UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

FLUID: A Software Development Kit for Adaptive Tabletop Applications

by

Evangelos Poutouris

MSc dissertation submitted in partial fulfillment for the

Master of Science degree in Computer Science

Heraklion, July 2020

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
DEPARTMENT OF COMPUTER SCIENCE

FLUID: A Software Development Kit for Adaptive Tabletop Applications

by **Evangelos Poutouris**

in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

APPROVED BY:

Author: Evangelos Poutouris

Supervisor: Constantine Stephanidis, Professor, University of Crete

CONSTANTINOS STEPHANIDIS
Digitally signed by CONSTANTINOS STEPHANIDIS
Date: 2020.07.03 14:44:23 +03'00'

Committee Member: Antonis Argyros, Professor, University of Crete

Digitally signed by ANTONIOS ARGYROS
Date: 2020.07.03 14:46:00 +03'00'

Committee Member: Xenophon Zabulis, Principal Researcher, FORTH-ICS

XENOFON ZAMPOULIS
Digitally signed by XENOFON ZAMPOULIS
Date: 2020.07.03 14:41:25 +03'00'

Director of Graduate Studies: Antonis Argyros, Professor, University of Crete

Digitally signed by ANTONIOS ARGYROS
Date: 2020.07.03 14:46:37 +03'00'

Heraklion, July 2020

To my loved one and my parents for their love and patience
To my friends and colleagues for always being there for me
To my mentor and my lab for supporting me guiding me during my studies

Abstract

The exponential growth of Internet of Things (IoT) devices and services along with the emergence of the Ambient Intelligence (AmI) paradigm unveiled new potentials for the domain of domestic living, blurring the imaginary line between “the computer” and the surrounding “’intelligent’ environment”. The diversity of artifacts in the ’intelligent’ living room introduces a plethora of interaction challenges, stemming from the need for cross-platform migration of the available applications, and highlights the requirement for a unified interaction environment for the residents.

As a result of the advancements in surface computing, physical monitors that were initially used in ‘intelligent’ environments, can now be replaced by common household surfaces, such as coffee-tables, benches, and walls, which can act as displays with the help of projection systems. However, user interaction and usability of such systems is still subject to the current limitations of surface computing technologies.

To address these interaction issues, this thesis introduces FLUID, a web-based software development kit (SDK) that provides an application hosting environment, explicitly customized to be deployed on surface displays, which binds all surface computing applications under a common roof. Additionally, FLUID grants the developers the ability to compose and manage in real-time flexible and adaptable ‘surface-computing-web-applications’ through a rich application programming interface (API). Additionally, FLUID offers a set of utilities that empowers intelligent space designers to monitor and evaluate the behavior of their applications by simulating different scenarios of use.

In comparison to related work, FLUID offers an innovative, context-aware, user-centric environment that not only hosts surface computing applications, but also resolves two prominent surface computing issues: the occlusion of graphical user interfaces from physical objects, as well as user interface cluttering. This is achieved through a sophisticated and configurable virtual space management mechanism, through which hosted applications get reorganized in an intelligent manner, ensuring a smooth User Experience (UX) to their end-users.

In addition to the FLUID development kit, this thesis describes the design process and its functionality, elaborates on the implementation details,

presents two case studies that made use of it and discusses the results of an expert-based evaluation study in the context of the Intelligent Home of ICS-FORTH.

Keywords: Surface Computing, Adaptive User Interfaces, Ambient Intelligence, Intelligent Home

Περίληψη

Η ιλιγγιώδης εξέλιξη των συσκευών και υπηρεσιών του «Διαδικτύου των Πραγμάτων» (Internet of Things - IoT), σε συνδυασμό με την εμφάνιση του υποδείγματος της Διάχυτης Νοημοσύνης, προσφέρσει νέες δυνατότητες στον τομέα της οικιακής διαβίωσης, εξαφανίζοντας τη διαχωριστική γραμμή μεταξύ του «υπολογιστή» και του «'έξυπνου' περιβάλλοντος». Η ποικιλομορφία των πιθανών έξυπνων συσκευών που εντάσσονται σε ένα «έξυπνο σαλόνι» εισάγει πληθώρα προκλήσεων στη διαδικασία αλληλεπίδρασης που απορρέουν από την ανάγκη μετακίνησης και 'παρουσίασης' των διαθέσιμων εφαρμογών σε διαφορετικά υπολογιστικά συστήματα. Αυτό σκιαγραφεί την απαίτηση για ένα ενοποιημένο οικοσύστημα διάδρασης για τους χρήστες-ενοίκους.

Χάρη στην πρόοδο της υπολογιστικής τεχνολογίας που επιτρέπει την διάδραση μέσω φυσικών επιφανειών (Surface Computing), οι συσκευές απεικόνισης (οθόνες) που αρχικά χρησιμοποιούνταν σε έξυπνα οικιακά περιβάλλοντα αντικαθίστανται σταδιακά από κοινές οικιακές επιφάνειες, όπως τραπέζια, πάγκοι ή τοίχοι, που έχουν την δυνατότητα να παρουσιάζουν πληροφορία με τη βοήθεια συστημάτων προβολής. Ωστόσο, η ευχρηστία αυτών των συστημάτων περιορίζεται, λόγω των χαρακτηριστικών των σημερινών τεχνολογιών του Surface Computing.

Για την αντιμετώπιση αυτών των ζητημάτων, η παρούσα μεταπτυχιακή εργασία παρουσιάζει το FLUID, μια διαδικτυακή βιβλιοθήκη ανάπτυξης λογισμικού (SDK), η οποία προσφέρει ένα περιβάλλον ενσωμάτωσης εφαρμογών Surface Computing ειδικά προσαρμοσμένο για χρήση σε υπολογιστικές επιφάνειες. Το FLUID παρέχει στους προγραμματιστές τη δυνατότητα να συνθέσουν και να ελέγξουν σε πραγματικό χρόνο ευέλικτες και προσαρμόσιμες εφαρμογές, μέσω μιας πλούσιας προγραμματιστικής διεπαφής εφαρμογών (Application Programming Interface). Επιπλέον, το FLUID συνοδεύεται από ένα σύνολο βοηθητικών εργαλείων που επιτρέπουν στους σχεδιαστές έξυπνων χώρων να προσομοιώνουν διαφορετικά σενάρια χρήσης ώστε να παρακολουθούν και να αξιολογούν τη συμπεριφορά των εφαρμογών τους.

Σε σχέση με τη σχετική βιβλιογραφία, το FLUID προσφέρει ένα καινοτόμο περιβάλλον, με επίκεντρο τον χρήστη, το οποίο όχι μόνο φιλοξενεί εφαρμογές, αλλά και επιλύει δύο σημαντικά προβλήματα του Surface Computing: την 'έμφραξη' των γραφικών αντικειμένων από φυσικά αντικείμενα και την «άναρχη» μετακίνηση τους σε περιπτώσεις περιορισμένης διαθεσιμότητας

χώρου. Αυτό επιτυγχάνεται μέσω ενός εξελιγμένου και διαμορφώσιμου συστήματος διαχείρισης εικονικού χώρου, το οποίο ενισχύει την ‘έξυπνη’ αναδιοργάνωση των εφαρμογών, εξασφαλίζοντας στους τελικούς χρήστες των εφαρμογών μια ομαλή εμπειρία χρήσης.

Πέραν της βιβλιοθήκης ανάπτυξης λογισμικού FLUID, η εργασία αυτή περιγράφει τη διαδικασία σχεδίασης και τη λειτουργικότητά του SDK, αναλύει τις λεπτομέρειες υλοποίησης, παρουσιάζει δυο μελέτες εφαρμογής και καταλήγει συνοψίζοντας τα αποτελέσματα ενός πειράματος ευρετικής αξιολόγησης, το οποίο έγινε με τη συμμετοχή ειδικών ευχρηστίας στο πλαίσιο της «Έξυπνης Κατοικίας» του ΙΠ-ΙΤΕ.

Λέξεις κλειδιά:, Surface Computing, Προσαρμόσιμες Γραφικές Διεπαφές Χρήστη, Διάχυτη Νοημοσύνη, Έξυπνη Κατοικία

Contents

List of Figures

List of Tables

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) [1], the World Wide Web (WWW) has become mature enough to serve as a platform for creating rich interactive applications.

The success of the Web 2.0 [2] and the Web of Things (WoT) [3], [4] 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. [5], [6], [7]). These technological accomplishments (e.g. the use of client-side scripting languages, the advance of Web Services, public APIs) shaped up a new concept, the mashups [8], [9]. Mashup web applications, and more specifically in the context of this work the user interface mashups – are small web applications or parts of an application that integrate interfaces at different levels of the application stack from different resources on the web.

Moreover, the recent re-emergence of machine learning [10] and the use of neural networks [11] has led to the advancement of computer vision [12] and, as a result, a big step forward towards object detection and recognition [13]. That, along with the increasing interest in Ambient Intelligence (AmI) environments over the years, is interrelated with the need for new and advanced object detection systems used in smart environments (e.g. intelligent homes) to encapsulate and control a wide range of connected devices and artificial services, offering smart behavior by responding to events (e.g. turn on the lights when the user enters a room in an intelligent home) under a common application.

1.1 Overview

An ambient intelligence environment is expected to contain a wide range of technologically augmented artifacts able to present intelligent, multimodal and context-aware applications along with various sensors to unobtrusively pay attention to users in order to understand their intentions, and ultimately decide which actions to make. Nowadays, most of these displays are common household surfaces such as coffee-tables and dinner-tables, benches, walls, even mirrors. These surfaces are virtually augmented with projectors, input sensors and presence detection mechanisms. However, most scientific projects and commercial products regarding augmented physical surfaces (e.g. smart tables and walls) are deprived of their standard use as common surfaces which incorporate real physical objects in favor of displaying user interfaces. However, in common household environments, tables or walls should retain their standard use. Therefore, in a modern intelligent home, computer applications that are displayed on such surfaces must naturally coexist with stacked physical objects and at the same time retain their functionality.

One of the main challenges for tabletop applications originate from the Human Computer Interaction (HCI) domain and refer to the application's usability [14] [15]. That is because a surface can be rather large for a human to be able to reach in every direction. A usable interface has three main characteristics, (i) it should be easy-to-use and easy to familiarize by the user, (ii) it should be easy to achieve an objective through the user interface and (iii) it should be easy to be remembered by a user when subsequently used. In order to achieve usability, it is important to create interactive applications through the iterative design process that HCI advises. It is essential for designers to utilize the User-Centered Design (UCD) approach to define their functional requirements in order to create a successful user experience [16]. Furthermore, common issues of tabletop applications like user interface occlusion from physical objects and user interface cluttering which stem from the surface computing paradigm will be discussed in this thesis.

To that end, this thesis proposes FLUID (**F**orce **L**ayout **U**ser **I**nterface **D**evkit), a software development kit which empowers designers and developers to compose, develop, distribute, monitor and evaluate rich and adaptable application mashups that are intended for virtually augmented physical surfaces. Apart from providing a tabletop application host mechanism and a toolset for creating and evaluating mashups, FLUID aims to address the challenges of surface computing by providing: (i) a configurable space management algorithm which prevents user interface cluttering and (ii) a sophisticated user interface presentation layout which prevents occlusions from

detected physical objects and at the same time provides a more suitable interaction medium to the end-user, and (iii) a unified environment for application hosting which enables interoperability across multiple surfaces in the context of the Intelligent Livingroom.

FLUID provides two distinct APIs containing functions for customizing the behavior of the system. The first API targets the developers of the Livingroom's smart detection artifacts in order to control several aspects of the surfaces (e.g. control the positioning of a surface's application based on the position of the end-user, or provide the position and size of any physical objects that are present). The second API provides various functions in order to help application developers in the development and the installation of FLUID-enabled applications.

1.2 Thesis structure

The rest of this master thesis is divided in six (6) main chapters as indicated in the table of contents. The structure is as follows:

- **Chapter 2** presents an extensive review of the background theory on the introduction of Ambient Intelligence in Smart Livingroom environments and how it affects the everyday needs and activities of the home residents as well as a literature review on modern Surface Computing theory, Second Screening and Web Application Mashups. Also, existing technology regarding tabletop and second screen applications are explored further. Finally, the importance of utilizing unified environments is discussed and how this laid the groundwork for FLUID SDK.

- **Chapter 3** presents the AugmenTable system that started as a proof of concept for this thesis, following a set of motivating scenarios, and subsequently describes the requirements of FLUID SDK system architecture.

- **Chapter 4** is dedicated to an elaborate analysis of the FLUID SDK, followed by an analysis of the fundamental components and the main functionality aspects of the system.

- **Chapter 5** describes the implementation of the system by illustrating the conceptual model and the representation of the data model that was adopted, and by providing an overall high-level description of the functionality of the three (3) main facets of the SDK.

- **Chapter 6** illustrates some use cases of the SDK in developing tabletop applications for Smart Home Spaces such as the Intelligent Livingroom and the Intelligent Kitchen of the AmI-Home.

- **Chapter 7** discusses the evaluation process followed, the scenarios and tasks that were created to this purpose, and the findings that were extracted.

- **Chapter 8** presents a summary of this work, as well as a discussion on future feature updates and performance fixes. Finally, future projects that stem from FLUID SDK will be discussed.

Chapter 2

Literature Review

2.1 Background Theory

2.1.1 Ambient Intelligence in the context of the Intelligent Livingroom

With the advancement of Internet of Things (IoT) [1] and in conjunction with cloud computing [17], there has been an abundance of web-enabled devices and services for smart homes [18] that performed a complete makeover of the face of domestic life. In such technologically cutting-edge homes, individual users in their everyday lives interact frequently with various smart artifacts at the same time. The main objectives of smart homes, according to [19], are the automation of tasks that might be complex or annoying for inhabitants (e.g., control home appliances), saving energy, enhancing the feeling of security, improving comfort and supporting independent living for elderly or disabled people. In that regard, the environment of the living room has gained much attention by researchers, since inhabitants spend a considerable amount of time there with their family and friends, and as a result many daily activities are linked to it (i.e., watching TV, reading, socializing, relaxing) [20]. Starting from the early 2000s, the EasyLiving project [21] presented a technological architecture intended for smart environments that provides user detection, identification and tracking, device tracking, and several demo applications regarding room control, media control, remote user sessions, and a universal mouse controller.

Figure 1: Document discussion in the EasyLiving livingroom

Illumiroom [22] and RoomAlive [23] applied spatial augmented reality techniques by employing depth cameras and projectors in order to perform projection mapping in the room's space and physical objects, thus enabling new interaction possibilities and turning the living room into an enhanced and immersive experience.

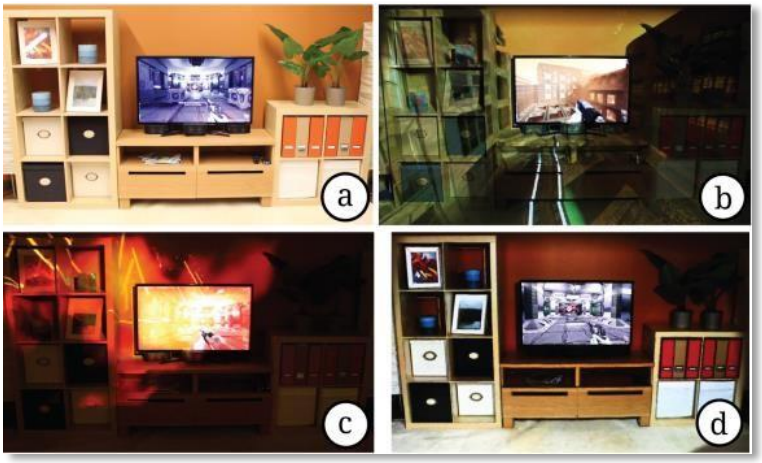


Figure 2 The Illumiroom, augmenting the appearance of an environment.

Figure 3: The RoomAlive project: a spatial augmented reality system

Moreover, services and applications adapt in a context-sensitive manner [19], while interaction goes beyond traditional desktop-oriented interaction techniques (e.g., mouse, keyboard, and touch), endorsing natural modalities such as eye-tracking [24], freehand gestures [25], etc. In a living room space, the most commonly found device is the television. Watching TV is a regular daily activity that takes up large part of people's leisure time [26]. In some cases, people consider the TV as a companion, letting it play in the background regardless of their activity [27]. There are people that even talk to this device, although they know that they will never get an answer [28]. Nowadays, as Internet technology and satellite broadcasting change the way people consume

television content, the medium continues to evolve, solidifying its position as one of the most important inventions of the 20th century [29].

2.1.2 Second Screening

Besides the TV being the primary display of an “intelligent” living room, additional commercial display devices (e.g., smartphones, tables, laptops) or other flat surfaces capable of displaying information by using projecting technologies, (e.g., coffee tables, benches, walls) act as second screens that supplement TV content [30], [31] or give additional feedback to the house residents [22], [32]. As suggested by [33], [34], the TV no longer demands our full attention. Instead, while watching TV, consumers are doing complementary activities (e.g., looking for information) across multiple displays, usually through the device that is closest to them. In addition to such exploration tasks, studies conducted by [35], [36] claim that the usage of second screens provides enhanced attention to TV programs, fosters co-discussion among users regarding TV news and social media, encourages critical thinking and permits personalized advertising [37]. Additionally, [38], [39] reports that the embodiment of gamification techniques (e.g., scoring systems where the user earns points for being an active commentator) can raise participation via socially generated commentary. In addition to that, [40] claims that users actively interact and engage with the viewing ecosystem by using second screens which are also intended to be used as input devices, thus promoting the bidirectional interactive TV (iTV) concept [41], [42]. Specifically, many approaches use second screens as hosts that contain interfaces that not only display information but also remotely control the TV and the overall viewing environment (see next section). While second screening has benefits [43], it also has a fair number of drawbacks. The most important is cognitive fatigue, especially after extensive use [44], whereas problems concerning decision making and usability issues come next [45], [46]. When used effectivelly, second screens should mainly aid the user visually by showing complementary content and not replace entirely the main screen [47], especially when the user is working on a cognitively demanding primary task [48]. Moreover, the TV should remain the most significant device in a viewing ecosystem and the user should remain always in control of it [49]. Finally, context-sensitive prompts can motivate users to actively participate [50] and the interface should always be optimized for the target screen [51] as well as be personalized for the current user [52].

2.1.3 Surface Computing

While second screens hosted in commercial devices, such as smartphones and tablets, inherently support user input (i.e., touch), common surfaces such as a coffee table or a wall can be transformed into interactive mediums with the use of ICT. Such technologically enhanced artifacts follow the paradigm of surface computing [53] and grant multimodal interaction (i.e., touch and object recognition) via unobtrusive sensing technologies. The terms “Surface Computing” and “Surface Computers” [53] were coined by Microsoft in the PixelSense project [54] and describe means of (multitouch) interaction based on common physical surfaces. From an interaction perspective, as [55] reports, such systems (e.g., table top or wall-projected interfaces) present design challenges because interaction resembles more real-world object manipulation than traditional mouse-based interaction with computers. Different manipulation styles have their own pros and cons; specifically, as [56] suggests, direct input enables rich interpersonal and natural interactions, allowing users to understand their actions (especially during multi-user setups), while indirect input through mouse devices allows a more comfortable and ergonomic interaction, enabling easy access to all regions of the surface. Finally, when multitouch manipulation is supported, (i) the touch interaction paradigm should support some level of personalization [57] and (ii) the role of multitouch surfaces along with appropriate vocabularies of gestures have to be defined for each application [58]. Visualization is another challenging aspect of surface computing, due to the occlusion problems that occur when a physical object is placed on top of a UI element; to that end, various research approaches aim to adapt the overall interface so as to eliminate such cases.

2.1.4 User Interface Mashups

An interesting topic which has emerged with the accelerating development of the Web of Things and Web 2.0 are the web mashups. A web mashup is a web application that combines data, logic or/and interfaces from multiple sources. Daniel et al. [59] states that a typical composite application is developed starting from reusable data, application logic, and/or user interfaces which in most cases are typically are already sourced from the Web.

The typical architecture of a mashup consists of three layers [59]–[61]:

Data Component. Monitors and handles the data that flow in an ambient environment. Data in this application are sent, stored or received using the JSON format.

Logic Component. Provides the underlying functionality of the logic service using RESTful services.

Presentation Component. Uses web browser specific technologies (HTML, CSS and Javascript), so the developer can develop user interfaces.

In intelligent environments, user interface mashups are called physical mashups which are in fact, composite web application following the typical definition of mashups as described above, which also involve smart things and virtual web services [59].

One of the novelties of a UI mashup is that it can integrate user interfaces sourced from the web to the presentation layer. A UI mashup is able to combine various components in the presentation layer of the application stack, while reusing data and synchronization elements from the involved user interfaces. The output of this integration is a newly produced Web application. This approach is very beneficial when the expense to develop a new application is high, or when the user interface of an application is over-complicated. As demonstrated in (see Figure 4 from [59]) the user interface mashup model with inter-component communication and synchronization includes three elements:

Component Operations, which allows central monitoring the user interface components.

User Interface events, which are responsible to inform the other user interface components regarding state changes.

Shared Memory, which is responsible to enable the exchange of data among the user interface components.

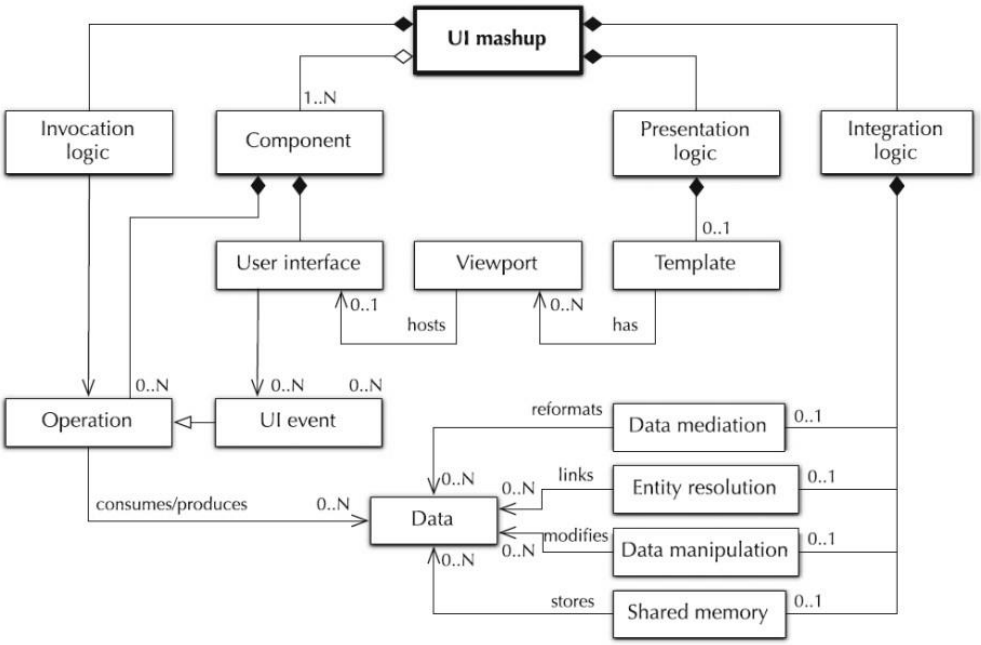


Figure 4: User Interface mashup model with inter-component communication

A simple 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 5). This architecture consists of an 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 widgets 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.

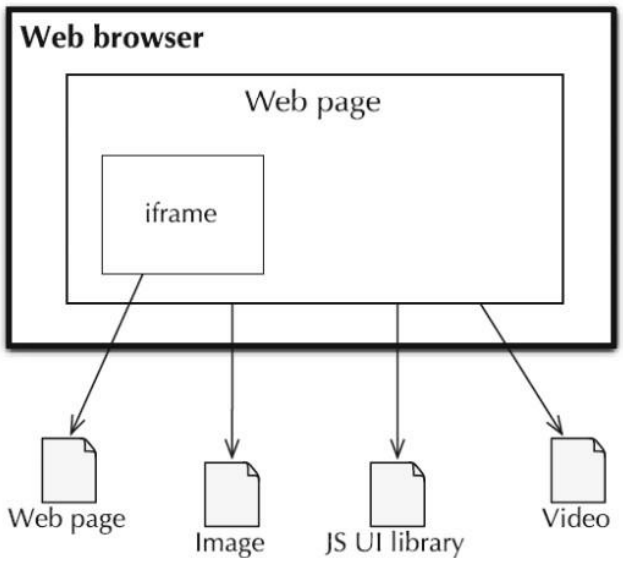


Figure 5: Web - User Interface mashup

2.2 Related Work

Over the years, several research demos and commercial products have been developed, each trying to aid in various types of tasks of a specific environment. Many vendors including Microsoft have tried to enhance the user experience inside the living room by augmenting every surface. Also, many researchers have developed second screen applications that are either companion apps, enhancing the overall experience of a specific system (e.g. a movie player) or even multiscreen systems that share information in order to aid the user in his every day needs. Finally, in terms of tabletop systems, the vast majority of them are created for collaborative purposes, while others explore various usability aspects that stem from the surface computing paradigm.

In this section, several topics of related research relevant to the present work will be reviewed, in order to effectively understand the latest technologies and their limitations which this thesis tries to overcome. In more details, this section is divided in three categories of related work, i.e., intelligent living room environments, surface computing systems and second screen applications.

2.2.1 Surface Computing Systems

Back in 2002, the authors of [62] employed a polar-coordinate system and fisheye view in order to support collaborative interaction on a circular table top

interface that give users the full capability to relocate, re-orient, scale, and layout the documents in the circular interface, as well as support users' focus during collaborative group meetings around the table (see Figure 6). Aside from the fisheye view, this system also provides a graph layout mechanism for displaying information, making it one of the earliest works that support this metaphor. However, the system can only be used free of objects as an occlusion prevention mechanism does not exist.

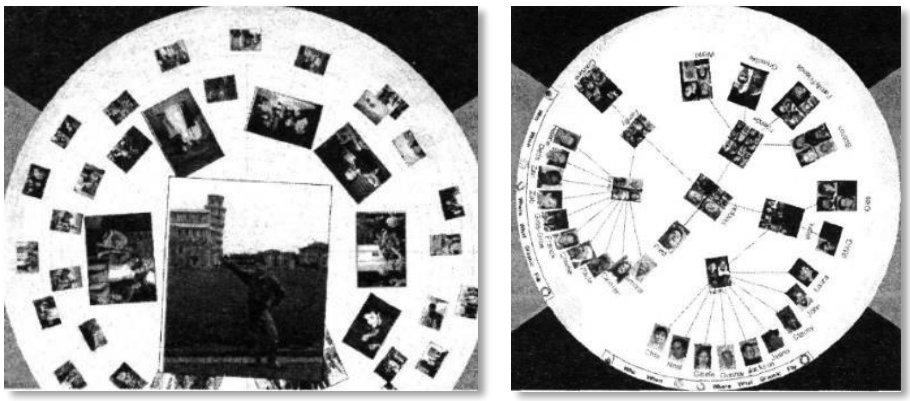


Figure 6: Visualization Techniques for circular tabletop interfaces

MirageTable [63] is an augmented reality curved table that provides virtual 3D model creation, interactive gaming with real and virtual objects, and a 3D teleconferencing experience by enabling real-time stereoscopic 3D digitization of the user sitting in front of the table along with the physical items that are present on its surface (see Figure 7). This particular tabletop system managed to combine physical objects with virtual graphics with the help of augmented reality.

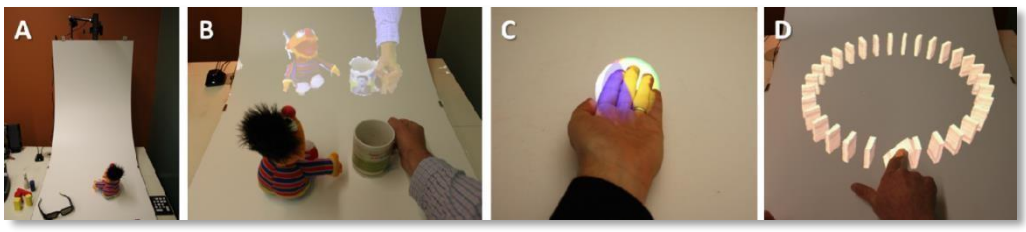


Figure 7 The MirageTable: a curved surface augmented reality system

Drift Table [64] is a coffee table that aims to support ludic activities in the home, such as, for example, geographical landscape viewing (e.g., explore the countryside, travel to a friend's house). It offers a small viewport showing a slowly changing aerial view of the British landscape, while shifting weights on

the table changes its apparent height, direction, and speed (see Figure 8). Amongst others, its authors found that ludic activities increase social engagement and are usually interleaved with everyday utilitarian ones.

Figure 8: The Drift Table

Kirk et al. [65] report the findings of a field study regarding an augmented surface for families that enabled the management (e.g., viewing, archiving) of digital content such as photos (see Figure 9) . People regardless of their age were very much engaged with the table top system paradigm because of the open nature of interaction, which supported multitouch and the integration of physics principles (e.g., photos pushing each other when colliding). This prevented the occlusion of virtual objects in between them, however occlusion prevention by physical objects is not supported by this system.

Figure 9: Kirk et al, interactive tabletop

The work in [66] enables interfacing with a TV set using low-effort gestures detected (using computer vision) over a predefined area of the surface of a coffee table. Lastly, FingerTalk [67], while studying the collaborative decision

making of users interacting cooperatively with a touch-enabled table surface, showed that interactive tables show much promise for supporting flexible and fluid ways of creating and discussing digital documents (see Figure 10).

Figure 10: FingerTalk table - with the calendar interface

According to [68] when the user works on the tabletop next to a physical object and opens a context menu, its size, layout and position can cause it to overlap with objects nearby. In such a case, either the object or the menu has to be moved. Whichever one of these options can disrupt the workflow and possibly lead to a suboptimal workspace arrangement. In that regard the authors proposed a user-drawn path menu which allows a customizable arrangement of the menu items and is therefore easily adaptable to the available tabletop display space (see Figure 11). That approach resulted in the reduction of time using menus. However, menus are not the only user interfaces that can be occluded by physical spaces.

Figure 11: Interactive tabletop using User-Drawn Path menus

The authors of [69] have introduced a new metaphor which stands for interactive environment-aware display bubbles, that aims at significantly

enhancing the flexibility, interactivity and adaptivity of displays (see Figure 12). Interactive bubbles can be instantiated anywhere on demand. Their freeform shapes, the displayed content and the illumination can freely be redefined using a laser pointer. The novel display metaphor is well suited for a wide range of use cases and applications including: Collaborative design, engineering, modeling and visualization, presentation meetings, edutainment and brainstorming sessions.

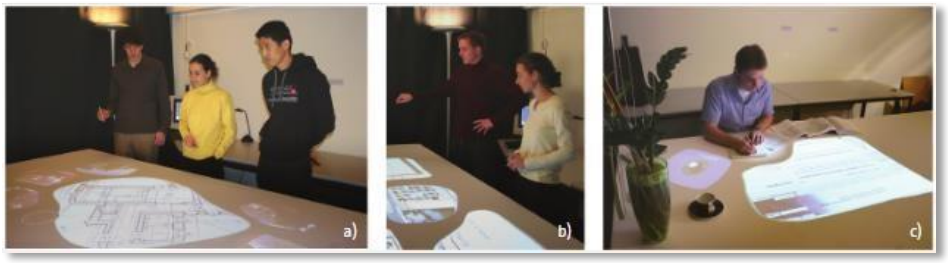


Figure 12: Interactive Environment-Aware Display Bubbles

Concurrent interaction with physical and digital media is ubiquitous in knowledge work. Although tabletop systems increasingly support activities involving both physical and digital media, patterns of use have not been systematically assessed. So, the authors of [70] present a hybrid tabletop that includes physical and digital media, while they address issues about interactive space management which is done by efficiently managing tabletop space and effectively assisting semantic organization which commonly involves creating groups of items (see Figure 13). Other than that, this system does suffer from the physical object occlusion problem, although the authors state that only a small number of studies have considered the problem of occlusions generated by physical objects on a display surface which is important to note and is considered a bothersome phenomenon.

Finally, aiming to address the overall occlusion problem that takes place on heavily cluttered table tops, [71] introduces an access-supporting occlusion management algorithm that identifies the visible regions of the display suitable for showing digital content (see Figure 14). Currently this system only provides an algorithm for occlusion detection and user interface placement recommendation and have not evaluated their results with actual displayed applications.

Figure 13: Hybrid tabletop including physical and digital media

Figure 14: Occlusion Prevention with placement recommendation

On the other hand, SnapRail [72] recognizes the occluding physical object's footprint and rearranges the occluded virtual elements over a ring-shaped rail widget that appears around the physical object. Their interface identifies the virtual elements under a physical object on a tabletop surface and places them along the ring-shaped rail widget. The user can also manipulate the virtual elements by using with the widget. Their prototype implementation has some technical limitations. The touch surface is not sensitive enough and as a result the system cannot distinguish a small physical object from a finger. Also, the size of each rail widget is fixed at the time it first appears and cannot be altered. Finally, virtual space management is another issue, as user interfaces are placed in an absolute manner and can occlude one another.

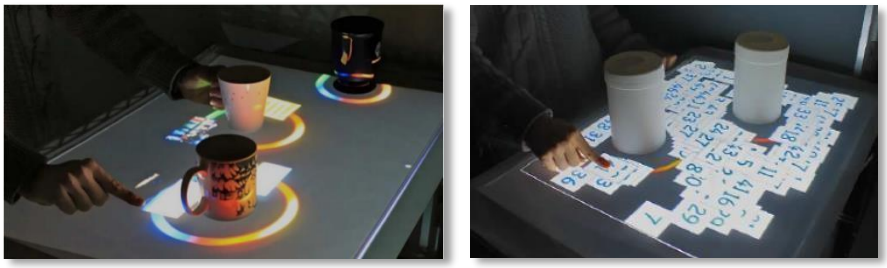


Figure 15: The SnapRail widget, managing the occluded user interfaces

In addition to research prototypes, many vendors have commercialized such devices [73], [74], [75]. However, these systems are mostly large monitors enabling touch interaction and pressure detection, which are intended to be used as displays for a computer as they do not provide any dedicated software. On the other hand, in 2005, Microsoft released its Surface Table along with the PixelSense Interface [54] which offered four main components: (i) direct interaction, multi-touch contact, (iii) a multi-user experience and (iv) object recognition. The software came prepacked with basic but customizable applications including a photo viewer, music player, virtual concierge and some mini-games (see Figure 16).

Figure 16: Microsoft Surface Table with Pixelsense Interface

2.2.2 Second Screen Applications

In [34], the authors created a companion application for multi-episode series that creates story arcs. Their approach is focused specifically on TV series and uses the second screen to support the contents of the main screen with supplementary commentary and character progression (see Figure 17). However, there is no bidirectional interaction between the companion app and the main device.

Figure 17: Second screen use cases

GRASP [76] presents a spatially aware mobile and wall display visualization environment that utilizes the enormous space of large displays and adds to this the flexibility, display capabilities, and interaction modalities of mobile devices (see Figure 18). They specifically focus on graph visualization, exploration and manipulation and on how graph-specific tasks can be supported with a combination of smartphones and physical surfaces; based on the findings of an evaluation study, the authors highlight the benefits of distributed interaction among combined displays, such as mobile and wall displays.

Figure 18: The GRASP system - a mobile toolbox as a second screen

The authors of [77], created a multi-screen cloud social TV system that consists of a video watching app that supports local and remote users, where they can even collaboratively edit the original content by inserting text, pictures and audio in order to generate new content, as well as “video teleportation” functionality between the devices present in a living room (see Figure 19). It adopts the cloud computing paradigm to encapsulate media services in the backend and provides attractive multi-screen and social features in the front-end.

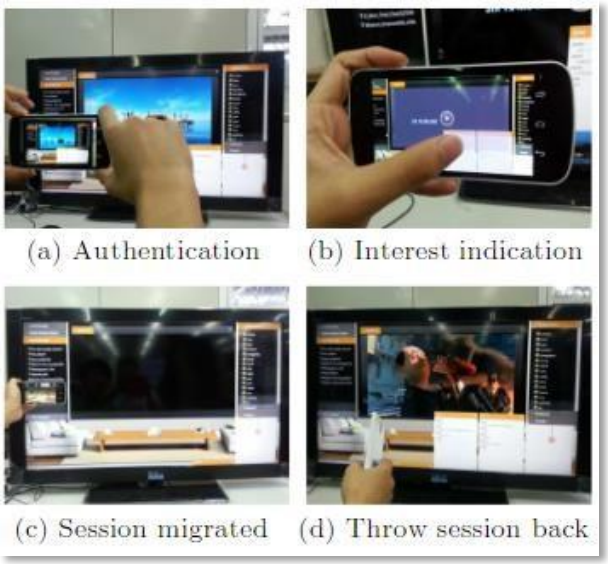


Figure 19: Multi-Screen Cloud Social TV

The authors of [78] present a second screen approach that offers the opportunity to extend the model of a smart TV operation by offering a far richer interaction mechanism than conventional remote-control devices (see Figure 20). It also allows for the dynamic reconfiguration of the UI to reflect a user’s needs and desires. This could be a matter of personal preference or could represent a major enhancement of possibilities for those with significant impairments which affect their interaction with smart TV’s services. With a move away from infrared communication between TV/set-top box and remote, the control device would not have to be pointed directly at the television set to function, allowing it to be used in a more casual ‘lean back’ position or even from a different room. This is another facet to smart TV where broadcast programs and TV services co-exist in parallel, rather than one having to displace the other on a single television display (see Figure 20).

Figure 20: Second Screen and remote controller on mobile device

HouseGenie [79] is a universal monitor and controller of networked devices through touch-screen phone in smart home environments (Figure 21). It enables the users to intuitively monitor and control all the home devices in 2D-panoramic view. It also provides a direct-manipulation home controlling experience, where people could control all devices in order to fulfill their tasks through simple “click” and ”drag & drop” actions. It also supports several multiple modalities in controlling the devices. Finally, it allows users to monitor and control the networked devices of their home anytime and from anywhere.

Figure 21: HouseGenie - a second screen universal monitor and controller

In addition to research prototype systems, many commercial applications are available offering second screen utilities to enhance the experience of certain TV programs, as well as in other areas such in education through eLearning applications.

The EuroVision Song Contest [80] provides is a mobile second screen application (see Figure 22) which offers extra functionality that enhances the viewer's experience during the contest exhibition by offering: (i) information about the participants with all the latest news and videos, as well as details and commentary about their journey to the contest's country, (ii) a voting system that is available in the voting session of the contest where the users cast their vote about their favorite song, and (iii) live streams and a camera mode with Eurovision Selfie filters.

Figure 22: EuroVision Second Screen Voting Application

LiveRugby [81] is a second screen application for sports enthusiasts who want to know more about a rugby match in real time. The system provides a details section with real time game statistics, a live overview display of the field, live commentary and a phase replay system which is able to replay past phases like penalties and touchdowns. Moreover, the application is supplied with a schedule section that displays all the past, current and upcoming rugby matches along with their outcome score.

Finally, Kahoot! [82] is a game-based system hosting a variety of learning games and trivia quizzes, in the form of multiple-choice questions. The content of these quizzes enhanced with videos, images or diagrams are hosted on a TV system acting as the main screen, and each contestant uses their own mobile second screen Kahoot! app in order to answer.

Figure 23: LiveRugby Second Screen Application

Figure 24: Kahoot,an educational oriented quiz system

2.3 Discussion

Evidence from the present 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 tabletop user interfaces, an approach that incorporates the state-of-the-art technology such as Web of Things and UI mashups is required. Furthermore, based on the related tabletop systems that where studied, occlusion prevention, space management and application hosting are three significant challenges that are not addressed in any of these systems in conjunction. Towards contributing in this direction, FLUID SDK proposes a mechanism which enables the embedding of HTML UI mashups of published web applications from the smart home in a way which makes it possible to use them in surface computing

systems flawlessly along with common physical objects, preventing occlusions, user interface cluttering and ensuring an 'overall' stable user experience.

		Applications			Surface Computing		Service	
		Hosts Applications	Multi-Screen	Tabletop Display	Occlusion Prevention	Space Management	Configurable Behavior	Provides API
Surface Computing Systems	**Circular Tabletop**	✓		✓		✓	✓	
	MirageTable	✓		✓	✓			
	DriftTable			✓				
	Kirk Interactive Tabletop	✓		✓		✓		
	FingerTalk Table	✓		✓				
	Path-menu interactive Table			✓		✓		
	Interactive Display Bubbles	✓	✓	✓		✓	✓	
	Hybrid Media Table			✓	✓			
	Placement Recommendation Table			✓	✓			
	SnapRail	✓		✓	✓			
	Microsoft Pixelsense	✓		✓	✓	✓	✓	
Second Screen Systems	**GRASP**	✓	✓				✓	
	Cloud Social TV		✓				✓	
	iTV Remote	✓						
	HouseGenie	✓	✓				✓	
	EuroVision Song Contest App		✓					
	LiveRugby							
	Kahoot!		✓				✓	
	FLUID SDK	✓	✓	✓	✓	✓	✓	✓

Table 1: Related work compared to FLUID SDK

2.3 Discussion

Chapter 3

Requirements Elicitation

FLUID SDK's philosophy revolves around designers and developers, hence it was of foremost importance to focus on facilitating their wants and needs, as well as making a scalable system. Requirements of this thesis started with the AugmenTable concept, a smart coffee-table that was created as a computer science project for Human-Computer Confluence course which was lated deployed in the Intelligent Livingroom of Human Computer Interaction Laboratory of ICS-FORTH. AugmenTable sparked the idea of having a tabletop environment where digital applications could coexist naturally with physical objects. That along with motivating scenarios that were presented, led to the system requirement elicitation. The presented project of this thesis aims to support developers of smart environments that need to create a graphical user interface (GUI) application that will aggregate multiple user interfaces available in the environment.

3.1 The AugmenTable coffetable

AugmenTable (see Figure 25) is a stylish commercial 52 × 35 inch coffee table made of wood with a smooth, non-reflective white finish, which in addition to its intended use for placing objects on top of it (e.g., cups, plates, books), acts as a large projection area where secondary information can be presented from a small form factor projector embedded in the ceiling. Its physical attributes (i.e., color, dimensions) and placement (i.e., in front of the couch) enable users to clearly see the projected content on demand without being overwhelmed when interacting with the main display (i.e., AmITV [83]).

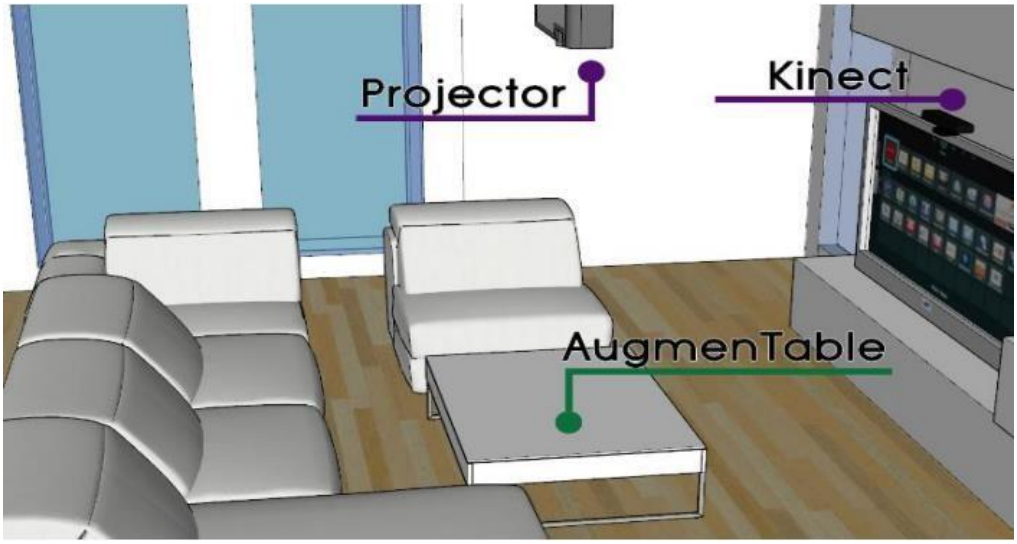


Figure 25: A 3D representation of AugmenTable coffee table

Through a second Kinect sensor installed on top of the TV, facing directly at the table's surface, AugmenTable becomes a touch-enabled surface that can recognize the physical objects placed on it. Additionally, a vibration motor is located under the table in order to provide haptic feedback to the users when deemed appropriate, e.g., deliver a silent alarm that indicates food delivery is on its way (e.g., three short vibrations) when the TV is playing a critical scene and the notification service decides not to explicitly interrupt the user. Finally, the force-sensitive resistors and load sensors embedded in SmartSofa's seat and back permit the monitoring of the seated users as well as of their posture while seated. Depending on whether the user leans toward the table, relaxes on their back, or lies down completely, the information displayed on AugmenTable, as well as the available interactive controls, adapt their appearance and/or their location to better accommodate the user's needs (e.g., interactive controls remain hidden when out of reach, the brightness level is set to low when the user is not looking toward the AugmenTable).

3.2 Motivating Scenarios

Following the process of requirement elicitation, one of the most widely used methods that can systematically contribute to the process of developing requirements is personas and narrative scenarios [84]. In the case of the

conceptualization and design of the “FLUID SDK”, was decided on two factors, (i) the proof of concept of AugmenTable that was mentioned earlier and (ii) a set of motivating scenarios which would offer an opportunity to empathize with potential users (i.e., personas) and express ideas in specific contexts.

George - the persona of the following scenarios, is a full-stack engineer member of an engineering team that creates tabletop applications and his current job is to develop and test a surface computing and second screen application about a movie player intended for an Intelligent Livingroom.

Scenario 1. Deploy mashup layouts on physical surfaces

George has created some components (e.g. play/pause, volume up/down, etc.) that act as a controller of the TV’s movie player. The goal is to use FLUID SDK to create a menu that contain these components, so as a first step he follows the documentation that is provided as a guide in using the system’s APIs. With the help of the API’s documentation, George developed a mini-application consisted of a menu with the aforementioned components and by using the FLUID SDK API he manages to send his application to be displayed on AugmenTable coffee-table. Following the same procedure George develops another mini-application and with the help of FLUID API he sends his new application to be displayed on the smart wall.

Scenario 2. Simulate a use case scenario of the movie player

George has developed an early version of his application. Now, he wants to test the behavior of his system by simulating some use cases with physical objects. The scenario that he has created is about a man watching his favorite movie while eating dinner. George has installed and launched his system. Then he decides to add one plate, one cup and one water bottle. The user interfaces react accordingly by evading the physical objects that were placed. However, George needs one more item to place in order to simulate the use case, which of course is not available in the laboratory. This item is a pizza. So, George decides to use the simulation tools that are provided with the system to add a virtual one. After adding such a large object, George noticed that some user interfaces were hidden due to insufficient space, as the table hosted many physical objects.

Scenario 3. Evaluating the logic and usability of the movie player

Later that day, George decided to test the decision making of his system by evaluating the results from the use case test he made earlier. So, he uses the logging tool of FLUID SDK to see which items were hidden during that time and why. He notices that some of these elements were not supposed to get hidden and through log inspection. George found that the elements that got hidden were the pause button and the volume-down button from the movie player's controls interface. Normally, the movie player's controls should be always present on the screen, as their importance is rather high, however the system decided to hide those elements when the table's virtual space got low. To that end George altered the importance property of these elements through the 'Manager App' and repeated the same simulation. Upon placing the physical objects on the table, the system's decision was to keep the controls present.

3.3 System Requirements

The main objective of the system is the orchestration of the applications by integrating them in a complex User Interface, namely UiCollection, that is used to create adaptive, context-aware and multi-modal user interfaces. FLUID is a software development kit consisted of three main components which when combined together, implement the above functionality and bind all the individual applications together. The basic functionality of the SDK includes: (i) Application Hosting, (ii) Physical Occlusion Prevention, (iii) Desktop Space Management, (iv) Surface-Migrating User Interfaces, (v) Application control through REST APIs, (vi) Monitoring and Simulation.

3.3.1 Application Hosting

The intelligent living room environment of AmI laboratory of ICS Forth already hosts smart home applications on the TV [83], the smart wall above the TV and the coffetable [85], offering a user-friendly experience. Across the various artifacts, these applications need to communicate and collaborate in a seamless manner, leading the users into thinking they are interacting with an ecosystem, rather than a group of isolated units. Each application should be able to launch on demand on every artifact it applies, while also being capable of transitioning between them without losing its state. A custom application hosting system capable of facilitating applications in a dynamic manner is

required. The SDK must provide a set of functions dedicated to the developers of the Intelligent Home, which enable the distribution and control of their applications to the surface artifacts of the intelligent living room.

3.3.2 Physical Occlusion Prevention

As discussed in [72], [85], a table is a place where people perform daily activities such as having coffee, reading a book, and enjoying a hobby. During those activities, people place various objects, such as paper, books, and coffee mugs on the table. Tabletop computers and displays are expected to enhance these activities. There is, however, a fundamental problem with tabletop computers: physical objects often hide virtual elements on the tabletop display, making it difficult for the user to see (visibility) and handle them (manipulability). As a solution the SDK must support collision detection between the occlusions of the physical objects that reside on the surface and the application components. Then the system must automatically move the elements outside the occlusion zones of the physical objects, thus preventing the occlusion problem.

3.3.3 Desktop Space Management

Another fundamental problem with tabletop computers, which can also be noticed on the tabletop systems that are discussed (in Chapter 2.1.3), is “application cluttering”, which occurs when a surface hosts a big amount of user interfaces at the same time. This particular problem can dramatically hinder the usability of the system resulting in an unpleasant user experience. To ensure stable usability levels, the system must provide a desktop management algorithm that is able to hide specific user interfaces when a certain threshold level of space availability gets surpassed. The application components which get hidden by the algorithm must be chosen based on their significance value, which is determined by the developer.

3.3.4 Surface-Migrating User Interfaces

After addressing the Desktop Space Management problem several user interfaces may get hidden, resulting in information loss. To address this issue, the SDK should be able to provide to the developer functionality which enables the migration of user interfaces to other surfaces of the smart home. This way,

hidden interfaces of the table can be moved to the nearest available host (e.g. Smart Wall).

3.3.5 Monitoring, Simulation and Evaluation

According to literature, the development stage of software applications is always accompanied by software errors, which in most cases hinder the process of completing a development task [86]. But debugging is hard. Debugging distributed applications is still harder, and debugging distributed applications deployed across the Internet is downright daunting. The authors of [86] state that the development of new services has been held back by this difficulty and that more powerful debugging tools are needed. Simulation and small-scale test deployments help developers evaluate aggregate system behavior in a relatively easy environment. With a simulator, the developer or the designer of a FLUID application has full power to repeat the same execution process across multiple experiments, and the state of each application process is available locally for examination. In order to assist the designers in the process of creating and testing their applications, FLUID SDK provides a set of monitoring and simulation tools that enable the control of the hosted tabletop user interfaces and simulating situations with several physical items.

Chapter 4

The FLUID SDK

The FLUID SDK consists of two main components, namely the FLUID Host and the FLUID Server, as well as the FLUID Manager utility tool.

The FLUID Host is the first main component of the SDK. It acts as the operating system and is responsible for displaying the end-user application on physical surfaces and also act as an interaction medium between end-users and applications. It is also responsible for preventing occlusions when the user places objects on-top of the host's physical surface. The FLUID Host has been created using the Angular 8.0 framework.

The FLUID Server is the second main component of the SDK which is responsible for enabling communication between end-user applications and the FLUID Hosts. In order to enable this functionality, it provides two distinct API services. The HOST-API, which is exposed as an AmI-Home service that is used by the intelligent artifacts to set the host's physical context (e.g. Physical objects, user position, etc.), and the UI-API which is used by the end-applications in order to be displayed to the host's surface and be properly controlled. The FLUID Server is equipped with a database system in order to store and load all host data. The FLUID Server has been created with Node.JS-Express technology and the database system is a MongoDB.

The FLUID Manager is a utility application that is built in order to provide a set of monitoring, debugging and evaluating tools to the designer/developer of the end-user applications. It is considered an end-user application; thus, it uses the HOST-API and the UI-API to monitor and control each and every aspect of all FLUID Hosts that are deployed on the intelligent living room. The FLUID Manager has been created using Angular 8.0 framework.

As illustrated in Figure 36, the end-user applications that are displayed on physical surfaces can be hosted on multiple instances of the FLUID Host. The hosts communicate with the FLUID server via HTTP requests and Web Sockets to send and receive data in JSON format. When an HTTP request is performed, the server stores the appropriate information to the database and fires an event (see section 5.4.4). For example, when a particular host has insufficient physical space to display the currently open visual elements, it hides some of

them. This action triggers an event which is received by the application controller of the visual elements that got hidden in order to respond with an action (e.g. a logic controller of the movie application is configured to show the hidden user interfaces of the table to the smart wall above the TV).

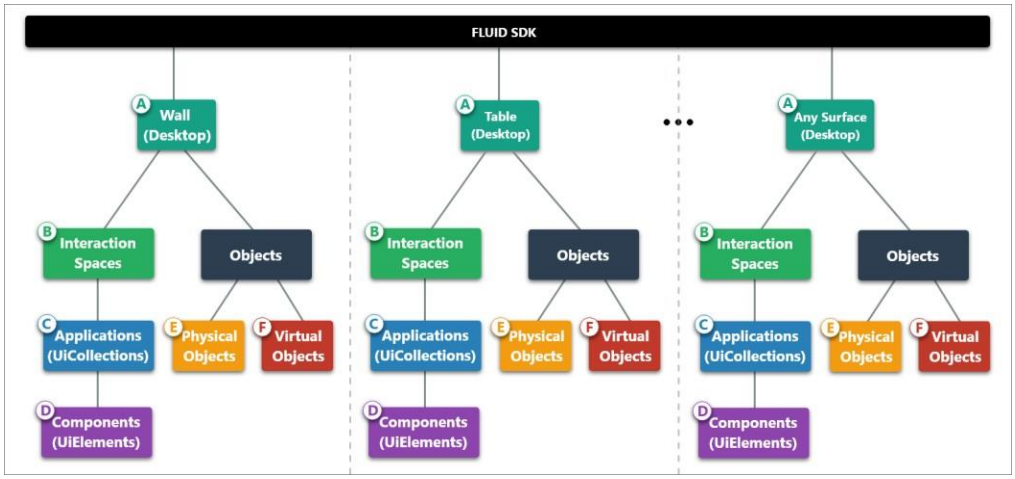


Figure 26: Representation of the FLUID SDK Desktop components hierarchy

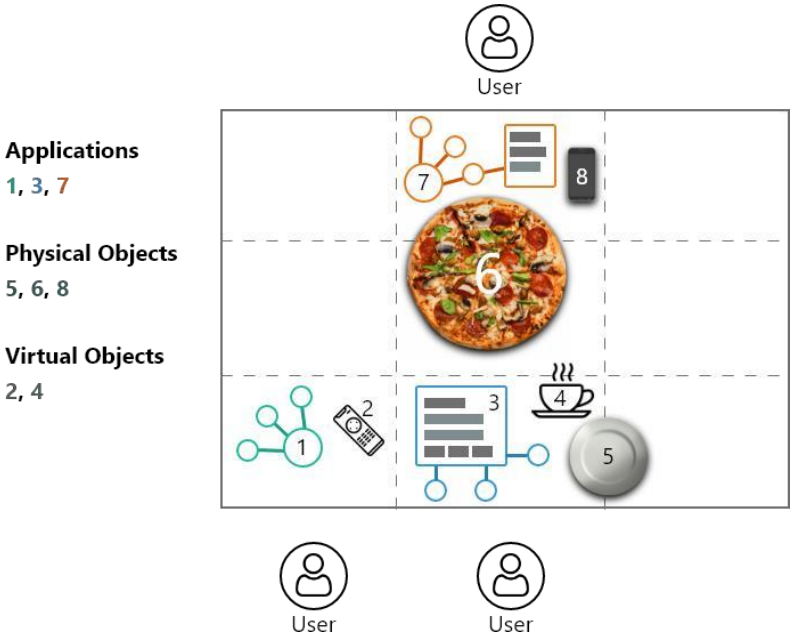


Figure 27: FLUID Host (Desktop) example

4.1 Fundamental Components

Given the aforementioned issues regarding surface computing (see section 2.1.3), following approach focused on creating a sophisticated application host - appropriate for large displays - that supports the projection, the arrangement and the management of AmI Home's surface computing applications.

Towards that direction, five concepts are introduced, namely (A) Desktop, (B) Interaction Space, (C) Applications, (D) Physical Objects, and (E) Virtual Objects.

A. **FLUID Host (Desktop).** A Desktop refers to the entire surface that is used to project applications. In more detail, the desktop's components can be projected on one or more physical surfaces. Each surface displays their own instance of a Desktop.
B. **Interaction Space**. An interaction space is an area of a surface which can host and manage application components and is intended for one end-user. Each Desktop is split in several interaction spaces in the form of a grid (see Figure 27).
C. **Applications**. The Ami-Home provides a plethora of applications that are available for use in the Intelligent spaces. They are introduced in the environment though an Application Repository that is part of the Unify studio [85]. These GUI applications are mainly exposed as partial applications called Application Components or Widgets, that are autonomous and can coexist in conjunction to other applications.
    i. **UiElement**. A UiElement is a single Application Component that is part of an application. This can be either a window or a small widget which consists of a button or some information. Finally, a UiElement supports collision detection with other elements, thus avoiding occlusions between UiElements and objects (physical and virtual).
    ii. **UiCollection**. A UiElement can be related to one or more UiElements in terms of parent-child relationships. Those that are related are displayed together, bound by edges, thus creating a graph layout representation. Such a group of UiElements is called UiCollection. An application hosted in a Desktop consists of one or many UiCollections.
D. **Physical Objects**. Normally nearly all physical surfaces of a home may host several household items such as vases, photo frames, silverware and plates placed on top of tables, paintings, frames and clocks pinned

on walls and so on. These physical objects occlude parts of a surface; thus, making it is unsuitable for displaying applications on these occluded zones. A “physical Object” is the occluded space of a physical surface.

E. **Virtual Objects.** A Virtual Object is the simulation of a Physical Object and is used by the AmI-Home developers to simulate use cases, so they can test the behavior of their applications when an Object Detection System is not available.

4.2 FLUID Host (Desktop) Functionality

The FLUID Host (here called Desktop) can host second screen applications. For example, in the context of the intelligent living room, the Desktop used on the coffee table hosts applications which provide supplementary information and control to the corresponding main screen application which runs on the iTV (e.g. movie player, radio player, RSS feed, email Viewer, Weather viewer, home monitoring and control). A Desktop is organized into interactive areas, where each one of them is intended for one end-user and offers a set of functional capabilities which can be partially or fully exploited by the end-application developer. The next part of this section will address the functionality that is provided to the end-user of FLUID empowered application.

General Functionality

The end-user during the use of a FLUID-Host-enabled surface has the opportunity to manipulate and organize the position of the application components to his/her liking (e.g. drag the lights-control application closer, in order to use it) and even send some applications to other FLUID-Host-enabled surfaces that are close (e.g. send the weather application to the smart wall). Additionally, the user can expand certain application components in order to view more information and/or collapse other expanded components in order to reserve space. The application developer can provide all FLUID SDK’s functions to be used by the end-user explicitly in the form of commands through user interfaces (e.g. user expands the ‘home control menu’ clicking on it) or be provided implicitly by the application’s logic controller in the form of events (e.g. the ‘movie player control menu’ gets expanded automatically when a movie has ended).

In more detail, the available functional capabilities that can be provided through a FLUID application are:

Expand/Collapse. Applications deployed to a FLUID Host are displayed in a graph layout representation (UiCollection). Moreover, each application component (UiElement) is actually a graph-node that has a number of parent nodes and children nodes. The end-user is able to select a specific node and expand it in order to view its children nodes, or do the opposite, collapse it in order to hide its children and conserve space. By default, the end-user is able to perform this functionality by clicking on a UiElement, although this can be overridden by the application developer and can be actually provided internally from their applications (e.g. through a button which is located inside the application component).

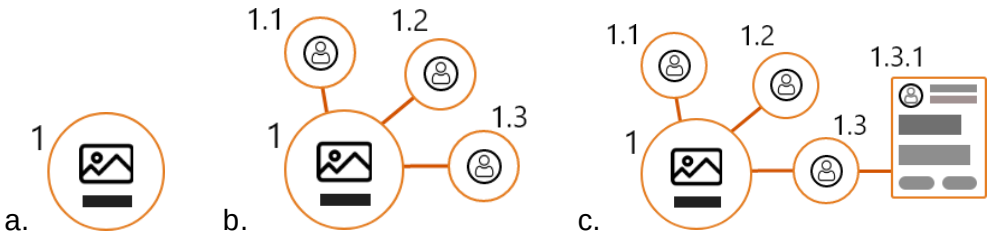


Figure 28: Expand/Collapse application components (UiElements)

***a:** Fully Collapsed, **b:** Expanded Node 1, **c:** Expanded Node 1 and Node 1.3*

Rearrange. Having the ability to drag & drop UiElements, the end-user can manually rearrange the displaying applications of a Desktop to his liking. Additionally, this feature can be further exploited by the application developer by providing a number of predefined positions of his/her UiCollections for the user, accessible through the application's user interface. This enables the end-user to choose and auto-arrange the application components accordingly.

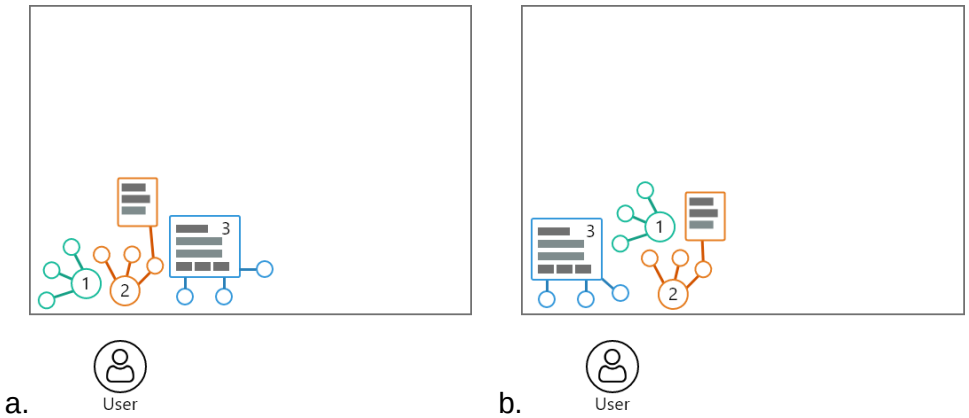


Figure 29: Rearrange applications (UiCollections)

***a:** Initial position of UiCollections, **b:** New Positions after 'rearrangement'*

Focus. This feature is provided automatically by the system. The board exploits the information coming from the reachability and position services. By combining this information indication of whether the content of the desktop should be rearranged is provided. For example, AugmenTable uses the smart couch user posture detection to apply 'focus' to the UiElements when a user leans over the coffee-table, thus, all application components move towards the user in order to be reachable.

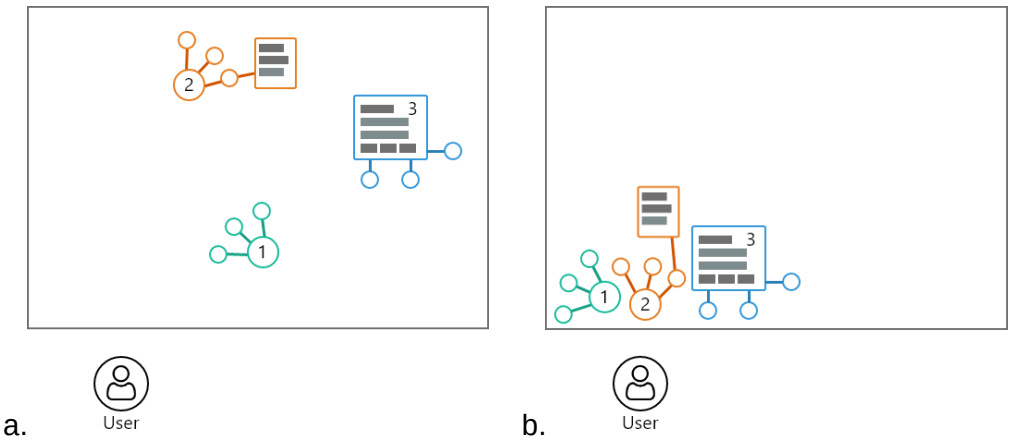


Figure 30: Focus applications (UiCollections)

***a:** Initial state, **b:** 'Focused' state*

Pin. This function allows the developer to ‘pin’ certain applications to specific positions on a Desktop. The ‘pin’ feature is useful to situations where an application is ‘read-only’ and does not provide interaction to the user (e.g. notifications), thus the developer may choose to bind such applications away from the active interaction spaces of a Desktop. Alternatively, the developer may exploit the ‘pin’ function and provide a ‘pin’ button inside an application component (UiElement), in order to be used by the user on demand.

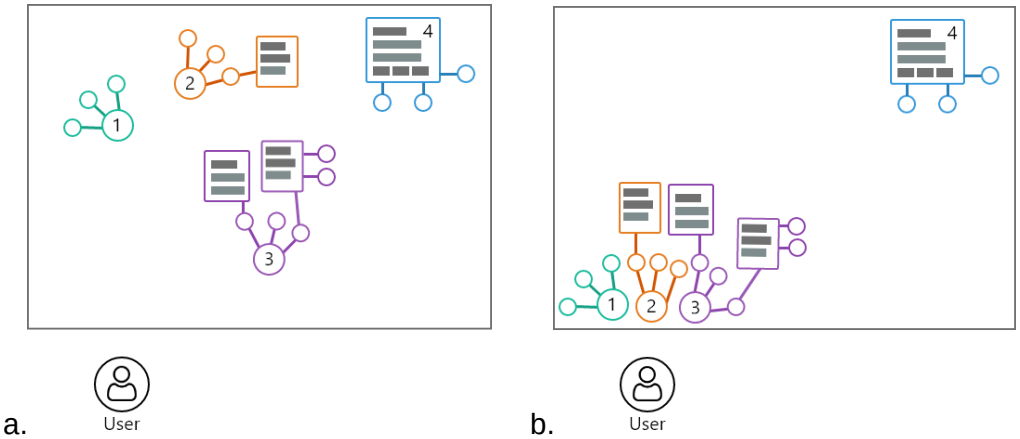


Figure 31: A 'pinned' application (UiCollection)

***a:** Initial state, **b:** After ‘focus’, application-4 did not move because it’s ‘pinned’*

Resize. This feature allows the developer to ‘resize’ certain applications to specific proportion. The ‘resize’ function is useful in situations where applications are further away from the user. In that regard the developer may scale-up an application in order to help text information become more readable to the user. Alternatively, the developer may exploit the ‘resize’ function and provide a ‘resize’ button inside an application component (UiElement), in order to be used by the user on demand.

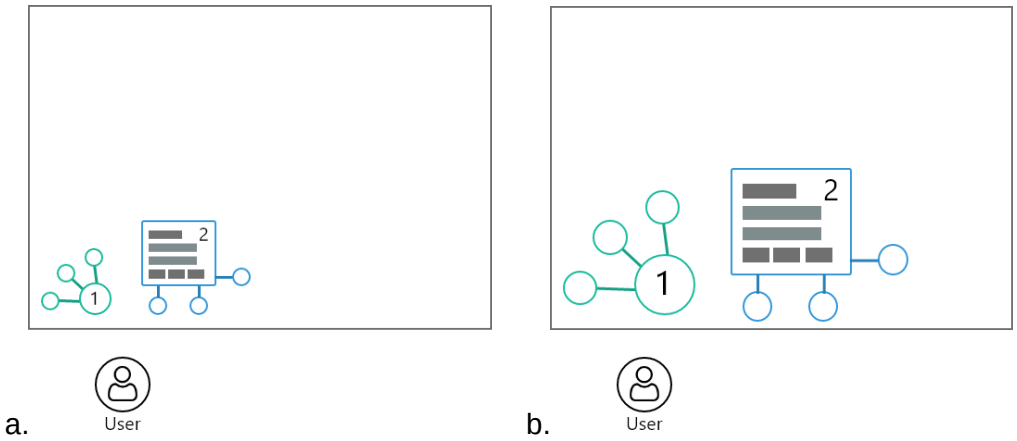


Figure 32: 'Resize' applications (UiCollections)

a: Initial dimensions, b: New dimensions after 'Resize'

Alternative View (Split). As discussed earlier, UiElements can range from a simple widget containing a button, to a full window containing sections of information and application components. However, a window can be large and cumbersome in situations where the surface contains objects. In that regard the developer can provide different layouts of an application which can be either selected manually by the user, or be applied when an event fires (e.g. the email tabletop application is a window which splits into a graph layout UiCollection when the table gets overcrowded with objects).

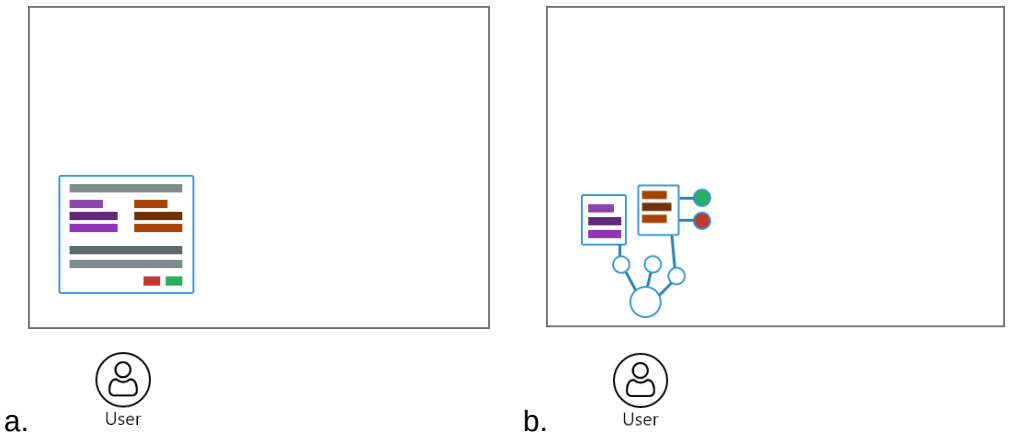


Figure 33: Alternative View (Split) of an application

***a:** Default View, **b:** Alternative View 'Split'*

Migrate. This feature enables the migration of UiCollections from one available Desktop to another. Application migration is very useful when a surface has insufficient interactive space to display user interfaces, thus the application developer can utilize this functionality to send an application (UiCollection) to another available Desktop that is present in the intelligent space.

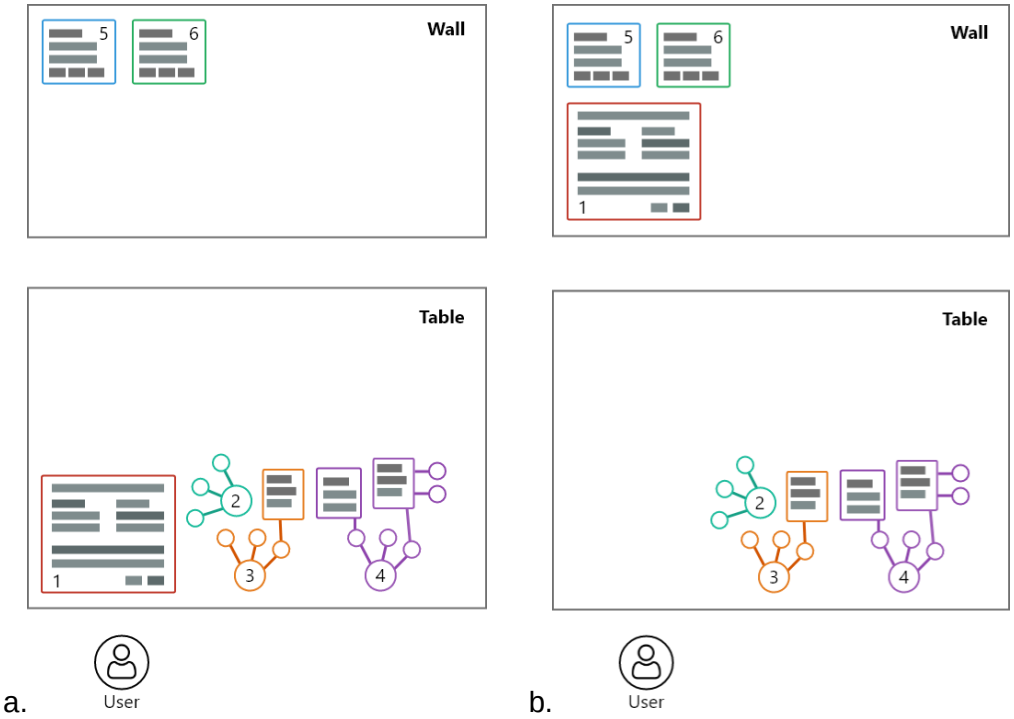


Figure 34: Migration of an application (UiCollection) from a Desktop to another

***a:** Initial state, **b:** Application-1 'migrated' from the table to the wall*

Bind to Object. In general, UiElements are bound together with edges in a parent-child relationship, forming a graph layout (a UiCollection). Aside from that, FLUID SDK supports binding between Objects (Physical & Virtual) and UiElements. This feature becomes handy in situations where the developer wants to augment specific objects with information (e.g. in the smart kitchen bench, when the system detects a pot on the stove, it automatically binds UiElements to the pot containing information about the stove's status).

4.2 FLUID Host (Desktop) Functionality

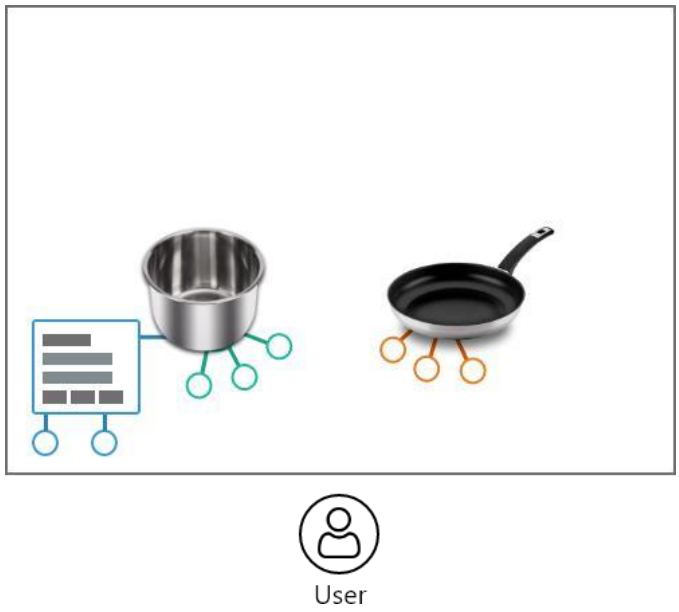


Figure 35: Applications (UiCollections) bound to Physical Objects

UiElement	
Functionality	**Description**
Move	Changes the position of the UiElement
Resize	Changes the dimensions of the UiElement
Pin	Pins the UiElement to a specific position. The end-user cannot move the UiElement while pinned
Bind	Binds the UiElement to a specific Physical or Virtual Object (becomes child node of an Object)
Show/Hide Border	Set's the visibility property of a UiElement's border
Change Border Color	Changes the UiElement's border color

Table 2: Functional Capabilities of an application component (UiElement)

UiCollection	
Functionality	**Description**
Migrate	Moves a UiCollection from one Desktop to another
Focus	Moves a UiCollection towards the end-user
Split	Splits a UiCollection into two or more UiCollections
Show/Hide Edges	Set's the visibility property of the UiCollection's edges between its UiElements
Change Edges Color	Changes the color of the UiCollection's edges

Table 3: Functional Capabilities of an application (UiCollection)

Virtual Object (Developer only)	
Functionality	**Description**
Resize	Changes the dimensions of the Virtual Object
Change Image	Sets the image of a Virtual Object

Table 4: Functional Capabilities of a Virtual Object

4.2 FLUID Host (Desktop) Functionality

Chapter 5

The FLUID SDK Implementation

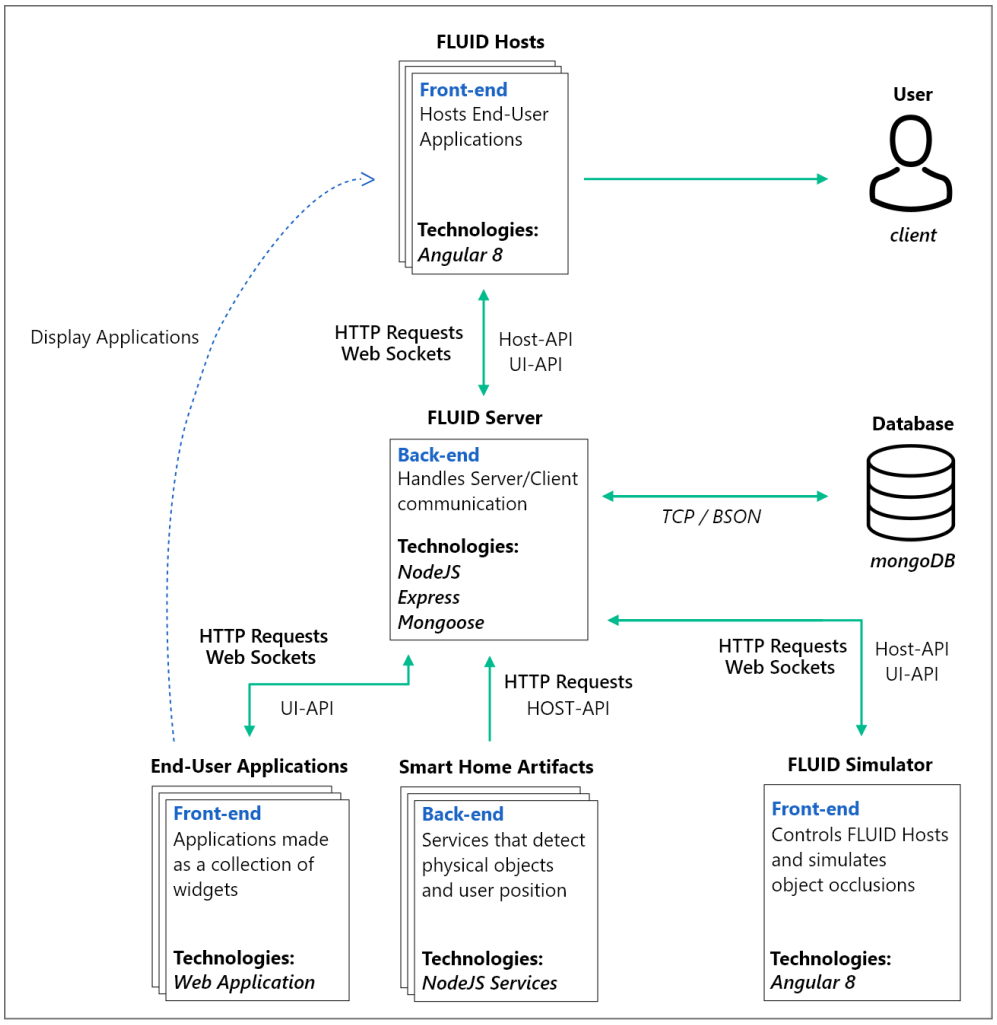


Figure 36: The FLUID SDK

5.1 Data Modeling

One of the most important tasks for the development of FLUID SDK was to conceptualize an entity model schema for the system platform. The system

uses MongoDB as a means of saving its current state. Data retrieved by the database is in JSON format and can be saved in document style [87] (see Figure 37). These documents contain actual data used by FLUID SDK and can vary in number of fields, size and content (e.g. a file that contains the state of an application which gets displayed on a FLUID host inside the intelligent living room).

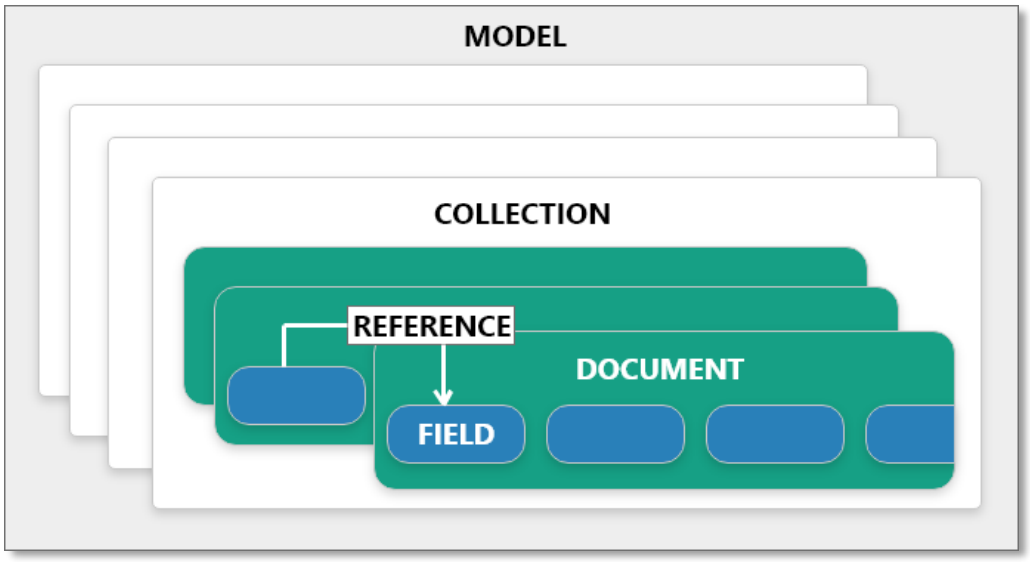


Figure 37: The document style of mongoDB

FLUID SDK uses mongoDB [88], [89] as the schema used is scalable and does not need a defined structure beforehand, which is perfect for modeling data and the relationships between them which can be altered easily whenever needed.

5.1.1 FLUID SDK Entities

After the collection of functional requirements, the basic entities have been extracted and are described in detail in this section (see Figure 38).

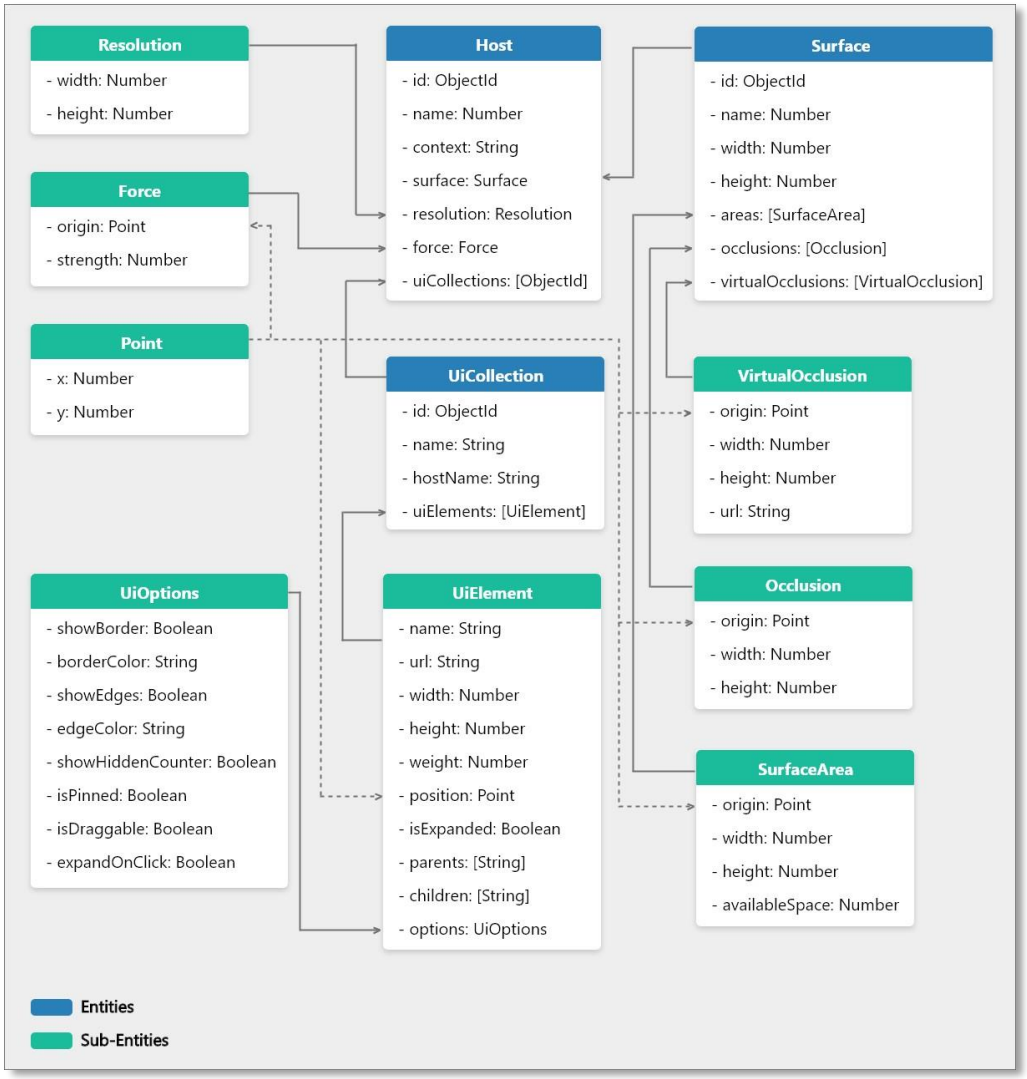


Figure 38 The conceptual data model of FLUID SDK

Point. The Point is a sub entity that describes a two-dimensional point in pixel format.

Field	Type	Description
x	Number	The point's 'x' position in pixels.
y	Number	The point's 'y' position in pixels.

Table 5 The Point Sub-Entity Model

Resolution. The Resolution is a sub entity that describes the resolution of a screen surface in pixel format.

Field	Type	Description
width	Number	The resolution width in pixels.
height	Number	The resolution height in pixels.

Table 6 The Resolution Sub-Entity Model

Force. The force is a sub entity that describes the force that is induced to application components that are displayed on a surface in order to move them in the force's direction.

Field	Type	Description
origin	Point	The force's point of origin.
Strength	Number	The force's strength.

Table 7 The Force Sub-Entity Model

SurfaceArea. The SurfaceArea is a sub-entity that is part of the Surface entity. A Surface is split in interactive and non-interactive zones. Each zone has an available space percentage counter that will eventually assist in the decision-making process of the space management Algorithm which will hide uiElements in order to maintain the space availability.

Field	Type	Description
origin	Point	The area's origin point (top-left).
Width	Number	The area's width in pixels.
Height	Number	The area's height in pixels.
availableSpace	Number	The area's available space percentage.

Table 8 The SurfaceArea Sub-Entity Model

Surface. The Surface is an Entity that describes the physical properties and the occlusions of a physical surface artifact that is being used to display applications. A Surface is in fact an artifact that belongs to the AmI-Home infrastructure.

Field	Type	Description
id	ObjectId	The Surface‘ unique id number.
Name	String	The Surface name.
width	Number	The Surface width in pixels.
Height	Number	The Surface height in pixels.
areas	[SurfaceArea]	The Surface areas (grid cells).
occlusions	[Occlusion]	The Surface occluded areas by physical objects.
virtualOcclusions	[virtualOcclusion]	The Surface occluded areas by virtual objects.

Table 9 The Surface Entity Model

Host. The Host entity describes the display devices that host FLUID applications. In contrast to Microsoft Windows, a Host is like a desktop and the windows are the UiCollections – containing applications. A Host can manage and display several UiCollections. The position of the displaying applications can be determined by a force property that is bound to each application component of each application. The force of the Host is managed by the AmI-Home artifacts.

Field	Type	Description
id	ObjectId	The unique id number of the Host.
name	String	The Host’s name.
context	String	The Host’s context in the smart home.
surface	Surface	The Host’s Surface information.
resolution	Resolution	The Host’s Resolution information.
uiCollections	[ObjectId]	The UiCollection IDs of the applications that are being displayed.
force	Force	The force that is applied to the host’s application elements.

Table 10 The Host Entity Model

Occlusion. The occlusion sub-entity describes the properties of an occluded area for every physical object that get detected by a detection system on a Host's physical surface.

Field	Type	Description
origin	Point	The occlusion's point of origin (top-left).
width	Number	The occlusion's width in pixels.
height	Number	The occlusion's height in pixels.

Table 11 The Occlusion Sub-Entity Model

VirtualOcclusion. The VirtualOcclusion sub-entity is in fact a pseudo-occlusion defined by an application programmer. This particular sub-entity can be used by the application developer to simulate occlusions to observe situations with multiple items in different scenarios. The VirtualOcclusion has an image property and can be displayed on a surface in contrary to the original Occlusion sub-entity which cannot be displayed.

Field	Type	Description
origin	Number	The virtual occlusion's point of origin (top-left).
width	String	The virtual occlusion's width in pixels.
height	String	The virtual occlusion's height in pixels.
url	String	The virtual occlusion's image url.

Table 12 The VirtualOcclusion Sub-Entity Model

UiElement. The UiElement Sub-Entity describes the container that is used to display the user interface of an application component or widget. The UiElements are in fact nodes of a bigger graph that together knit an entire application in the form of a graph layout. UiElements are able to move on a surface, either by the end-user or by the application manager, through events. Such events can be collisions between nodes or occlusions, or commands that are given from the AmI-Home artifacts through the FLUID Server.

Field	Type	Description
name	String	The element's name.
url	String	The element's url path.
width	Number	The element's width in pixels.
height	Number	The element's height in pixels.
weight	Number	The element's importance level (higher values corresponds to higher importance).
position	Point	The element's position point (top-left).
isExpanded	Boolean	Indicates if the element's children are visible
parents	[String]	The names of the parent elements.
children	[String]	The names of the children elements.
options	UiOptions	The element's extra options.

Table 13 The UiElement Entity Model

UiCollection. The UiCollection entity describes an application that is being displayed on a FLUID Surface Host. A UiCollection is in fact a collection of UiElements that are part of the same application. Each UiCollection is a unique object and can be displayed on one FLUID Host at a specific time. However, UiCollections can migrate from one Host to another.

Field	Type	Description
id	ObjectId	The unique id number of the collection.
name	String	The name of the collection.
hostname	String	The name of the collection's host.
uiElements	[UiElement]	The elements of the collection.

Table 14 The UiCollection Entity Model

UiOptions. The UiOptions sub-entity describes various options about a UiElement's user interface. This sub-entity will assist in the user interface design decision making of an application. By default, an application is split in a number of UiElements that are bound to each other with visible edges and border shadow. These elements have also the ability to expand on click and are able to automatically move on collision with other UiElements. The

application developer has the ability to override these functions by using this sub-entity.

Field	Type	Description
showBorder	Boolean	Indicates a uiElement's border visibility.
borderColor	String	Indicates a uiElement's border color.
showEdges	Boolean	Indicates a uiElement's edges visibility.
edgeColor	String	Indicates a uiElement's child edges color.
showHiddenCounter	Boolean	Indicates if a uiElement's hidden children counter is visible.
isPinned	Boolean	Indicates if a uiElement is pinned in its default position.
isDraggable	Boolean	indicates if a uiElement is able to be moved by the user.
expandOnClick	Boolean	Overrides default expansion of a uiElement when clicked by the user.

Table 15 The UiOptions Sub-Entity Model

5.2 FLUID Host – Frontend

The FLUID Host is one of the three main parts of the SDK. It acts as the application player that is used as a display medium for end-user application components on top of smart surfaces. This particular system, being a front-end application, is built upon Angular 8.0 technology. Application components can be distributed on various FLUID Hosts that are deployed in a smart facility via an application programming interface named UI-API (see section 4.3.1) that is part of the FLUID Server. These application components must be bundled together as a graph representing an application, or part of an application. In order to group and display the components, the FLUID Host has a built-in Graph Layout which was created by using the D3.js library, a library that is mainly intended for building big data visualizations for the web.

5.2.1 Data-Driven Documents (D3) Library

When building visualizations, designers often employ multiple tools simultaneously. This is particularly true on the web, where interactive visualizations combine varied technologies: HTML for page content, CSS for aesthetics, JavaScript for interaction, SVG for vector graphics, and so on.

Data-Driven Documents (D3) [90] is a novel representation-transparent approach to data visualization for the web (see Figure 39). D3 enables direct inspection and manipulation of a native representation, the standard document object model (DOM), rather than hiding the underlying scene-graph within a toolkit-specific abstraction. With D3, users can selectively bind input data to arbitrary document elements, applying dynamic transformations which generate and modify content.

D3 is not a traditional visualization framework. Rather than introducing a novel graphical grammar, D3 solves a different, smaller problem: the efficient manipulation of documents based on data. Thus D3's core contribution is a visualization "kernel" rather than a framework, and its closest analogues are other document transformers such as jQuery [91], CSS and XSLT [92] . As the document model directly specifies graphical primitives, D3 also bears a resemblance to low-level graphics libraries such as Processing and Raphael. For high-level capability, D3 includes a collection of helper modules that sit on top of the selection-based kernel; these modules are heavily influenced by prior visualization systems, including Protovis [93].

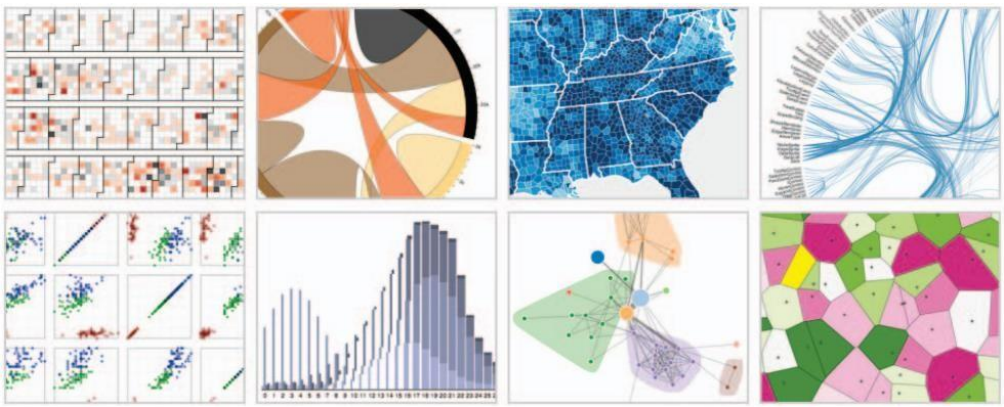


Figure 39 Interactive visualizations built with D3.

5.2.2 Force Layout with D3 JS

D3.JS is commonly used for data visualization purposes, however its graph layout capabilities (see Figure 40a), as well as its collision detection mechanism (see Figure 40b), make this library a great candidate for the application representation of FLUID Host. D3.JS visualizations are in fact groups of svg images that are bound to certain properties and get rendered in real time. Data Visualization designers bind their data to properties like position, width, height, color, text, etc. When these properties change, the svg images get refreshed and redrawn on the screen, making neat and responsive visualizations. FLUID SDK exploits this functionality by binding application components on-top of D3's force-directed graph mechanism, thus creating 'fluid' application mashups.

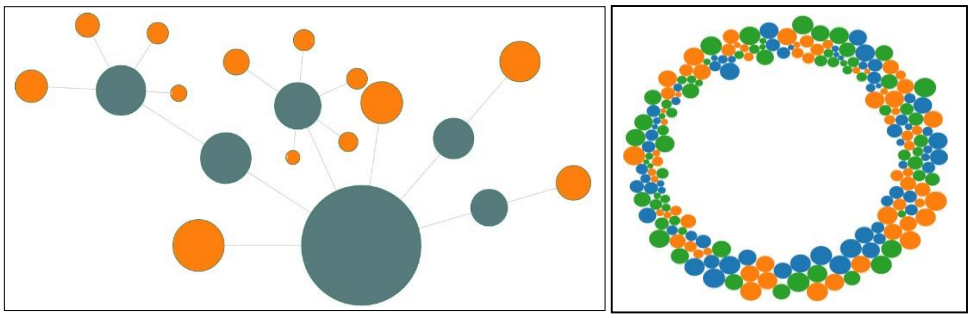


Figure 40: D3.JS: (a) Graph Layout, (b) An Occlusion Simulation

In more detail, a force-directed graph in D3.JS world is a list of node elements that are bound together with edges. Each node has a set of properties that define its position on the canvas, its size and its color and a representative name. In addition, each node has two lists of strings containing the names of the node's children and parents. These two lists are used by D3's simulation mechanism to create the edges that bind the sibling nodes together.

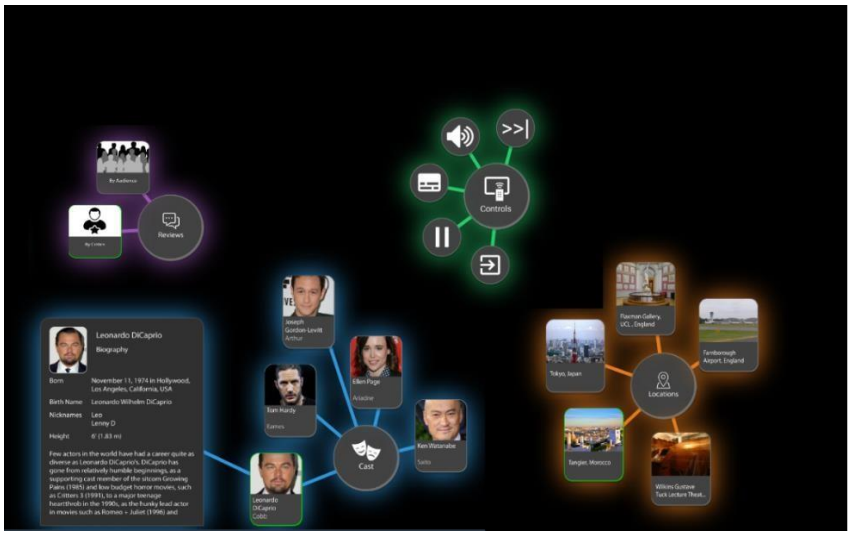


Figure 41: Application Mashup using FLUID SDK

5.2.3 Virtual Space Management Algorithm

One of the key aspects of FLUID Host is to adapt to situations where the physical surface gets occupied by one or more physical objects. There is a high chance that these physical objects occlude several application components making them unusable to the end-user. By default, the force layout mechanism of the host moves the occluded components out of the occlusion zone. However, it is common in household spaces like the living room to have multiple items on a table resulting in a cluttered mess of physical objects along with virtual application components. In order to prevent such scenarios, the FLUID Host is equipped with a virtual, custom-made space management algorithm, originally created for this thesis in order to 'hide' existing application components when the virtual space of a FLUID Host does not suffice to display them. The algorithm consists of two rules, the rule of expansion and the rule of reduction of an application component. More details about the rules are presented below.

Rule of Expansion. According to the rule of expansion***, 'only one path of expansion is permitted at a given time'***. This means that at any time, any already expanded node can only have one of its children nodes expanded. When the user decides to expand a node that already has an expanded sibling, then the expansion path resets, and the node tree of the previously expanded sibling-node collapses. The role of this rule is to reduce application cluttering when the number of user interfaces displayed on a surface becomes high.

Expansion Example:

Consider a force layout graph consisting of 14 nodes where each node represents a UiElement (see Figure 42). This graph is a part of an application (a sub-graph) that resides on top of an interactive space area. Each node is named depending on the Hierarchy starting from R (root) to 1.2.2.2. The expanded nodes are highlighted with a green border to represent a valid path of expansion and red border to represent an invalid state.

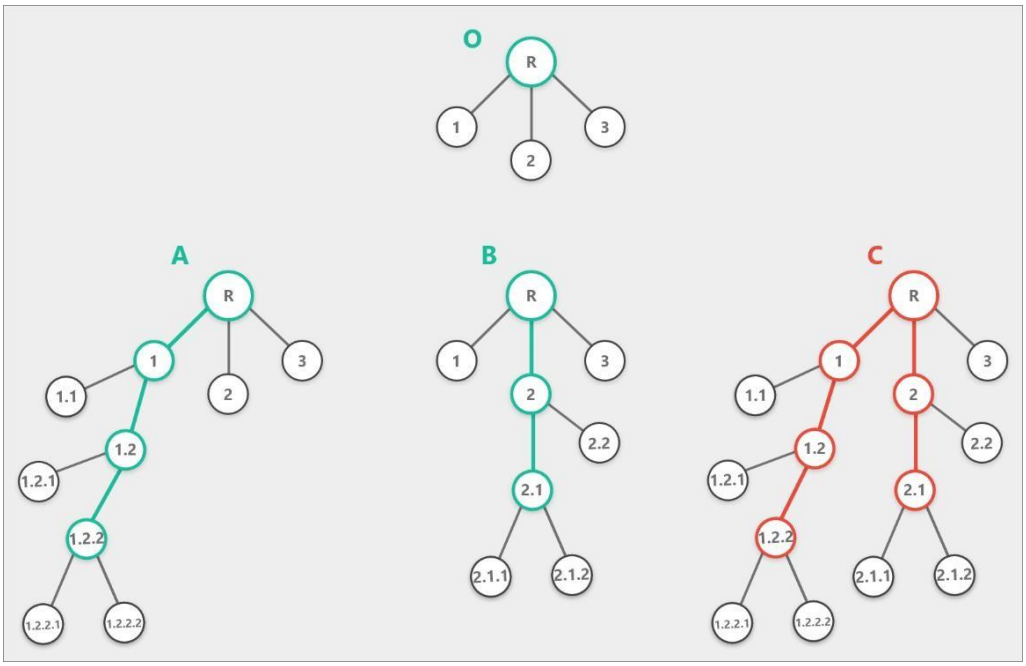


Figure 42: Rule of Expansion Example

O: Initial state where only node R is expanded, A: Valid expansion path starting from node R to node 1.2.2, B: valid Expansion paths starting from node R to node 2.1, C: Invalid state as more than one expansion paths is not possible by default

The Rule of Expansion is by default active, but it is optional. In situations where the application developer wants to display more than one ‘expansion paths’ (e.g. see Figure 42C) he may choose to bypass this rule by overriding the UiElement’s ‘expansion’ functionality (triggered by default when the end-user clicks on a UiElement) and expand them manually inside the end-application’s logic controller. This feature is present in ‘Partially Update a UiElement’ function (see section 5.3.2.9) where the application developer can manually set the expansion state of a UiElement and disable its default expansion functionality at any given time, thus overriding the ‘rule of expansion’.

Rule of Reduction. According to the rule of reduction, upon insufficient space, a UiElement gets reduced (a reduction means that a UiElement gets hidden itself, not to be confused with 'collapse of a node' which means that a UiElement's children get hidden). This action depends on three factors**:**

- **Expansion state** (whether a node is expanded)
- **Weight Value** (a number representing the node's importance)
- **Level of Hierarchy** (how deep is the node in terms of tree-hierarchy)

The rule of reduction is performed for each UiElement of an 'interaction space area' every time the available virtual space of that area gets below 20% due to many physical/virtual objects and UiElements. The rule is performed in a top down order for each UiElement recursively from the lowest hierarchy node to the highest. Starting from top to bottom the first UiElements to be reduced will be the ones with the lowest weight value. The last UiElements to be reduced are the expanded nodes.

Reduction Example:

Consider a force layout graph consisting of 12 nodes where each node is a UiElement (see Figure 43). This graph is a part of an application (a sub-graph) that resides on top of an interactive space area. Each node is named depending on the Hierarchy starting from R (root) to 3.2.3 and has a weight value (shown on its edge). The expanded nodes are highlighted with a darker border and thicker edges that represents the expansion path.

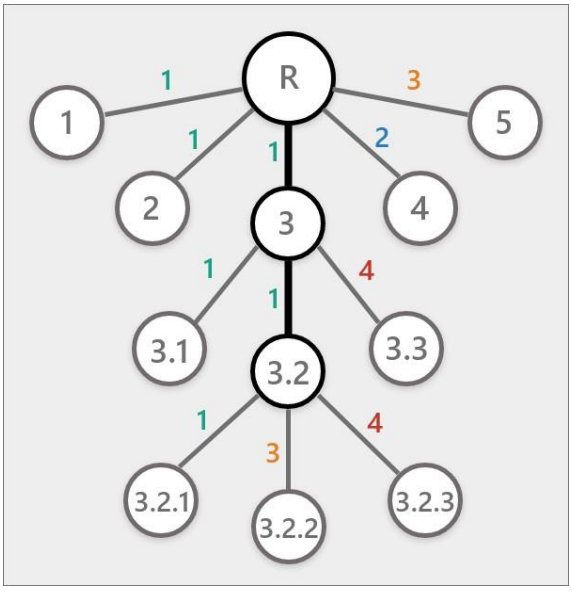


Figure 43: Force Layout Graph Example with Weights

As the containing area gets filled with application components and physical items and when the free-space threshold gets below 20%, the reduction algorithm initiates recursively for every node that is part of the area.

Table 16, Figure 44). According to the reduction order, the first nodes to be reduced, starting from the top of the Hierarchy are the ones with the lowest weight (with a value of 1). These nodes are 1, 2, 3.1 and 3.2.1. Then the next nodes to be reduced are the ones with weight value 2 and so on. The last nodes to be reduced are the expanded nodes 3.2, 3 and the root node R defying their weight values and hierarchy levels resulting in a bottom-up reduction order.

Order	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th
Node	1	2	3.1	3.2.1	4	5	3.2.2	3.3	3.2.3	3.2	3	R

Table 16: Reduction Order Example

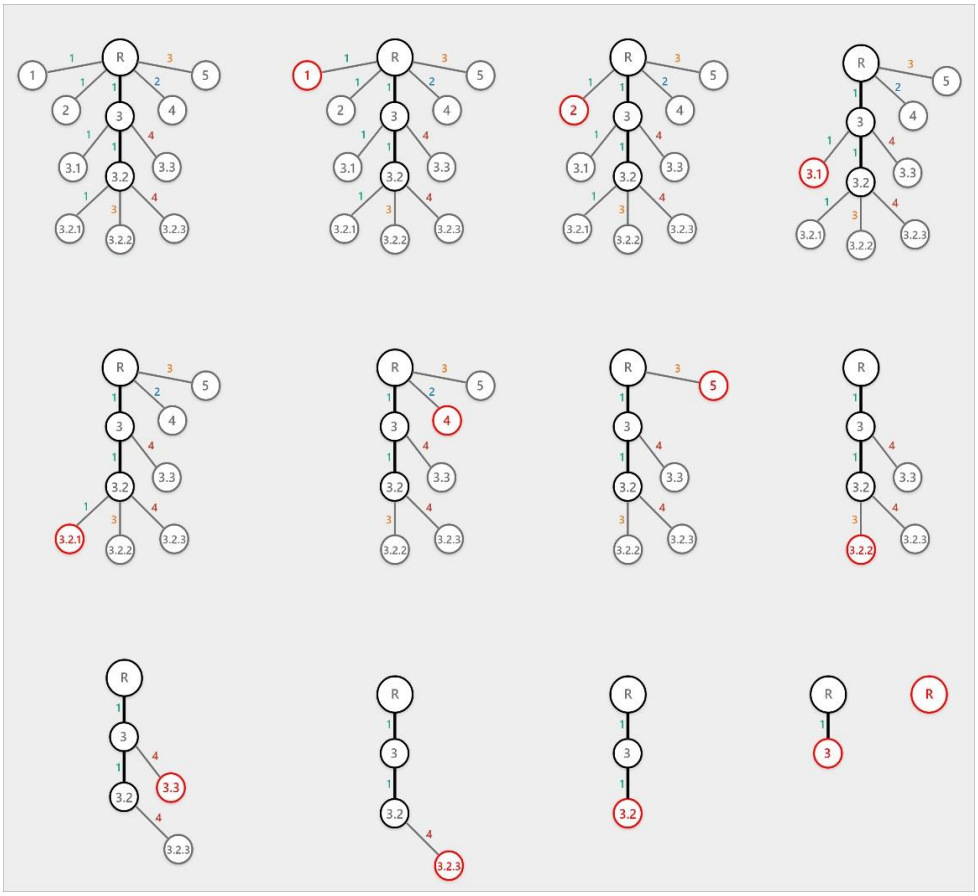


Figure 44: Reduction Example (Figure for the images below)

Figure 45: Example of UiElement reduction due to insufficient space

5.2.4 Asynchronous Communication with End-Applications

Asynchronous communication is an extremely valuable feature that further enhances interoperability and discoverability, since it enables the environment to exhibit intelligent behavior by reacting to events that occur within it. Unfortunately, the REST architecture enables one-way communication from the client to the server and does not provide a standardized event mechanism. In the AmI-Home environment, ensuring asynchronous communication amongst the various artifacts and end-user applications is perfomed by AmI-Solertis [94] suite which introduces a custom universal Event Federator server that is built on top of the Redis [95] Pub/Sub [96] mechanism. However, to support other intelligent environments in general, FLUID SDK provides asynchronous communication through common Web-Socket technology.

In order to perform these links, each FLUID Host creates its own event channel through Redis and publishes messages when an event occurs. On the other hand, end-applications subscribe to their Host's channel in order to listen for messages. This way the end-applications can respond and adapt to several events that occur inside the FLUID Host desktop. For example, many application components of a Host may hide when many physical objects enter the Host's territory. In these situations, the Host sends a message to the subscribed receivers (the end-application controllers) that a specific application component got hidden due to insufficient space.

Data pass through the event federator are in JSON Format [87]. The structure of the event-message as well as a detailed list of the FLUID Host events is presented below.

```
{
   "author": "AugmenTable",
   "message": {
      "event": "string",
      "data": object
   }
}
```

Event	Data	Description
elementHidden	uiElementName (String)	Occurs when a UiElement get hidden due to unavailable space.
pinnedElementOccluded	uiElementName (String) isOccluded (Boolean)	Occurs when a “pinned” UiElement gets occluded by a Physical/Virtual Object.
forceChanged	force (Force)	Occurs when the force of the Host is change by a smart artifact.
elementStateChanged	uiElementName (String) isExpanded (Boolean)	Occurs (by default) when the user clicks on a UiElement with children
elementMovedByUser	uiElementName (Sting) newPosition (Point)	Occurs when the user drag & drops a UiElement to a specific location.
objectDetected	position (Point) width (Number) height (Number) name? (String-Optional)	Occurs when a Physical or Virtual Object gets placed on a surface (and gets detected)

Table 17: List of the available events that occur on a FLUID Host

5.3 FLUID Server – Backend

The FLUID SDK has a web server to enable communication between FLUID Hosts, End-Applications and AmI-Home artifacts. The server provides two distinct REST APIs [97] build with Node.JS and Express library by using standard HTTP call methods (see Table 18) to distribute data. These two APIs provide functions that can be called by FLUID end-applications, tools and utility systems of AmI-Home (e.g. FLUID Manager) and AmI-Home’s smart artifacts to send or request information about the FLUID Hosts that are deployed in the

smart facility. The APIs are fully documented using swagger [98]. A developer can view a list of the API functions by navigating to the swagger documentation page (see Figure 46, Figure 47). For each API function the user can view the function's detailed description, its prerequisite parameters for the function request and details about the function's response. In addition, swagger provides a testing mechanism which gives the user the ability to call the functions and view the request parameters of the call, as well as the response parameters that are returned because of the function call.

As discussed in [99], the REST Architecture enables one-way communication from the client to the server. After an API function call, the server must notify the appropriate FLUID Host system. To that matter, the Server and the Hosts use Web Sockets [100] to pass data around asynchronously.

This section provides a detailed description of every function of both APIs and describes how the server uses Web Sockets to listen and fire events that enable information exchange with FLUID Hosts and the rest of AmI-Home ecosystem.

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

Table 18: Standard HTTP methods used in FLUID SDK

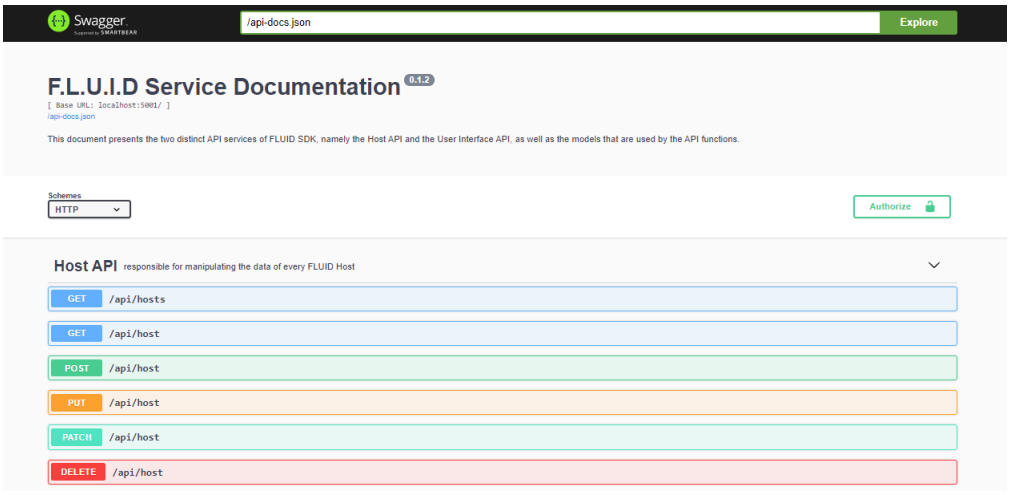


Figure 46: The Swagger Interface, containing a list of service endpoints

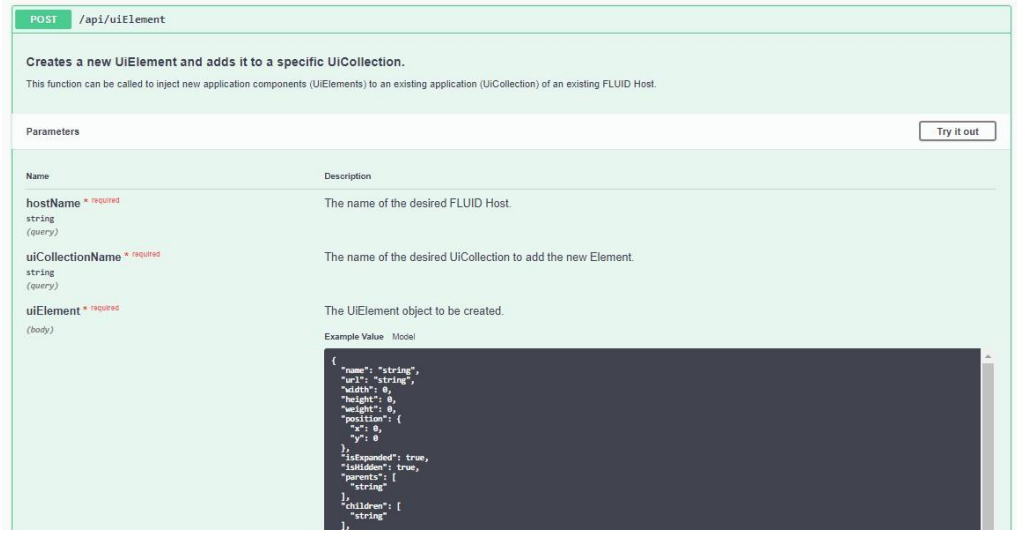


Figure 47: The Swagger Interface, example of a service endpoint.

5.3.1 The Host Application program interface (Host-API)

The first API, named Host API, is responsible for data manipulation of every FLUID Host. The main users of this API are smart artifact services that provide information about the end-user of a host and utility tools that present information and/or control the FLUID Hosts. For example, the Intelligent Livingroom incorporates a smart couch, embedded with force and pressure

sensors [101] that are connected to a Raspberry Pi [102]. The smart couch can detect if a resident is using the couch and the way that user sits on it. Therefore, the raspberry pi utilizes the Host-API to alter the force value that binds the position of AugmenTable's [85] application components when the end-user leans towards the table. In other words, it uses the API to implicitly move the applications towards the end-user's interaction space. In general, this API can be used to fetch data about FLUID Host instances as well as functions that create, update and delete these instances. More details about each function of Host-API is given bellow.

5.3.1.1 Get Hosts Function

This function can be used by external systems to retrieve a list of the available FLUID Hosts that are deployed in the AmI-Home, along with their context of use. The purpose of this function is supplementary. The caller can use the information (e.g. the name of a host) that is provided by this function to retrieve more details about a specific Host (see Get Host Information Function).

GET https://fluidserver.ics.forth.gr/api/hosts

Returns the names and context of all fluid hosts.

Parameters: None

Example Response:

```
[
  {
    "name": "FluidHost",
    "context": "demo"
  },
  {
    "name": "AugmenTable",
    "context": "Intelligent Livingroom"
  },
  {
    "name": "KitchenCounter",
    "context": "AmI-Kitchen"
  },

]
```

5.3.1.2 Get Host Information Function

This function can be used to retrieve all details about a specific FLUID Host. It can be used for display purposes (e.g. FLUID Manager uses this function to display a Host's properties in the Status Screen).

GET https://fluidserver.ics.forth.gr/api/host

Returns information about a specified FLUID host.

Parameters:

Field	Type	Required	Description
hostName	String (query)	True	Contains the requested Host's name

Example Response:

```
{
  "name": "FluidHost",
  "context": "demo",
  "surface": {
    "name": "FluidSurface",
    "width": 1500,
    "height": 800,
    "areas": [
      {
        "origin": {
          "x": 0,
          "y": 0
        },
        "width": 1500,
        "height": 800,
        "availableSpace": 100
      }
    ],
    "occlusions": [],
    "virtualOcclusions": []
  },
  "resolution": {
    "width": 1920,
    "height": 1080
  },
  "force": null
}
```

5.3.1.3 New Host Function

This function creates a new FLUID Host. It is currently used by the FLUID Host system on startup to notify the server of its existence.

POST https://fluidserver.ics.forth.gr/api/host

Creates and Saves a new Host.

Parameters:

Field	Type	Required	Description
host	Host (body)	True	The new Host object containing information to be changed.

Request Example:

```
{
  "name": "ExampleHost",
  "context": "ExampleContext",
  "surface": {
    "name": "ExampleSurface",
    "width": 2000,
    "height": 1000,
    "areas": [
      {
        "origin": {
          "x": 0,
          "y": 0
        },
        "width": 200,
        "height": 200,
        "availableSpace": 100
      }
    ],
    "occlusions": [],
    "virtualOcclusions": []
  },
  "resolution": {
    "width": 1920,
    "height": 1080
  },
  "uiCollections": [],
  "force": {
    "origin": {
      "x": 0,
      "y": 0
    },
    "strength": 0
  }
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.1.4 Fully Update Host Function

This function is responsible for updating the properties of an already existing FLUID Host. Being a PUT function, the developer of the application must provide all properties of the Host model, thus performing a full update.

PUT https://fluidserver.ics.forth.gr/api/host

Fully update an already existing Host.

Parameters:

Field	Type	Required	Description
host	Host (body)	True	The new Host object containing information to be changed.

Request Example:

```
{
  "name": "ExampleHost",
  "context": "UpdatedField",
  "surface": {
    "name": "UpdatedField",
    "width": 2000,
    "height": 1000,
    "areas": [
      {
        "origin": {
          "x": 0,
          "y": 0
        },
        "width": 200,
        "height": 200,
        "availableSpace": 100
      }
    ],
    "occlusions": [],
    "virtualOcclusions": []
  },
  "resolution": {
    "width": 1920,
    "height": 1080
  },
  "uiCollections": [],
  "force": {
    "origin": {
      "x": 0,
      "y": 0
    },
    "strength": 0
  }
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.1.5 Partial Update Host Function

This function can be used to update an already existing FLUID Host. Being a PATCH function, the developer of the application can decide which properties of the host entity wants to update, thus performing a partial update.

PATCH	https://fluidserver.ics.forth.gr/api/host
Partial update an already existing Host.	

Parameters:

Field	Type	Required	Description
host	Host (body)	True	The new Host object containing information to be changed.

Request Example:

```
{
  "name": "ExampleHost",
  "surface": {
    "name": "IkeaTable",
    "width": 20,
    "height": 10,
  },
  "resolution": {
    "width": 1920,
    "height": 1080
  }
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.1.6 Delete Host Function

This function is responsible for removing an already existing Host. It is currently called by FLUID Hosts on shutdown to notify the server about its removal. Using this function externally will command the FLUID Host to perform a shutdown.

DELETE https://fluidserver.ics.forth.gr/api/host

Removes an already existing Host.

Parameters:

Field	Type	Required	Description
hostName	String (body)	True	Contains the requested Host's name.

Request Example:

```
{
  "name": "ExampleHost"
}
```

Response: Status code (200: Success, 505: Server Error)

5.3.1.7 Add Force to Host Function

This function can be used as an alternative way to Update Host function or to alter directly a FLUID Host's force value. Changing the force value of a host moves its application components towards the force's origin. This function is used by smart artifacts that detect user position and user focus.

POST https://fluidserver.ics.forth.gr/api/host/force

Sets a force value to a specific Host.

Parameters:

Field	Type	Required	Description
hostName	String (query)	True	Contains the requested Host's name.
force	Force (body)	True	The Force value to be applied.

Request Example:

```
{
  "force": {
    "origin": {
      "x": 10,
      "y": 200
    },
    "strength": 10
  }
}
```

Response: Status code (200: Success, 505: Server Error)

5.3.1.8 Remove Force from Host Function

This function clears the force value of a FLUID Host. Like the Add force function, it is mainly used by smart artifacts that detect user position and user focus.

DELETE https://fluidserver.ics.forth.gr/api/host/force
Removes the force value from a specific Host.

Parameters:

Field	Type	Required	Description
hostName	String (body)	True	Contains the requested Host's name.

Request Example:

```
{
  "hostName": "ExampleHost"
}
```

Response: Status code (200: Success, 505: Server Error)

5.3.1.9 Get Occlusions of a Host Function

This function can be used to retrieve information about the detected physical objects on a Host's surface. Currently, this function is called on startup by

FLUID Hosts in order to initialize space availability, thus restricting access to application components on the occluded desktop areas.

GET https://fluidserver.ics.forth.gr/api/host/occlusions

Returns the occluded areas of a specific Host.

Parameters:

Field	Type	Required	Description
hostName	String (query)	True	Contains the requested Host's name.

Response Example:

```
[
  {
    "origin": {
      "x": 50,
      "y": 10
    },
    "width": 310,
    "height": 400
  }
]
```

5.3.1.10 Set Occlusions of a Host Function

This function is responsible for setting the physical objects that get detected at the surface of a FLUID Host. This function is exclusively used by object detection systems.

POST https://fluidserver.ics.forth.gr/api/host/occlusions

Sets the occluded areas of a specific Host.

Parameters:

Field	Type	Required	Description
hostName	Host (query)	True	Contains the requested Host's name.

Request Example:

```
[
  {
    "origin": {
      "x": 50,
      "y": 10
    },
    "width": 310,
    "height": 400
  },
  {
    "origin": {
      "x": 300,
      "y": 100
    },
    "width": 50,
    "height": 100
  },
]
```

Response: Status code (200: Success, 505: Server Error)

5.3.1.11 Get VirtualOcclusions of a Host Function

This function can be used to retrieve information about the virtual objects of a FLUID Host's surface. On startup, a FLUID Host uses this function to get the virtual object data, thus displaying the virtual objects and restricting access to application components on the virtually occluded areas of its desktop space.

GET https://fluidserver.ics.forth.gr/api/host/virtualocclusions

Returns the virtually occluded areas of a specific Host.

Parameters:

Field	Type	Required	Description
hostName	String (query)	True	Contains the requested Host's name.

Response Example:

```
[
  {
    "name": "vOcclusion1",
    "occlusion": {
      "origin": {
        "x": 10,
        "y": 30
      },
      "width": 110,
      "height": 220
    },
    "url": "files/image1"
  },
  {
    "name": "vOcclusion2",
    "occlusion": {
      "origin": {
        "x": 120,
        "y": 330
      },
      "width": 10,
      "height": 220
    },
    "url": "files/image2"
  }
]
```

5.3.1.12 Set VirtualOcclusions of a Host Function

This function is responsible for setting the virtual objects that get displayed on the surface of a FLUID Host. This is currently called by FLUID Manager's Occlusion manager every time the user adds, removes or moves a virtual object.

POST https://fluidserver.ics.forth.gr/api/host/occlusions

Sets the virtually occluded areas of a specific Host.

Parameters:

Field	Type	Required	Description
hostName	Host (query)	True	Contains the requested Host's name.

Request Example:

```
[
  {
    "name": "vOcclusion1",
    "occlusion": {
      "origin": {
        "x": 10,
        "y": 30
      },
      "width": 110,
      "height": 220
    },
    "url": "files/image1"
  },
  {
    "name": "vOcclusion2",
    "occlusion": {
      "origin": {
        "x": 120,
        "y": 330
      },
      "width": 10,
      "height": 220
    },
    "url": "files/image2"
  }
]
```

Response: Status code (200: Success, 505: Server Error)

5.3.2 The UI Application Program Interface (UI-API)

The second API, named UI-API, is addressed to the developers of the end-applications and is used to populate and control applications on FLUID Hosts. The user can utilize this API to create, edit and remove applications in a user interface mashup format that is called UiCollection. This enables the distribution of the applications across several FLUID Hosts that are deployed in the smart spaces of the AmI-Home. More details about each function of UI-API is given bellow.

5.3.2.1 Get the UiCollections of a Host Function

This function can be used to retrieve a list of names of the UiCollections that are deployed in a specific Host. The purpose of this function is supplementary. An external utility or tool can use the name of a UiCollection provided by this

function to retrieve more details about that specific UiCollection. This function is currently used by FLUID Manager's Application Viewer.

GET https://fluidserver.ics.forth.gr/api/uiCollections Returns a list of the UiCollection names of a specific Host.

Parameters:

Field	Type	Required	Description
hostName	String (query)	True	Contains the requested Host's name.

Response Example:

```
[
  "Netronio-Cast",
  "Netronio-Locations",
  "Netronio-Reviews",
  "Netronio-Controls",
  "Netronio-Soudtrack",
]
```

5.3.2.2 Get a UiCollection Function

In order to retrieve detailed information about a specific UiCollection of a specific Host, the developer of the application can call this function. The function returns information for every UiElement contained in the UiCollection.

GET https://fluidserver.ics.forth.gr/api/uiCollection Returns a UiCollection from a specific Host.

Parameters:

Field	Type	Required	Description
uiCollectionName	String (query)	True	Contains the requested UiCollection's name.
hostName	String (query)	True	Contains the requested UiCollection's name.

Response Example:


```
{
  "name": "Netronio",
  "hostName": "AugmenTable",
  "uiElements": [
    {
      "name": "Netronio-Cast",
      "url": "netronio.ics.forth.gr/cast",
      "width": 200,
      "height": 200,
      "weight": 10,
      "position": {
        "x": 0,
        "y": 0
      },
      "isExpanded": false,
      "parents": [],
      "children": [
        "Netronio_cast_dicaprio",
        "Netronio_cast_murphy",
        "Netronio_cast_hardy"
      ],
      "options": {
        "showBorder": true,
        "borderColor": "green",
        "showEdges": true,
        "edgeColor": "green",
        "isPinned": false,
        "isDraggable": true,
        "expandOnClick": true
      }
    }
  ]
}
```


5.3.2.3 Add a UiCollection Function

As discussed earlier, a FLUID application is in fact a ui-mashup of application components that are bound together in a force enabled graph layout, rather than the traditional grid or absolute layout. The information that is needed to create these mashups is stored in UiCollection objects. The developer can use this function to populate the FLUID Hosts with end-user applications.

POST	https://fluidserver.ics.forth.gr/api/uiCollection
Creates a new UiCollection and adds it to a specific Host.	

Parameters:

Field	Type	Required	Description
uiCollection	UiCollection (body)	True	Contains the requested Collection's info.

Request Example:

```
{
  "name": "string",
  "hostName": "string",
  "uiElements": [
    {
      "name": "string",
      "url": "string",
      "width": 0,
      "height": 0,
      "weight": 0,
      "position": {
        "x": 0,
        "y": 0
      },
      "isExpanded": true,
      "parents": [
        "string"
      ],
      "children": [
        "string"
      ],
      "options": {
        "showBorder": true,
        "borderColor": "string",
        "showEdges": true,
        "edgeColor": "string",
        "isPinned": true,
        "isDraggable": true,
        "expandOnClick": true
      }
    }
  ]
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.4 Fully Update a UiCollection Function

This function is responsible for updating an already existing UiCollection. Being a PUT function, the developer of the application must provide all properties of the UiCollection model, thus perform a full update. This function is very useful

in situations where the developer needs to perform a full instance update regarding the application components of a UiCollection. For example, consider a situation of an application that provides secondary information about a movie in real time. The developer can utilize this function when the end-user is watching a movie and he decides to watch something else, thus every application component of the movie application need to be changed or removed from the FLUID Host's instance.

PUT https://fluidserver.ics.forth.gr/api/uiCollection

Fully update an already existing UiCollection and from a specific Host.

Parameters:

Field	Type	Required	Description
uiCollection	UiCollection (body)	True	Contains the requested UiCollection's info.

Request Example:

```
{
  "name": "string",
  "hostName": "string",
  "uiElements": [
    {
      "name": "string",
      "url": "string",
      "width": 0,
      "height": 0,
      "weight": 0,
      "position": {
        "x": 0,
        "y": 0
      },
      "isExpanded": true,
      "parents": [
        "string"
      ],
      "children": [
        "string"
      ],
      "options":  {
        "showBorder": true,
        "borderColor": "string",
        "showEdges": true,
        "edgeColor": "string",
        "isPinned": true,
        "isDraggable": true,
        "expandOnClick": true
      }
    }
  ]
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.5 Partially Update a UiCollection Function

In general, the state of an application changes a lot during its use by an end-user. There are situations where the state of an application changes constantly in real time. In that regard, the application developer may partially update his UiCollection by using this function. In contrast to “Fully update a UiCollection”, this function is preferred in situations where the update must be performed on specific application components of the whole application, leaving the remaining components intact. On the other hand, when “Fully update a UiCollection” is called, the new instance of the UiCollection fully replaces the older one.

PATCH https://fluidserver.ics.forth.gr/api/uiCollection

Partially update an already existing UiCollection and from a specific Host.

Parameters:

Field	Type	Required	Description
uiCollection	UiCollection (body)	True	Contains the requested UiCollection’s info.

Request Example:

```
{
  "name": "Netronio",
  "hostName": "AugmenTable",
  "uiElements": [...]
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.6 Delete a UiCollection Function

When closing an application, the developer can use the “Delete a UiCollection” function to remove all application components of his application from the FLUID host it was deployed to.

DELETE https://fluidserver.ics.forth.gr/api/uiCollection

Delete an already existing UiCollection and from a specific Host.

Parameters:

Field	Type	Required	Description
uiCollectionName	String (body)	True	Contains the requested Collection's info.
hostname	String (body)	True	Contains the requested host's name.

Request Example:

```
{
  "name": "Netronio",
  "hostName": "AugmenTable"
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.7 Add a UiElement Function

This function can be called to inject new application components (UiElements) to an existing application (UiCollection) of an existing FLUID Host.

POST https://fluidserver.ics.forth.gr/api/uiElement

Creates a new UiElement and adds it to a specific UiCollection.

Parameters:

Field	Type	Required	Description
uiCollectionName	String (query)	True	Contains the requested UiCollection's name.
hostname	String (query)	True	Contains the requested host's name.
uiElement	String (body)	True	Contains the requested UiElement object.

Request Example:

```
{
      "name": "string",
      "url": "string",
      "width": 0,
      "height": 0,
      "weight": 0,
      "position": {
        "x": 0,
        "y": 0
      },
      "isExpanded": true,
      "parents": [
        "string"
      ],
      "children": [
        "string"
      ],
      "options": {
        "showBorder": true,
        "borderColor": "string",
        "showEdges": true,
        "edgeColor": "string",
        "isPinned": true,
        "isDraggable": true,
        "expandOnClick": true
      }
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.8 Fully Update a uiElement Function

This function can be used to edit the details of a UiElement. Being a PUT function, the developer of the application must provide all properties of the UiElement model. Developers may choose to use this function when they want to make a full update of a UiElement. If the user wants to partially update an element, he may use the PATCH version of this function (see 4.3.2.9).

PUT	https://fluidserver.ics.forth.gr/api/uiElement
Fully update an already existing uiElement and from a specific uiCollection.	

Parameters:

Field	Type	Required	Description
uiCollectionName	String (query)	True	Contains the requested Collection's name.
hostname	String (query)	True	Contains the requested host's name.
uiElement	String (body)	True	Contains the requested uiElement object.

Request Example:

```
{
      "name": "string",
      "url": "string",
      "width": 0,
      "height": 0,
      "weight": 0,
      "position": {
        "x": 0,
        "y": 0
      },
      "isExpanded": true,
      "parents": [
        "string"
      ],
      "children": [
        "string"
      ],
      "options": {
        "showBorder": true,
        "borderColor": "string",
        "showEdges": true,
        "edgeColor": "string",
        "isPinned": true,
        "isDraggable": true,
        "expandOnClick": true
      }
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.9 Partially Update a uiElement Function

Yet another function that is responsible for updating an existing UiElement. Being a PATCH function, the developer of the application is not required to provide all the properties of the UiElement, but only the properties to be updated, thus performing a partial update. This particular function is of great importance, due to the fact that the developer can utilize it for different update scenarios. For example, this function can be used to alter some of the options of the UiElement, or change its expansion state.

PATCH https://fluidserver.ics.forth.gr/api/uiElement

Partially update an already existing uiElement and from a specific uiCollection.

Parameters:

Field	Type	Required	Description
uiCollectionName	String (query)	True	Contains the requested Collection's name.
hostname	String (query)	True	Contains the requested host's name.
uiElement	String (body)	True	Contains the requested uiElement partial object.

Request Example:

```
{
      "name": "string",
      "url": "string",
      "isExpanded": true,
      "options": {
        "edgeColor": "string",
        "isPinned": true,
        "isDraggable": true,
        "expandOnClick": true
      }
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.3.2.10 Delete a UiElement Function

This function can be used to remove a UiElement from a UiCollection of a specific FLUID Host. This function is responsible for updating the children and parent properties of other UiElements regarding the removed element. When a UiElement is removed, its children elements get removed also.

DELETE https://fluidserver.ics.forth.gr/api/uiElement

Delete an already existing uiElement and from a specific UiCollection.

Parameters:

Field	Type	Required	Description
uiCollectionName	String (body)	True	Contains the requested Collection's name.
hostname	String (body)	True	Contains the requested host's name.
uiElementName	String (body)	True	Contains the requested uiElement name.

Request Example:

```
{
  "name": "uiCollection1",
  "hostName": "Host2",
  "uiElementName": "uiElement1234"
}
```

Response: HTTP Status code (200: Success, 505: Server Error)

5.4 FLUID Manager – Monitoring and testing tool

5.4.1 Overview

This section outlines the details of the FLUID Manager's components and functionalities. FLUID Manager is the third subsystem of the FLUID SDK. As already mentioned, one of the main features of the SDK is to provide to the application developers and designers a way to control, monitor and test the behavior of their end-user applications by simulating real life use case

scenarios with physical and/or virtual objects on top of the computing surface along with their projected applications. This system is a web-application, built using the Angular 8.0 Framework, and it uses the UI-API and HOST-API (see sections 4.3.1 and 4.3.2) which are provided with the SDK to monitor and control the applications that are hosted on the available FLUID Hosts. This system was designed to be a responsive web application following the “mobile first design” [103] approach. Currently, the system is implemented in its mobile form.

To assist in the testing process, FLUID Manager provides a set of tools, namely, the ‘Host Status Viewer’, the ‘Force Controller’, the ‘Application Manager’, the ‘Occlusion Manager’ and the ‘Log Viewer’. These tools are accessible through the Simulator’s navigation menu (see Figure 48). More information about each of the tools is presented below.

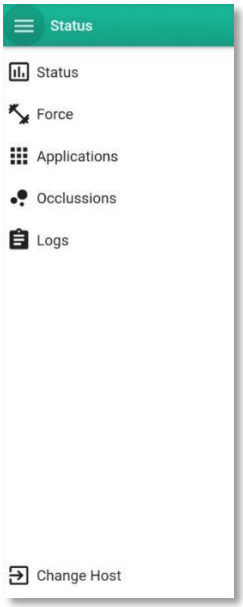


Figure 48: The navigation menu of FLUID Manager

5.4.2 Selecting a FLUID Host

Upon using the system for the first time, users are presented with a list of the available FLUID Hosts that are deployed in the AmI-Home facility. For each of the available Hosts, the system provides a representing image, the host’s name and context and current status (see Figure 49). After selecting the desired Host, the user is directed to the Host Status Screen (see section 4.4.3). In order to

select another Host, the user can also navigate to the Selection screen directly from the navigation menu.

5.4.3 FLUID Host Status Viewer

Upon selecting a Host from the selection screen, the user is directed to the Status Viewer (see Figure 50), where he can view information regarding the selected Host in real time. In more detail, the user can view the Host's Surface details and resolution and information about the current state of the Host like the number of UiCollections and UiElements, the number of physical and virtual objects that are detected on the surface, the number of expanded, collapse and hidden UiElements, as well as the point of origin of the Host. Finally, the user can view information about the Host's reserved space percentage for each interaction area of the surface.

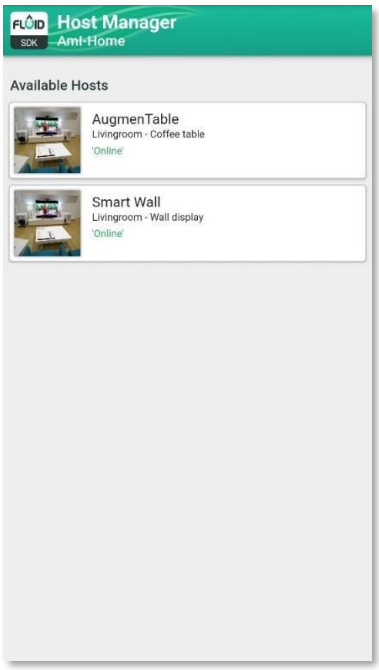


Figure 49: The FLUID Manager Host Selection Screen

Figure 50: FLUID Manager Host Status Screen

5.4.4 Force Controller

Another feature of the FLUID Manager is the ability to control a host's Force value. In the context of the AmI-Home's Living Room, the Force value of the coffetable's Host is controlled by a system embedded in the smart couch that detects the user's presence and posture. However, in many debugging or testing situations using the actual smart artifacts might not be possible due to availability issues. In that regard, the designer of the end-application can utilize the FLUID Manager's Force Control functionality to alter the Force value of a Host manually.

The force control screen contains a rectangular space which represents the surface of the FLUID Host (see Figure 51). That space contains the "force knob", a circular 'draggable' component with a dumbbell icon, that represents the force's point of origin. The user can view the force's origin properties, namely it's x and y parameters. In order to alter the force's point of origin, the designer of the end-application can use the touchscreen of the mobile device to drag and drop the "force knob" in the desired position. This action triggers the "Add Force value" function of Host-API (see section 5.3.1.7) which causes

the application components of the actual FLUID Host to move towards the force's new point of origin. With this functionality the user can simulate end-user focus on the host's surface.

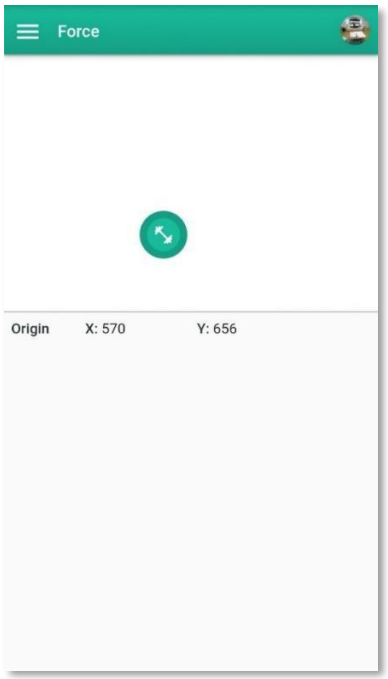


Figure 51: The Force Control Screen

5.4.5 Application Manager

FLUID Manager offers the ability to review and control the applications components of a Host. This is performed through the Application Manager Screen. The manager provides a list view of every application component that is present. Each item of the list contains real-time information about the application component's attributes, namely, its name, size, absolute position on the surface, weight value, expansion state, visibility and the number of children components, as well of the number of its parent components (see Figure 52). When the user clicks on a list item, it expands and provides additional information about the names of its children and parents, as well as a control panel which can be used to alter the UiElement's state and attributes. Upon selecting an item from the item list, the corresponding application component that is present on the FLUID host gets highlighted.

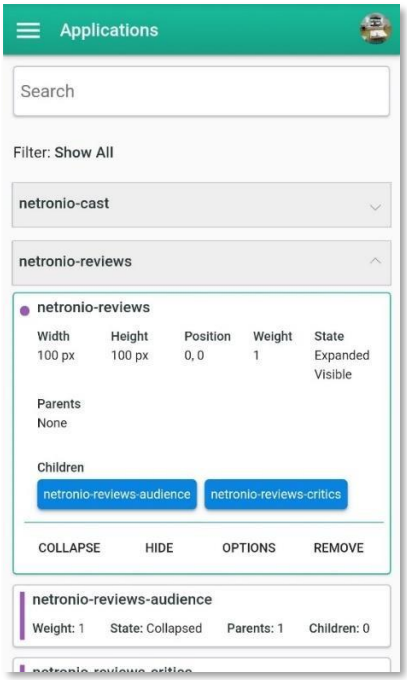


Figure 52: FLUID Manager's Application Manager

Application controls. As mentioned above, the selected application component item provides several control options. In more detail, the user can expand or collapse the application component, can select to hide it or even remove it entirely from the Host. Additionally, the user can also edit the component's details (its name, URL, width, height, weight, position, its children components and its parent components as well). Finally, the user can navigate to a component's child or parent item by clicking on the desired child/parent button.

Figure 53: Application Controlls of a uiElement in Application Manager

Search. At the top section of the Application Manager's Screen, the user can use the provided search bar to filter out UiElements by their name. As the user types in the search bar, the UiElements with names that don't contain the provided search query get filtered out automatically.

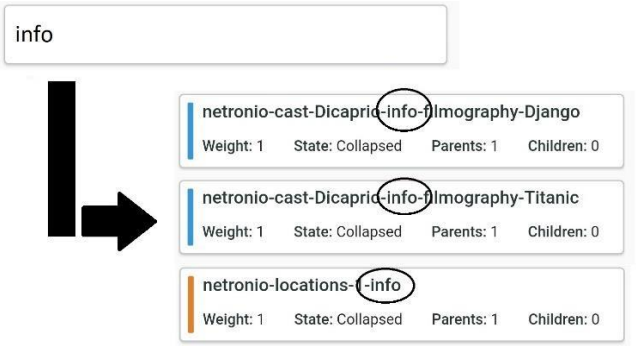


Figure 54: Search Results in FLUID Manager's Application Manager

Filtering. Finally, the Manager provides another filter option that filters out the list of UiElements by their current visibility and expansion state. The available filter options are: 'show all' components, 'show expanded' components, 'show collapsed' components and 'show hidden' components.

Figure 55: Filtering results on FLUID Manager's Application Manager

5.4.6 Occlusion Manager

One of the most important aspects of the SDK is the ability to provide physical object monitoring and surface occlusion simulation. As discussed earlier, a FLUID application developer may want to test the behavior of his applications in the presence of physical objects, even on extreme scenarios where the surface is covered with multiple and different items.

As an example, consider a developer testing a movie application that uses the table to display secondary information and controls about the movie. According to the scenario, the system is used by a family of two (2) persons watching a movie in the living room while eating their dinner. On top of the table the residents have placed two (2) plates, one (1) large pizza, a napkin and two (2) teacups. Simulating similar situations with multiple physical objects could be cumbersome, time consuming or even expensive. In that regard the user can alternatively use virtual objects and place them in the position of his liking. The Occlusion Manager was created as part of the FLUID Manager to assist in that process.

Physical and Virtual Object Viewer

Starting from the top of the Occlusion Manager Screen, the user can see a view of the host's surface enriched with information provided by the FLUID Server about the host's occluded areas. All physical occlusions are displayed as a blue box that represents the outline of the detected physical object (see Figure 56). In addition to that, the user can enrich the host's surface with virtual objects and can also manipulate their position by dragging and dropping them using the touchscreen.

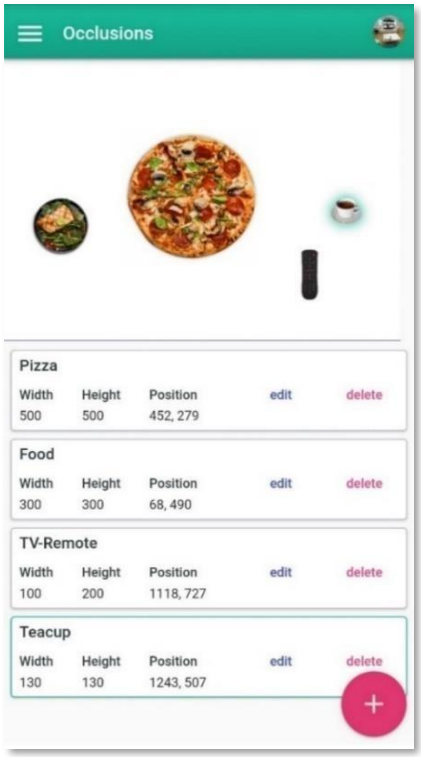


Figure 56: The occlusion manager of FLUID Manager

New Virtual Object. The user can add new virtual objects by clicking the add-button, which is located at the bottom right of the screen. After pressing the add- button, a modal window appears (see Figure 57) that prompts the user to enter information about the new virtual occlusion. Then the user must provide a name, select an image that represents the virtual object through numerous predefined images of household items and food (e.g. a pizza, a plate, a smartphone, a tv-remote etc.). He must also provide a width and height value and finally set the position of the virtual occlusion. After all information is provided, the user can save the new virtual object by clicking on the save button. Then the modal window closes, and the new virtual object is added both on the occlusion manager viewer and on the FLUID Host's surface.

Physical and Virtual Object List. The user is also able to view a list containing information for each physical and virtual object that is present on the host's surface. For each item of the list the user can view the object's information, namely the type of the object (physical or virtual), its name (if applicable), its width and height and finally its position on the surface. By selecting a list item, the object viewer highlights the corresponding object. Moreover, the system provides the ability to edit an existing virtual object's information and/or to remove it from the host by using the edit and delete button respectively.

Virtual Object Collection. Finally, the Occlusion manager offers the ability to save the list of virtual objects the user has created or load an already saved list whenever necessary. The user can save or load a list of virtual objects via the options menu by clicking the options button on the top right side of the screen. The saved lists are stored locally in the user's web-browser local storage.

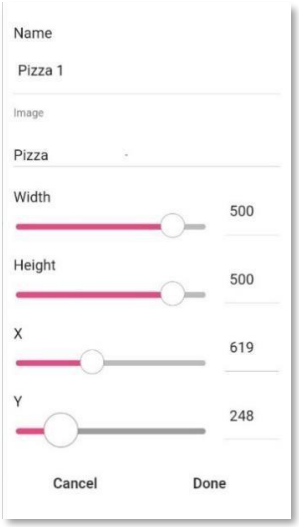


Figure 57: New / Edit Virtual Object Modal Window.

5.4.7 Log Viewer

With the ever-increasing size and complexity of surface applications in conjunction with a relatively large number of physical and virtual objects, monitoring and evaluating FLUID applications and their behavior to certain situations can be rather difficult. UiElements tend to get hidden by the Space Management Algorithm (see Section 5.2.3) when space availability is low.

In order to assist in the evaluation of a FLUID application in the Intelligent Livingroom, the FLUID Manager is equipped with the 'Log Viewer' (see Figure 58), a logging mechanism that provides detailed information about the events that occur on a Host in real time.

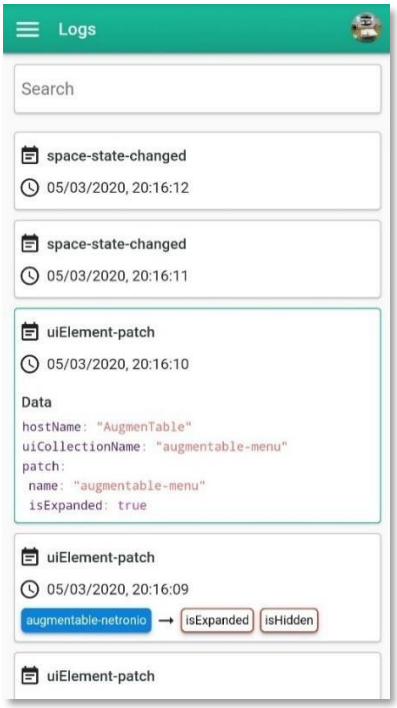


Figure 58: FLUID Manager Log Viewer Screen

Log List. The log viewer contains a list of Log Items. Each log provides information about an event which happens on the selected FLUID Host. The user can see the event's type (e.g. 'uiElement-patch', 'uiElement-new', 'host-force-changed' etc.), its timestamp and a set of tags that represent the event's outcome. By clicking on the Log item, the user can view the raw data regarding the event.

Search. At the top section of the Log Viewer's Screen, the user can use the provided search bar to filter out Logs. The search query applies to every

attribute-key and attribute-value regarding the log. For example, the user can type a date to select the logs with that specific timestamp, or filter out the Logs that don't contain an attribute with a specific key (e.g. 'isHidden', 'isExpanded', 'weight'), or an attribute with a specific value.

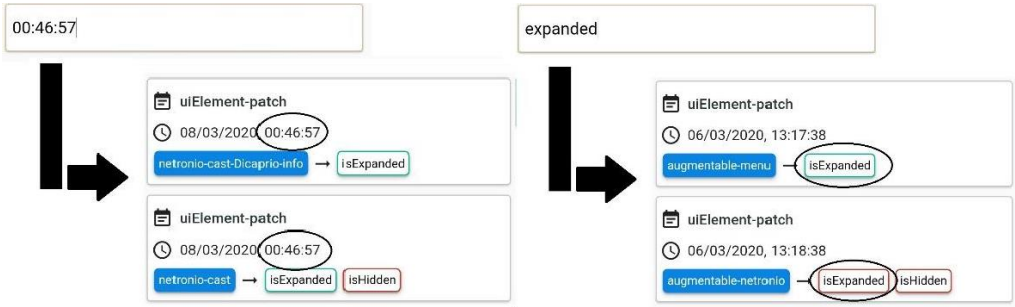


Figure 59: Search query results in FLUID Manager's Log Viewer

5.5 Additional Functional Capabilities

All the aforementioned systems have been designed to be scalable, flexible and resilient. As a result, there are various design aspects, functionalities and techniques that are supported by FLUID SDK which can help the developer create smarter and more user-friendly applications. In this section, a number of additional functional capabilities that stemmed from the design of the SDK will be discussed.

5.5.1 Wrapper API Services

As described in section 4.3, the server provides two REST APIs that control the applications hosted on FLUID Hosts as well as control of the FLUID Hosts themselves. These two APIs are created using a CRUD design pattern. Usually, AmI-Home's APIs are created using a command programming pattern rather CRUD. The main reason for choosing the aforementioned approach for developing the SDK was to give application developers the freedom to create their own Wrapper APIs. The users can utilize the SDKs functions to create custom commands.

```
moveUiElement(hostName, uiCollectionName, uiElementName, newX, newY){
  let data = {
    name:uiElementName,
    position: {
      x:newX, y:newY
    }
  };

  let params = new HttpParams()
    .set("hostName", hostName)
    .set("uiCollectionName", uiCollectionName);

    return this.http.patch<any>("http://fluidServer.ics.forth.gr/api/uiElement", data, {"params": params});
}
```

Figure 60: Wrapper function example for moving a uiElement.

5.5.2 Zoomable Application Components

Many tabletop applications as described in Chapter 2 have the ability to alter the proportions of their user interfaces to make them more usable and accessible to end-users on certain distances and angles. This functionality is also supported by FLUID SDK. Developers may use the “update UiElement” PATCH function of the UI-API to alter the width and height of any application component. The FLUID Host handles size changes by transitioning from the old proportion to the new one with CSS animation, resulting in a zoom effect. However, in order for this to work, developers must ensure that their application components are fully responsive to size change.

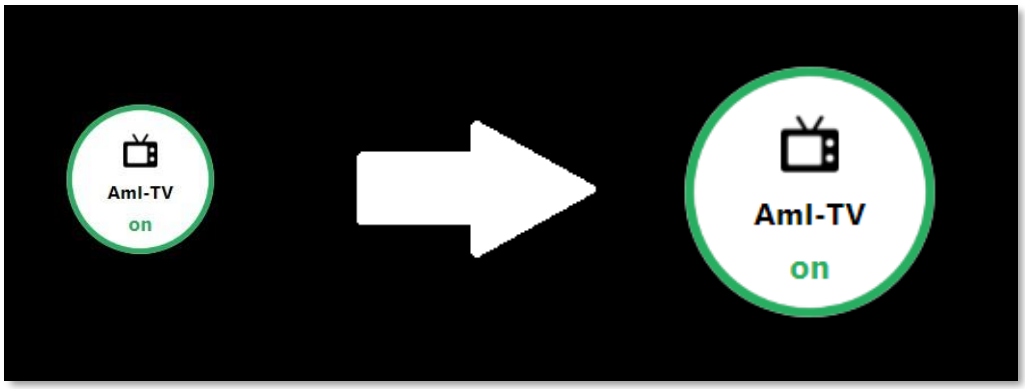


Figure 61: Zoomable Components example

5.5.3 Static Absolute Layout

According to the user interface layout standard, the absolute layout allows to specify exact locations (x/y coordinates) of its children components. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning. As described in 5.2.2, the FLUID Host displays applications on a custom force layout which is in fact an absolute layout with volatile elements. Developers may choose to create applications that contain components which are bound to certain static positions, following the standard absolute layout. Normally, when deploying application components, the developer can provide a starting position. Then the FLUID Host manages the position of the component from that point and on. If this is not desired, the developer can set the 'isPinned' property of the UiElement to 'true'. By using this functionality, the movement of the UiElement gets disabled when colliding with physical objects or when a force is applied.

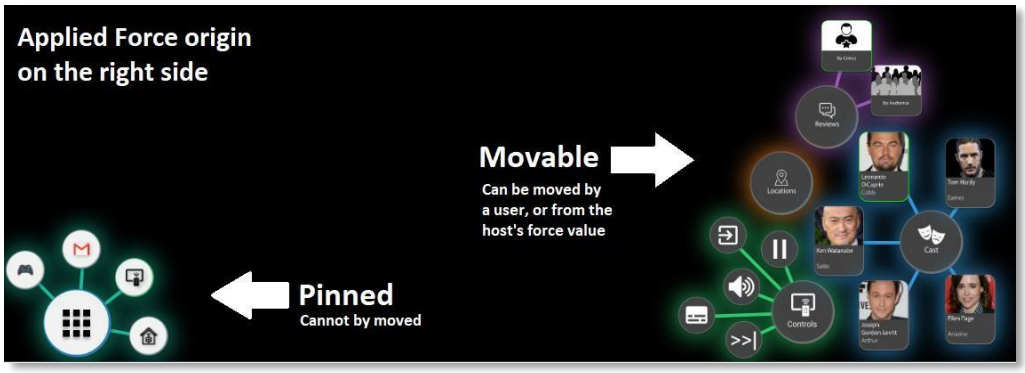


Figure 62: Absolute Layout by pinning UiElements

5.5.4 Alternative Application View

In many situations the developer may want to distribute large components on a Host. Large components tend to fill available space quicker and when the available space threshold gets exceeded, the application components tend to get hidden. In situations like these the developer may utilize the event mechanism of AmI-Home to get notified when an application component hides. Then he may use the "add UiElement" function to display an alternative set of smaller application components instead. When space is available again, the developer may revert the old large view back to the desktop.

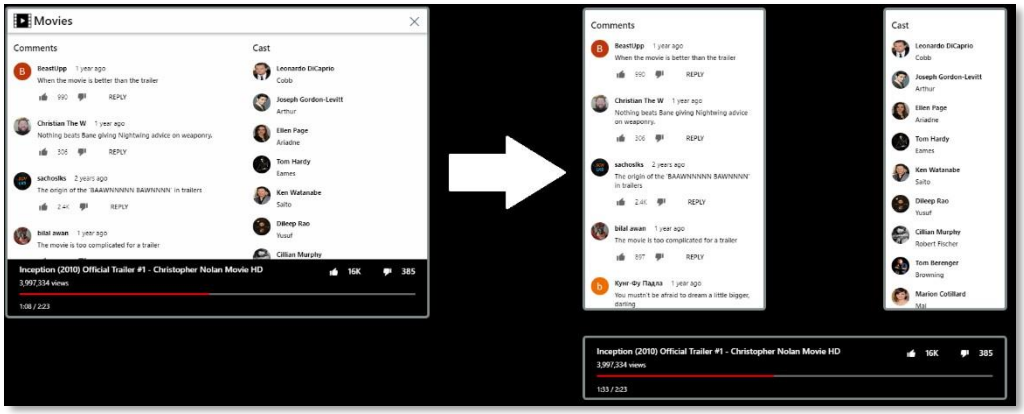


Figure 63: Alternative view example

5.5.5 Migration of Application Components

FLUID SDK is built to support, control and enable communication between multiple Hosts concurrently. The user can take advantage of this functionality to build multimonitor applications in the space of the smart home. For example, a developer may have a system consisting of 3 tabletops (e.g. a living room with a coffee table and two side-tables). Each of these tables have their own FLUID Host instances for hosting applications. When building the end-application logic, the developer can utilize the “update UiCollection” PATCH function of the UI-API to change the “hostName” property, resulting in a migration of application components from one FLUID Host to another.

5.5.6 Binding Application Components to Virtual and Physical Objects

As discussed in Chapter 2, many smart table systems utilize physical objects as part of the application itself. FLUID SDK is designed to support this functionality also. In order to bind an application component to an object, the developer must assign the physical object as a parent element to that application component. This works because, internally, the FLUID Host treats physical and virtual objects as pinned, abstract application components.

5.5 Additional Functional Capabilities

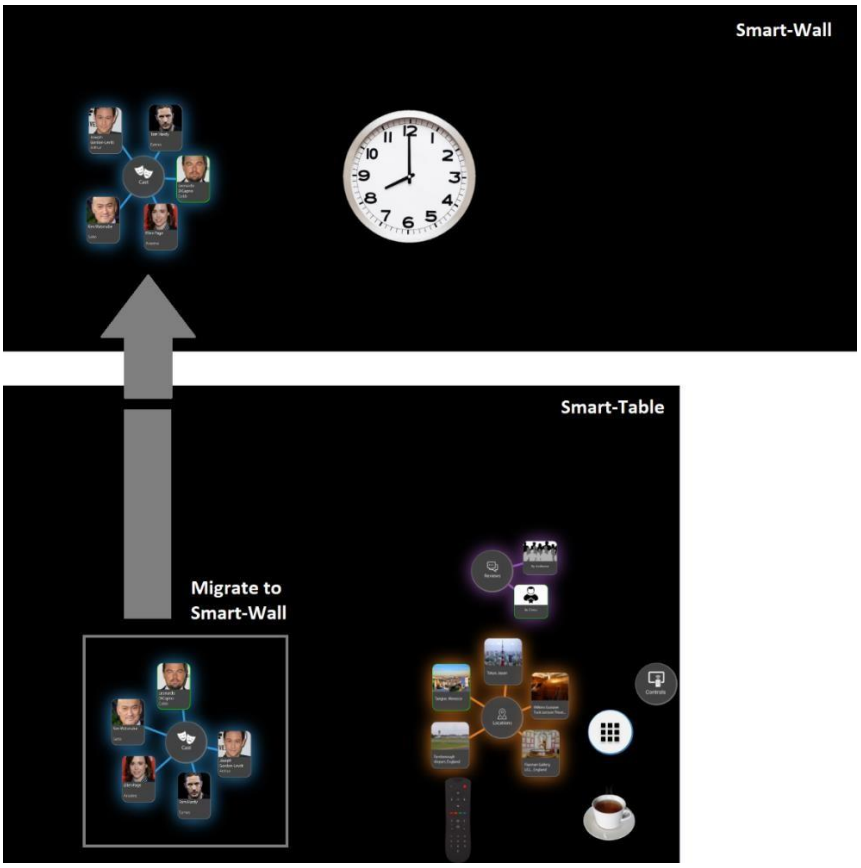


Figure 64: UiCollection Migration Example

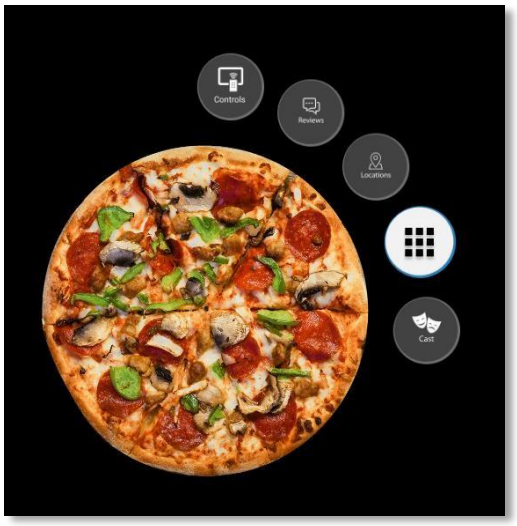


Figure 65: Bound UiElements on a Virtual Object

Chapter 6

FLUID SDK Use Cases

6.1 AumenTable Suite

The AugmenTable suite was developed as a proof of concept for FLUID SDK. The proposed system makes use of some applications. The available applications that AugmenTable suite contains are:

- YouTube app: Navigate and watch a video, display the comments section of a video as well as write a comment.

- Gmail app: An application giving access to the functionalities of the Google e-mail app, to receive, view and send e-mails.

- Google calendar: An application giving access to the functionalities of the Google Calendar app, with task/event reminders and calendar viewing.

- Smart-Home Control: An application that displays the status of the smart home's devices and appliances. The application also provides a limited control over the smart devices. Specifically, the user has access to:
    1. The lights of the home: controlling the intensity of the room.

    2. The Coffee machine: viewing the status of the machine and turning it on and off.

    3. The water heater: viewing the status of the boiler and turning it on and off.

    4. Video door entry system: View an incoming guest at the front door of a smart home and let him enter the house or dismiss the call.

6.2 Netronio Movie Player

The authors of AmI-TV [83] proposed a multimodal and service-oriented system which aimed to redefine the role of TVs inside ambient intelligence environments. Their work includes many TV applications, each offering a variety of features (news, weather, email, notifications, music player, radio player and movie player). Currently, AmI-TV's movie player is a mini application which permits users to browse through a library of movies and TV-series, select the desired movie and watch it in real time. The system also offers personalization and movie recommendation based on the user's profile.

This work was introduced in the early stages of the Intelligent Living Room development which consisted of a smart TV, various presence detection sensors and a Google home assistant. Nowadays, the Intelligent Living Room has a plethora of new smart artifacts capable of enhancing the resident's experience. Among these artifacts are the Smart-Wall, the Smart-Couch and the AugmenTable system that was described above.

Taking advantage of these new artifacts in addition to the AmI-Tv, Netronio Movie Player aims to deliver an even richer movie-theater experience by incorporating movie controls and additional information about the movie in real time, but also ensuring at the same time a smooth and unobtrusive movie experience. Amongst its core functionalities, Netronio's movie player is able to display real-time secondary information about the scene that takes place at every point of time in a movie. This secondary information includes details about the scene's shooting place, a list of the scene's actors, details about the scene's soundtrack, ratings and reviews, as well as various facts about the scene and the movie in general.

Netronio is currently in the design and development process using the FLUID SDK in order to distribute various user interfaces on AugmenTable, coffetable and across the Smart-Wall display. Also, many use-case scenarios are being considered, by optimizing the user interfaces when AugmenTable is also used to host physical objects (for example, when the user uses the table to eat and watch his movie).

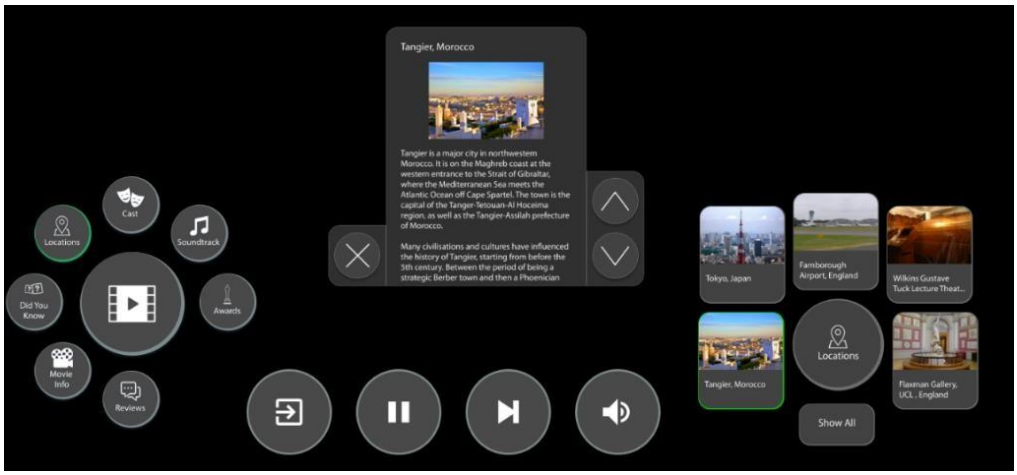


Figure 66: Netronio - AugmenTable (Controls and location information)

6.3 Smart Kitchen Assistant

Many research efforts have focused on augmenting kitchens with a variety of digital media to create rich, interactive experiences for users cooking in the kitchen. Some work has focused on providing awareness to support multi-tasking activities in the kitchen. For example, the Counter Intelligence project of MIT [104] has augmented a kitchen with ambient interfaces to improve the usability of the physical environment. Their augmented reality kitchen can assist users in determining temperatures, finding things, following recipes, and timing intermediate steps during meal preparation. Other work has focused on capturing or using digital interactive recipes that can guide users through a step-by-step cooking process. The authors of [105] have developed a system that automates the creation of web-ready multimedia recipes in a kitchen. By operating one of the foot-switches, a user can capture images of the cooking workplace with voice memos and organize into a multimedia recipe. Such digital recipes can provide a more interactive experience than that from reading a paper-based recipe book. Another project, named CounterActive [106], which utilizes digital recipes to teach people how to cook by projecting multimedia content onto a touch panel-like interactive kitchen counter. Also, the Intelligent Kitchen project [107] proposes a human activity recognition system in the kitchen, using data mining approach to infer next human action by considering the past human behaviors. Rather than augmenting kitchens with a variety of digital media to create interactive cooking experiences, the smart kitchen introduced in [107] is focused on promoting healthy cooking by raising

nutritional awareness during the cooking process, while leaving the decision about how to cook to users.

Following a similar path, the AmI-Home's Intelligent Kitchen is equipped with a variety of smart artifacts and various sensors that will assist in the process of helping the house residents cook recipes. In more detail, the kitchen is equipped with two interactive surfaces with touch support as well as a smart bench with object detection and an interactive wall-display above the bench. The AmI-Home developers can utilize these artifacts along with the available sensing technologies to design and implement applications that can assist the users in creating recipes. In the context of a kitchen, the countertop is a surface that host various objects and tools. In that regard, the developer can take advantage of FLUID SDK's layout capabilities to enrich the smart bench with widgets that assist in the cooking process. As an example, the developer can bind application components to physical objects (e.g. pots, pans, kitchen tools) which are present on the table, that provide information about the objects as well as components that act as controls for the objects. When things get cluttered on the table, the developer can migrate the widgets to the wall display.

Chapter 7

Heuristic Evaluation

The objective of this chapter is to report the outcomes of a preliminary evaluation targeted to assess the FLUID SDK. The main goal of this process is to evaluate if the system optimizes the design method in intelligent environments. This section will present (i) a brief analysis of the Heuristic Evaluation Process, (ii) details about the heuristic evaluation experiment that was performed on FLUID SDK, (iii) the usability-issues that were discovered from the experiment.

Heuristic evaluation [108] is a method for finding usability problems in a user interface design by having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (the “heuristics”). Heuristic evaluation thus falls into the general category of usability inspection methods together with methods like pluralistic usability walkthroughs [109], claims analysis [110], and cognitive walkthroughs [111], with the main difference being that it is less formal than the other methods and intended as a “discount usability engineering” [112] method. Independent research has found heuristic evaluation to be extremely cost-efficient [113], confirming its value in circumstances where limited time or budgetary resources are available. The goal of heuristic evaluation is finding usability problems in an existing design (such that they can be fixed). One could thus view it as a “debugging” method for user interfaces.

During this process of individual sessions with each expert, the main objective is to extract identified findings and merge them into a list of usability issues. Afterwards, each evaluator is asked to provide severity ratings for each list item individually. These ratings, which are used to prioritize the issues according to their severity, may range from zero (0) to four (4), where the first is categorized as “not a usability problem” and the last as a “usability catastrophe”. In addition, every issue was also graded with respect its ease-of-fix, using a scale ranging from zero (“would be extremely easy to fix”) to three (“would be difficult to fix”), indicating the amount of effort that needs to be spent to be addressed.

7.1 The Process

Apart from further investigating FLUID SDK's possible usability issues, the iteration of the heuristic evaluation included assessing the ease of interaction with the system and the comprehension of the UI (User Interface). In order to efficiently conduct this experiment, a set of actual use case scenarios containing a number of tasks were created, which were given to five (5) expert evaluators, who were engaged in this process.

The evaluation experiment was divided into five stages: (i) the preparation, (ii) the introduction, (iii) the actual experiment, (iv) the usability issue extraction and rating and (v) the usability issue ease-of-fix rating.

An important aspect of this experiment is that it had to be explicitly executed remotely with the help of online videocall and screen sharing technologies due to a global pandemic which caused a quarantine lockdown at the time the evaluation was conducted. Therefore, this experiment also provided useful feedback on conducting heuristic evaluation in an online modality.

Since it was important to browse a big part of the system for this experiment, use case scenarios and tasks were created to lead users into interacting with the biggest part of FLUID SDK. These scenarios were given to the users one by one in an electronic presentation-like format. Before proceeding with initiating the experiment with the introduction stage, one FLUID Host was launched to the screen of the user along with an instance of the FLUID Manager, launched to his smartphone.

After the preparation stage was completed, the users were invited to an online video-call, one by one in separate sessions, and were introduced to the purpose of the experiment, as explained in the previous section, followed by a brief introduction to the system and its functionalities.

During the actual experiment, the scenarios were read aloud to each user as well as displayed in an electronic format besides the experiment artifacts, so users could turn to read them again at any time. Since that was a heuristic-interaction evaluation experiment with experts in this domain, there was no need to offer hints or help and the session time was not recorded. However, all users were prompted to speak aloud during each session, since their comments constituted a basis for identifying easier the usability issues of the system.

After finishing the session, each user was asked to extract system's usability issues based on Nielsen's recognized usability principles such as Learnability,

Efficiency, Memorability, Low Error Rate and User Satisfaction [114]. These issues were afterwards merged into three separate lists, which contained all unique findings. A list that contains issues for FLUID Host environment, one list for FLUID Manager Mobile App and lastly a list for the FLUID Demo applications of the experiment. Each user was given a copy of this list to provide a severity rating to each issue, according to their preference. These ratings, which are used to prioritize the issues according to their severity, may range from zero (0) to four (4), where the first is categorized as “not a usability problem” and the last as a “usability catastrophe”. The final severity rating was produced by computing the average from each user rating.

Finally, the identified issues, along with the final severity rating, were forwarded to the development team, which ranked them according to the ease-of-fix scale, in order to designate the amount of effort needed to address them.

7.2 Use Case Scenarios

Each user was given five (5) scenarios, containing a total of ten (10) tasks. These scenarios represent possible use cases of FLUID SDK in its actual target environment and cover a wide range of the system’s functionality

Introduction

You are a member of the designers’ team of an intelligent home project and you have designed and developed a movie player application for the Intelligent living-room. This particular application consists of a main screen (the TV) which acts as the movie player and two secondary screens (the smart coffee-table and the smart-wall) which display supplementary information and control about the movie in real time. Your job is to evaluate the behavior of the second screen applications of the movie player by simulating real life use cases.

Scenario 1

Task 1. As a first step, you want to view information about the AugmenTable coffeetable’s current state. Examine how many user interfaces are hosted in total and how many of them are collapsed.

Task 2. Next, you want to view information about the Smart-Wall’s current state. How many occlusions are hosted on the wall’s surface?

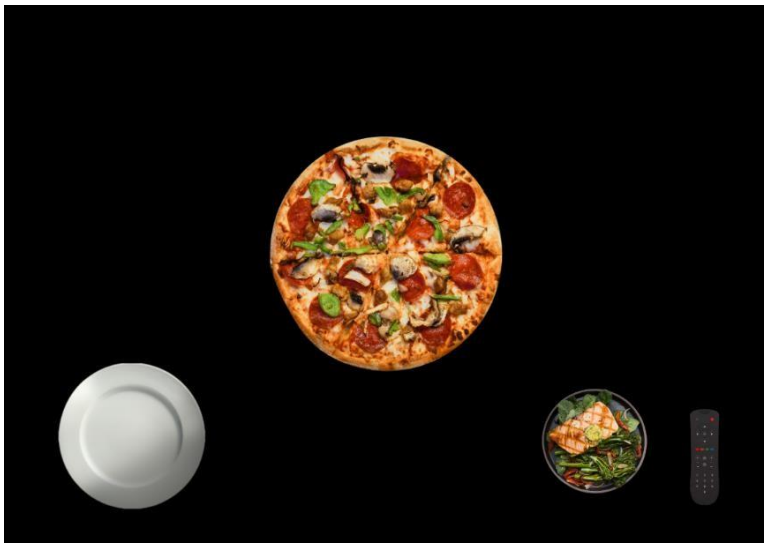
Scenario 2

After reviewing the current state of the two FLUID Hosts, you now want to test the behavior of AugmenTable's hosted user interfaces. You decide to simulate a situation of a family eating dinner while watching a movie.

Task 1. You want to simulate some **occlusions** by placing a number of virtual objects on the table's surface. Add the following:

- one (1) "pizza" on position 'x: 400, y: 400'
- one (1) "plate" on position 'x: 200, y:500'
- one (1) "food" on position 'x: 800, y: 500'
- one (1) "tv-remote" on position 'x: 700, y: 400'

Task 2. You decide that you want to rearrange the virtual objects a little bit more. Move the virtual objects to match the positions of the following picture:



Scenario 3

Task 1. Next, you want to focus all uiElements to the bottom-center interaction space of the table. In order to do so, apply the appropriate **force** value.

Scenario 4

Task 1. Next, you decide to view some details about the Application Interfaces that are deployed on the table. Find the "Netronio-Locations" application component and view its current position.

Task 2. Next, find and ‘expand’ the “Netronio-Cast” component and then find and ‘expand’ the “Netronio-Cast-Dicaprio” Component.

Scenario 5

During your last action, some user interfaces got hidden due to insufficient space. In this scenario, you want to understand the reasons behind that event in order to evaluate the correctness of that ‘ui hiding’ action.

Task 1. Use the Simulator’s Logging system to find the names of the application components that got hidden.

Task 2. Now that you know the names of child elements that got hidden you want to view their weight values.

Task 3. Finally, you decided that you want to view AugmenTable’s space availability. What is the space availability of the ‘bottom-center’ interaction space?

7.3 Evaluation Findings

The heuristic evaluation process revealed twenty-four (24) usability issues, which are categorized and sorted based on their severity ratings, as given by the expert evaluators.
Table 19 presents the extracted issues, along with their severity and ease-of-fix ratings.

The final severity score for each usability issue was extracted by combining and averaging individual scores provided by each expert. The final score ranges between two (2) and three (3). Six (6) issues were suggestions, ten (10) issues were minor usability problems, whereas the remaining seven (7) were ranked as moderate issues and thus the most important to fix and finally one (1) issue was ranked as a usability catastrophe, however it’s ease-of-fix score is very low. As a last step, the development team designated the amount of effort needed to address these issues, by providing an ease-of-fix score. According to
Table 19, Table 20 and Table 21, fixing these issues requires a small to moderate amount of effort.

FLUID Host		
Heuristic	Severity Rating	Ease-of-fix
When migrating ‘part’ of an application from a Host to another, there should be an indication of where the migrated ‘part’ is, rather than the “+1” indicator	2	1
Experts suggested that there should a ‘UiElement migration attribute’ used by the space managements algorithm to migrate UiElements when space availability gets low	3	2
Edge-force should be modifiable in order to be able to keep child-UiElements close to their parent-UiElements	2	0
The system should provide pinch and zoom functionality on UiElements	0	2
Collision detection should be better optimized as rectangular UiElements get slightly occluded by other elements or objects	4	1
FLUID Host should support panning	0	3
An expert suggested that there should be a UiElement Migration functionality on every Host, not only from end-application-logic or from FLUID Manager	2	3

Table 19 Usability issues with severity and ease-of-fix ratings of FLUID Host

FLUID Manager		
Heuristic	Severity Rating	Ease-of-fix
Application controls ‘Expand’, ‘Hide’, ‘Options’, ‘Remove’ should be better clarified as clickable buttons	3	0
Experts suggested that there should be an option for ‘Migrate a UiElement’ inside the Applications List	2	2
The application list should provide a way to view the depth level of the UiElements	2	1
Experts stated that Surface model information should not be displayed first on the Status Screen, as it is not that important	2	0
The space availability grid-section of Status Screen is not interactive, but it seems otherwise. It should be redesigned	2	1
The Host Selection Menu Button is not easy to find. It should change position.	2	0
The side menu should indicate the selected screen	2	0
Modal windows inside the application should take all screen as this is standard for mobile applications	2	0
There should be a correct indication of the selected FLUID Host on every screen	3	0
Surface availability percentage on status should support different Hosts instead of only showing the table availability	3	1
An expert suggested that the Force Screen and the Virtual Objects Screen should be merged in one Screen	0	2
There developer should be able to control the edges visibility and color through the Application manager	0	0

Table 20: Usability issues with severity and ease-of-fix ratings of FLUID Manager Applpication

FLUID Demo applications		
Heuristic	Severity Rating	Ease-of-fix Score
Upon launching an application from FLUID Host (A) to be displayed on FLUID Host (B), the user should be presented with feedback about his action	3	1
Parent UiElements should become smaller in size when they get expanded	0	0
The menu application should be always close to the user	3	3
It should be clear to the user, on how to close/shut down a ‘part’ of an application that has migrated to another FLUID Host	3	1
An expert suggested that the menu should automatically closed after some actions	0	0

Table 21: Usability issues with severity and ease-of-fix ratings of FLUID Demo applications

Chapter 8

Summary and Future Work

8.1 Summary

A couple of decades ago, a house that aids its residents with daily routine, an office that assists employees to work effectively or a smart class seemed like things that could only be found in movies using visual effects. All these concepts became reality with the progress of technology and the realization of Ambient Intelligence. Intelligent environments are complex ecosystems that are expected to have multiple screens that display the graphical user interface that has been imported into the system. Many of these screens are physical surfaces that utilize projection technologies to display user interfaces that contain information. Naturally, these common surfaces are also being used to host physical objects on top of them, thus occluding the information of applications and in some situations making them unusable.

To address these issues, this thesis has proposed FLUID, a software development tool that can be used to compose, host and manipulate flexible web applications on smart surfaces which act as secondary displays of an intelligent space (i.e. intelligent living room). From an engineering perspective, FLUID SDK: (i) acts as a host for applications on physical surfaces, (ii) prevents the occlusion of its user interfaces from physical objects ensuring a seamless interaction with the end-users, (iii) provides a rich API with useful functions for developing a unified host environment for multi-screen applications (iv) offers a set of tools for monitoring, controlling, and evaluating surface applications.

8.2 Future Work

Since FLUID is an SDK, it already provides all means to create smart applications for physical surfaces, however creating the layout and behavior of an application is currently performed manually by a programmer, therefore, offering a semi-automated tool for layout and behavior creation is worth further research and development.

Moreover, the SDK can be further enhanced by accommodating machine learning algorithms in order to further reason with the space management procedure. Finally, further investigation should be conducted in order to explore and define the SDK's limits in terms of both functionality and system flows regarding user experience.

8.2.1 FLUID Manager for desktop

As stated in section 5.7, FLUID SDK's Simulator application was designed as a mobile first approach and currently is developed for mobile use only. The next step is the implementation of its desktop version. Hence, the developers can program and debug their applications side by side with the desktop variant of the FLUID Manager.

The desktop version of the FLUID Manager should provide each functionality that is available in the mobile version, the Host selection, Host status, Force manager, Application Controller, Occlusion manager and Log Viewer. Additionally, the desktop variant should also provide a dashboard screen that contains information about all the aspects of a FLUID Host, its current status, current visible user interfaces, the latest logs and metrics and statistics about the events that occur in general, to help the developers understand which events are triggered the most and which not in order to better optimize the behavior of their applications.

8.2.2 Layout Builder

FLUID SDK's layout building is currently handled by the application developer, hardcoded in the form of JSON files. These files can get very long and complex, resulting in obvious hindrances in the development of the FLUID applications. In that regard FLUID SDK should provide a Layout Building tool.

The Layout builder should provide a rich editor which enables the creation of σto create, load and update FLUID mashups in the form of the JSON files that is supported by the system.

8.2.3 Surface Management using Machine Learning

Currently, FLUID SDK features a Space Management Algorithm that is responsible for maintaining stability over the amount of space taken by the available user interfaces in conjunction with the physical objects that are present. This means that some user interfaces are going to be hidden,

depending on their level in the graph-hierarchy and level of importance. Both of these two factors are predetermined by the application's designer.

Even then, there are situations where the result might not be suitable for each individual end-user. For some users some aspects of an application might be more significant than the predetermined importance that is assigned by the designer. In that regard, machine learning can be utilized to alter the importance values of the user interfaces by learning the habits of each end-user through profiling techniques.

8.2.4 User Based Evaluation of FLUID SDK and FLUID applications

Up until now, the SDK has been evaluated by UI-UX experts in order to find its major flows and usability issues. In the future, an alpha test of the system should be conducted by developing a full application using the SDK in order to discover potential further design flaws of the system in the application development perspective. After the completion of a Demo FLUID application, a user-based evaluation should be conducted in order to investigate usability factors and issues that could stem from FLUID applications.

8.2 Future Work

References

[1] F. Xia, L. T. Yang, L. Wang, and A. Vinel, “Internet of Things,” *Int. J. Commun. Syst. Int. J. Commun. Syst*, vol. 25, pp. 1101–1102, 2012, doi: 10.1002/dac.2417.

[2] D. Lewis, “What is web 2.0?,” *XRDS Crossroads, ACM Mag. Students*, vol. 13, no. 1, pp. 3–3, Sep. 2006, doi: 10.1145/1217666.1217669.

[3] D. Guinard and V. Trifa, “Towards the Web of Things: Web Mashups for Embedded Devices.”

[4] D. Guinard, “Mashing up your web-enabled home,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6385 LNCS, pp. 442–446, doi: 10.1007/978-3-642-16985-4_42.

[5] A. Kamilaris, V. Trifa, and A. Pitsillides, “HomeWeb: An application framework for Web-based smart homes,” in *2011 18th International Conference on Telecommunications, ICT 2011*, 2011, pp. 134–139, doi: 10.1109/CTS.2011.5898905.

[6] T. Kubitza, A. S.- Computer, and undefined 2017, “meSchup: A platform for programming interconnected smart things,” *ieeexplore.ieee.org*.

[7] “Jaimin Patel and Gaurang Panchal. An iot-based portable... - Google Scholar.” [Online]. Available: https://scholar.google.com/scholar?hl=en&as_sdt=1%2C5&q=Jaimin+Patel+and+Gaurang+Panchal.+An+iot-based+portable+smart+meeting+space+with+real-time+room+occupancy.+In+Yu-Chen+Hu%2C+Shailesh+Tiwari%2C+Krishn+K.+Mishra%2C+and+Munesh+C.+Trivedi%2C+editors%2C+Intelligent+Communication+and+Computational+Technologies%2C+volume+19+of+Lecture+Notes+in+Networks+and+Systems.+Springer+Singapore.&btnG=. [Accessed: 18-Apr-2020].

[8] I. R. Floyd, M. C. Jones, D. Rathi, and M. B. Twidale, “Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software.”

[9] B. Shneiderman and C. Plaisant, “Designing the user interface: strategies for effective human-computer interaction,” 2010.

[10] “Introduction to Machine Learning - Ethem Alpaydin - Google Books.” [Online]. Available:

https://books.google.gr/books?hl=en&lr=&id=tZnSDwAAQBAJ&oi=fnd&pg=PR7&dq=machine+learning&ots=F2ZW60dwzi&sig=44odWIR9cs9cJXcDoEVZlx0ix3g&redir_esc=y#v=onepage&q=machine learning&f=false. [Accessed: 18-Apr-2020].

[11] "Neural Networks and Analog Computation: Beyond the Turing Limit - Hava T. Siegelmann - Google Books." [Online]. Available: https://books.google.gr/books?hl=en&lr=&id=n-jpBwAAQBAJ&oi=fnd&pg=PR9&dq=neural+networks+computer+science&ots=5Ytgz-ipXl&sig=u0BYLCH2A5-yhM6I5NBy_0VHe_o&redir_esc=y#v=onepage&q=neural networks computer science&f=false. [Accessed: 18-Apr-2020].

[12] "Computer Vision: Algorithms and Applications - Richard Szeliski - Google Books." [Online]. Available: https://books.google.gr/books?hl=en&lr=&id=bXzAlkODwa8C&oi=fnd&pg=PR4&dq=+computer+vision&ots=g_3954pFDE&sig=zvW0EwkhPb3wYzMIDNLsbTRynvk&redir_esc=y#v=onepage&q=computer vision&f=false. [Accessed: 18-Apr-2020].

[13] S. Y. Bao, M. Sun, and S. Savarese, "Toward coherent object detection and scene layout understanding," *Image Vis. Comput.*, vol. 29, no. 9, pp. 569–579, Aug. 2011, doi: 10.1016/j.imavis.2011.08.001.

[14] J. Garrett, *Elements of user experience, the: user-centered design for the web and beyond*. 2010.

[15] B. Shneiderman and M. L.-U. http://www. usability. gov/guidelines/[05. 04. 2010], "Research Based Web-Design & Usability-Guidelines (2006)."

[16] "Carsten R¨ocker. User-centered design of intelligent... - Google Scholar." [Online]. Available: https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=Carsten+R¨ocker.+User-centered+design+of+intelligent+environments%3A+Requirements+for+designing+successful+ambient+assisted+living+systems.+page+8.&btnG=. [Accessed: 18-Apr-2020].

[17] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology."

[18] B. L. Risteska Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *Journal of Cleaner Production*, vol. 140. Elsevier Ltd, pp. 1454–1464, 01-Jan-2017, doi: 10.1016/j.jclepro.2016.10.006.

[19] M. Miraoui, R. Cherif, N. Rtimi, and C. Tadj, “Context-aware services adaptation for a smart living room,” in *2014 World Symposium on Computer Applications and Research, WSCAR 2014*, 2014, doi: 10.1109/WSCAR.2014.6916780.

[20] A. Chapman, “What we can learn from… Living rooms,” *Early Years Educ.*, vol. 18, no. 2, pp. ii–iv, Jun. 2016, doi: 10.12968/eyed.2016.18.2.ii.

[21] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, “Easyliving: Technologies for intelligent environments,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2000, vol. 1927, pp. 12–29, doi: 10.1007/3-540-39959-3_2.

[22] B. R. Jones, H. Benko, E. Ofek, and A. D. Wilson, “IllumiRoom: Peripheral projected illusions for interactive experiences,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2013, pp. 869–878, doi: 10.1145/2470654.2466112.

[23] B. Jones *et al.*, “RoomAlive: Magical experiences enabled by scalable, adaptive projector-camera units,” in *UIST 2014 - Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, 2014, pp. 637–644, doi: 10.1145/2642918.2647383.

[24] C. Hennessey and J. Fiset, “Long range eye tracking: Bringing eye tracking into the living room,” in *Eye Tracking Research and Applications Symposium (ETRA)*, 2012, pp. 249–252, doi: 10.1145/2168556.2168608.

[25] S. S. Lee, J. Chae, H. Kim, Y. K. Lim, and K. P. Lee, “Towards more natural digital content manipulation via user freehand gestural interaction in a living room,” in *UbiComp 2013 - Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2013, pp. 617–626, doi: 10.1145/2493432.2493480.

[26] C. Fabrigoule, L. Letenneur, J. F. Dartigues, M. Zarrouk, D. Commenges, and P. Barberger-Gateau, “Social and Leisure Activities and Risk of Dementia: A Prospective Longitudinal Study,” *J. Am. Geriatr. Soc.*, vol. 43, no. 5, pp. 485–490, May 1995, doi: 10.1111/j.1532-5415.1995.tb06093.x.

[27] E. A. Vandewater, D. S. Bickham, J. H. Lee, H. M. Cummings, E. A. Wartella, and V. J. Rideout, “When the Television Is Always On,” *Am. Behav. Sci.*, vol. 48, no. 5, pp. 562–577, Jan. 2005, doi: 10.1177/0002764204271496.

[28] M. Aslama and M. Pantti, “Talking alone,” *Eur. J. Cult. Stud.*, vol. 9, no. 2, pp. 167–184, May 2006, doi: 10.1177/1367549406063162.

[29] D. H. Shin, Y. Hwang, and H. Choo, “Smart TV: Are they really smart in interacting with people? Understanding the interactivity of Korean Smart TV,” *Behav. Inf. Technol.*, vol. 32, no. 2, pp. 156–172, Feb. 2013, doi: 10.1080/0144929X.2011.603360.

[30] M. Lochrie and P. Coulton, “Sharing the viewing experience through second screens,” in *EuroiTV’12 - Proceedings of the 10th European Conference on Interactive TV and Video*, 2012, pp. 199–202, doi: 10.1145/2325616.2325655.

[31] M. R. Morris, “Benefits and Challenges of Tabletop Peripheral Displays.”

[32] D. Geerts, R. Leenheer, D. De Grooff, S. Heijstraten, and J. Negenman, “In front of and behind the second screen: Viewer and producer perspectives on a companion app,” in *TVX 2014 - Proceedings of the 2014 ACM International Conference on Interactive Experiences for TV and Online Video*, 2014, pp. 95–102, doi: 10.1145/2602299.2602312.

[33] G.-T. with Google and undefined 2012, “The new multi-screen world: Understanding cross-platform consumer behavior.”

[34] N. Šoskić, N. Kuzmanović, M. Vidaković, and G. Miljković, “Second screen user experience: A new digital television frontier,” in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014 - Proceedings*, 2014, pp. 1057–1060, doi: 10.1109/MIPRO.2014.6859725.

[35] T. Feltwell *et al.*, “‘I’ve been manipulated!’: Designing second screen experiences for critical viewing of reality TV,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2017, vol. 2017-January, pp. 2252–2263, doi: 10.1145/3025453.3025833.

[36] H. Gil de Zúñiga, V. Garcia-Perdomo, and S. C. McGregor, “What Is Second Screening? Exploring Motivations of Second Screen Use and Its Effect on Online Political Participation,” *J. Commun.*, vol. 65, no. 5, pp. 793–815, Oct. 2015, doi: 10.1111/jcom.12174.

[37] K. Chorianopoulos, G. Lekakos, and D. Spinellis, “Intelligent user interfaces in the living room,” in *Proceedings of the 8th international conference on Intelligent user interfaces - IUI ’03*, 2003, p. 230, doi: 10.1145/604045.604083.

[38] B. Cardoso and J. F. de Abreu, “Indagante: A proposal for a social multiplatform game to motivate Interaction in the living room,” in

Communications in Computer and Information Science, 2017, vol. 689, pp. 105–116, doi: 10.1007/978-3-319-63321-3_8.

[39] S. Basapur, H. Mandalia, S. Chaysinh, Y. Lee, N. Venkitaraman, and C. Metcalf, “FANFEEDS: Evaluation of socially generated information feed on second screen as a TV show companion,” in *EuroiTV’12 - Proceedings of the 10th European Conference on Interactive TV and Video*, 2012, pp. 87–96, doi: 10.1145/2325616.2325636.

[40] M. Obrist, R. Bernhaupt, and M. Tscheligi, “Interactive TV for the home: An ethnographic study on users’ requirements and experiences,” *Int. J. Hum. Comput. Interact.*, vol. 24, no. 2, pp. 174–196, Feb. 2008, doi: 10.1080/10447310701821541.

[41] P. Vorderer, S. Knobloch, and H. Schramm, “Does Entertainment Suffer from Interactivity? The Impact of Watching an Interactive TV Movie on Viewers’ Experience of Entertainment,” *Media Psychol.*, vol. 3, no. 4, pp. 343–363, 2001, doi: 10.1207/S1532785XMEP0304_03.

[42] K. Chorianopoulos and G. Lekakos, “Introduction to social TV: Enhancing the shared experience with interactive TV,” *Int. J. Hum. Comput. Interact.*, vol. 24, no. 2, pp. 113–120, Feb. 2008, doi: 10.1080/10447310701821574.

[43] M. Satyanarayanan, “Pervasive computing: Vision and challenges,” *IEEE Personal Communications*, vol. 8, no. 4. pp. 10–17, Aug-2001, doi: 10.1109/98.943998.

[44] A. Van Cauwenberge, G. Schaap, and R. Van Roy, “‘tV no longer commands our full attention’: Effects of second-screen viewing and task relevance on cognitive load and learning from news,” *Comput. Human Behav.*, vol. 38, pp. 100–109, Sep. 2014, doi: 10.1016/j.chb.2014.05.021.

[45] “ERIC - ED348029 - Choosing a Display Format for Instructional Multimedia: Two Screens vs. One., 1992-Feb.” [Online]. Available: https://eric.ed.gov/?id=ED348029. [Accessed: 18-Feb-2020].

[46] D. H. Shin and F. Biocca, “Explicating user behavior toward multi-screen adoption and diffusion: User experience in the multi-screen media ecology,” *Internet Res.*, vol. 27, no. 2, pp. 338–361, 2017, doi: 10.1108/IntR-12-2015-0334.

[47] T. Neate, M. Jones, and M. Evans, “Mediating attention for second screen companion content,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2015, vol. 2015-April, pp. 3103–3106, doi: 10.1145/2702123.2702278.

[48] K. Z. Gajos and K. Chauncey, “The influence of personality traits and cognitive load on the use of adaptive user interfaces,” in *International Conference on Intelligent User Interfaces, Proceedings IUI*, 2017, pp. 301–306, doi: 10.1145/3025171.3025192.

[49] E. Anstead, S. Benford, and R. J. Houghton, “Many-screen viewing: Evaluating an olympics companion application,” in *TVX 2014 - Proceedings of the 2014 ACM International Conference on Interactive Experiences for TV and Online Video*, 2014, pp. 103–110, doi: 10.1145/2602299.2602304.

[50] “Online Entertainment| Disconnecting, Connecting, and Reconnecting: How Chinese Television Found Its Way Out of the Box | Keane | International Journal of Communication.” [Online]. Available: https://ijoc.org/index.php/ijoc/article/view/5726. [Accessed: 18-Feb-2020].

[51] H. Z. Tan, W. Zhao, and H. H. Shen, “Adaptive user interface optimization for multi-screen based on machine learning,” in *Proceedings of the 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design, CSCWD 2018*, 2018, pp. 872–877, doi: 10.1109/CSCWD.2018.8465348.

[52] M. Fink, M. Covell, and S. Baluja, “Social-and Interactive-Television Applications Based on Real-Time Ambient-Audio Identification.”

[53] M. Conti *et al.*, “Looking ahead in pervasive computing: Challenges and opportunities in the era of cyberphysical convergence,” *Pervasive Mob. Comput.*, vol. 8, no. 1, pp. 2–21, Feb. 2012, doi: 10.1016/j.pmcj.2011.10.001.

[54] M. Mhatre, S. Choudhari, N. Chourasia, and N. Mumbai, “MICROSOFT PIXELSENSE-INTERACTIVE SURFACE COMPUTING PLATFORM FOR SAMSUNG SUR40.”

[55] “Ambient Television: Visual Culture and Public Space - Anna McCarthy - Google Books.” [Online]. Available: https://books.google.gr/books?hl=en&lr=&id=qxj3t835slEC&oi=fnd&pg=PR9&dq=McCarthy,+A.+Ambient+Television:+Visual+Culture+and+Public+Space%3B+Duke+University+Press:+Durham,+NC,+USA,+2001%3B+ISBN+0-8223-2692-2.&ots=QrlyzCgpMZ&sig=JtfYL693QQtBgYJKxFEJ8E7BIKg&redir_esc=y#v=onepage&q&f=false. [Accessed: 19-Feb-2020].

[56] V. Ha, K. M. Inkpen, R. L. Mandryk, and T. Whalen, “Direct intentions: The effects of input devices on collaboration around a tabletop display,” in *Proceedings of the First IEEE International Workshop on Horizontal*

Interactive Human-Computer Systems, TABLETOP'06, 2006, vol. 2006, pp. 177–184, doi: 10.1109/TABLETOP.2006.10.

[57] I. A. Zaiţi, Ş. G. Pentiuc, and R. D. Vatavu, "On free-hand TV control: experimental results on user-elicited gestures with Leap Motion," *Pers. Ubiquitous Comput.*, vol. 19, no. 5–6, pp. 821–838, Aug. 2015, doi: 10.1007/s00779-015-0863-y.

[58] M. Lochrie and P. Coulton, "Mobile phones as second screen for TV, enabling inter-audience interaction," in *ACM International Conference Proceeding Series*, 2011, p. 1, doi: 10.1145/2071423.2071513.

[59] F. Daniel and M. Matera, *Mashups: Concepts, models and architectures*. Springer Berlin Heidelberg, 2014.

[60] M. Schneps-Schneppe, D. N.-I. J. of Open, and undefined 2013, "About Home Gateway Mashups."

[61] A. Koschmider, V. Torres, and V. Pelechano, "Elucidating the Mashup Hype: Definition, Challenges, Methodical Guide and Tools for Mashups."

[62] F. Vernier, N. Lesh, and C. Shen, "Visualization techniques for circular tabletop interfaces," in *Proceedings of the Workshop on Advanced Visual Interfaces AVI*, 2002, pp. 257–263, doi: 10.1145/1556262.1556305.

[63] H. Benko, R. Jota, and A. D. Wilson, "MirageTable: Freehand interaction on a projected augmented reality tabletop," in *Conference on Human Factors in Computing Systems - Proceedings*, 2012, pp. 199–208, doi: 10.1145/2207676.2207704.

[64] W. W. Gaver *et al.*, "The Drift Table: Designing for ludic engagement," in *Conference on Human Factors in Computing Systems - Proceedings*, 2004, pp. 885–900, doi: 10.1145/985921.985947.

[65] D. S. Kirk, S. Izadi, O. Hilliges, S. Taylor, A. Sellen, and R. Banks, "At home with surface computing?," in *Conference on Human Factors in Computing Systems - Proceedings*, 2012, pp. 159–168, doi: 10.1145/2207676.2207699.

[66] R. D. Vatavu and S. G. Pentiuc, "Interactive coffee tables: Interfacing TV within an intuitive, fun and shared experience," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5066 LNCS, pp. 183–187, doi: 10.1007/978-3-540-69478-6_24.

[67] Y. Rogers, W. Hazlewood, E. Blevis, and Y. K. Lim, "Finger talk: Collaborative decision-making using talk and fingertip interaction around

a tabletop display," in *Conference on Human Factors in Computing Systems - Proceedings*, 2004, pp. 1271–1274, doi: 10.1145/985921.986041.

[68] D. Leithinger and M. Haller, "Improving Menu Interaction for Cluttered Tabletop Setups with User-Drawn Path Menus."

[69] D. Cotting and M. Gross, "Interactive environment-aware display bubbles," in *UIST 2006: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 2008, pp. 245–254, doi: 10.1145/1166253.1166291.

[70] J. Steimle, M. Khalilbeigi, M. Mühlhäuser, and J. D. Hollan, "Physical and digital media usage patterns on interactive tabletop surfaces," in *ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010*, 2010, pp. 167–176, doi: 10.1145/1936652.1936685.

[71] E. Freeman and S. Brewster, "Messy Tabletops: Clearing Up the Occlusion Problem," in *Conference on Human Factors in Computing Systems - Proceedings*, 2013, vol. 2013-April, pp. 1515–1520, doi: 10.1145/2468356.2468627.

[72] G. Furumi, D. Sakamoto, and T. Igarashi, "SnapRail: A tabletop user interface widget for addressing occlusion by physical objects," in *ITS 2012 - Proceedings of the ACM Conference on Interactive Tabletops and Surfaces*, 2012, pp. 193–196, doi: 10.1145/2396636.2396666.

[73] "Touch Monitor SmartMedia - SmartMediaUSA, Inc." [Online]. Available: https://www.smartmediaworld.net/products/interactive-monitors. [Accessed: 19-Feb-2020].

[74] M. Owens, "IdeaExchange@UAkron Smart Coffee Table," 2016.

[75] "Official Home of Microsoft Surface Computers, Laptops, PCs, 2-in-1's & Devices – Microsoft Surface." [Online]. Available: https://www.microsoft.com/en-us/surface. [Accessed: 19-Feb-2020].

[76] "GraSp: Combining Spatially-aware Mobile Devices and a Display Wall for Graph Visualization and Interaction - Kister - 2017 - Computer Graphics Forum - Wiley Online Library." [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13206. [Accessed: 18-Feb-2020].

[77] Y. Jin, T. Xie, Y. Wen, and H. Xie, "Multi-screen cloud social TV: Transforming TV experience into 21st century," in *MM 2013 - Proceedings of the 2013 ACM Multimedia Conference*, 2013, pp. 435–436, doi: 10.1145/2502081.2502257.

[78] L. Cruickshank, E. Tsekleves, R. Whitham, A. Hill, and K. Kondo, “Making interactive TV easier to use: Interface design for a second screen approach,” *Des. J.*, vol. 10, no. 3, pp. 41–53, 2007, doi: 10.2752/146069207789271920.

[79] Y. Suo, C. Wu, Y. Qin, C. Yu, Y. Zhong, and Y. Shi, “HouseGenie: Universal monitor and controller of networked devices on touchscreen phone in smart home,” in *Proceedings - Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing in Conjunction with the UIC 2010 and ATC 2010 Conferences, UIC-ATC 2010*, 2010, pp. 487–489, doi: 10.1109/UIC-ATC.2010.14.

[80] “App - Eurovision Song Contest.” [Online]. Available: https://eurovision.tv/app. [Accessed: 02-May-2020].

[81] “LiveRugby app — second screen or settling an argument? – Seven League.” [Online]. Available: https://sevenleague.co.uk/2012/02/05/liverugby-app-second-screen-or-settling-and-argument. [Accessed: 03-May-2020].

[82] “Kahoot! | Learning games | Make learning awesome!” [Online]. Available: https://kahoot.com/. [Accessed: 02-May-2020].

[83] N. Anyfantis *et al.*, “AmITV: Enhancing the role of TV in ambient intelligence environments,” in *ACM International Conference Proceeding Series*, 2018, pp. 507–514, doi: 10.1145/3197768.3201548.

[84] L. Nielsen, “Engaging personas and narrative scenarios.”

[85] Leonidis *et al.*, “Ambient Intelligence in the Living Room,” *Sensors*, vol. 19, no. 22, p. 5011, Nov. 2019, doi: 10.3390/s19225011.

[86] D. Geels, G. Altekar, and S. Shenker, “Replay Debugging for Distributed Applications.”

[87] C. Chasseur, Y. Li, and J. M. Patel, “Enabling JSON Document Stores in Relational Systems.”

[88] “The most popular database for modern apps | MongoDB.” [Online]. Available: https://www.mongodb.com/. [Accessed: 21-Feb-2020].

[89] K. Chodorow and M. Dirolf, “MongoDB: The Definitive Guide.-Description Based on Print Version Record.-Includes Index,” 2010.

[90] M. Bostock, V. Ogievetsky, and J. Heer, “D3 data-driven documents,” *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, 2011, doi: 10.1109/TVCG.2011.185.

[91] K. De Volder, “JQuery: A generic code browser with a declarative configuration language,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2005, vol. 3819 LNCS, pp. 88–102, doi: 10.1007/11603023_7.

[92] J. Clark, “XML to PDF by RenderX XEP-http://www.renderx.com, XSL to PDF and XSL to Postscript formatter XSL Transformations (XSLT),” 1999.

[93] M. Bostock and J. Heer, “Protovis: A graphical toolkit for visualization,” in *IEEE Transactions on Visualization and Computer Graphics*, 2009, vol. 15, no. 6, pp. 1121–1128, doi: 10.1109/TVCG.2009.174.

[94] A. Leonidis, D. Arampatzis, N. Louloudakis, and C. Stephanidis, “The AmI-Solertis system: Creating user experiences in smart environments,” in *International Conference on Wireless and Mobile Computing, Networking and Communications*, 2017, vol. 2017-October, doi: 10.1109/WiMOB.2017.8115850.

[95] “Redis.” [Online]. Available: https://redis.io/. [Accessed: 21-Feb-2020].

[96] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003, doi: 10.1145/857076.857078.

[97] L. Li, W. Chou, W. Zhou, and M. Luo, “Design Patterns and Extensibility of REST API for Networking Applications,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 1, pp. 154–167, Mar. 2016, doi: 10.1109/TNSM.2016.2516946.

[98] “The Best APIs are Built with Swagger Tools | Swagger.” [Online]. Available: https://swagger.io/. [Accessed: 21-Feb-2020].

[99] “REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces - Mark Masse - Google Books.” [Online]. Available: https://books.google.gr/books?hl=en&lr=&id=eABpzyTcJNIC&oi=fnd&pg=PR3&dq=rest+api&ots=vzWx06efJz&sig=XvY3ODtil-FaD1XyV1p_zR2pP8I&redir_esc=y#v=onepage&q=rest api&f=false. [Accessed: 21-Feb-2020].

[100] “The WebSocket API (WebSockets) - Web APIs | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Accessed: 21-Feb-2020].

[101] “Pressure Sensors | TE Connectivity.” [Online]. Available: https://www.te.com/usa-en/products/sensors/pressure-sensors.html. [Accessed: 21-Feb-2020].

[102] “Teach, Learn, and Make with Raspberry Pi – Raspberry Pi.” [Online]. Available: https://www.raspberrypi.org/. [Accessed: 21-Feb-2020].

[103] “Chapter 4: Responsive Web Design, Discoverability, and Mobile Challenge | Kim | Library Technology Reports.” [Online]. Available: https://www.journals.ala.org/index.php/ltr/article/view/4507. [Accessed: 21-Feb-2020].

[104] L. Bonanni, C. H. Lee, and T. Selker, “Attention-based design of augmented reality interfaces,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2005, pp. 1228–1231, doi: 10.1145/1056808.1056883.

[105] I. Siio, N. Mima, I. Frank, T. Ono, and H. Weintraub, “Making recipes in the kitchen of the future,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2004, p. 1554, doi: 10.1145/985921.986130.

[106] W. Ju, R. Hurwitz, T. Judd, and B. Lee, “CounterActive: An interactive cookbook for the kitchen counter,” in *Conference on Human Factors in Computing Systems - Proceedings*, 2001, pp. 269–270, doi: 10.1145/634067.634227.

[107] Y. Nakauchi, T. Fukuda, K. Noguchi, and T. Matsubara, “Intelligent kitchen: Cooking support by LCD and mobile robot with IC-labeled objects,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 2464–2469, doi: 10.1109/IROS.2005.1545346.

[108] J. Nielsen, “Finding usability problems through heuristic evaluation,” in *Conference on Human Factors in Computing Systems - Proceedings*, 1992, pp. 373–380, doi: 10.1145/142750.142834.

[109] B. Randolph, “Walkthroughs: Efficient collaborative testing,” *IEEE Softw.*, vol. 8, no. 5, pp. 94–95, 1991, doi: 10.1109/52.84220.

[110] “Qualitative artifact analysis | Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction.” [Online]. Available: https://dl.acm.org/doi/abs/10.5555/647402.725295. [Accessed: 07-Apr-2020].

[111] C. Lewis, P. Poison, C. Wharton, and J. Rieman, “Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces,” in *Conference on Human Factors in Computing Systems - Proceedings*, 1990, pp. 235–242, doi: 10.1145/97243.97279.

[112] “Usability Engineering - Jakob Nielsen - Google Books.” [Online]. Available: https://books.google.gr/books?hl=en&lr=&id=95As2OF67f0C&oi=fnd&p

g=PR9&dq=Nielsen,+J.+Usability+Engineering.+Academic+Press,+San+Diego,+CA,+1992.&ots=3bGDznco_p&sig=bp1aY6wN3ZH5Hx6FZrRUTFOz1AE&redir_esc=y#v=onepage&q&f=false. [Accessed: 07-Apr-2020].

[113] R. Jeffries, J. R. Miller, C. Wharton, and K. M. Uyeda, “User interface evaluation in the real world: A comparison of four techniques,” in *Conference on Human Factors in Computing Systems - Proceedings*, 1991, pp. 119–124, doi: 10.1145/108844.108862.

[114] M. Matera, F. Rizzo, and G. T. Carughi, “Web usability: Principles and evaluation methods,” in *Web Engineering*, Springer Berlin Heidelberg, 2006, pp. 143–180.