Field	Description
id: type: ObjectID	The unique ID of the Room.
name: type: String, required	The name of the Room.
floor: type: String	The floor of the Room.
devices: type: [type: Artifact.ObjectID]	The devices that are installed in the Room.

Table 4.9: The Room model

Composition

The *composition* entity describes the unified mashup (see section 2.3.3 & 3.3) that is constructed with a set of pluggable UI components. This entity takes into consideration the required specification of the Application as well as the artifact specification in order to recommend applications. During the first step of this process, the user is asked to create a layout through an intuitive editor and then choose the UIs that can be imported into this mashup. The Composition model is displayed in Table 4.10.

Field	Description
id: type: ObjectID	The unique ID of the Composition.
name: type: String, required	The name of the Composition.
dateCreated: type: Date	The data Composition was created.
creator: type: Owner.ObjectID	The creator of the Composition.
thumbnail: type: String	The thumbnail of the Composition.
description: type: String	The description of the Composition.
tags: type: ObjectID	The primary and secondary tags of the Composition.
artifacts: type: ObjectID	The artifact that the UI will be displayed into.
amiView: type: complex	The layouts that was created for the composition.
stats type: complex	Several statistic values that are used into the UInify.

Table 4.10: The Composition model

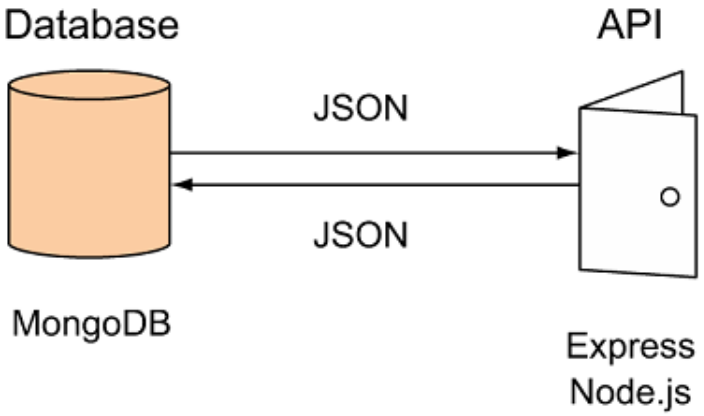

Figure 4.6: Express and Node.js for data retrieval from MongoDB database

4.2 UInify Backend technologies

UInify is using a MongoDB database to pass data around. Figure 4.6 shows the starting point, a REST API built with Express and Node.js, which is used to enable interactions and retrieve data from the database. Express uses typical HTTP methods as described in table 4.11, in order to get, add or delete the requested data.

Method	**Description**
GET	Retrieve requested data from a specified resource.
POST	Send data to a server to create or update a resource.
PUT	Send data to a server to create or update a resource.
DELETE	Deletes the specified resource

Table 4.11: Typical HTTP methods used in UInify

In order to follow a Model-View-Controller (MVC) architecture in the REST API, the presentation of the model and the controlling methods had to be separated. As the entities and the relationships between them have been established in the database conceptual model presented previously, the mongoose models can be created, which is the structure of the data in the databases' document.

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const compositionSchema = new Schema({
  name: {type: String, required: true},
  thumbnail: {type: String, required: true},
  date: Date,
  creator: {type: Schema.Types.ObjectId, ref: 'owner'  },
  description: String,
  artifact: {type: Schema.Types.ObjectId, ref: 'artifact', required:
     true },
  tags: {
    primaryTag:{type: Schema.Types.ObjectId, ref: 'tag' },
    secondaryTags: [{type: Schema.Types.ObjectId, ref: 'tag' }]
  },
  amiview: [{
    name:  String,
    thumbnail: String,
    layout: {type: Schema.Types.ObjectId, ref: 'layout_template' },
    date: Date
  }],
  stats: {
    useCounter: Number,
    lastUsed: Date
  },
});

module.exports = mongoose.model('composition', compositionSchema, '
    composition')
```

Listing 4.1: Composition model

Models are defined using the Schema interface defined by mongoose [15]. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection. Schemas not only define the structure of the document, but also define document instance methods, static Model methods and document life-cycle hooks called middleware. By using middleware, such as save, delete or find, which are functions that have control during execution of asynchronous functions, mongoose can communicate with the database.

With the aid of models, controllers that fetch the data from database can be defined. A sample controller for fetching all the Compositions from the database is presented in Listing 4.2. This logic will be executed when requests received in "api/compositions".

```
exports.getCompositions = (req, res, next) => {
  Composition.find()
  .populate("creator")
  .populate({path: "amiview.layout", model: "layout_template"})
  .populate({
    path: 'artifact',
    populate: {
      path: 'room',
      model: 'room'
    }
 })
 .populate({
  path: 'artifact',
  populate: {
    path: 'type',
    model: 'artifact_types'
  }
})
  .then(documents => {
    console.log(documents);
    res.status(200).json({
      message: "Compositions fetched successfully!",
      compositions: documents
    });
  });
};
```

Listing 4.2: HTTP request to mongoose for all the Compostions

Server

The server.js (see Listing 4.3) executes the node.js server in a specific port chosen by normalise port function (usually port 3000). Server listens to this port for HTTP requests.

```
1  const app = require("./backend/app");
2  const debug = require("debug")("node-angular");
3  const http = require("http");
4
5  const normalizePort = val => {
6    var port = parseInt(val, 10);
7
8    ...
9  };
10
11 ...
12
13 const onListening = () => {
14   const addr = server.address();
15   const bind = typeof addr === "string" ? "pipe " + addr : "port " +
       port;
16   debug("Listening on " + bind);
17 };
18
19 const port = normalizePort(process.env.PORT || "3000");
20 app.set("port", port);
21
22 const server = http.createServer(app);
23 server.on("error", onError);
24 server.on("listening", onListening);
25 server.listen(port);
```

Listing 4.3: server.js

Any node web server application will create a web server object. This is done by using createServer function. the Server object returned by createServer function is an EventEmitter. When an HTTP request hits the server, node calls the request handler function in order to deal with the transaction with request and response objects. The main methods that handle these requests developed in the file app.js (see Listing 4.4).

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");

const compositionRoutes = require("./routes/composition");
...
const layoutsRoutes = require("./routes/layout");

const app = express();

...

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept, Authorization"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PATCH, PUT, DELETE, OPTIONS"
  );
  next();
});

app.use("/api/compositions", compositionRoutes);
...
app.use("/api/layouts", layoutsRoutes);

module.exports = app;
```

Listing 4.4: Bundling app in app.js

Bundling app.js

Model, Controllers and routes are bundled together in app.js file (see Listing 4.4). This file holds the Express app by taking advantage of its features. Furthermore, it defines the paths used and forwards the requests in the correct route. In addition, in this file we handle the Cross-Origin Resource Sharing error, that is caused by the different hosts that the client and the server are into. To resolve this we add the Headers of "Access-Control-Allow-Origin", "Access-Control-Allow-Headers" and "Access-Control-Allow-Methods" in our Http requests to allow several extra features in the application.

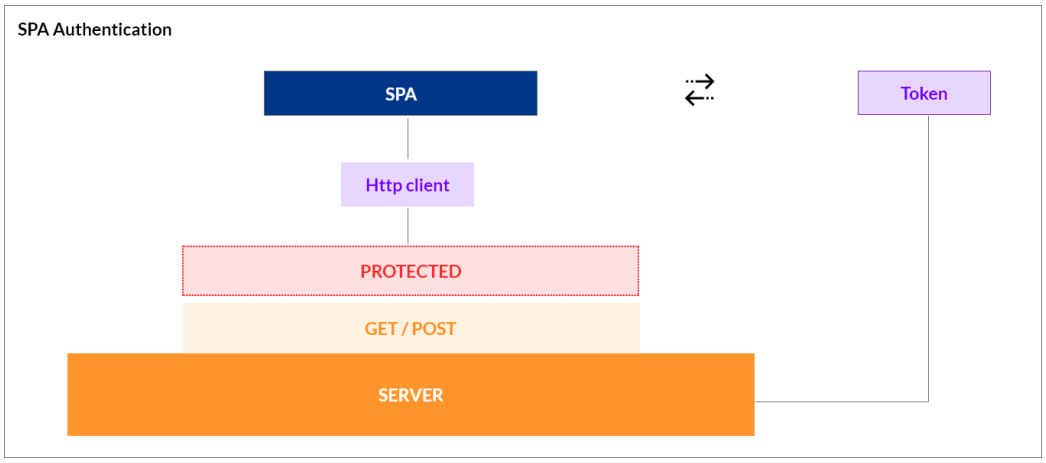

Figure 4.7: SPA authentication with jwt tokens

User authentication

A simple user authentication component was used in order to restrict the access in UInify. For this, a simple register and login form was created to store users and let them sign in the system. The system in order to securely store the password of the user, uses bcrypt [4]. BCrypt is a JavaScript library which utilise a password hashing function. It incorporates a salt to protect against rainbow table attacks and remains resistant due to its mechanisms to brute force attacks.

In UInify the backend (node and angular) are stateless, so to achieve authentication we have to use Javascript Web Token [12] as represented in the figure 4.7. This token is generated in the server upon a successful login. That token is sent back to the browser, where it can stored in the angular app (in the local storage).This token is attached in all request (in the request header) and the application can identify the user in that way. The validation and the creation of the token is only possible in the server. Other request that does not have this token will be rejected.

4.3 UInify Frontend technologies

Today, front end frameworks and libraries that support front-end development are increasing vastly. Angular which is designed to work with data directly in the front end while using HTML as a template language, is one of them. Angular CLI, the command line interface for the Angular makes it easy to create an application that already works, right out of the box, while following the best practices.

The user interacts with a Single Page Application (SPA). On a SPA, after the initial page load, no more HTML gets sent over the network. Instead, only data gets requested from the server (or sent to the server). Therefore, SPAs increase User Experience by minimizing page reloads and the amount of bandwidth

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Observable } from 'rxjs';

const envPath = 'http://localhost:3000/api/';

@Injectable({providedIn: 'root'})
export class ApiService {

  constructor( private _http: HttpClient) {}

  get(path: string, params: HttpParams = new HttpParams()):
    Observable<any> {
    return this._http.get(envPath + path, { params });
  }

  put(path: string, body: Object = {}): Observable<any> {
    return this._http.put(envPath + path, JSON.stringify(body));
  }

  post(path: string, body: Object = {}): Observable<any> {
    return this._http.post(envPath + path, body);
  }

  delete(path): Observable<any> {
    return this._http.delete(envPath + path);
  }
}
```

Listing 4.5: Rest method request service

required.

REST method requests service

One of the features of angular are services, where we can enable the communication with the back end with REST method requests. The service that aids with the communication is shown in the snippet below.

This service handles all the requests from all the individual services requesting data, and forwards them to backend which returns the appropriate JSON object for the request.

Chapter 5

UInify platform

This chapter outlines the details of the various front-end components that have been implemented in the context of UInify. Next, it will report an overview of the provided functionality (i.e. Dashboard, menus) as well as the main components that the designers interact within the UInify platform, to create UI mashups. The main components are: (i) **Dashboard** (ii) **Composition Repository and Details**, (iii) **Application Repository and Details**, (iv) **AmiView editor**. Additionally, snapshots of the underlying functionality will be provided to adequately illustrate the interaction with the user.

5.1 Overview

The UInify platform has been designed as an online system supporting intelligent space designers in delivering complex user interfaces in ambient environments. UInify aims to offer a simple and intuitive user interface which designers can easily interact with. Otherwise stated, the ultimate task is to reduce the complexity and the time that is required to accomplish a design task for a smart environment. This section will provide information about the basic utilities of the system, namely: (i) **Login** and **Register**, (ii) **Menus**, (iii) **Dashboard**, that the user interacts with. In the following sections, the functionality of the UInify platform, accessible only by **registered** users, will be described in detail.

5.1.1 Register and Login

End Users may register to the system in a secure way by selecting the registration option from the landing page and providing the following information: a username, an email and a password. Using a similar form (see Figure 5.1), the user must log in to the system to have access to its functionality.

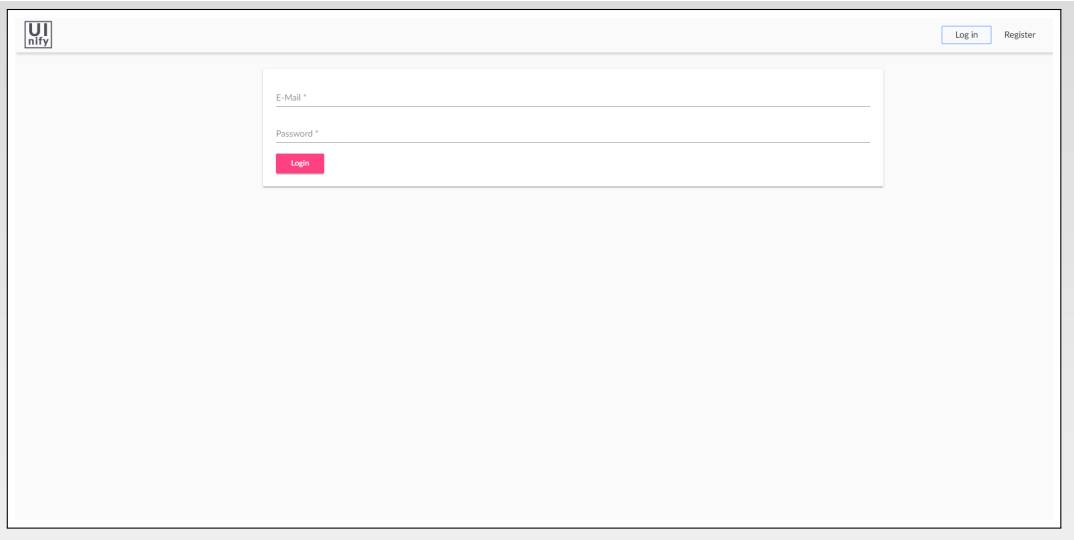

Figure 5.1: The login form

5.1.2 Dashboard

The end user is able to navigate to a user-friendly interface, that is a visual display of data used to monitor and aid in the understanding of the environment. Furthermore, the UInify platform provides an interactive display that allows users (i.e., designers) to explore: (a) information about their Compositions (i.e., how many exist in the ecosystem) (b) information about the graphical user interface (GUI) applications that are imported in the environment (see Figure 5.2).

In more detail, the end users can view some metrics concerning the current state of the ambient environment whose data are presented in their personalized dashboard, such as: the number of: (i) active *applications* that are exposed from AmI Solertis, (ii) the total UI *rules*, user has defined and being used from the decision making component, (iii) the available *compositions* that have been created from UInify and (iii) finally, the number of *intelligent spaces* UInify is controlling.

On the bottom left side, user can browse through some popular applications and application components. Firstly, the **popular user interfaces**, that as the name indicates are the GUI applications that have been used the most from the end-users to create compositions and secondly, the **recently created user interfaces**, which are the applications that have been recently used in a composition. The user can interact with both of these components through a carousel component, and select them to view their details (see section 5.2.2).

On the right side of the page, the user can explore the recent compositions that have been introduced into the system, in the shape of the table. Every row starts with the **name** of the composition (i.e. AmiTV Launcher), the **date** that the composition was created, the **username** of the creator of the composition, the

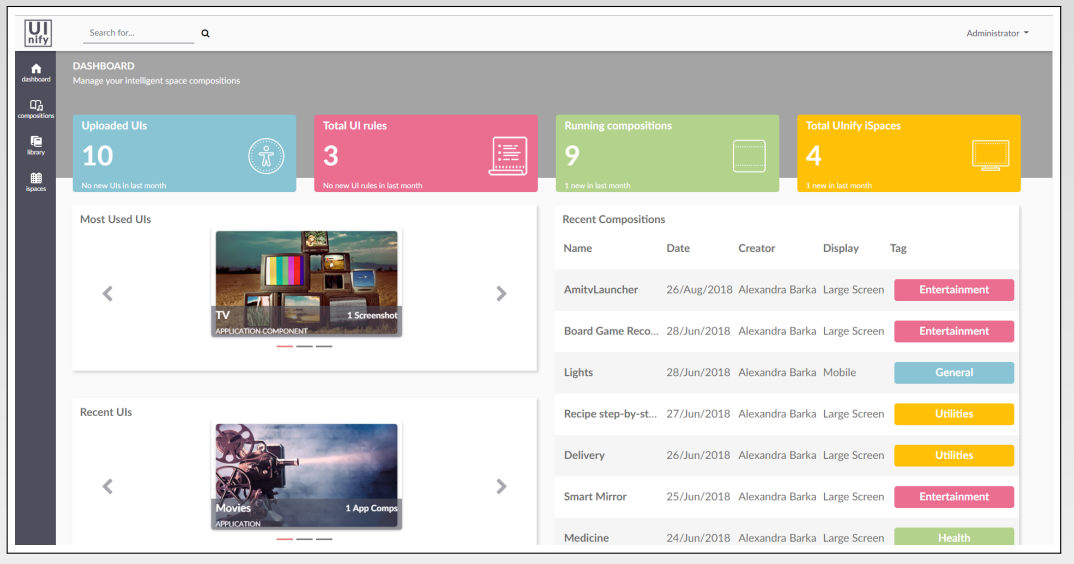

Figure 5.2: The UInify's platform dashboard

display device that the composition is designed for and finally the compositions' **primary tag**.

5.1.3 Main and secondary menus

The system offers an intuitive user interface that categorizes the available functionality of the main components under four menu items in the Dashboard menu (see Figure 5.3), namely: (i) **Dashboard** (ii) **Compositions** (iii) **Library** (iv) **Ispaces**.

Figure 5.3: The UInify's main dashboard menu

Figure 5.4: The UInify's secondary menu

Dashboard

A dashboard with a visual display of statistics and general information about the system.

Compositions

A repository with the compositions which have already been deployed in the system. The end user can explore the available and create new compositions through this page.

Library

The applications' repository, which displays the graphical user interfaces that have been imported to the system.

Ispaces

The intelligent spaces repository, which displays the ambient intelligent spaces that have been imported to the system.

Furthermore, the system provides a secondary top menu (Figure 5.4) where the user can perform some **auxiliary** actions such as *searching*, browsing her *profile* or *logging out* the platform.

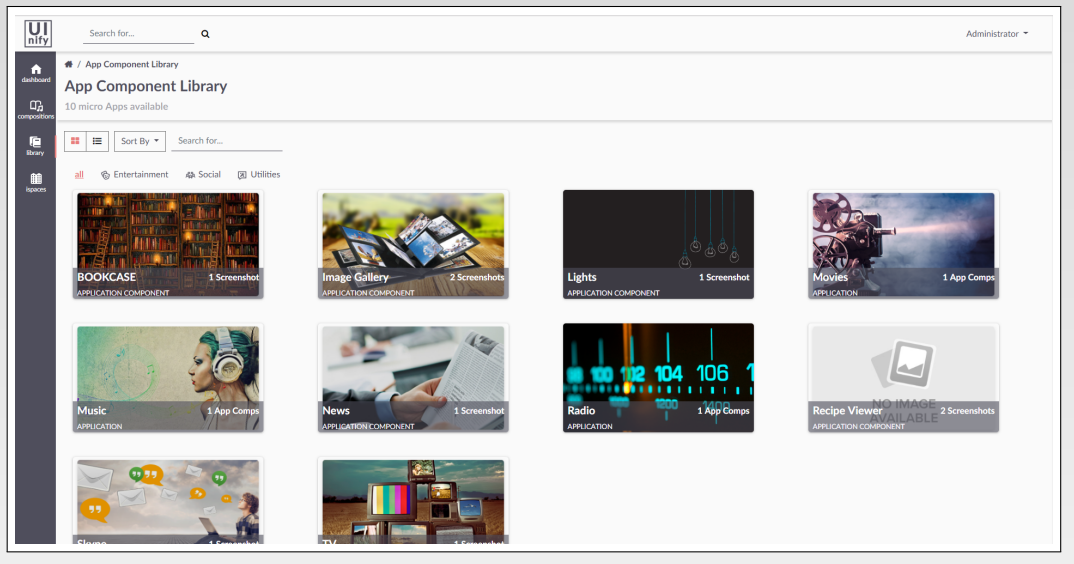

Figure 5.5: The application repository in tile view

5.2 Applications

The applications are the components identified as a graphical user interface and have been imported to the system. To that end, UInify provides an exploration tool to permit to users to browse through the list of available applications (i.e., AMI TV, Recipe step-by-step). For each application, the user can view additional information and screenshots of the available components.

Generally, the GUI applications fall under two categories, the **"Applications"** and **"Application Components"**. The "Applications" (see section 3.6) category , can expose several components, namely "Application Components" (see section 3.6) that can be used independently from the default application (e.g. the Music Player component of the Music application). To that end, under the "Applications" items the end user can browse and use in her composition all the "Application Components" that are exposed from the application's API.

5.2.1 Application Repository

The end user is able to navigate through the applications and interact with the introduced GUI in two different views - *tile* and *list* view - by selecting the desired view from the filters menu (see Figure 5.7). For each item of the application repository, in the tile view, a small characteristic image is provided along with the important information, namely: the name of the application (or application component), the number of screenshots and the type of the application (see Figure 5.5). In the list view, a short description of the application is additionally provided (see Figure 5.6).

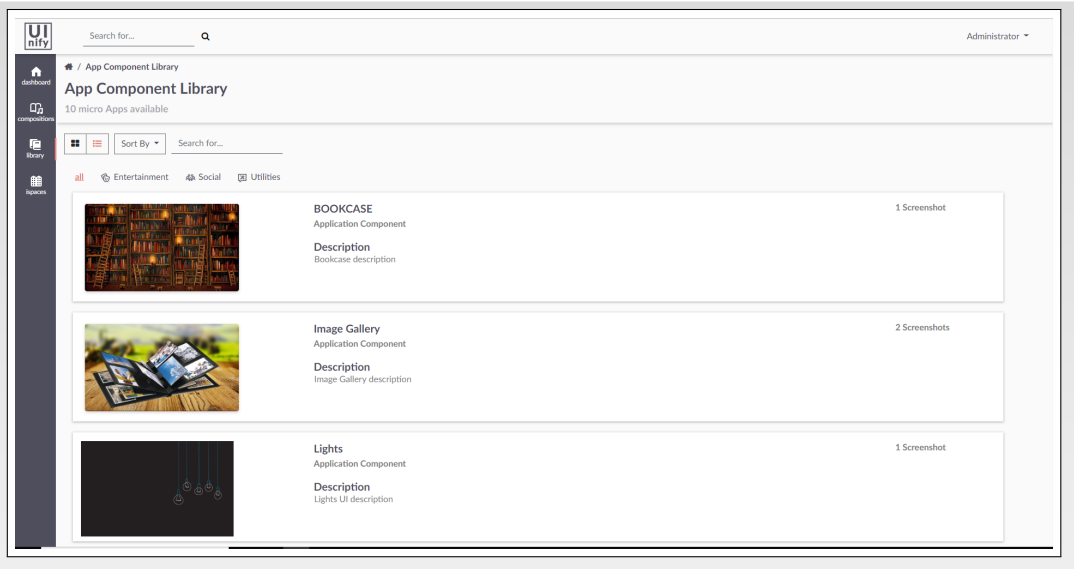

Figure 5.6: The application repository in list view

Furthermore, in the filter menu several sorting and filtering tools are provided (see Figure 5.7) for a more refined selection. The end user can sort all the available applications alphabetically, by date or by usage. Also, the system provides a search feature for a quick query of the desired application. Finally, the user can filter a tag or category, which means that she can select the category under which some application fall into.

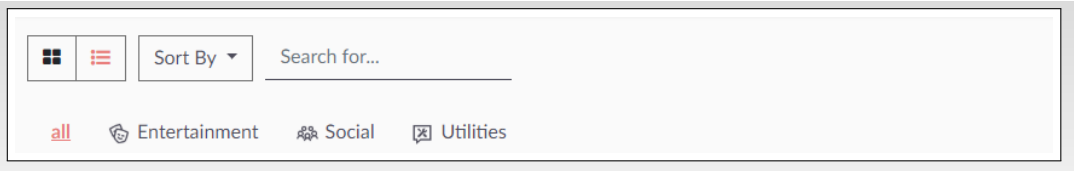

Figure 5.7: The available sorting and filtering components for application repository

5.2.2 Application details

By selecting a specific item of the list, the user is able to view additional information regarding this item including a detailed description of the item and the creator of the item (see Figure 5.8). In the top bar the end user can view the Title of the item as well as the associated tags.

After the top bar, there is a section, where the details of the application (or application component) are displayed. The detail component includes a detailed description, the creator and the date created. Below details, there is a section with the available application components (if any exist).

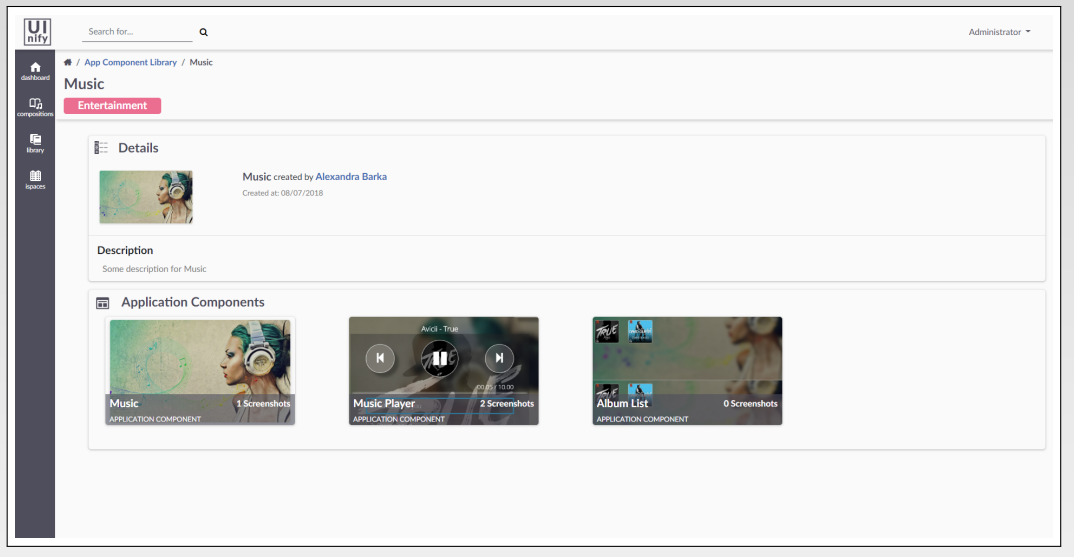

Figure 5.8: The "Music" Application has two application components

In the "Application Components" section, the end user can see screenshots (see Figure 5.9) from the component in use. In the future, the system will provide a demo link where the end user can have the user experience of the exposed component to decide whether to use it or not.

5.3 Compositions

An important requirement of the UInify platform is the introduction of new complex user interfaces in the intelligent space. Towards this goal, a set of UI components was created to to assist in that process, namely the (i) **Composition repository**, (ii) **Composition details**, and (iii) **Composition introduction**, which provides the functionality to introduce a new amiView (see next chapter).

5.3.1 Composition Repository

The end user is able to navigate through the compositions repository. The compositions are the complex user interfaces that have already been created with the tool by all registered users. The end user can view the HTML UI Mashups in two different views - tile and list view - by selecting the desired view from the filter menu. For each item of the composition repository, in the tile view, a small characteristic image is provided along with some important information (i.e. display device, room) (see Figure 5.10). In the list view, a short description of the application is additionally provided.

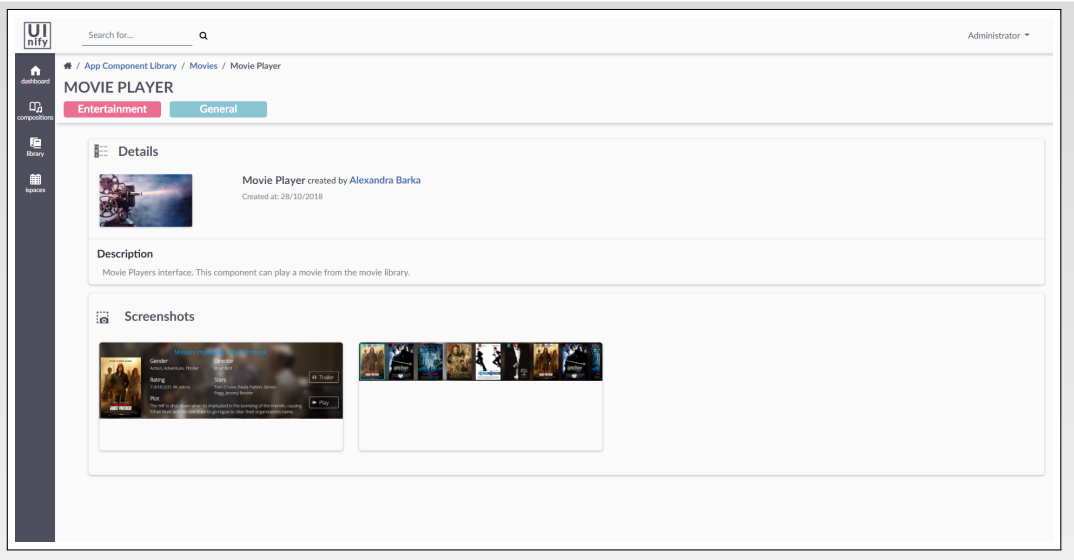

Figure 5.9: The "Movie player" Application Component as displayed in Uinify

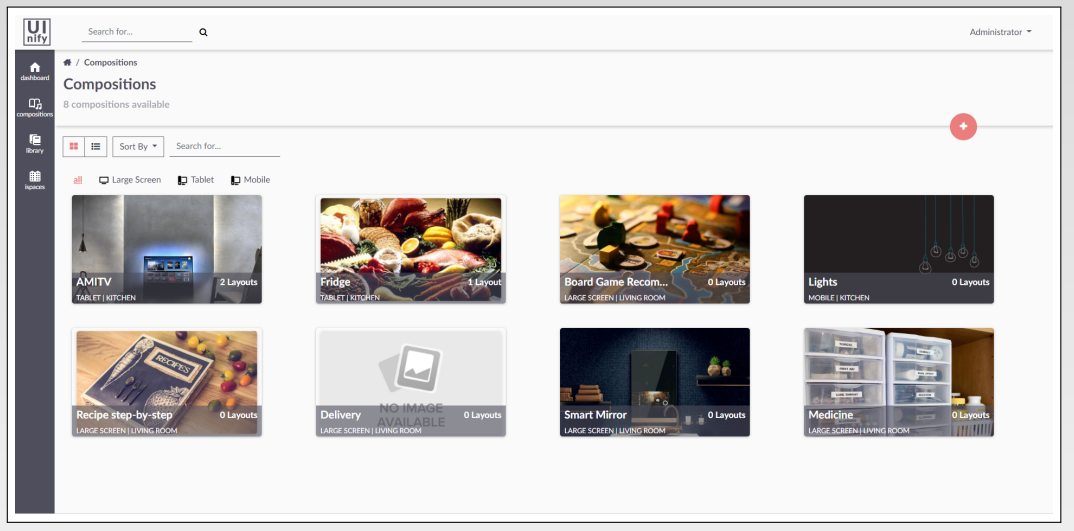

Figure 5.10: The Composition repository - tile view

For each item of the composition repository, in the tile view, a small characteristic image is provided along with some important information (see Figure 5.10). In the list view, a short description of the application is also provided. Finally, a similar filtering component as in the application repository user interface exist, which helps the user browse the compositions easily by sorting or/and filtering the list for a more refined selection.

5.3.2 Composition Details

By selecting a specific composition item from the list, the end user can view some additional information regarding this item (see Figure 5.11).

Figure 5.11: The AMITV composition details

Additionally, the selected composition item provides a list of the available layouts, namely AmiViews (see Figure 5.11), where the user can browse the different template layouts that have been created for the specific composition. Finally, the end user can create a brand new template layout, see the details of an existing one or delete the selected composition through this detailed interface.

5.3.3 Introducing a composition

The end user can create a new composition through an intuitive interface. The new composition can be introduced from the composition repository by clicking the on the top right button (see Figure 5.10).

After pressing this button a pop-up dialogue appears and prompts the user to select her desired options. Firstly, the user should select the Intelligent Space (see Figure 5.12) that the composition is intended to used for. In this user interface, all the intelligent spaces that are imported in UInify are displayed into a tile view.

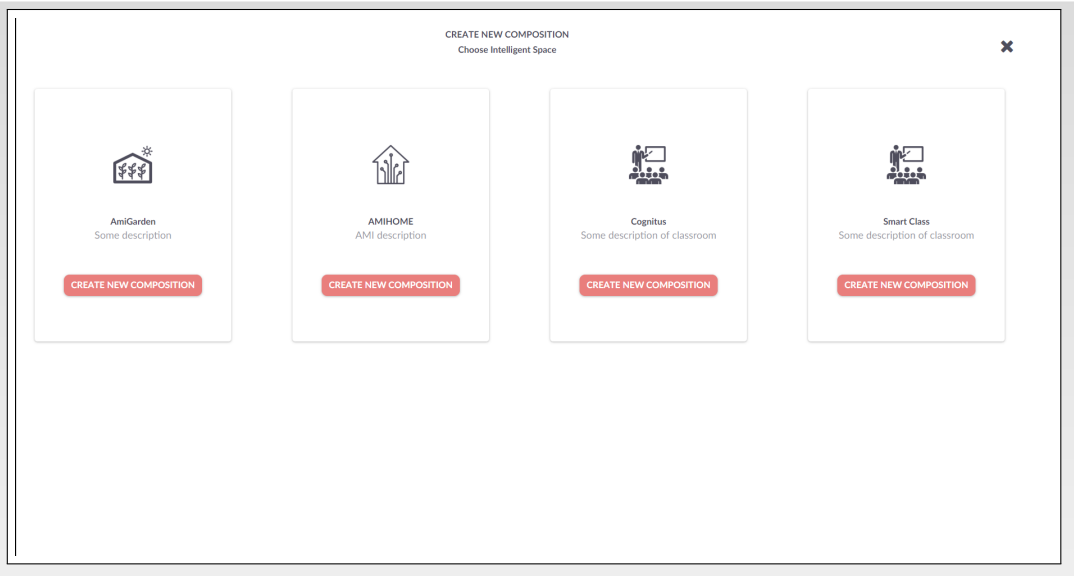

Figure 5.12: Choosing between intelligent spaces for the composition

When the end user selects one of the available intelligent space that has been imported to the system (i.e. AMIHOME), she redirects to the second step of the process, which is to select a room from the intelligent space, where the composition is going to be displayed into. This step is crucial to define the context and the appropriate web applications that can be displayed. Without this step, the system wouldn't be able to identify the room and the activities that take place there, to offer additional functionality by the reasoning component.

In the final step, the end user has to select a name, a primary tag which is mandatory and describes the type of the composition, an optional secondary tag and the specific artifact that the composition is intended to display into. Finally, the user has to click the Create Composition button and her choices will be saved into the database.

5.4 The AMIview editor: Introducing a layout

The core feature of the UInify platform is introducing a new **layout** to a composition which will be filled with web applications. UInify considers as a layout, the template user interface structure where the user can import the web applications into. When the application is deployed, it will be displayed in an HTML iframe tag with the properties that the user has defined in that step. The application-iframe mapping in the layout is called, in the UInify context, an **AmiView**.

In the details of a specific composition, the end user can find a list with the available amiViews, that already have been introduced to the system (see Figure 5.13). The user can create a new amiView, either by clicking the Create Static Layout or the "+" button at top of the screen.

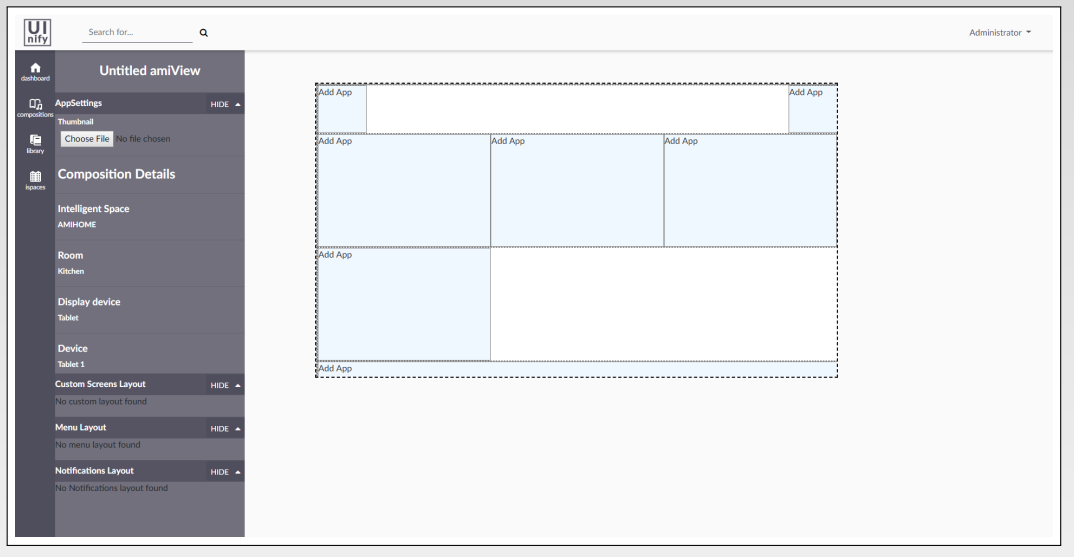

Figure 5.13: The UInify's editor

Furthermore, by selecting a specific amiView from the list, she will be redirected to the amiView editor (see Figure 5.13). Here, in this preliminary version of the editor the user can (i) create a template and (ii) add functionality in each layout.

Creating a new template layout

In order to create a new AmiView, the user presses the *Create a static layout* button and redirects to the editor. An empty layout, with the aspect ratio of the screen that the end user chose (scaled down by two), is then created (see Figure 5.13).

The end user can create rows and columns by setting the height and the width respectively. When the user finishes the layout, she can save it as a template and

proceed to the next step where she can add web applications in the cells that are created. The general flow of creating a composition is depicted in figure 5.14.

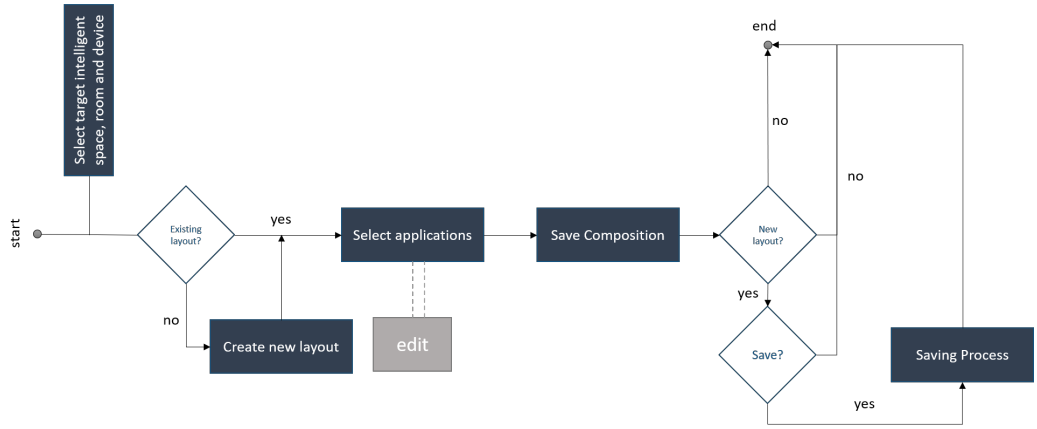

Figure 5.14: The flow of creating a new composition

In the future versions of the editor, the edit functionality will be implemented, where the user can edit an existing amiView, delete its cells or adjust the width and height of existing ones.

5.5 Utilities

In this section, some extra features that the UInify tool exposes will be described. These features are: (i) **Intelligent Space Repository** (ii) **Intelligent Space Details** (iii) **User Profile**, and (iv) **Universal Search**. These are additional information that can aid the user in the management of the smart environments. Intelligent Space repository can offer an overview of the compositions that already has been deployed in the space. The profile can offer an overview of the users' activities in the system and finally with the search component the user can query keywords in the system.

5.5.1 Intelligent Space Repository and Details

Intelligent Space Repository

The end user is able to navigate through the intelligent space repository, which contains the ambient spaces that have been imported to the UInify system. The end user can view the repository in two different views - tile (see Figure 5.15) and list view - by selecting the desired view from the filter menu.

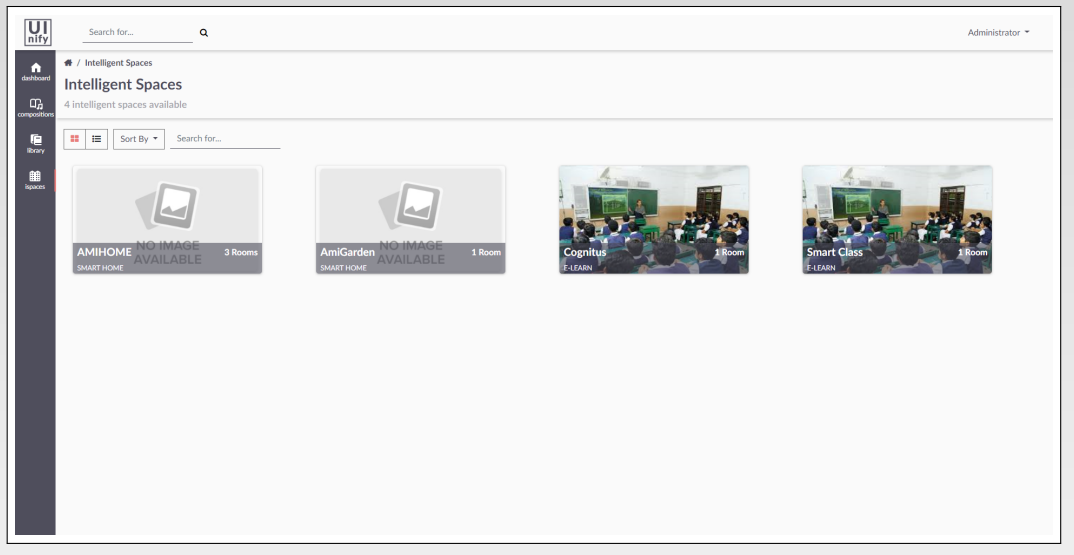

Figure 5.15: The Intelligent Space repository - tile view

Intelligent Space Details

Moreover, by selecting a specific item from the list, the end user can view some additional information regarding this space, including the ambient rooms and the number of composition on each room.

5.5.2 Profile

The user can see her profile, by pressing the “Profile” button in the dropdown menu in the top bar. In this page, the user can view details about their status and details, as well as the applications and compositions they created (if any).

5.5.3 Universal Search

The user is provided with a universal search component, where she can query the available applications, compositions and intelligent spaces.

Chapter 6

UInify Use Cases

The scope of this chapter is to demonstrate use cases where UInify can be successfully utilised, instead of developing a brand-new application from scratch. Firstly, this chapter will show the steps needed to build a multimodal and intelligent multimedia hub through the UInify application. Next, it will report two use cases: a) one to design an interactive application for the kitchen countertop and (b) the use in monitoring the home from the home screen of a smartphone. Finally, it will illustrate a use case outside the smart home environment, in the education domain, and how the functionality of the current system can be expanded to other areas, with presenting the implementation of CognitOS [90] application through UInify.

6.1 AMITV Launcher

6.1.1 Requirements

The designer wants to create an application for the living room TV that will serve as a multimedia hub. This application will provide access to movies, TV, music, news and Image Gallery, through a launcher. Each one of the links (e.g. music, TV) in the launcher, will provide access to a full view of the respective component.

- **Intelligent Space:** AMIHOME
- **Room:** Living Room
- **Artifact:** Large Screen (TV1)
- **Primary Tag:** Entertainment, General

6.1.2 Creating the launcher

In order to create the new UI mashup, firstly the designer should create a composition or utilize an existing one. For this example, the designer will create a brand

new composition by clicking the appropriate button in the composition repository, namely 'AMITV Launcher'. The user will be redirected to a pop-up dialogue, where she can fill in the basic information. Firstly, she will choose as an Intelligent space the AMIHOME environment. In the next dialogue, she will choose the living room as the room where the composition is intended to be used into. Then, the name (i.e. AMITV Launcher), the tags (i.e. Entertainment, General) and the Artifact (i.e. Large screen, TV1) must be filled in. The entire process is illustrated in Figure 6.1.

In the editor, the designer customizes the layout by adding appropriate placeholders as desired; she can create the grid, by adding rows and columns (as many as the chosen display device can fit) and defining their height and width respectively. After completing the template she can save it to template repository, to reuse it in other compositions. Next, she populates the placeholders with concrete application components from the application repository. Finally, she can save the repository in the database.

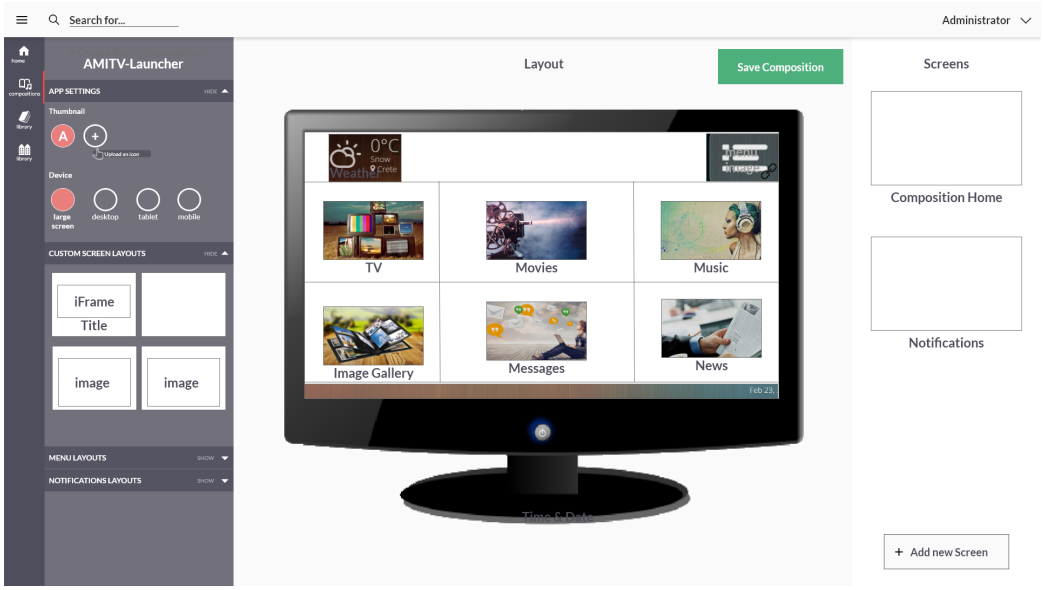

Figure 6.2: A concept layout for "AMITV Launcher"

6.2 Mobile Home Screen

6.2.1 Requirements

The designer wants to create a mobile application that will assist the user by providing information and quick access to home control functionality. This application will provide information about the status of the home, and more specifically about the status of the security and the safety components (e.g. fire alarm). Moreover,

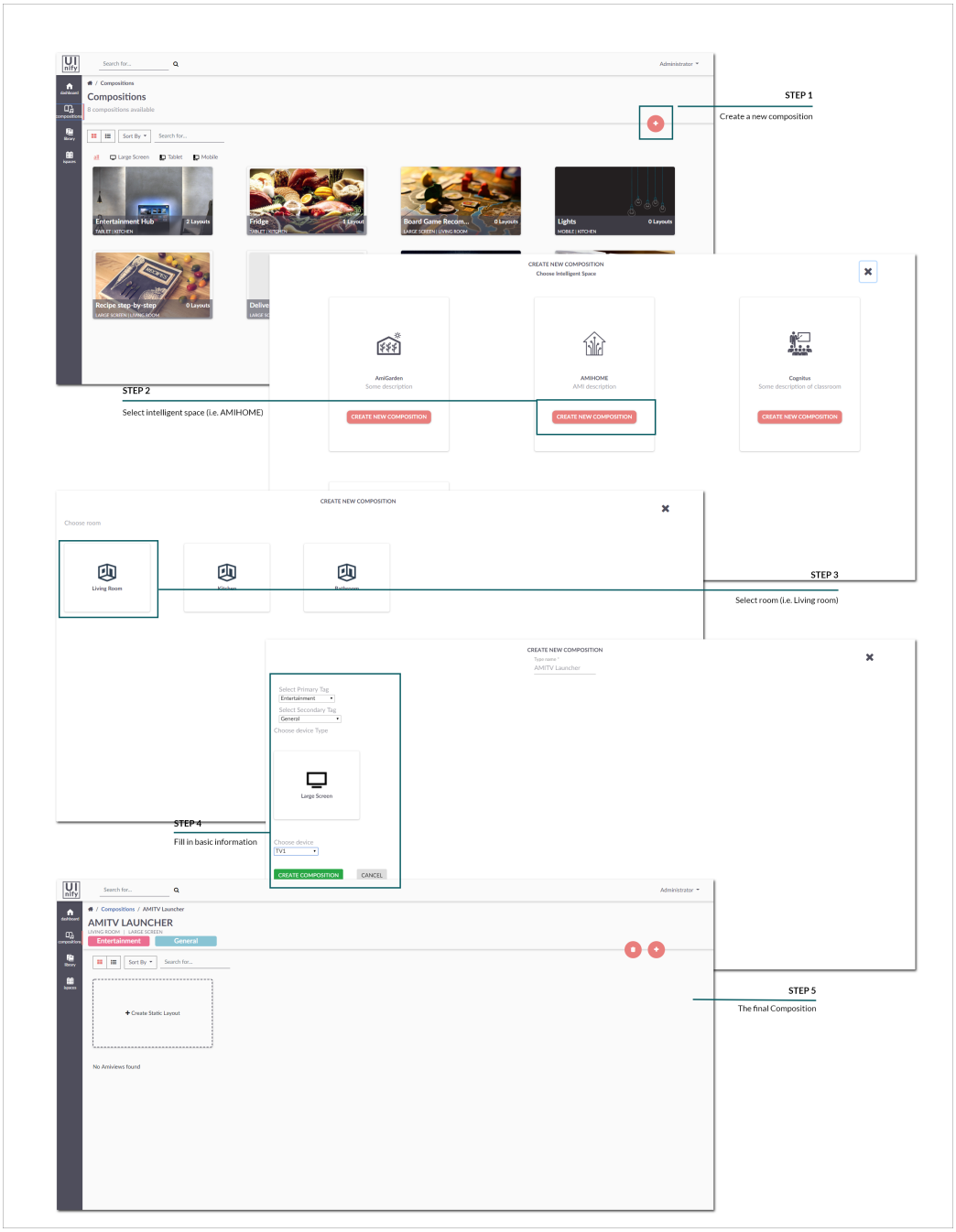

Figure 6.1: Creating the “AMITV Launcher” composition

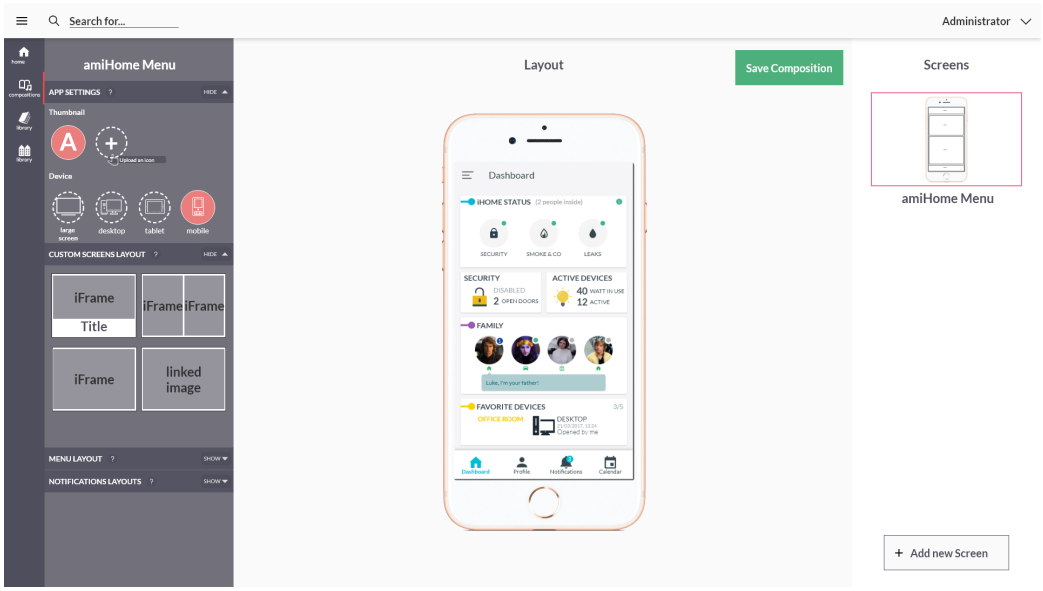

Figure 6.3: The mobile application as created in UInify

it will provide an overview of the active devices and a quick communication component for the family. Finally, it will provide a quick view of the favourite devices of the user (see figure 6.3).

- **Intelligent Space:** AMIHOME
- **Room:** ALL
- **Artifact:** Mobile (Mobile1)
- **Primary Tag:** General

6.2.2 Creating the mobile application

In order to create the new UI mashup, firstly the designer should create a composition or utilize an existing one. For this example, the designer will utilize the composition named “Home Controller”. The process of creating a new composition is similar to the process that is illustrated in Figure 6.1.

In the editor, the designer customizes the layout by adding appropriate placeholders as desired. For the mobile applications, the top and bottom parts of the screen are reserved for menu placements. The user can choose a menu layout from the left panel for the top and bottom menus (e.g. bottom menu with 4 placeholders for quick actions as in figure 6.3). Also, the user can create the main container grid, by adding rows and columns (as many as the container can fit) and defining their height and width respectively.

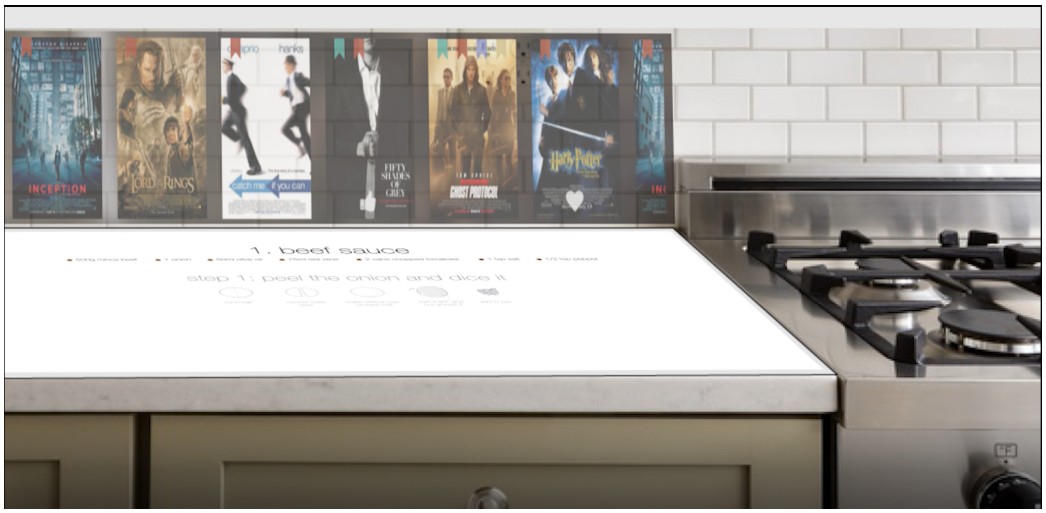

Figure 6.4: UInify concept in Kitchen Bench application

After completing the template she can save it to template repository, to reuse it in other compositions. Next, she populates the placeholders with concrete application components from the application repository. Finally, she can save the repository in the database.

In future versions of UInify, the user could create multiple screens for one individual application. UInify will allow the user to make connections from the respective elements of one screen to the rest of the screens.

6.3 Even more Mashups

6.3.1 Kitchen Countertop

The designer wants to create an ambient application for the Kitchen Countertop. In this scenario, the application will display the recipes' application on the countertop as well as the list of available movies on the wall, to entertain the user while she cooks. For this scenario, we will consider that the screen is one (countertop and wall), but in future versions, the interoperability of multiple devices will be allowed in the system. More specifically, the workspace should include multiple screens for tasks where the user interacts with more than one devices simultaneously within a room. The logic behind this is that all devices are linked to each other and can display related events at the same time.

- **Intelligent Space:** AMIHOME
- **Room:** Kitchen
- **Artifact:** Bench/Large Screen (Bench Display1)

- **Primary Tag:** Utilities

The following concept scenario is an example of the interoperability of the display devices. Two screens are used: the user prepares a meal in the countertop by using the recipe step-by-step application (which is displayed through a projector to the kitchen counter), while in the wall in front of her a movie is playing (through a diagonal projector). The movie pauses when an event of a skype call is pushed in the event federation component. If she decides to answer it, the user interface change and the screen is split in half for movies and Skype call applications.

With the same process, as discussed before a composite user interface for the kitchen countertop can be introduced in UInify. The user can create a composition (or use an existing one if it already exists) for the kitchen bench composition (with similar process as shown in figure 6.1). With the assumption that is mentioned before, that we consider the countertop and the wall as one screen, the user can make the layout by adding rows and columns as in the previous examples (see figure). The figure depicts the composition for the concept scenario.

6.3.2 CognitOS : A Student-Centric Environment for a Intelligent Classroom.

The designer wants to create a web-based working environment that hosts educational applications, namely CognitOS [31]. CognitOS is deployed in multitouch-enabled All-in-One PC which integrates various sensors (e.g. eye-tracker, camera, microphone).

- **Intelligent Space:** Smart Classroom
- **Room:** Intelligent classroom
- **Artifact:** Desktop
- **Primary Tag:** Education

More specifically, the desktop application offers four (4) basic virtual items for accessing the main functionality: (i) a **pile of books** for the book application that offers a shortcut to the student's collection of books (ii) a **pile of notebooks** for the notebook application that acts as a shortcut to student's collection of assignments, (iii) a **personal card** that provides access to the profile application and (iv) a **computer monitor** for launching the multimedia application.

In its core, the CognitOS application is a web launcher to four different widgets. This can easily translate into a UI mashup and create it as a composition, with the information and the process mentioned before (similar with the process followed in Figure 6.1).

Moreover, the editor can be used to create the actual launcher. This layout can be created by adding the following placeholders: First, we create two columns. The right column will be used for the Multimedia application. In the left column,

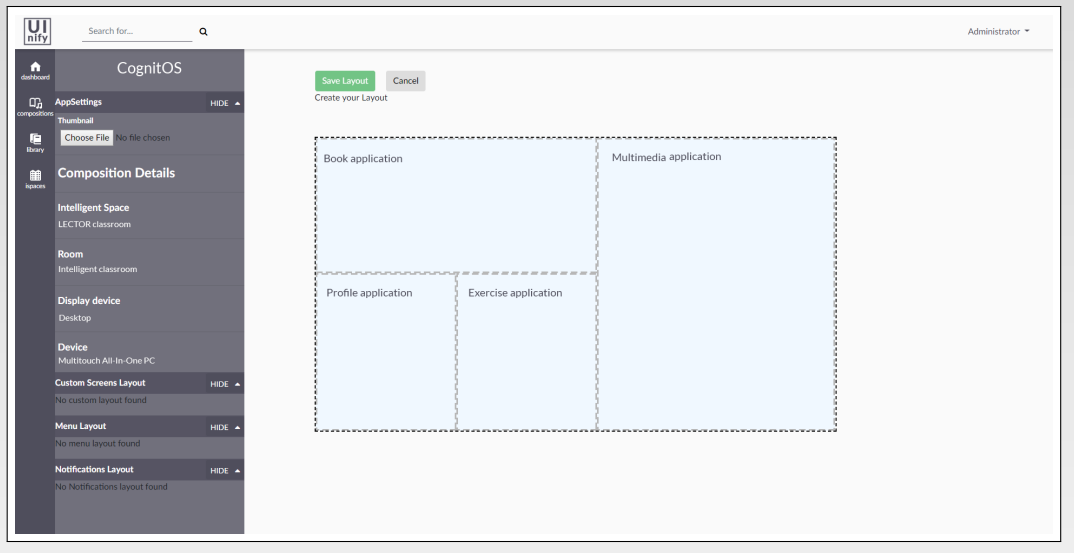

Figure 6.5: AmiView creation for CognitOS

we will create two equal size rows. In the top column, the Book application will be placed. The bottom left column will be separated into two equal columns. In the left part, the Profile application will be placed and in the right part, the Notebook application. The CognitOS AmiView is illustrated in figure 6.5 and the end result in figure 6.6.

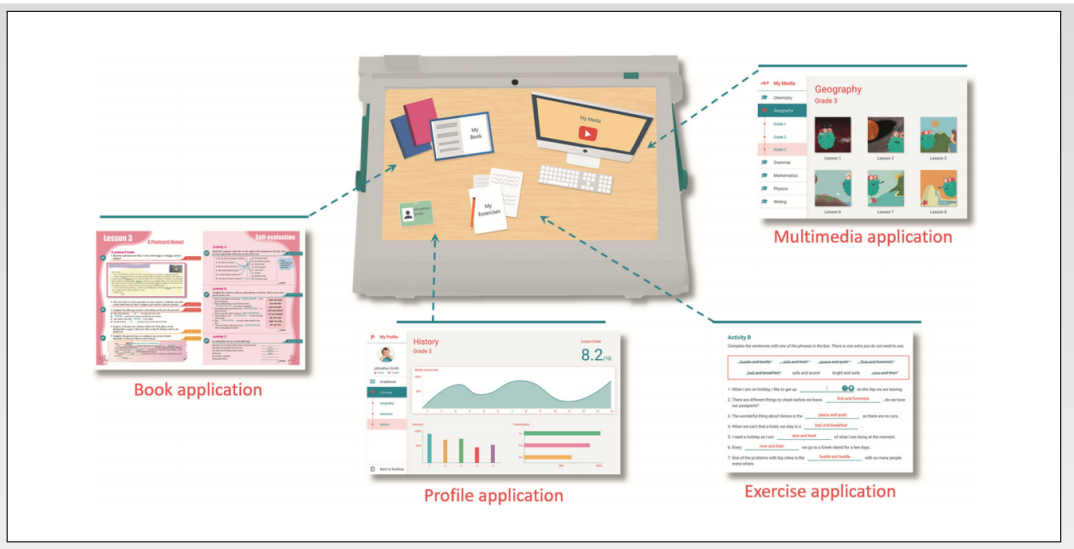

Figure 6.6: Snapshots from CognitOS applications

The CognitOS example demonstrates the expansion of the UInify uses, outside the intelligent home ecosystem. With the proper modulation of the applications and applications components, composite user interfaces for smart classrooms, greenhouses, transportation services, hospitals and eventually smart cities will be easily deployed in the ecosystem.

Chapter 7

Evaluation

The objective of this chapter is to report the outcomes of a preliminary evaluation that attempted to assess the UInify platform. The main goal of this process is to evaluate if the system optimizes the design method in intelligent environments. This section will present (i) the user-based interface evaluation that was followed in the experiment, (ii) the findings of the experiment and (iii) the preliminary results regarding the usability of the implemented User Interfaces as extracted from SUS questionnaires.

7.1 User-based Interface Evaluation

Based on our study, a platform with similar functionality, which enables the entire design and deployment of a composite user interface, does not exist. In this chapter the preliminary evaluation will be presented, which helps to assess the platform and highlight the benefits of using it.

As D.J. Nielsen suggested [89], in the first evaluation iteration, five (5) users can detect approximately 85% of the problems in the user interface. After the first iteration with 5 users, most of the usability problems can be fixed in a redesign. More experiments (at least three (3) according to [89]) can cover the entirety of the problems. Based on this conclusion, UInify was evaluated by five (5) users, who were asked to design and deploy a multimedia hub launcher in the living room of an intelligent home by following nine (9) scenarios.

For the experiment, the following assumptions were made; firstly that all the web applications used through the UInify platform, are already implemented and have been exposed as services to the system from AmI Solertis. Furthermore, the AmI environment and more specifically the Living room has a large screen (TV), where the composition can be displayed.

In the system evaluation, five (5) users of both genders (2 females and 3 males) participated. The users belonged to the age group of 25-30. All of the participants were graduate students in computer science, with different levels of expertise in ambient intelligent systems. One of them was experts in the domain, three of them

had moderate experience and one of them had little experience.

7.2 The experiment

7.2.1 Preliminaries

For the evaluation of a system to be successful, tasks/scenarios should be clear for the user and allow him to test and evaluate its major functionality. To this end, nine (9) tasks were decided for the evaluation of the UInify, which drive the user through a large portion of the system. Prior to the assignment, users were introduced to the purpose of the system, its capabilities and functions, the target audience and the process that they are requested to follow. To assist the assessors, the tasks were also recorded in an electronic presentation to be understandable and easily available. For the evaluation process, it was requested by the users to follow some scenarios of different difficulty. Moreover, they were asked to follow the thinking aloud method and they were encouraged to make comments. Finally, to better measure the process it was explained that the comments and the completion time of tasks, were recorded.

7.2.2 The process

The usability test that was conducted, had four stages: (i) the preparation, (ii) the introduction, (iii) the actual test and (iv) the debriefing session. Initially, the scenarios for the evaluation were decided. It was important to browse a large portion of the system for the users in order to have a better understanding of UInify platform. Before the evaluation phase began, a presentation was created for the user to read the scenarios. The scenarios and the SUS questionnaire were given in an electronic format (PowerPoint and Google form respectively).

The computer system was prepared with the UInify in its initial state. The experiment was conducted remotely with a Skype call. The test users were given a brief explanation of the purpose of the test. Afterwards, the test procedure was presented. It was made clear that the test evaluates the software and not the user. The users were asked to use the method thinking aloud, to make comments (positive or negative) for the system. Finally, it was explained that some comments and time will be written down for assessing the system and produce some graphs for statistical reasons.

During the test itself, the scenarios were read to the users and the experimenter was helping only when he was asked. After completing the scenarios, users were asked to fill a System Usability Scale (SUS) [40] questionnaire. The SUS questionnaire includes ten (10) questions with five response options ranging from “Strongly agree” to “Strongly disagree”. It is a Likert scale, providing a “quick and dirty”, yet valuable evaluation tool.

Finally, the experiment included a debriefing session during which the users prompt to give comments regarding the system. They were also asked for what

they liked most and whether they would use the system on a daily basis and for any suggestion that will result in the enhancement of the system.

7.2.3 Scenarios

Each user was requested to complete nine (9) scenarios covering the most important functionality of UInify.

Scenario 1

You are a smart environment designer and you have previously registered in UInify platform.

Login to your account using the following credentials:

- **Username:** Ami User
- **Email:** ami@ics.forth.gr
- **Password:** 123456

Scenario 2

Check in your profile if you created any compositions for the Smart Kitchen? How many?

Scenario 3

Find the recently created compositions and the most used UIs in UInify.

Scenario 4

You are a designer for ambient environments and you want to create a composition for the smart living room that will serve as an entertainment hub. You want to include a Music component, but you don't know if there is one in the system. For this component, you need to have a music player. See if such an application exists in the UInify library.

Scenario 5

Next, you want to know if there is already a composition for a media center for the Living Room. Browse through the appropriate components of the system to check if there is a similar composition.

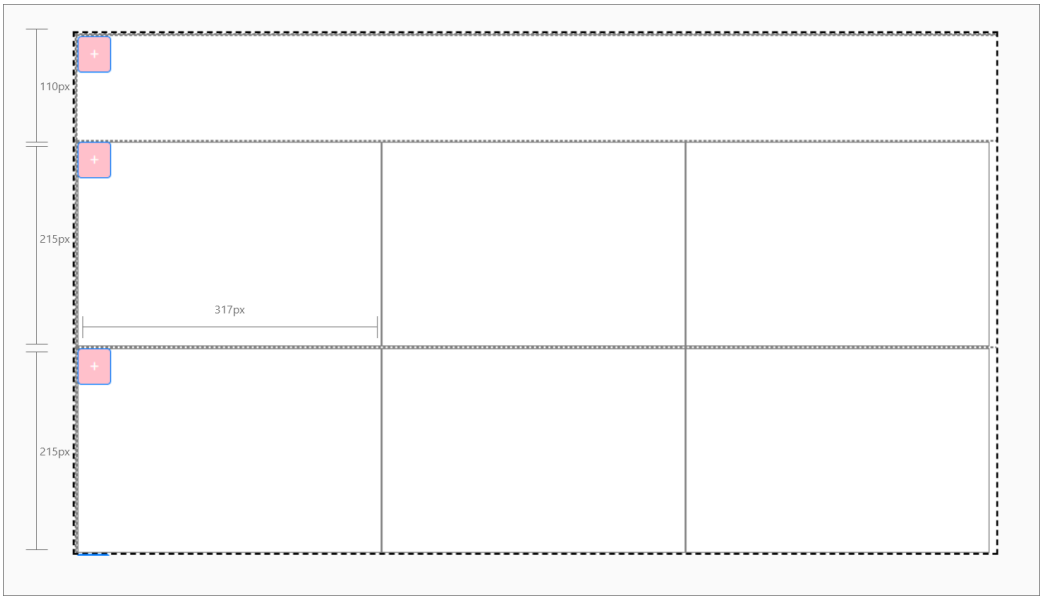

Figure 7.1: Figure for Scenario 7

Scenario 6

You decided that you want to design a launcher for your multimedia hub with the applications that already exist in the system. You will name it "AMI TV Launcher". The display device will be the TV1 of your AMIHOME Living room. It will be for Entertainment or general use.

Scenario 7

Start designing your AmiViews in the 'AMI TV Launcher' Composition (see Figure 7.1). Your launcher will have 3 rows. The first will be narrower with 110px height and the rest will have height of 215px. Moreover, the second and the third row should each have 3 columns. Each column will have width of 317px.

Scenario 8

After designing the template, you should proceed to adding applications in the template (see Figure 7.2). In the cell (2,1) you should add 'Music' application. In the cell (2,2) you should add 'TV' application. In the cell (2,3) you should add 'Movies' application. In the cell (3,1) you should add 'News' application. In the cell (3,2) you should add 'Image Gallery' application. Finally, in the cell (3,3) you should add 'Chat' application. Deploy your application in the system.

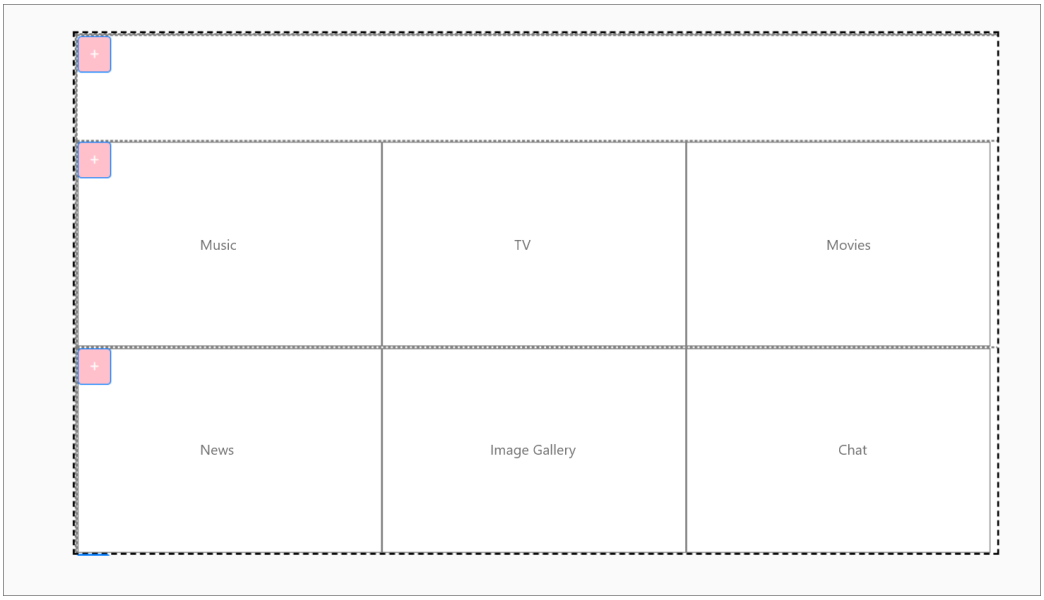

Figure 7.2: Figure for Scenario 8

Scenario 9

You have an idea about creating a recipe composition of your kitchen counter and you are interested whether a Recipe composition already exists or applications of this context have been imported to the system. Query the keyword 'Recipe' to check if any exist.

7.3 Performance Measurement

Each user was requested to complete nine (9) scenarios (see section above) covering the most important functionality of UInify.

7.3.1 Findings per scenario

In this section, the appropriate quantitative methods used to assess the UInify system will be presented. Both graphs that show the execution times of the scenarios and some worth-mentioning comments of the users will be presented. After the assessment per scenario, we extract the general problems that need to be revised in the UInify studio.

7.3.1.1 Scenario 1

are a smart environment designer and you have previously registered in UInify platform.

Login to your account using the following credentials:

- **Username:** Ami User
- **Email:** ami@ics.forth.gr
- **Password:** 123456

Execution Times

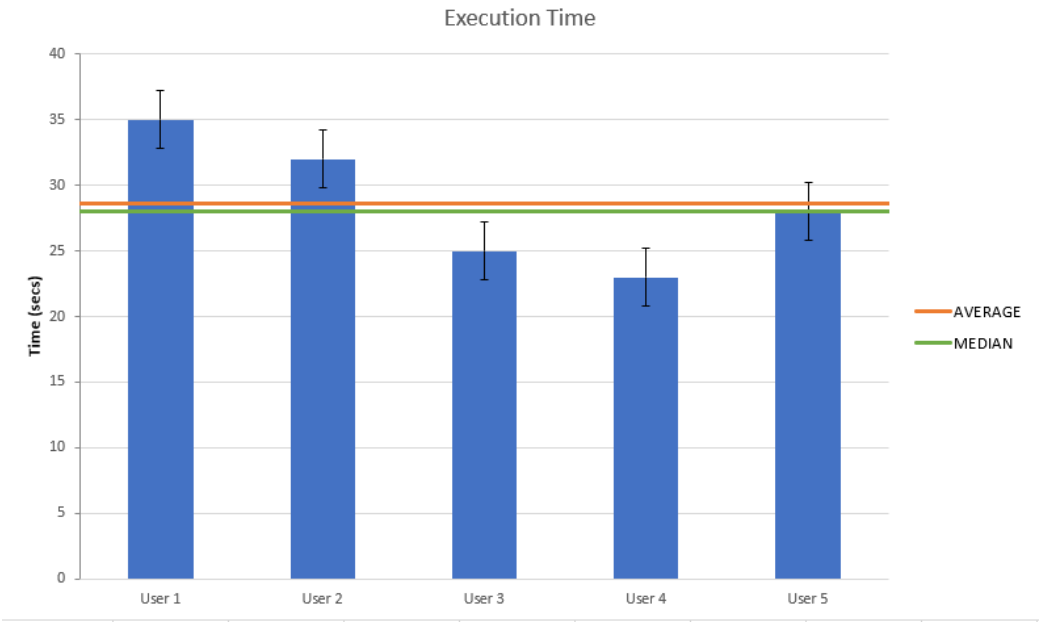

Figure 7.3: Execution times for Scenario 1

Comments

This was an easy scenario to introduce the users to the system. As depicted in Figure 7.3, they users executed really fast. All users successfully log in to the system. Only the third user didn't notice right away that the required field was e-mail and password and not a username. This may mean that the font size should be bigger and more noticeable.

Scenario 2

Check in your profile if you created any compositions for the Smart Kitchen? How many?

Execution Times

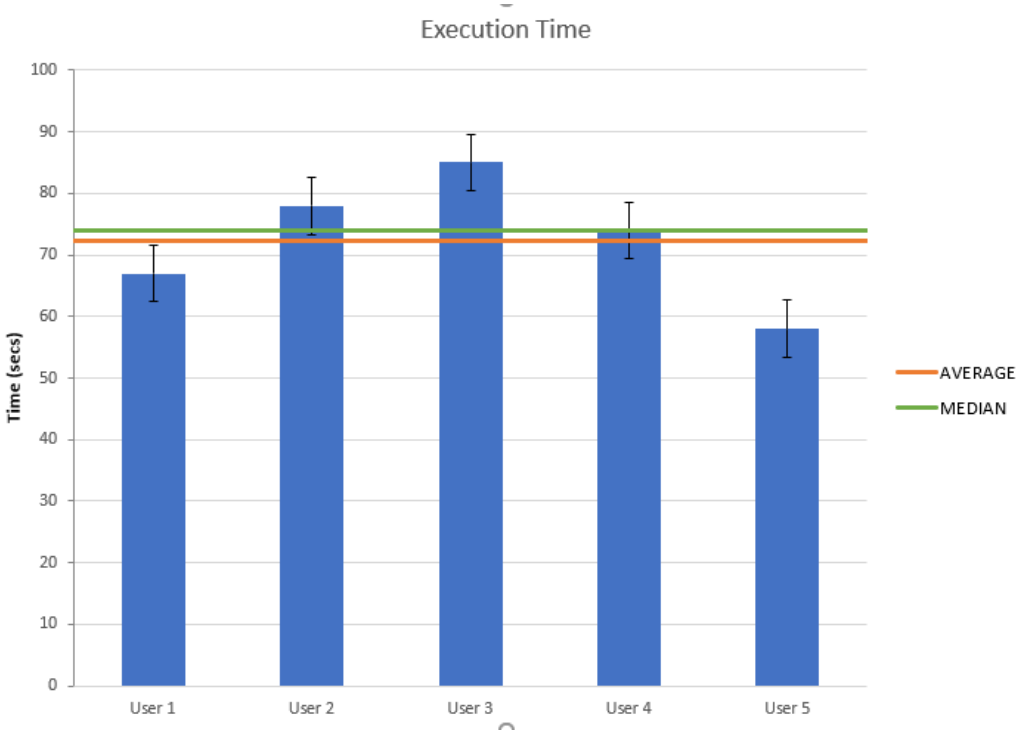

Figure 7.4: Execution times for Scenario 2

Comments

In general, the users didn't have a problem with this scenario. Three of the users (Users 2,3,4 in Figure 7.4), who did the longer time to execute the scenario, started looking in the composition category (they assumed that in the tiles the system would mention the creator probably). They all found the profile eventually, in the right top corner. Some users didn't notice at first the 'Kitchen' indicator in the boxes. One user mentioned that she wanted to click the Username and redirect to the profile immediately, and not open the dropdown menu.

Scenario 3

Find the recently created compositions and the most used UIs in UInify.
Execution Times

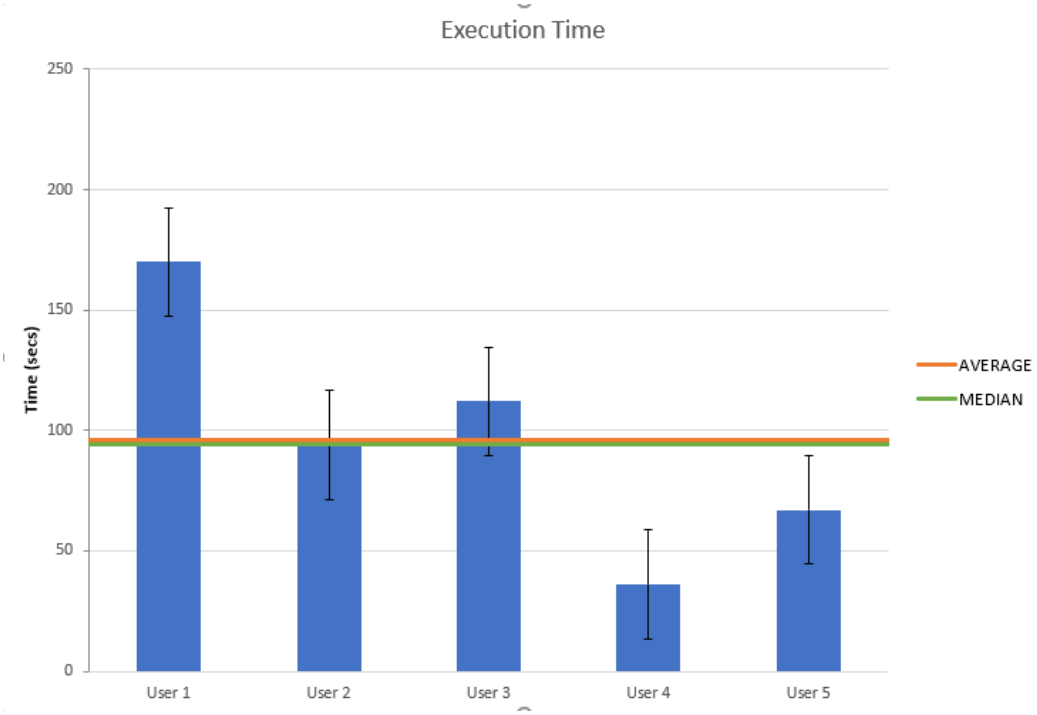

Figure 7.5: Execution times for Scenario 3

Comments

Almost all users tried to search their profile for the recently created compositions and the most used UIs. Many users browsed to library and compositions and sorted the results by Date or by Most Used. In the end, all users found these categories on the Dashboard. Some of the users commented that it was more of a scenario problem rather than a platform-oriented problem and this is depicted in execution times in figure 7.5.

Scenario 4

You are a designer for ambient environments and you want to create a composition for the smart living room that will serve as an entertainment hub. You want to include a Music component, but you don't know if there is one in the system. For this component, you need to have a music player. See if such an application exists in the UInify library.

Execution Times

Figure 7.6: Execution times for Scenario 4

Comments

This scenario was easy for most of the users. They had some difficulty with the terminology used (Applications - Compositions - AmiViews). Almost every user used the filtering option provided to find Entertainment applications. All users found the Music application and the Music Player application component in the UI library. The user 5, did the longer time (see figure 7.6), due to the fact that tried to create a composition rather than find the application. The user commented that she wanted a preview of the application in the creation process.

Scenario 5

Next, you want to know if there is already a composition for a media center for the Living Room. Browse through the appropriate components of the system to check if there is a similar composition.

Execution Times

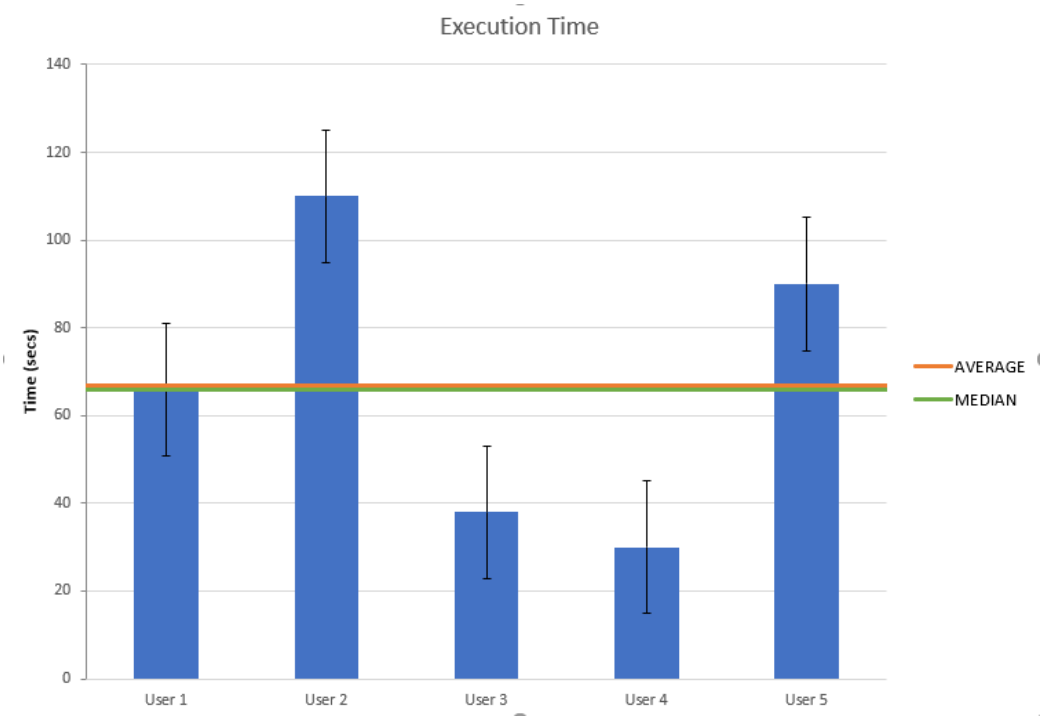

Figure 7.7: Execution times for Scenario 5

Comments

This scenario was relatively easy for the users. All the users browsed to the composition functionality immediately and checked if the appropriate composition exists. Some users used the searching functionality of the compositions to find a media center application. User 2 did the longest time, because she tried to search the compositions and 'play' with the keywords. The User 5 first searched for the composition in UI library and then in Composition repository.

Scenario 6

You decided that you want to design a launcher for your multimedia hub with the applications that already exist in the system. You will name it "AMI TV Launcher". The display device will be the TV1 of your AMIHOME Living room. It will be for Entertainment or general use.

Execution Times

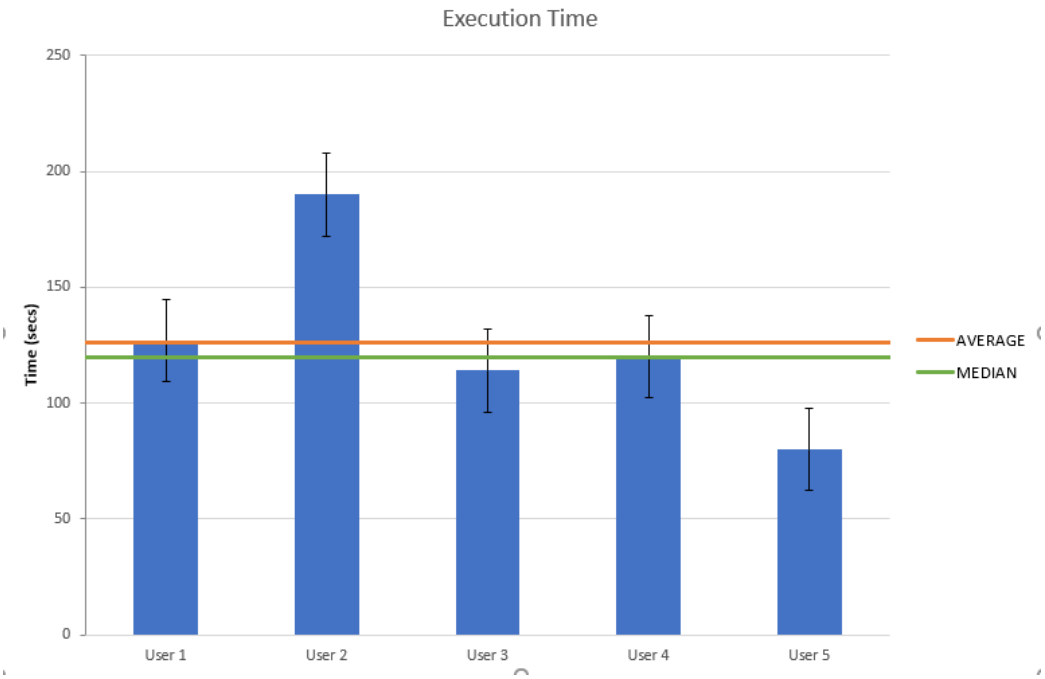

Figure 7.8: Execution times for Scenario 6

Comments

All the users browsed for the composition functionality and pressed the '+' button provided. Users thought that this wizard was intuitive and clear. One user suggested to include a recommendation system, which will auto-fill the fields and suggest grids and applications based on the previous compositions and choices of her. User 2, who did the longest time (see figure 7.8) followed the same process.

Scenario 7

Start designing your AmiViews in the 'AMI TV Launcher' Composition (see Figure 7.1). Your launcher will have 3 rows. The first will be narrower with 110px height and the rest will have height of 215px. Moreover, the second and the third row should each have 3 columns. Each column will have width of 317px.

Execution Times

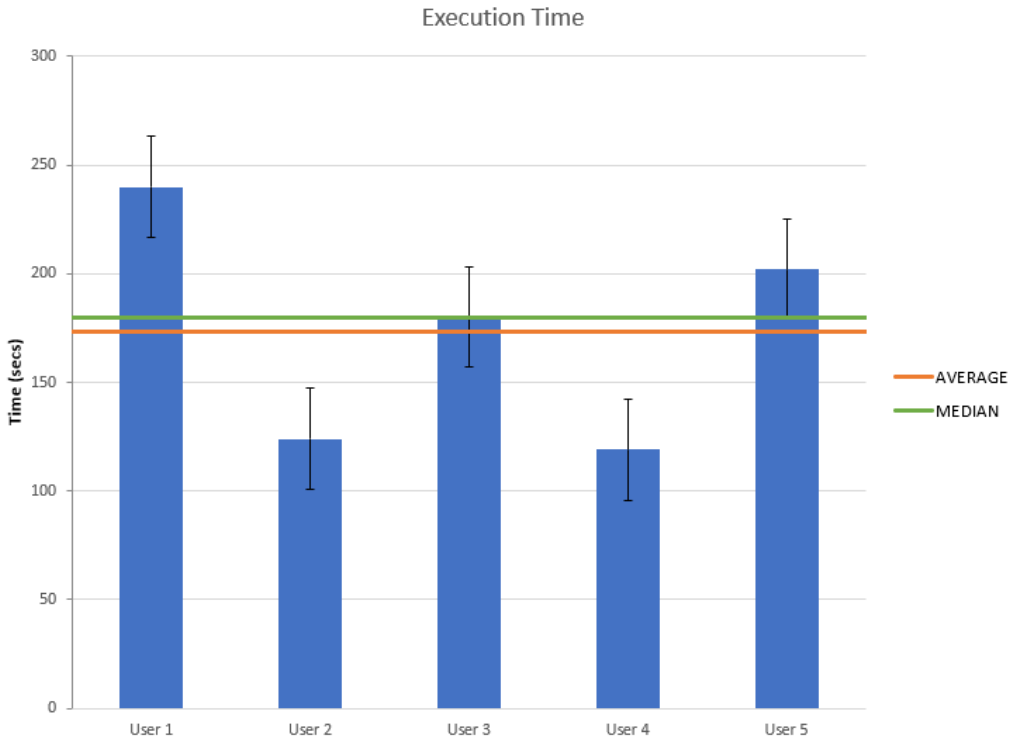

Figure 7.9: Execution times for Scenario 7

Comments

The users had some difficulty with this scenario, as depicted from execution times in figure 7.9. All users requested more indications and labels to follow the process of creating AmiViews. They felt unsure in the way that they add columns on the grid. Some users commented that they wanted to drag and drop already existing placeholders. They also requested for adjustable components and edit feature if the save their layout.

Scenario 8

After designing the template, you should proceed to adding applications in the template (see Figure 7.2). In the cell (2,1) you should add 'Music' application. In the cell (2,2) you should add 'TV' application. In the cell (2,3) you should add 'Movies' application. In the cell (3,1) you should add 'News' application. In the cell (3,2) you should add 'Image Gallery' application. Finally, in the cell (3,3) you should add 'Chat' application. Deploy your application in the system.

Execution Times

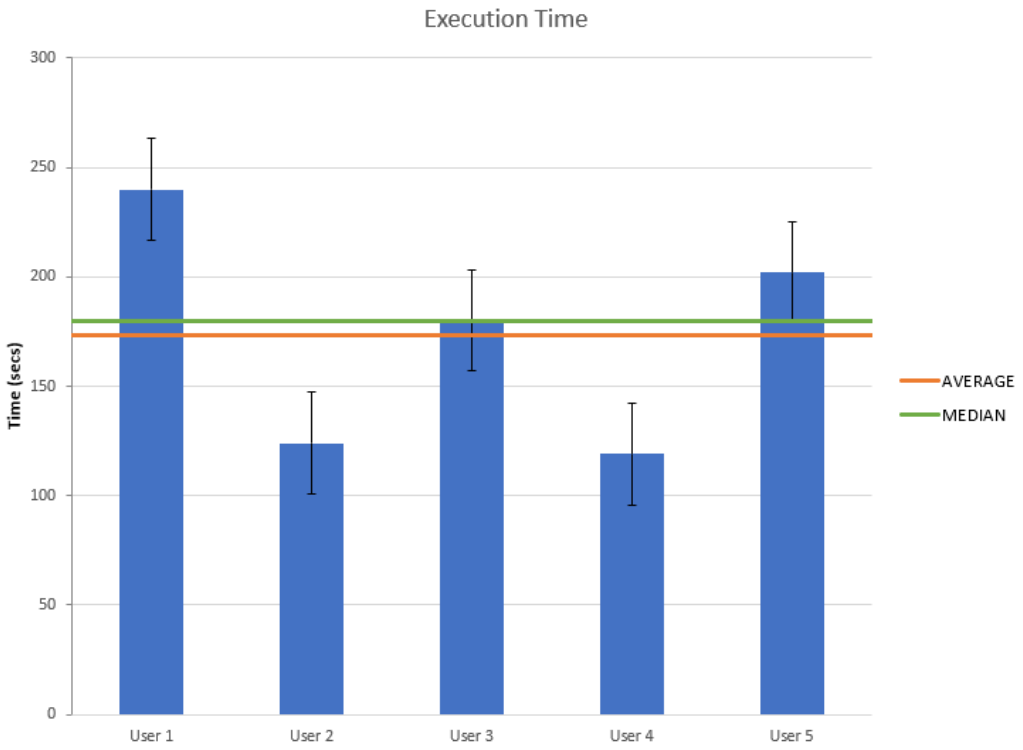

Figure 7.10: Execution times for Scenario 8

Comments

In this scenario, the users found intuitive the way that they could add an application to the placeholders. However, they all wanted indications, label or icons that their application indeed added to the placeholder. The execution times illustrate this confusion of the users in figure 7.10.

Scenario 9

You have an idea about creating a recipe composition of your kitchen counter and you are interested whether a Recipe composition already exists or applications of this context have been imported to the system. Query the keyword 'Recipe' to check if any exist.

Execution Times

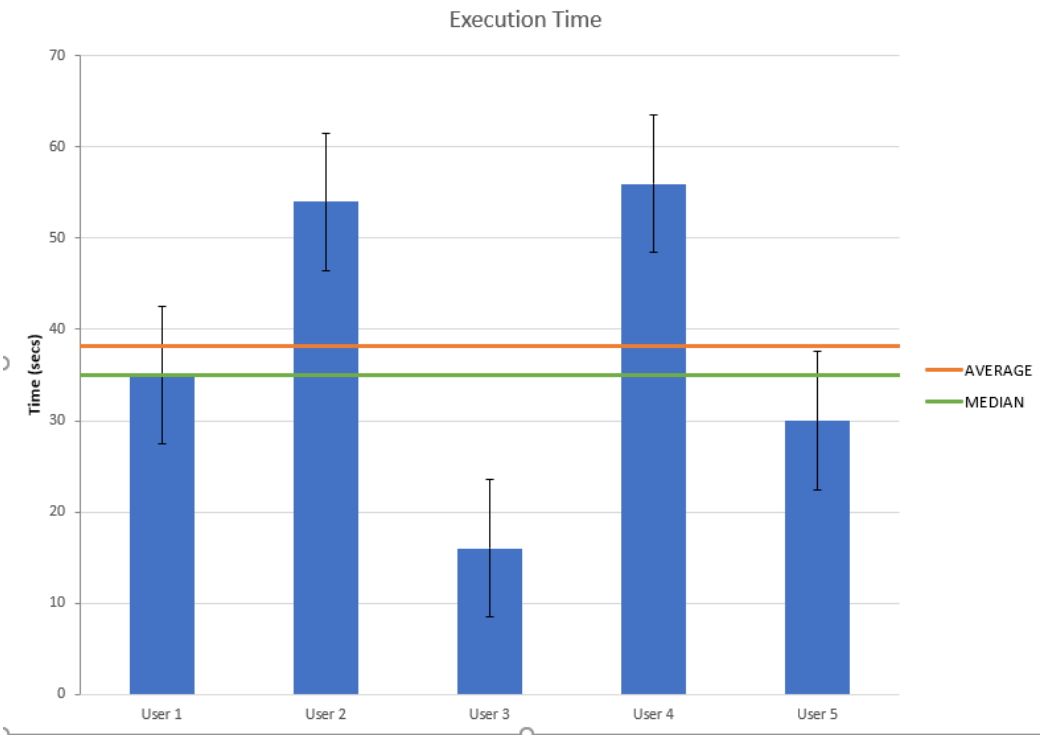

Figure 7.11: Execution times for Scenario 9

Comments

All users used the universal search component efficiently. One user thought to browse the composition repository and search in that page for the requested composition. The figure 7.11 illustrates the execution times for scenario 9.

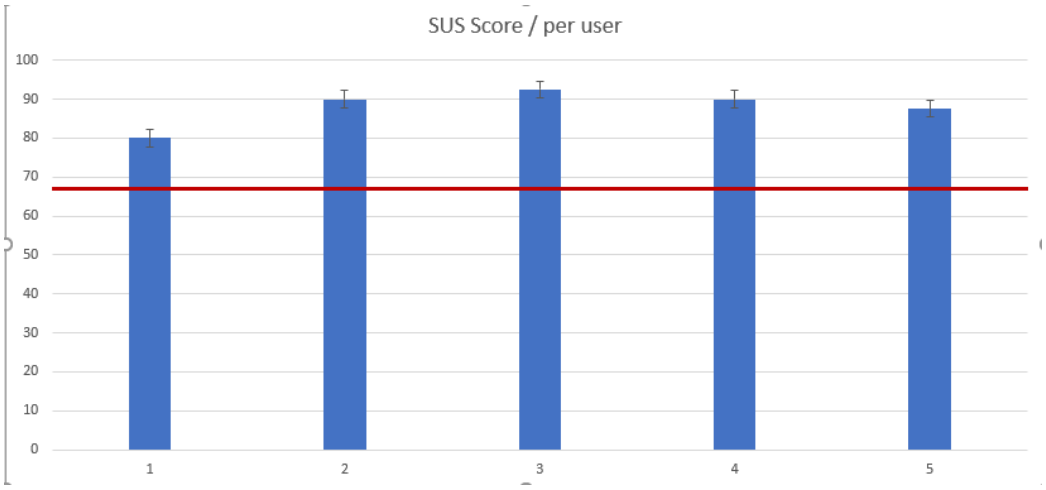

Figure 7.12: The System Usability Score of UInify per user

7.3.2 Discussion

As the findings suggest, the UInify was easy to use by the experiment users. All the users commented that the user interface design was very simple and intuitive and that resulted in an easy-to-use system. Even for users with a little ambient intelligent background, it did not take long to familiarize with the system and the entire concept. That is an important finding because the system could be used in its future versions by simple users without a technical background.

Some users had a difficulty to familiarize with the terminology used in the system.. One interesting comment from a user was that he would prefer the system to be more personalized to his actions and previous concepts. Overall, users found that the system was designed with familiar concepts and did not find something peculiar.

Two main problems were found on the system that needs redesign. Firstly all users suggested that the editor, that used for AmiView creations, needs better indications for its basic functionality. Almost every user said that they need a more flexible grid system as well as indicators for rows and column addition. Two users suggested a recommendation system in the editor as well as in the overall system, as it will be helpful to monitor their compositions. The second problem found was in the UI mapper. Users requested a more clear indication that their application was added (with text or/and icon) in the placeholders. They also wanted a preview of the device to check if the application was actually deployed.

The overall opinion of the users was that the UInify was an intuitive tool, with a pleasant and clear design. All users commented that they would use it if the appropriate infrastructure was well defined. Moreover, they thought that the UI of the application was intuitive and found the idea interesting. This is also depicted by the SUS score of eighty-eight (88) (see figure 7.12) which indicated that the tool was marked as highly usable.

Chapter 8

Summary and Future work directions

8.1 Summary

Fifty years ago, a house that aids its residents with daily routine, an office that assists user for work effectively or a smart class seemed like things that could only be found in movies using visual effects. All these concepts become reality with the progression of technology and the realization of Ambient Intelligence. Ambient environments are complex ecosystems that are expected to have multiple screens that display the graphical user interface that has been imported into the system. Given the vast number of devices and services in such environments, it is impossible for developers to build all-inclusive applications, which will enable users to thoroughly monitor and control the extensive variety of physical devices and artificial services that they own.

To that end, this thesis has proposed UInify, a framework that can compose flexible web applications in real-time that unify under a common roof the individual GUIs that control and/or monitor the hardware and software components of an intelligent space (i.e. smart home).

Summing up, from an engineering perspective the UInify framework: (i) introduces a unified user interface made from pluggable web interfaces that exist in the ecosystem, (ii) store and deploy the compositions as HTML UI mashups, (iii) provides an editor the designer can create the grid by adding appropriate placeholders (AmiView).

8.2 Future Work

The UInify platform has many aspects that are worth further research. Firstly, the system needs some redesign according to the users' comments that seemed to converge. UI mapper and Layout builder need a redesign with a more intuitive editor and labels to guide the users.

The process of introducing an AmiView can be more simple by offering a set of tools. That tools can be resized, zoom-in and zoom-out features for the placeholders. In addition, different types of placeholders should be created i.e. placeholder with a title, placeholders for menus, to make the process more fast and intuitive.

Moreover, the tool can include some simple machine learning algorithms, to recommend applications, similar composition layouts to users. The evaluation results showed that UInify Dashboard and all categories in general, can be more personalized to its user.

8.2.1 Dynamic Rules

One addition to the system could be the creation of amiViews with dynamic rules. That means, that the UInify could create the amiviews that fit in the context of the initialization information that the user made. The user can choose from the available layouts and customize it or use it on the fly. Moreover, the user will be available while using the system to press a button and change the layout of the placeholders, if they don't fit her needs.

In the evaluation experiment, one user that it would be useful when a user creates compositions to recommend existing layouts and choose between them for the composition.

8.2.2 Layout builder

UInify'S **Layout builder** currently is rather limited to one device per AmiView and should be further extended to support designers to design and deploy more composite user interfaces. Firstly, the layouts should be expanded and include more complex shapes than standard size rectangles. It also will help the user to control their layout if re-sizing and drag-and-drop component will make the editor even more intuitive and easy-to-use.

The composite designs for such an ambient environment should combine multiple devices for tasks performed in daily life. Another improvement should consider integrating complex artificial intelligent components to aid the whole process. These components can suggest to the end-users application component and/or template layouts to improve the creation of user interface and the overall user experience.

In the experiment, all users wanted indicators for the Layout Builder for the row and column addition. One user said that didn't understand the difference between colours in addition of rows and columns, and wanted text labels. Furthermore, users suggested undo and redo functionality, drag and drop components and adjustable rows and columns.

8.2.3 UI Mapper

UInify's UI Mapper can be improved with the aid of reasoning components. This reasoning component will limit the options of the applications for the chosen display device. This filtering will limit the applications with the best-fitted applications for the placeholder chosen of the AmiView by taking into consideration of previous preferences of the user, the room that the composition will be deployed, the context of the application and the artifact limitations (e.g. if the application needs a Kinect sensor to work correctly and the display device hasn't got one installed, this display device is not a match for the application).

In the evaluation experiment, all users suggested that they wanted a label or/and icon that the application was inserted in the placeholder. They also suggested that the system should recommend applications for the placeholders.

8.2.4 UInify player

The UInify player is a new component that will be essential to the system. This will be the client that will be installed in every system in the intelligent space of FORTH-ICS. Its purpose will be to communicate with the UInify studio and it will present the appropriate composition in every display device. This will provide the change composition button that is mentioned before, and it will be sent a request for a new calculation for the recommendation system, in order to provide the user with a new composition.

Bibliography

[1] Amazon alexa and echo, www.amazon.com.

[2] Angular, http://angular.io.

[3] Apple home application, https://www.apple.com/lae/ios/home/.

[4] bcrypt: Javascript library.

[5] Control4 | home automation and smart control systems, www.control4.com.

[6] The ECHO IV home computer: 50 years later | computer history museum.

[7] Evrythng home, https://evrythng.com/.

[8] Express.js, expressjs.com.

[9] Gideon | smart home.

[10] Google home, https://store.google.com.

[11] Ifttt | https://ifttt.com/.

[12] Introduction to json web tokens.

[13] MavHome: Managing an adaptive versatile home, CSE@UTA.

[14] The mean stack blog.

[15] Mongoose, https://mongoosejs.com/.

[16] Nest: Create a connected home, www.nest.com/uk/.

[17] Node.js, nodejs.org/en/.

[18] Openhab | empowering the smart home https://www.openhab.org.

[19] SmartThings www.smatthings.com.

[20] The top 10 IoT segments in 2018 based on 1,600 real IoT projects - IoT analytics.

[21] Wink | a simpler, smarter home, www.wink.com/.

[22] Electronic computer for home operation (echo): The first home computer. In *IEEE Annals of the History of Computing*, volume 16, pages 59–61. 1994.

[23] MIT house_n, 2000.

[24] Orange smart house | university of surrey, 2001.

[25] Emile Aarts and Jose Luis Encarnação. Into ambient intelligence. In Emile HL Aarts and Jose Luis Encarnação, editors, *True Visions - The Emergence of Ambient Intelligence*, pages 1–16. Springer, 01 2006.

[26] Emile Aarts and Frits Grotenhuis. Ambient intelligence 2.0: Towards synergetic prosperity. *J. Ambient Intell. Smart Environ.*, 3(1):3–11, January 2011.

[27] Gregory D Abowd, Aaron F Bobick, Irfan A Essa, Elizabeth D Mynatt, and Wendy A Rogers. The aware home: A living laboratory for technologies for successful aging. page 7.

[28] G. Acampora, D. J. Cook, P. Rashidi, and A. V. Vasilakos. A survey on ambient intelligence in healthcare. *Proceedings of the IEEE*, 101(12):2470–2494, Dec 2013.

[29] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015.

[30] Jan Alexandersson. i2home: Towards a universal home environment for the elderly and disabled. page 3.

[31] Margherita Antona, George Margetis, Stavroula Ntoa, Asterios Leonidis, Maria Korozi, George Paparoulis, and Constantine Stephanidis. Ambient intelligence in the classroom: an augmented school desk. 07 2010.

[32] Nikolaos Anyfantis, Evangelos Kalligiannakis, Achilleas Tsiolkas, Asterios Leonidis, Maria Korozi, Prodromos Lilitsis, Margherita Antona, and Constantine Stephanidis. AmITV: Enhancing the role of TV in ambient intelligence environments. In *Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference on - PETRA '18*, pages 507–514. ACM Press.

[33] L. Atzori, A. Iera, and G. Morabito. From "smart objects" to "social objects": The next evolutionary step of the internet of things. *IEEE Communications Magazine*, 52(1):97–105, January 2014.

[34] Juan Carlos Augusto. Past, present and future of ambient intelligence and smart environments. In Joaquim Filipe, Ana Fred, and Bernadette Sharp, editors, *Agents and Artificial Intelligence*, volume 67, pages 3–15. Springer Berlin Heidelberg.

[35] Andreas Auinger, Martin Ebner, Dietmar Nedbal, and Andreas Holzinger. Mixing Content and Endless Collaboration – MashUps: Towards Future Personal Learning Environments. In Constantine Stephanidis, editor, *Universal Access in Human-Computer Interaction. Applications and Services*, volume 5616, pages 14–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[36] Lynne Baillie and David Benyon. Place and technology in the home. *Computer Supported Cooperative Work (CSCW)*, 17(2):227–256, Apr 2008.

[37] Nazmiye Balta-Ozkan, Rosemary Davidson, Martha Bicket, and Lorraine Whitmarsh. Social barriers to the adoption of smart homes. 63:363–374, 12 2013.

[38] Christoffer Björkskog. Human computer interaction in smart homes, 2007.

[39] Manousos Bouloukakis, Christos Stratakis, and Constantine Stephanidis. Ami garden: Building an iot infrastructure for precision agriculture. volume 153, pages 69–80.

[40] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[41] Jean C Burgelman and Yves Punie. Information, society and technology. In Emile HL Aarts and Jose Luis Encarnação, editors, *True Visions - The Emergence of Ambient Intelligence*, pages 17–33. Springer, 01 2006.

[42] Federico Cabitza, Daniela Fogli, Rosa Lanzilotti, and Antonio Piccinno. Rule-based tools for the configuration of ambient intelligence systems: a comparative user study. *Multimedia Tools and Applications*, 76(4):5221–5241, February 2017.

[43] Jill Cao, Yann Riche, Susan Wiedenbeck, Margaret Burnett, and Valentina Grigoreanu. End-user mashup programming: Through the design lens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1009–1018, New York, NY, USA, 2010. ACM.

[44] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2010.

[45] Diane J. Cook. How smart is your home? 335(6076):1579–1581.

[46] Diane J. Cook, Juan C. Augusto, and Vikramaditya R. Jakkula. Ambient intelligence: Technologies, applications, and opportunities. 5(4):277–298.

[47] Joëlle Coutaz. User interface plasticity: Model driven engineering to the limit! In *Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '10, pages 1–8, New York, NY, USA, 2010. ACM.

[48] F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul. Understanding ui integration: A survey of problems, technologies, and opportunities. *IEEE Internet Computing*, 11(3):59–66, May 2007.

[49] Florian Daniel and Maristella Matera. *Mashups: Concepts, Models and Architectures*. Springer Publishing Company, Incorporated, 2014.

[50] Boris de Ruyter and Emile Aarts. Ambient intelligence: Visualizing the future. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 203–208, New York, NY, USA, 2004. ACM.

[51] Liyanage C. De Silva, Chamin Morikawa, and Iskandar M. Petra. State of the art of smart homes. *Eng. Appl. Artif. Intell.*, 25(7):1313–1321, October 2012.

[52] Google developers. Blocky | google developers.

[53] Alan Dix and Laura Cowen. HCI 2.0? usability meets web 2.0. page 2.

[54] Soufiene Djahel, Christoph Sommer, and Annapaola Marconi. Guest editorial: Introduction to the special issue on advances in smart and green transportation for smart cities. 19(7):2152–2155.

[55] K Ducatel, M Bogdanowicz, F Scapolo, J Leijten, and J-C Burgelman. Scenarios for ambient intelligence in 2010. page 58, 2001.

[56] D Evans. The internet of things: How the next evolution of the internet is changing everything. 1:1–11, 01 2011.

[57] Alois Ferscha, Stefan Resmerita, and Clemens Holzmann. Human computer confluence. In *Proceedings of the 9th Conference on User Interfaces for All*, ERCIM'06, pages 14–27, Berlin, Heidelberg, 2007. Springer-Verlag.

[58] Daniela Fogli, Rosa Lanzilotti, and Antonio Piccinno. End-user development tools for the smart home: A systematic literature review. In Norbert Streitz and Panos Markopoulos, editors, *Distributed, Ambient and Pervasive Interactions*, volume 9749, pages 69–79. Springer International Publishing.

[59] Jesse James Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders Publishing, Thousand Oaks, CA, USA, 2nd edition, 2010.

[60] Hans-W. Gellersen, Michael Beigl, and Holger Krull. The mediacup: Awareness technology embedded in an everyday object. In Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, pages 308–310, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[61] Neil Gershenfeld, Raffi Krikorian, and Danny Cohen. The internet of things. 291(4):76–81.

[62] Khusvinder Gill, Shuang-Hua Yang, Fang Yao, and Xin Lu. A zigbee-based home automation system. 55(2):422–430.

[63] Gartner IT Glossary. Glossary for connected home.

[64] trends.google.com Google Trends. Google trends, 2018.

[65] Dominique Guinard. Mashing up your web-enabled home. In Florian Daniel and Federico Michele Facca, editors, *Current Trends in Web Engineering*, volume 6385, pages 442–446. Springer Berlin Heidelberg.

[66] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. page 8.

[67] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer, Berlin, Heidelberg.

[68] Richard Harper. The connected home: The future of domestic life.

[69] Richard Harper. *Inside the Smart Home*. Springer-Verlag, 2003.

[70] J. P. Hubaux, S. Capkun, and Jun Luo. The security and privacy of smart vehicles. *IEEE Security Privacy*, 2(3):49–55, May 2004.

[71] Yucheng Jin, Chi Tai Dang, Christian Prehofer, and Elisabeth André. A multi-display system for deploying and controlling home automation. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, pages 399–402, New York, NY, USA, 2014. ACM.

[72] Andreas Kamilaris, Vlad Trifa, and Andreas Pitsillides. HomeWeb: An application framework for web-based smart homes. pages 134–139. IEEE.

[73] Hermann Kopetz. *Internet of Things*, pages 307–323. Springer US.

[74] Agnes Koschmider, Victoria Torres, and Vicente Pelechano. Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. page 8.

[75] Thomas Kubitza and Albrecht Schmidt. meSchup: A platform for programming interconnected smart things. 50(11):38–49.

[76] Shiu Kumar. Ubiquitous smart home system using android application. 6(1):33–43.

[77] M. O. Leavitt and B. Shneiderman. *Research-Based Web Design & Usability Guidelines*.

[78] Ying-tsung Lee, Wei-hsuan Hsiao, Chin-meng Huang, and Seng-cho T. Chou. An integrated cloud-based smart home management system with community hierarchy. 62(1):1–9.

[79] A. Leonidis, D. Arampatzis, N. Louloudakis, and C. Stephanidis. The ami-solertis system: Creating user experiences in smart environments. pages 151–158, Oct 2017.

[80] Daniel Lewis. What is web 2.0? *XRDS*, 13(1):3–3, September 2006.

[81] Sam Lucero. Iot platforms: enabling the internet of things. 2016.

[82] Eisaku Maeda, Yasuhiro Minami, Masato Miyoshi, Minako Sawaki, Hiroshi Sawada, Atsushi Nakamura, Junji Yamato, Takeshi Yamada, and Ryuichiro Higashinaka. The world of mushrooms: A transdisciplinary approach to human-computer interaction with ambient intelligence. 4(12):9.

[83] Deborah J. Mayhew. *Principles and Guidelines in Software User Interface Design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[84] Deborah J. Mayhew. The usability engineering lifecycle. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '99, pages 147–148, New York, NY, USA, 1999. ACM.

[85] McKinsey&Company. There's no place like a connected home.

[86] O'Reilly Media. What is web 2.0.

[87] Dr Miraz, Maaruf Ali, Peter Excell, and Rich Picking. A review on internet of things (iot), internet of everything (ioe) and internet of nano things (iont), 09 2015.

[88] Mohammad-Mahdi Moazzami, Guoliang Xing, Daisuke Mashima, Wei-Peng Chen, and Ulrich Herberg. SPOT: A smartphone-based platform to tackle heterogeneity in smart-home IoT systems. pages 514–519. IEEE.

[89] J Nielsen. Why you only need to test with 5 users. 01 2000.

[90] Anastasia Ntagianta, Maria Korozi, Asterios Leonidis, Margherita Antona, and Constantine Stephanidis. Cognitos: A student-centric working environment for an attention-aware intelligent classroom, 07 2018.

[91] Javier Palanca, Elena del Val, Ana Garcia-Fornes, Holger Billhardt, Juan Manuel Corchado, and Vicente Julián. Designing a goal-oriented smart-home environment. *Information Systems Frontiers*, 20(1):125–142, Feb 2018.

[92] Jaimin Patel and Gaurang Panchal. An iot-based portable smart meeting space with real-time room occupancy. In Yu-Chen Hu, Shailesh Tiwari, Krishn K. Mishra, and Munesh C. Trivedi, editors, *Intelligent Communication and Computational Technologies*, volume 19 of *Lecture Notes in Networks and Systems*. Springer Singapore.

[93] Chris J. Pilgrim. Improving the usability of web 2.0 applications. In *Proceedings of the Nineteenth ACM Conference on Hypertext and Hypermedia*, HT '08, pages 239–240, New York, NY, USA, 2008. ACM.

[94] Rajeev Piyare. Internet of things: Ubiquitous home control and monitoring system using android based smart phone. 2(1):5–11.

[95] Floyd I. R., Jones M. C., Rathi D., and Twidale M. B. Web mash-ups and patchwork prototyping: User-driven technological innovation with web 2.0 and open source software. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 86–86, Jan 2007.

[96] Carlos Ramos. Ambient intelligence: A state of the art from artificial intelligence perspective. In Jose Neves, Manuel Filipe Santos, and Jose Manuel Machado, editors, *Progress in Artificial Intelligence*, pages 285–295. Springer Berlin Heidelberg.

[97] Jorg Rech and Klaus-Dieter Althoff. Artificial intelligence and software engineering: Status and future trends. OCLC: 248670776.

[98] L. Rick. *Express.Js: Guide Book on Web Framework for Node.Js*. CreateSpace Independent Publishing Platform, USA, 2016.

[99] Edward Ross. Intelligent user interfaces: Survey and research directions. Technical report, Bristol, UK, UK, 2000.

[100] Carsten Röcker. User-centered design of intelligent environments: Requirements for designing successful ambient assisted living systems. page 8.

[101] Albrecht Schmidt. Interactive context-aware systems interacting with ambient intelligence. pages 159–178, 01 2005.

[102] Manfred Schneps-Schneppe and Dmitry Namiot. About home gateway mashups. 1(5):1–5.

[103] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. Pearson Addison Wesley, 2004.

[104] Statista. Number of apps available in leading app stores as of 1st quarter 2018.

[105] Yue Suo, Chenjun Wu, Yongqiang Qin, Chun Yu, Yu Zhong, and Yuanchun Shi. HouseGenie: Universal monitor and controller of networked devices on touchscreen phone in smart home. pages 487–489. IEEE.

[106] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. *Node.Js.* Betascript Publishing, Mauritius, 2010.

[107] MongoDB team. Mongodb, www.mongodb.com.

[108] Kaisa Väänänen-Vainio-Mattila and Minna Wäljas. Towards user-centered mashups: Exploring user needs for composite web services. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1327–1332, New York, NY, USA, 2011. ACM.

[109] Athanasios Vasilakos and Witold Pedrycz. *Ambient Intelligence, Wireless Networking, And Ubiquitous Computing.* Artech House, Inc., Norwood, MA, USA, 2006.

[110] Hongxu Yin, Ayten Ozge Akmandor, Arsalan Mosenia, Niraj K. Jha, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. Smart healthcare. 12(4):401–166.

[111] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. 1(1):22–32.

[112] Gottfried Zimmermann and Gregg Vanderheiden. The universal control hub: An open platform for remote user interfaces in the digital home. In Julie A. Jacko, editor, *Human-Computer Interaction. Interaction Platforms and Techniques*, volume 4551, pages 1040–1049. Springer Berlin Heidelberg.

[113] Gottfried Zimmermann and Gregg Vanderheiden. The universal control hub: An open platform for remote user interfaces in the digital home. In *Proceedings of the 12th International Conference on Human-computer Interaction: Interaction Platforms and Techniques*, HCI'07, pages 1040–1049, Berlin, Heidelberg, 2007. Springer-Verlag.