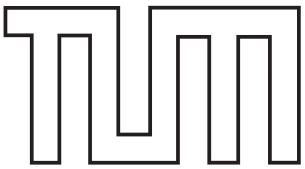


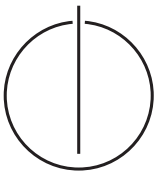
FAKULTÄT FÜR INFORMATIK

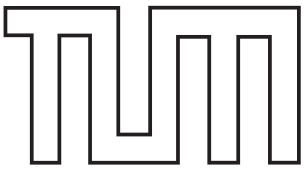
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Scenarios for Distributed User Interfaces for Single-User Recommender Systems

Wessam G. Abdrabo





FAKULTÄT FÜR INFORMATIK

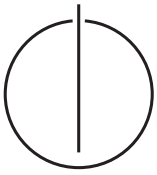
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Scenarios for Distributed User Interfaces for Single-User Recommender Systems

Szenarien für verteilte Benutzerschnittstellen für Empfehlungssysteme für einzelne Benutzer

Author:	Wessam G. Abdrabo
Supervisor:	Prof. Dr. Johann Schlichter
Advisor:	Dr. rer. nat. Wolfgang Wörndl
Submission Date:	November 15, 2015



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, November 15, 2015

Acknowledgments

Abstract

Contents

1. Introduction

1.1. Recommender Systems: Ubiquity and Distribution

The explosive dimension of digital content has raised the need for tools that could aid users in making informed decisions on consuming such content. Recommendation systems are such tools that have become an integrated part of everyday interaction with any digital content provider (such as Netflix, YouTube Spotify), online store (such as Amazon), or even Google search engine. During the past decade, it has become less and less uncommon for a user to get content tailored to her taste based on recommendation algorithms that are ever-evolving, providing sophisticated methods of implicitly building user profiles through eliciting the user's preferences, or by mining users' search histories or ratings for products and content. Recommendation algorithms varying from content-based which is done based on item-to-item similarity, to context-based which includes contextual information such as location and time in recommendation, to collaborative filtering which takes into account similar users' profiles, all provide a number of solutions to recommendation applications which usually vary in context, type of content, or kind of users. Moreover, continuous research is undergone to further develop recommendation matrices, such as novelity, diversity and serendipity, that could yield more enriching user experiences with recommended content.

On the other hand, user interface design for different recommendation phases (specially the presentation of recommendation results), is covered by an increasing amount of

research. However, given the ubiquity of recommendation systems, coupled with the increasing number of devices per user, as well as the different sizes, capabilities, platforms, and interaction modalities of such devices, little or no research is concerned with distributing the user interface of such system along the different devices used by the user in a given multi-device environment. Despite of known attempts to distribute the system itself along different computing machines, the use of distributed user interfaces in the domain of recommendation systems is an area that still is very much untackled by HCI or other researchers.
Our study identifies the need for recommendation system applications to possibly leverage the properties and capabilities of distributed user interfaces' techniques in hope of facilitating and enriching the user's experience with such systems.

1.2. A Case for Distributed User Interfaces

Looking back retrospectively to the evolution of Distributed Systems and Human-Computer Interaction (HCI), when computational tasks became more and more complex to be carried out by a single computing unit unless it could be doubled in performance, the alternative was to rather distribute the tasks to be done on a network of computing devices in parallel. Distributed user interfaces could be thought of as the counterpart of this very idea in the area of HCI. If computation executes on distributed entities, then it is only natural that user interfaces could behave similarly.
With the advancement of ubiquitous computing and the new trend of the ever-increasing number of devices per user, coupled with the demand for the inter-connectivity of these devices, users of interactive systems no longer perform tasks that reside mainly on a single device, but are rather confronted with situations where they have to interact with tasks across several platforms. A typical situation would be a user carrying out tasks in a multi-device environment that presents itself effectively to the user as a single

UI, but which is actually distributed along these platforms. Such situations represent typical cases of Distributed User Interfaces (DUIs) where "one or many parts or whole of one or many user interfaces are distributed in time and space depending on several parameters of the context of use, such as the user, the computing platform, and the physical environment where the task is carried out" [3]. Hence, DUIs represent an attempt to overcome the limitations of user interfaces that could be manipulated by single-user, on a single platform, in a fixed environment, providing no variations along these distribution dimensions.

In more concrete terms, DUIs enable end-users (who are rarely working alone, but collectively with other users, and who are no longer staying in the same locations while performing their tasks, simultaneously or not) "to distribute any user interface element, ranging from the largest one to the smallest one, across one or many of these dimensions at design- and/or run-time: across different users, across different computing platforms, and across different physical environments. In this way, end users could be engaged in distributed tasks that are regulated by distribution rules, many of them being currently used in the real world." [14]

Moreover, in a POST-WIMP(menus, windows, tool bars) era, there is a call for new methods of interaction with UIs that overpasses the traditional page or dialog oriented methods [13]. For example, Natural UIs [6!] define the content to be the actual interface with which the user could flexibly and directly interact through simple gestures. DUIs inherit such concepts and thus become candidate to replace more obsolete methods of interaction.

Evidently, UIs could no longer be thought of as concentrated, but rather as distributed across users, platforms, and environments, the three main dimensions of UI distribution. The new demand that DUIs have served in how user interfaces could evolve to meet the challenges of the increasingly distributed computer systems, as well as in providing fluid and naturalisitc methods of interaction with graphical systems, makes a case for

the integration of distributed user interfaces in the new generation of pervasive and interoperable interactive systems.

1.3. Scope and Limitations

Our study is concerned with the application of distributed user interfaces in single-user recommender systems. Our claim is that the distribution of recommender system's user interface leads to better user experiences. To apply our hypothesis, we chose the area of video recommendation. We built two high fidelity prototypes for video mobile recommendation applications: Monolithic Interface Recommender (MiRec), which a conventional mobile video recommendation application, and Distributed Interface Recommender (DiRec), which a distributed version of the mobile video recommender where the interface is distributed among the mobile device and a large-display screen. Since the goal of the study is not centered around the recommendation itself but rather the user interface distribution, the focus was not on developing a recommendation algorithm. Consequently, we used a rather simplistic recommendation of items based on top rated items in popular fields, and presented it in such a way that mimics actual recommendations.

1.4. User Study

To put our hypothesis to test, we conducted a closed user study in which participants were asked to use both MiRec and DiRec. Each participant was asked to perform a certain set of tasks on both applications including navigating through the recommended content, viewing videos' detailed information, and playing and rating of videos. Participants were then asked to express their direct impressions of the applications through a post-experiment user experience questionnaire.

1.5. Structure of The Work

This thesis aims at answering a main research question and number of derived questions that can be summed up as follows:
Main Research Question: *How can distributed user interfaces be used within the domain of single-user recommender systems in order to enhance the user experience in a multi-device environment?*
In order to answer this question, other research questions needed to be investigated:

- **Research Question 1**: *In which scenarios would single users of recommender systems need a distributed UI?*
- **Research Question 2**: *Which recommendation phases are best suited for UI distribution?*
- **Research Question 3**: *What should be the element/unit of UI distribution in a recommendation application?*
- **Research Question 4**: *What should be the span of control given to the user in deciding which tasks/components are to be distributed? And which UI distribution decision should be made by the system?*
- **Research Question 5**: *Which UI distribution strategy is best suited for recommendation scenarios?*

This work is structured as follows. In chapter 2, basics and related work are presented. Next, chapter 3 demonstrates a generic model for a distributed user interface for recommendation applications, as well as the high level design for a video recommendation application in a distributed UI multi-device environment. Chapter 4 explains the implementation details of the MiRec and DiRec mobile video recommendation prototype applications implemented for the user study. Chapter 5 presents the details of the conducted user study as well as the achieved results. Finally, chapter 6 provides a conclusion an discussion of our findings and possible future work.

2. Background and Related Work

In this section we provide the basics and background information in the area of distributed user interfaces and their application methods in different types of interactive systems, and further on, their attempted application in the area of group recommendation systems.

2.1. Distributed User Interfaces: Fundamentals, Approaches and Challenges

2.1.1. Fundamentals of DUIs: Definitions, Dimensions and Properties

Definitions of DUIs

Despite of the relative sparsity of applications for DUIs, the literature includes a number of definitions for what constitutes distributed user interfaces. Some of the provided definitions are synonymous, others are built on different foundations, which constitutes a problem of “a consensual ontology in the domain"[14]. In this section, we collect some of the well known definitions for DUIs and highlight how they complement each other.

Melchior [11] defines DUIs, in terms of their different dimensions, as “any application User Interface (UI) whose components can be distributed across different displays of different computing platforms that are used by different users, whether they are working at the same place (co-located) or not (remote collaboration).” [1,2,7,9].

A synonymous definition is provided by Demeure et al. [3], however, it is provided in the light of the time and space dimensions of DUIs and introduces the concept of the context-of-use of DUIs. They define DUIs as "those interfaces whose different parts can be distributed in time and space on different monitors, screens, and computing platforms, depending on several parameters expressing the context of use, such as the user, the computing platform, and the physical environment in which the user is carrying out her interactive task."

Perhaps one of the more comprehensive definitions is given by Vanderdonckt et al. [14] which also continues to draw on the idea of context-of-use. UI distribution concerns the repartition of one or many elements from one or many user interfaces in order to support one or many users to carry out one or many tasks on one or many domains in one or many contexts of use, each context of use consisting of users, platforms, and environments.

Context is defined as s triple of U P and E; user, platform and environment [14]. A single context of use is when a user is carrying out her task on a dedicated computing platform in a given environment. By expanding on a single context of use, DUIs are hence considered multi-device, multi-monitor, mutli-display, multi-platform and multi-user. [14]

From the previously stated definitions, one could draw on characteristics of a DUI system in terms of what changes they provide for the user's interaction with and use of the system. Blumendorf et al. [1] explain that a DUI system enables user to choose the dynamically the mode and devices of interaction based on the current context of use. A user of DUI system could also change their choice of devices dynamically and on demand. All of the user's behaviours are configurable, which should also be storeable and loadable at different times of use. A DUI system is also by definition multi-modal in the sense that it allows for the use of different devices simultaneously. And last, a DUI system should enable a user to share information with other users.

DUIs' definitions are often given in terms of the different platforms, environments, devices, and users they span, in other words, the dimensions of distribution. Elmqvist defines distributed user interfaces through the comprehensive definition: "A distributed user interface is a user interface whose components are distributed across one or more of the dimensions input, output, platform, space, and time" [4]. Thus, it is important to explain what is meant by these different dimensions of DUIs and to go in details of how the distribution of such dimensions are possible.

Dimensions of DUIs

Several studies draw on the different dimensions of DUIs [11], [3], [4], [14]. According to these studies, there are several dimensions to user interface distribution: user, time, space, input, output, platform, task, and element.
Perhaps one of the earliest studies of DUI dimensions was given by Demeure et al. by describing the 4C model, in which the distribution dimensions of user interface elements, which alters the elements' natural habitat (configuration), are given according to the four 'C' s: computation (what is distributed?), in other words the element of distribution, which could be the task or the platform, communication (when is it distributed?) or time, coordination (who is it distributed?) which is a variation on the user dimension, and configuration (from where and to where is the distribution operated? on the physical pixel level, or the logical level)[3].

The following is a presentation of the different dimensions in details, and how they are defined and mentioned in different studies.

Space. Both Elmqvist [4] and Melchior [11] agree on what defines the space dimension of DUIs. This dimension is concerned with whether the user interface is restricted to the same physical space/ geographic location, or if it could be distributed geographically.

In other words, space defines whether tasks/subtasks could be carried out in the same location, or distributed over different locations.

User. The user dimension is presented differently in different studies. Vanderdonckt [14] defines it in terms of "locus of distribution control"; whether it is in the hands of the user, the system, or in mixed-initiative way. This idea is originally defined in 4C model's coordination dimension which is a question of "who is distributing the interactive system"[3]. In other words, who is responsible for the detection of a need for distribution, computation of distributed alternatives, selection of a distribution strategy and finally the execution of this alternative. "For example, a user may decide that there is a need to do so and selects various portions of the UI which could then be migrated to other platforms she is using. When the task is finished, she may want to recall all migrated portions to restore the initial UI. In this case, detection, computation, and selection are user-initiated while execution is system-initiated" [3]. Melchior defines the user dimension differently. According to [11], the user dimension concerns how users interact with a system: whether the system involves single or multi-users, and whether the system users could be working competitively or coordinating to perform a certain task, concurrently or sequentially. Elmqvist, on the other hand, ignores the user dimension from his definition, claiming that "In other words, whether or not a DUI is used by a single user or multiple users is not pertinent to our definition."[4]

Platforms. This dimension is concerned with whether the user interface is executed on a single computing platform, or distributed along different platform [4]. What could be meant by a platform are devices with different architectures or operating systems Now, the focus is not on a multi device environment, which is an environment consisting of multiple devices or displays, but rather on how a single interface could be distributed in such an environment with each of the devices with its own interaction (input/output) modalities [4]. This idea is regaining much attention since the number

of devices a given user interacts with in a given environment are increasing and getting smaller in size. Vanderdinckt et al. mention that while there has been work done in the areas of muti-device UIs (UIs produced for several devices simultaneously) and UI migration(transferring UIs from one device to another while maintaining task continuity), “less work has been however devoted towards dividing a UI across devices, displays, or platforms, where they are used by the same user or shared by different users . . . This includes use of multiple monitors on a same computing platform by a single user, use of multiple platforms by a single user with synchronisation between, exchange of information between platforms belonging to different users, moving information between displays on a single platforms, partition of tasks across displays for a single user, sharing common information on a common display while keeping some information private on a own platform" [14].

Input. Input is consider one of the key aspects in multi-device scenarios which involves managing interaction with the different devices with their different input modalities [4]. One method of handling input in multi-device environments is by using input redirection “where the input events from one device are sent to another device in the environment.”[4]

Output. The output dimension is concerned with how the system will present the content to the user, or allow the user to control the content presentation in a multi-device environment. In other words, will the content be presented and will always reside on one device, or redirecting this content from one device/platform to another will be made possible. This is what is defined as content or display redirection [4].

Task. Many of the studies that tackle the different dimension of DUIs discuss the atomic element of UI distribution. For some, the distribution could be done at the level of UI widgets [14], and it could even go down to the level of individual pixels

[3]. For others however, what defines the basic level of distribution is usually a task. Lopez et al. [8] defines a task in this context “as the set of actions the user performs to accomplish an objective." Vanderdonckt et al. [14] mention that to be fully distributed, “one or many tasks should be considered to be carried out simultaneously or not in a distributed way.” Luyten et al. [17,18] introduced the notion of situated task in order to model how a task could be distributed into several sub-tasks to be carried out by one user, but on different platforms in the same environment over time.

Time. The time dimension is another crucial aspect of distribution and also another that is defined differently in different studies. In the 4C model, it is referred to as the communication aspect of distribution that answers the question of “when the distribution happens” [3]. Demeure et al. explains that a UI distribution is either static of dynamic depending on when the distribution takes places. If the UI distribution is static (compile or development time), this means that the distribution is pre- configured in the system and can not be changed unless the system is changed/recoded and recompiled. While dynamic UI distribution (run-time) allows for the change of the UI distribution scheme or strategy at run-time, for example, when on a given user input or behavoir, without the need to recode or recompile the system [3]. In dynamic systems, when the UI requests the loading of a new component, “the control might select from a list of UIs with different capabilities of supporting various habitats, choosing the one that most closely matches the new habitat” [3]. For example, if a user starts a UI on a mobile device, the system “might load a minimal display component to guarantee proper presentation.” [3]

A different definition to the time dimension is given by Melchior as “some sub-tasks are carried out during different time intervals, depending on who is contributing to the task." [11]. While Elmqvist et al. [4] define time as in interactive systems as the aspect

that allows the UI elements to "execute simultaneously (synchronously), or distributed in time (asynchronously)."

Properties of DUIs

User interfaces of interactive systems hold a number of properties in order to be considered fully distributed. The following is a set of some of the DUI properties that were mentioned in different studies.

Portability. A UI is said to be portable if "the UI as a whole or elements of the UI can be transferred between platforms and devices by means of easy user actions." [8]

Decomposability. A DUI system is said to be decomposable if the set of elements conforming the UI could be executed independently across different platforms, while keeping their functionality.[8]

Simultaneity. This property refers to the ability of different UI elements of the same DUI system to be managed in "the same instant of time" across different platforms. [8]

Continuity. is the system's ability to maintain its state while transferring elements of its UI from across different platforms in the distributed environments.[8]

Consistency. A DUI system is consistent if it manages all of its distributed components in the same manner.[8]

Flexibility. with a flexible DUI system, the user could perform the same action on different platforms in different ways.[8]

Multi and Cross. a DUI system is said to be multi and cross: device, modal, user and application [1], addressing more than one UI element at once on different dimensions, while cross refers to the dynamic change of elements at run-time.

Usability. Usability of DUI i a crucial property to achieve. A number of studies are dedicated to how to make DUIs, with the challenge they present to users, to become more and more usable, easy to learn and to interact with. Considering the different dimensions of DUIs (i.e. varying devices, multiple modalities, different users, multiple applications), it might be difficult for users to keep their mental model of the interactive system up-to-date [1]. The user is challenged by interacting with a dynamic and widely distributed system of possibly not obviously related elements that can also change for not immediately obvious reasons. Which requires special consideration for the usability and intelligibility of DUIs [1].

Controllability. denotes the span of control of the user over all aspect of the distributed UI. Thus, users should be enabled to configure, reconfigure, save configuration of the system which should statically or dynamically enable them to tailor the distribution of UI to their needs. One approach to provide users controlability is through the use of a meta-UI which is suggest by both the 4C model [3] and adopted by Blumendorf et al. [1].

Continuous Interaction. With this aspect, the interaction with the different components of the distributed systems should be continuous and undisturbed by the shifts between applications, devices, or displays. Hence the actual changes should be kept to a minimum so as to provide "a continuous and consistent user experience without confusion and should conserve the state of the interaction"[1].

Configuirability One of the strengths a UI distributed across multiple devices has over a monolithic system is the ability to flexibly add or remove computing resources as circumstances dictate. This refers to the dynamic aspect of a DUI system [2]. So another main challenge is to allow the dynamic handling, selection and adaptation of different UI variations at runtime by the user of the system. A key factor is the user who is required to be able to influence this process and override any kind of developer configuration [1]. This raises the challenge of uncertain and unknown run-time contexts which should be handled by the system.

Splittability. of UI components in a given container of a distributed system is the ability of such components to be presented separately "depending on the constraints imposed by the user's task corresponding to the container".[3]

Migratablility. Migratable user interfaces are defined in [4] user interfaces that allow distribution at a UI component level through migration, such that an abstraction layer redirects parts or the whole of the interface along the application's hosts. Different techniques are used to achieve migratability. For instance, one that is similar to what we use in our study is what is used by Bandelloni and Paterno [5] use a migration server to replicate the runtime state and adapt the interface accordingly. Whereas a model based approach for migrating UIs is taken by Mori et al. [41]. Blumendorf et al. [1] refer to complete versus partial migration/replication, where only part of the UI is involved.

Plasticity. is the ability of a UI to adapt to a new interface in; specially in a multi-device environment with different input and output capabilities. Thevenin and Coutaz [67] defined it as "the capacity of a UI to withstand variations in both the device and its physical environment while preserving usability." In practice, this means that a plastic interface should be able to adapt to different screen sizes (mobile device, laptop,

wall-sized display), as well as to different input devices (touch, mouse, voice, gesture, etc).

2.1.2. DUIS: Models, Toolkits, Frameworks, and Studies

While much research tackles problems that are of a DUI nature, few authors take the conceptual step to generalize these problems into models, frameworks, and toolkits supporting DUI development [4]. In this section, we survey some of the studies that aimed at providing DUI models, design guidelines, frameworks, applications and toolkits, as well as a reference to some of the existing DUI systems.

When it comes to DUI models, an early model was proposed by Demeure et al. in 2005 and then refined in 2008 [3] into the 4C model which is perhaps the most well known and widely referenced in most studies. The 4C model consists of four basic components—computation, com- munication, coordination, and configuration—that captures the what, when, who, and how aspects of the distribution. Melchior et al. [11] propose a model-based approach that "capture the abstract operations and requirements necessary for typical DUI systems and toolkits." One of the earliest reference models for DUI is the CAMELEON-RT which is a middleware software infrastructure for distributed, migratable, and plastic interfaces [4, 16]. Another approach is to combine software engineering methods with DUI models. As early as 1996, Graham et al. [24] proposed a distributed version of the model-view-controller (MVC) paradigm. Vandervelpen and Coninx [70] apply model-based methods to user interface design, similar to Mori et al. [41]) but specifically targeted at heterogeneous device environments. Luyten and Coninx [37] also target such environments, but take a bottom-up approach focused on designing user interface elements that support seamless distribution. Finally, the recent views, instruments, governors, and objects (VIGO) model [34] can be used to build distributed ubiquitous instrumental interaction applications.

Manca et al. [9] describes a model-based approach for the desciption of DUI on top of an extension of the MARIA language. The study explains that there are 4 levels at which the distribution could occur: elements could be distributed across multiple devices, elements could be assigned to one given device, elements could be replicated in multiple devices, or elements could be by either one device or another. (here we could see which approach. sometime we replicate and sometimes we choose one device.)

Froberg et al. [5] defines the MARVE framework for building DUI systems. The goal is to provide an approach as similar as possible to the traditional way of ui components placement, allowing for a smooth transition from GUI to DUI development. (we could mention the systems developed using MARVE). Similarly, presentation modes are defined for components atomic presentation refers to when a component can only be visible on a single device at a time, mirrored refers when a component is placed on two or more devices at any given time, and cloned presentations refers when a component is placed on two or more devices at any given time, but each component is unique and not interconnected with the source component.

Jetter et al. [6] propose the ZOIL design paradigm for DUIs.

The domain of Distributed User Interfaces (DUI) is still in evolution and there exist no toolkit allowing the creation of DUIs. In most pieces of work, there is almost no genuine DUI.[12]. DUIs toolkits are important for ubiquitous computing, where data and computation is integrated into everyday objects and activities.[4]
Melchior et al. [10] provides a toolkit to support distribution at both design-time and run-time with very fine and coarse- grained granularity and to support replicable distribution while being compliant with the DUI goals as in [5].

A number of user surveys were conducted in various studies to investigate users interaction with DUIs. [14] Beale Edmondson [4] conducted user surveys to determine the user behavior induced by using a DUI: they identified the importance of having multiple carets and the complexity of multi-tasking and they suggest design implications for using DUIs in order to support distributed tasks. In particular, they stressed the importance of a multi-tasking model that is partially built at the local level of a single user and at the global level across users when collaboration exists. The global scenario should be also dissolved into local scenario in order to preserve the consistency between common tasks and individual tasks.
[14] Tan Czewinsky [23] found out that physical discontinuities had no effect on performance, but found a detrimental effect from separating information within the visual field, when also separated by depth. Due to the multiplicity of interaction techniques in DUIs, Nacenta et al. conducted a study to compare the efficiency of six techniques for moving objects from a platform (e.g., a tablet) to another one (e.g., a tabletop) in four different distance ranges and with three movement direc- tions. Their study suggests that spatial manipula- tion of data was faster than pressure-based tech- niques.

[3] One the one hand, several DUI systems exist such as IAM [6], i-Land [17], Stanford Interactive Mural [11], Aura [16], ConnecTables [18,19], Dygimes [20], DistriXML [9]. [4] The ConnecTable is a table- centric information appliance for seamless coupled/decoupled collaboration [65]. iStuff is a physical UI toolkit for UbiComp that incorporates a wide range of physical components to be used in an interactive workspace [3]. Similarly, the u-Texture [36] physical panel can be used to effortlessly build smart environments from simple and easily configurable components. 4 N. Elmqvist. The BEACH system [64] is one such infrastructure and was used for the ConnecTable project. Aura [58] is a software architecture for supporting dynamic variability of computational resources in UbiComp environments. Similarly, the Gaia [56] middleware infrastructure supports

resource management in physical and interactive computing spaces (called Active Spaces). Additional frameworks include MediaBroker [39, 40], a recent toolkit for building peer-to-peer DUIs [38], and a visualization toolkit for distributed collaboration [33].

Moreover, a number of studies [13], [14] consider the challenging task developers of DUIs face and a offer a number of design guidlines, patterns and anti-patterns that should be considered on designing interfaces for distributed in multi-device environemnts.A number of studies provide Design Patterns

2.1.3. Dual Display:LD/SD

The whole section is taken from [7]

A promising approach in interacting with large public displays has been the use of ubiquitous cell phones which not only offer a means to interact with displays, but increasingly offer a small, but high quality screen to complement the larger public display.

Extending interactive large displays (LD) with small devices (SD) such as PDAs or smart phones has been discussed in earlier research efforts [3, 6]. The main idea behind this approach, which is also known as the Dual Display approach [2], is to execute a user interface across LD and SD to take advantage of input and output capabilities of both device types at the same time. Dix and Sas [3] argue that such an approach could help designers to solve GUI design issues due to multi-user interaction with large public displays.

What to show on LD/SD

Current research work with large public displays rests on the assumption that interaction feedback and user requested information (output data) can be presented on

LD, SD, or a combination of both. Furthermore, it is assumed that coupling LD with SD during interaction helps to reduce the load of information presentation on the LD and increases users' ability to manage content on large displays, mainly because of users' inherent experience in using their phones. What seems to be missing from the current research work is identifying differences in design requirements for interactive and non-interactive widgets depending on whether they are placed on LD or SD. There is lack of clear guidelines on how users respond to placement of elements in a user interface on LD or SD.

Simlar Goal to our study!
Our primary research question is to verify if users benefit from executing an application across large displays and small devices taking advantage of input and output capabilities of both devices, i.e. LD and SD. In other words we would like to understand if and how splitting interface entities (user interface widgets) across LD and SD affects user task performance when interacting with applications designed for large public displays.

Which widgets to show on which display. LD, SD, LD-SD. To free up resl estate on LD, use SD. For multi-user interaction use LD. Mirrored mode: redundancy and solves neither problem.
Building on top of Norman's seven stage model of interaction, we have identified four different ways that interactive and non-interactive widgets can be distributed across the mobile and large displays.

the use of gestures -> cite woerndl

2.1.4. Challenges and Concerns of DUIs

Concerns for UI distribution decribed in [11] Concern 1. Development of distributed user inter- faces: the development of DUI is not supported by usual tools. Most of the time, developers have to man- age the development in their own way. A lot of time is spent on the development of DUIs mostly the distrib- uted aspects. Concern 2. Support for distribution of user interfaces at running time: existing DUIs are limited to predefined applications and domains of application which lead to little support for the various possibilities of distribution. Concern 3. Support for multi-user collaboration: mul- ti-user applications are developed in different ways de- pending on the use and domain of application. The lack of a common base is slowing down the development. Concern 4. Execution control in the distributed envi- ronment: the control of the distribution is a real prob- lem when managing DUI systems [4]. The limitations are high especially with a fixed level of granularity. Some systems can replicate windows while not being able to replicate widgets. Others can manipulate wid- gets one at a time but no group of widgets. Concern 5. Network transparency: The distribution of the UIs has to be network transparent in the sense that the user should not have to worry about network details such as IP address, user network and network settings. Concern 6. Lack of description of the distributed do- main and models: The researches around multi-user ap- plications and distributed user interfaces are very spe- cifics to the needs of the developers and are almost never documented or badly documented. Model-based Approach. .

[12]The problem is that the granularity of UI distributed elements is often coarse-grained; it is not possible to distribute at the widget level.

???Benefits of DUI [2]
Distribution Enables Redirectable Interfaces In a DUI construction, interfaces can be

redirected and shipped around electronically so that the user can interact using a proxy device. In our system, this has been useful for minimizing physical interactions when these interac- tions are undesirable and to allow users to effect changes without incurring the costs of device switching. Distribution Helps Interfaces Scale In Both Directions The devices available in a given environment are deter- mined by a combination of factors that include the avail- able infrastructure, concerns about portability and users' personal preferences. By allowing UIs to be broken down into distributable pieces, the UI can better accommodate both device-rich and device-starved situations. Since a DUI-based system allows users to easily multiplex UI ele- ments across devices in space or time, the system can take advantage of additional devices, while offering fallbacks (like cycling through UIs) for device-starved environments.

2.2. Recommender Systems

Tasks

2.3. DUI in Recommender Systems

nothing except: cite the group recommendation system

3. Design for a Single-User Recommendation Application in a DUI Environment

In the previous chapters, the foundations of recommender systems were introduced, as well as the basics of distributed user interfaces, their definitions, properties and the different UI distribution dimensions. In this chapter, a design for a single-user recommendation application in a distributed UI environment is proposed.

The motivation of the proposed design could be best described through the following scenario: A user of a recommendation application receives recommendations on his/her mobile device. The user of such application might be willing to migrate the recommended content to be consumed on a different device in his/her environment, as his/her mobile device's battery might be expiring, or the consumption of the recommended item would be more convenient on the other device.

The actualization of the previous scenario depicts a multi-device (and possibly multi-platform) environment, in which the flow of control (logic) and application's user interface are decoupled in a way that allows for the distribution of UI components along the different devices. In other words, the user of such system is provided with a distributed solution, which enables him/her to perform tasks on whichever device in this environment (by for example migrating the UI components between the different devices) independently of where the application is running, and of the constraints

presented by the different platforms running the application.

The following section starts by describing a generic model for UI distribution of recommendation applications, followed by a description of more specific scenarios relative to our distributed video recommender application.

3.1. Generic Model for Recommender Systems UI Distribution

As debriefed earlier, there are different dimensions to UI distribution. The proposed design is described through different UI distribution scenarios with respect to the following UI distribution dimensions: time, user, platform, and task. This section describes how the different distribution dimensions could alter the design decisions for a system whose UI is targeted for distribution.

3.1.1. Regarding Distribution Dimensions on Designing Distributed UIs

Time: When to Distribute?

The aspect of when the UI elements of an interactive system are to be distributed (statically at compile/load time or dynamically at runtime) is a key design decision. One way to distribute the UI is to have the distribution decisions made prior to execution, without providing the ability to alter these decisions. If the UI elements are to be distributed dynamically, for example, based on the user's needs that can only be known at runtime, a design decision to make in such case is to prepare a set of UI configurations that the system could use on loading components to adjust the UI accordingly. In such case, the system delays the final decision to which UI elements to distribute, instead of having this option preconfigured.

User: Who Initiates Distribution?

In most of the scenarios, identifying whether the user or the system initiates the distribution is essential to the design. Creating a scenario in which the user detects a need for UI distribution, computes and selects an alternative scenario, requires the system to be more adaptable than the case when the system initiates the distribution. When a user initiates the UI distribution, they could select components to be displayed on the systems' platforms. Later, they could also reside to the possibility of restoring the UI to its original state, or keep the new configuration saved for later use. Alternatively, the system designer could detect a need for distribution and configure the system to initiate the UI distribution on, for example, carrying out specific tasks, or if the system is loaded with a specific configuration.

Platform: Where to Distribute?

It can be fairly assumed that a distributed scenario is probably going to involve different platform. In every distributed scenario, on which platform the UI components are decided (by system or user, dynamically or statically) to reside would alter how the UI needs to be adjusted to best fit the new platform. For example, allowing a UI to be transferred between a limited display of a PDA and a larger display such as a tabletop would require adjustment for how the different UI components are to be adjusted to fit the limitations of each platform.

Task: What to Distribute?

One perspective to look at the distribution of recommender system is to identify the recommendation tasks that could be distributed. For a system to be distributed, one or many tasks should be considered to be carried out simultaneously or not in a distributed way. A task could be distributed into several subtasks to be carried out

by one user, but on different platforms, in the same environment, over time. For the proposed distributed recommender system design, the following tasks are considered for distribution: presentation of recommended items, item consumption, recommended content filtering, recommended content rating, and sharing of recommended contents along different users of the system.
UI components constituting the interactive systems through which a user could undertake such tasks could be thought of as the unit of UI distribution. In the distribution of such tasks, full or part of the UI can be transferred among platforms and devices through a simple user action, such as a gesture. For user input and interaction with the system, the use of gesture, such as panning and swiping, is considered to reduce cognitive overhead [cite]. The decomposability of the system components enables one or more of the distributed UI elements to be executed independently without losing their functionality. It is also important to ensure the consistency of performing the different distributed actions; i.e. to ensure that a distributed action (for example phase of recommendation) is supported on the different platforms. Such actions can be carried out in various ways, hence, increasing the flexibility of the system.

3.1.2. Generic Scenarios for UI Distribution

Presented in this section is a set of generic scenarios for UI Distribution of interactive systems that are thought to be applicable for recommender systems. The need for UI distribution is detected in each scenario, and later, distributed alternatives to the regular scenarios are computed, with an execution strategy provided for such alternatives. In all scenarios, we consider the recommendation tasks as the core of the distribution scenario. We assume the existence of multiple platforms and devices in each scenario. Moreover, in all of the scenarios, we evaluate the time distribution and user/system distribution dimensions.

Migrating Item Consumption

In almost all types of recommender systems applications, there is an item to consume. Therefore, one of the main distribution scenarios to consider is the distribution of the task of recommended items' consumption. Unlike the regular scenario, where the user of a recommender system is presented with the recommended items and is able to select and consume an item on the same device, a distributed alternative would be to present the recommended content on one device while giving the user the ability to consume the content on the other device. As shown in figure 3.1, this scenarios could be triggered by the user performing a gesture on one of the items presented on the presentation node (i.e. the node that would contain the presentation of the recommended content). The item consumption task is now migrated to be carried out on the other device/platform. Here comes the question of time of distribution of the

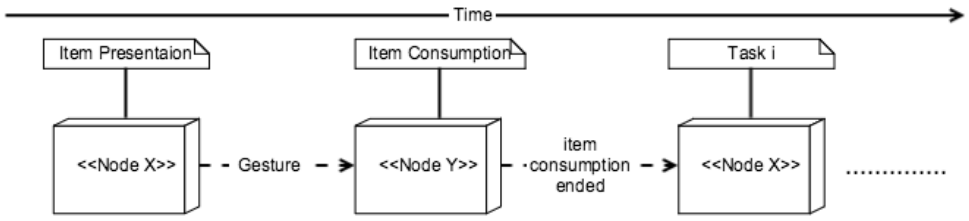


Figure 3.1.: Migrating Item Consumption from Presentation Node to Consumption Node.

item consumption along the two platforms, as well as the question of who (user or system) makes the decision of which UI components of this scenario are to be displayed on which platform. These questions add variations to the scenario on hand.
The system could load the UI components with a pre-configuration allowing the user to only perform the action to be done in a specific manner. Alternatively, the system could dynamically at run-time elicit from the user a distribution scheme for the UI. For example, the user could be asked to choose which UI component (relevant to this

scenario) are to be migrated and which are to reside. The system delays the decision of which UI components to show on which platform. Beside the item consumption task distribution, this variation of the scenario demonstrates the time distribution and user distribution dimensions.

Performing Parallel Activities

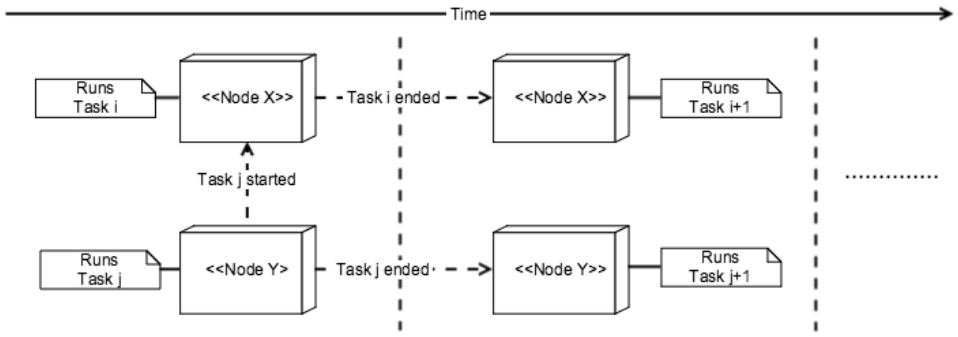


Figure 3.2.: Caption

In a distributed environment, presenting the user with different platforms through which the user could perform tasks simultaneously could be leveraged to perform parallel activities. In this case the distribution is achieved through distributing different interface components to be executed asynchronously. In contrast to the non-distributed scenario in which the user usually can not perform UI related tasks in parallel (asynchronously), but would usually wait for a UI task to be done before he/she could start the next (synchronously). Figure 3.2 is showing how a task could start running on a given node in the distributed environment, denoted here by task j on node Y, which triggers task i to start on node X. In this case, tasks are run in parallel and the user can perform these activities independently from each other and simultaneously. After the tasks end on their given node, execution of proceeding tasks could be carried out also in a distributed manner. The system could dictate this on the user and present the

distribution strategy to perform these tasks in parallel not as an option but as how the system functions. Alternatively, the system could let the user configure which activities are to be done in parallel. This configuration is saved and loaded whenever these actions are to be performed. The user could also choose to restore the system's original configuration or let the system be the initiator of the distribution.

Overview And Detail Presentations

In a distributed environment with multiple displays, LD-SD modes are used to show different versions of the presented content to the user. There is always a concern on using different display modes of what to choose to present to the user on which display/mode and what would be the basis for this choice. There is also the question of how to present the content differently on each mode based on its capabilities.
The presentation of the recommended content could be done at a different level of granularity. Fine granularity is offered on SD while coarse level of presentation is provided on the LD. This way, we provide the user with alternative ways to view the content. At a glance, he/she could get an overview of the provided content with a graphical indicator for the recommendation score on the LD. If more details are needed, the SD would provide more detailed information. This concept of UI distribution is what comes to be known as overview-detail coupling [cite].
Figure 3.3 suggests presenting a detailed presentation of content on the SD on which the user could get all available detailed information of the provided content (for example in a form of a detailed list), while on the LD, the user could be presented with an overview of the same content (for example, in the form of picture icons with titles presented with in different sizes). Selecting an item on either mode later could lead to the execution of a proceeding task on either platform such as item consumption or viewing more information for the item.

The previous design is to be inherited by the system, which makes the UI distribution

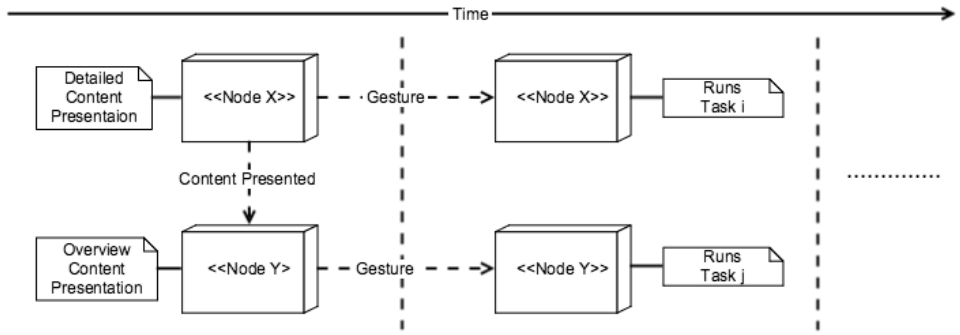


Figure 3.3.: Overview-Detail Presenations

strategy in this scenario initiated by the system and not the user. It also implies that the distribution of UI components related to this scenario are statically known to the system at compile/load time without the possibility of having them dynamically configured or reconfigured at run-time. Alternatively, the system could ask the user if this configuration makes sense to him/her, and then adds this option as a setting in the system and load the components accordingly. That would be a case of time distribution.

Content Filtering

As the user is presented with a wide array of recommended content, he/she might be willing to filter his/her choice of what to consume of this content. On a multi-device/platform environment, the filtering task could be distributed along the different nodes of the system. As shown in figure 3.4, the user could initiate a gesture on items on node X to be transferred/redirected to node Y. By the end of this process, the user ends up with a presentation of all the selected/filtered items on node Y. The user could next select one of these items to view more details. This is an example of synchronous task distribution along different platforms.

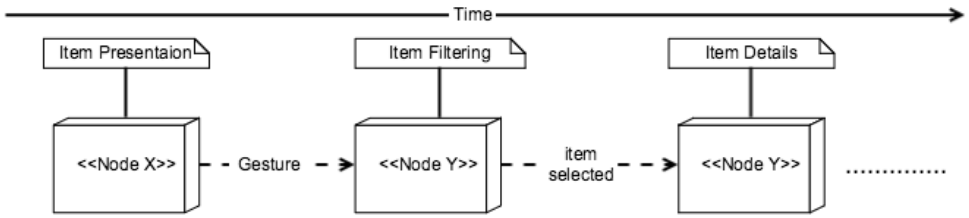


Figure 3.4.: Recommended Content Filtering

Content Redirection

Content (output) redirection is a key feature in a multi-device environment [cite]. It is argued to be more important in case of multi-user systems, however, it could still be leveraged for single-user scenarios. Item consumption redirection is considered one special case of content redirection. Other examples would be redirecting recommended content details to be presented between SD and LD displays. Figure 3.5 shows how node X presents the recommended content, while on performing a gesture on any of the items, this content could be transferred/redirected to be presented on node Y. This usually means the distribution of UI components representing this content is what enables the redirection of content seamlessly between the two presentation nodes. The user could possibly then proceed with item consumption on the same node or on a different node.

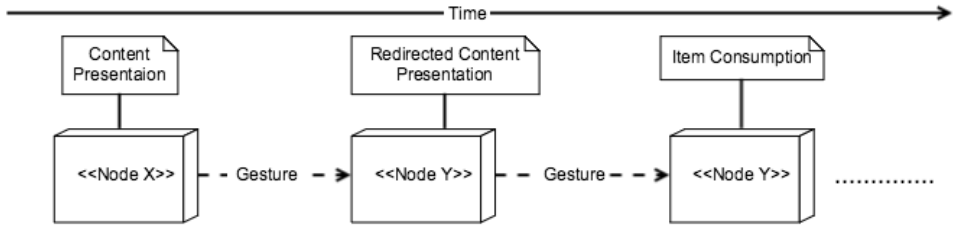


Figure 3.5.: Content Redirection

This scenario is inherently initiated by the user since he/she would be detecting and executing the need for UI distribution of a specific component or group of components along the different platforms. It is also inherently time distributed, since the system dynamically reconfigures the UI based on the user's input that could only be elicited at run-time. After performing such task, the user could select to have the system reconfigure the UI to its original state before the redirection took place.

Migration of Items Between Users

Another special case of content redirection is the migration of a list of recommended items (or an item in this list) from one user of the system to one or more other user. This could be thought of as a list of favourited items that the user creates, or a list of top-rated, or last-view items that the system keeps track of on behalf of the user and keeps them available in his/her profile. Sharing of such content between system users could be initiated by performing a gesture on this list, and getting this list to be transferred to a designated user/s. This is a case of user distribution in which the user detects the need for distributing his/her favourites list. Hence the initiator of the distribution in this scenario is the user rather than the system.

3.2. Design for a Single-User Video Recommender System in a Distributed UI Environment

The proposed design is for a single-user, multi-device video cast recommendation application, in which the user interface and recommendation subtasks are distributed along the devices. The goal of the design is to leverage the capabilities of the distributed UI environment in order to find a new way to present the recommendation tasks and results to the user. In this section, we show how the recommendation tasks such as presenting the recommended items, item consumption, item rating, and others could

be done in a distributed fashion. Our design is mainly targeted at benefiting from both platforms to make the post-recommendation phase of such system more efficient and more user-centric.

3.2.1. UI Distribution In a Single-User System

Our goal is the UI distribution of a single-user recommender system. The case for UI distribution of group recommenders in a multi-user environments has been made by a number of studies [cite]. The group recommendation involve multiple users with multiple devices. And since the main goal of a group recommendation is to reach consensus about a recommendation, distributed UI components that could be shared among the group members to reach this consensus is needed.

Less research have been made to make a case for the use of UI distribution of for single-user recommender systems. In such system, since a single user is involved in the process, and the consensus activity does not exist, it could be more challenging to think of what the distribution could be useful for in a single-user scenario. The main question on designing for such a system is to answer the question of why a single user would need a distributed environment; one with multiple devices and platforms. Why and if the action of recommendation for a single user with its various phases could be carried out on multiple devices/platforms, and whether the overhead(refer to the overhead described in earlier chapter) of distribution such a system is of an added value. The main challenge would be the introduction of a distribution mechanism for this system in such a way that does not hinder the process, and that would actually serve it. Also, the distribution should be done in such a way that does not introduce an overhead of shifting the user's attention from one platform/device to the other.

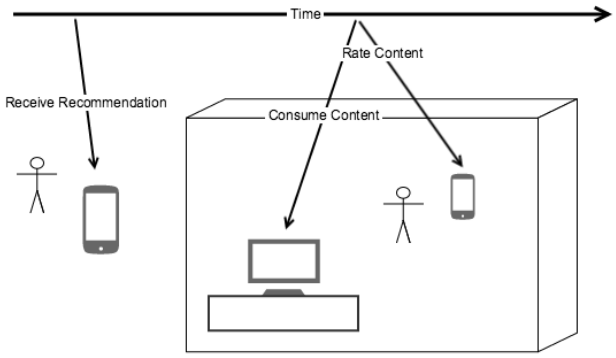


Figure 3.6.: Overall view of the system's environment

3.2.2. Platforms and Environment

To address some of the challenges mentioned in the previous section about designing for single-user in a distributed UI environment, we started by thinking of a scenario in which the user would be in need of, or would prefer to, use multi-devices to carry out the recommendation tasks. This lead us to think of a scenario that would start in a mobile environment. Figure 3.6 describes how the user receives recommendation for a new video cast on his/her mobile device while for example in commute. Since video content is usually lengthy and would be better viewed on a larger display than the limited space available on a mobile device display, the consumption of this content is thought to be better migrated to a large display device. The user could then migrate the consumption of this content to the LD once he/she reaches the environment that contains the LD (office, home, etc..). The application would then continue to operate in SD-LD mode; i.e. distributed among the mobile device (SD) and an LD, such that the user would be able to watch the video on the LD. In the following section, the distribution of video consumption as well as other recommendation subtasks are described.

3.2.3. Recommendation in a Distributed UI Environment

In this section, concrete UI distribution scenarios of pre and post recommendation tasks are provided. The reason for which tasks or UI components are selected to be displayed on which display is elaborated, as well as pinpointing the cases of time and user distribution in each concrete scenario.

Pre-Configuring UI Distribution Options

This scenario presents the initiation point of the system, in which the user is given an option to pre-configure the different options the system offers for UI distribution, and hence be the initiator of UI distribution. What this offers is the ability to delay the decision of which UI components to present on which platform making the system distributed in time. Moreover, it gives the user the option of detecting a need for distribution and deciding on the UI distribution scheme he/she would prefer instead of delegating this task to the system. This is made possible by presenting the user with a Meta UI in which he/she are asked to drag and drop the components of their choice to the target platform. After this step is done, the user is also asked if he/she would like to save this configuration for later use and setting it as the main option or if it should only be used once, in such case, every time the system reboots, this Meta UI will be loaded to elicit the user's distribution preferences. On the system side, this scenario is a case of time distribution, as the system could then decide dynamically on a UI distribution scheme.

User Profile Creation

The pre-recommendation phase usually starts by the elicitation of a user profile. Creating a profile usually involves filling out information forms, and likely rating some prototype items or entering preferences. This step would differ from one system to the

other. For the video recommender, this involves asking the user directly to enter basic information and to rate a list of topics and interests to give a background about the user's preferences. The assumption in this step is that filling out textual forms as well as rating would better be performed on the handheld device than a larger display. The user is assumed to have more control over input for smaller devices. Therefore, this task starts at in the SD. After the user finishes the profile creation step on the SD, the recommended content is then displayed on both SD and LD displays. This scenario

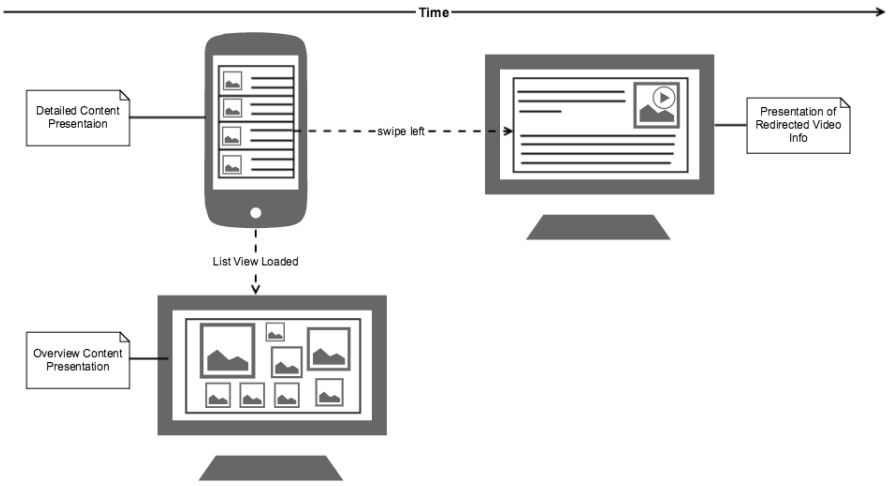


Figure 3.7.: Presentation of Recommendation Results in LD-SD Modes.

would be presented as such given that the user has selected to have the profile creation step done on the SD in the pre-configuration step described earlier.

Presentation of Recommendation Results

The presentation of recommended video casts is shown in parallel on the SD and LD, however, in different formats. We propose to present the recommended content at a different level of granularity for each platform; fine granularity is offered on the mobile device while coarse level of presentation is provided on the LD. As shown in

figure 3.7, for the mobile device, a detailed list of all the recommended videos, together with detailed information about the video, are shown in tabular form with different categorization. On the LD, an overview presentation is shown for the recommended items that scored the highest for the user without details, however shown in different sizes to indicate the recommendation score. This scenario is initiated by the system, hence the user would not have the ability to change the UI distribution scheme for presenting the results.

Recommended Item Details Presentation

For presenting the user with detailed information about the recommended videos, the recommendation list offers a link to a detailed information page on the mobile device in a master-detail fashion. As the user clicks on the item in the list, the user enters the detailed information page on the mobile device.
Before loading the details-view for the first time in on the SD, the user is prompted with a dialogue asking if he/she would like to redirect the details view to be displayed on the LD, or to show the details in parallel on both displays. Later, the user is also asked if this configuration is to be saved for later use. Based on the user's choice, the details view is either loaded on the SD or LD or both.
Alternatively, the user could be the initiator for the UI distribution by deciding to redirect the content as the need is detected. As shown in figure 3.7 content redirection is possible for viewing details information on the LD by applying a simple swipe-right gesture on any of the video items on the SD list. Consequently, the video details content will be redirected to be displayed in parallel on the LD.

Recommended Item Consumption

Similar to the recommended item details scenario, item consumption redirection is also given as an option to the user by asking the user if redirecting item consumption to the

LD is his/her preferred option or if it should the content should be played on the SD, and saving this configuration for later use. Consequently, if the user selects to consume the content on the LD, playing the recommended videos is done as depicted by figure 3.8. On the video details page, the user performs a pan gesture on the video image, which then triggers the migration of the video consumption from the mobile device to the LD. The video player automatically stars on the LD, providing the user with all controls for the video playback. Alternatively, the system could only enable playing

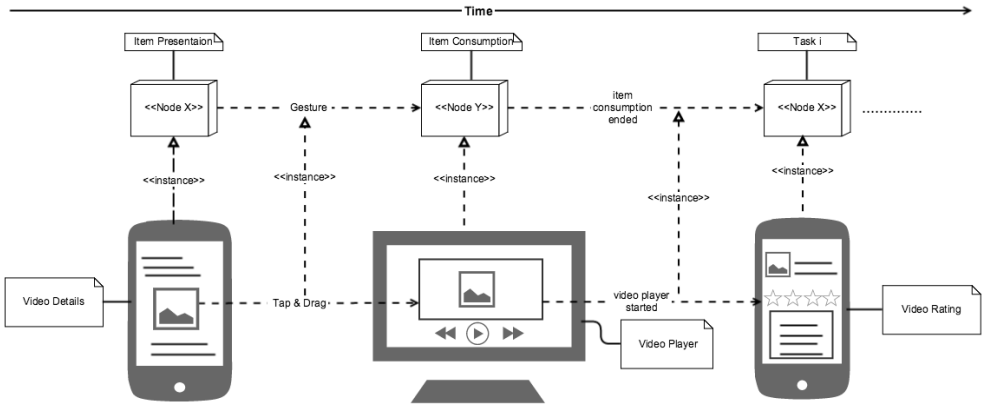


Figure 3.8.: Recommended Items Consumption.

the video on one display, SD (as done on the non-distributed version), or would only enable playing the video by redirecting it to the LD.

Rating of Recommended Items

There are two different proposed options for the rating task. The first option is distributed in time and done in parallel with the video playback. As shown in figure 3.8, after the video playback starts automatically on the LD as described in the previous section, the LD triggers the mobile device to display the rating page for the user on the SD. Hence, the two tasks could be carried out simultaneously by the user. Alternatively, the user could also be triggered to enter the rating on the mobile device after the

video has ended playing or if stopped by the user on the LD. In such case, there is no parallelization of the tasks, however there is still synchronisation between the tasks that are carried out synchronously on the different platforms. In such case, the system would be the initiator of distribution rather than the user.
The rating, similar to the user profile creation, include user input such as indication of likes or entering textual comments and reviews, which is also believed to be best done on a mobile device with a better controlled input, for the user's convenience.

Filtering Recommended Items

The filtering for the recommended items, although not a main task in recommendation, is believed to add to the value of the system in the proposed distributed UI environment. It is one of the tasks that leverages from the availability of the different platforms for the benefit of the user's experience. As shown in figure 3.9, the filtering is done by performing a left swipe gesture on the video item in the list view on the mobile device. This gesture redirects the content of the video to the LD. The display of the content

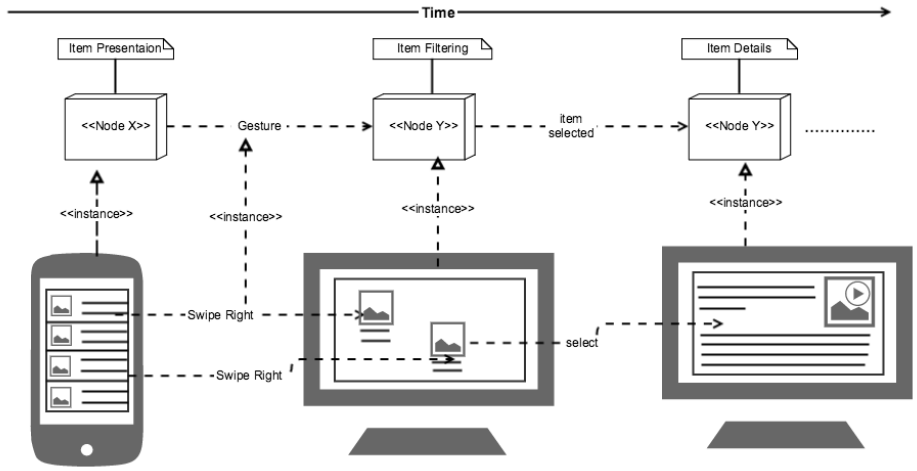


Figure 3.9.: Filtering of Recommended Items.

on the LD is also done in a overview-detail coupling manner as described in earlier section. After the user is done filtering (redirecting selected content from SD to LD), LD will contain all the selected items displayed in an overview fashion. More details could be access through the SD or by also clicking on the item on the LD.

Redirecting Favourites Lists

Unlike all previously described scenarios which involve a single user of the system, this scenario involves two or more users. On the SD, the user selects lists which hold his/her favourite items, top-rated items, or last watched items. On applying a long-press on the list, the user is prompted with a list of users from which he could select one or more users to transfer this list to. This type of scenario, similar to content redirection and item consumption redirection, is initiated by the user as found wanted , hence, the scenario is consider to distributed by user.

Table 3.1 sums-up the presented scenarios with respect to all the mentioned UI distribution dimensions. In the following chapter, the implementation details of a selected subset of the distributed scenarios are described.

Scenario	**UI Distribution Dimension**			
	User	**Time**	**Task**	**Platform**
Pre-Configuration of UI Distribution Options	Yes	Yes	Yes	Yes
User Profile Creation	No	Yes	Yes	Yes
Presentation of Recommendation Results	No	Yes	Yes	Yes
Recommended Item Details Presentation	Yes	Yes	Yes	Yes
Recommended Item Consumption	Yes	Yes	Yes	Yes
Filtering Recommendations	No	No	Yes	Yes
Redirecting Favourites Lists	Yes	No	Yes	Yes

Table 3.1.: Summary of UI Distribution Scenarios with respect to UI Distribution Dimensions: User, time, task and platform.

4. Implementation of a Prototype for Distributed-UI Scenarios

In chapter 3, a conceptual design for a single-user video recommendation application in a distributed UI environment was introduced. The goal of this design is to improve the recommendation process by leveraging the capabilities of the different platforms along which the UI is distributed. To put the design to test, a high-fidelity prototype was developed to mimic the functionalities of the complete system, hence, making it possible for evaluation through real user experience. A subset of the suggested distribution aspects with respect to the different distribution dimensions (user, task, time, space..etc) which was presented in the design chapter is selected for implementation. The selection criteria of the scenarios to be implemented was to select the ones that would be verifiable using the setup we had in mind for the user study. Since our user study involves only single user, scenarios that need more than one user at a time to verify (such as distributing a favourites list between users) where left out from the implementation. Moreover, our focus is the distribution of the post-recommendation phase. Therefore, tasks and activities involved in the pre-recommendation are not implemented. A recommendation engine is also not included since the focus of the study is not with recommendation generation. Instead, we provide the user with a set of pre-selected video items. This section provides the set of functionalities included in the prototype, as well as an explanation of implementation details of the prototype, together with explanations for the rationale behind using the frameworks, programming languages,

platforms and tools used in the course of implementing this prototype.

4.1. Prototype Versions and Functionalities

The prototype of a video recommendation application was implemented as a mobile application. The implementation of the prototype was done with the user study in mind. Since the planned study is designed to test our hypothesis of whether the UI distribution of such a system is of added value, the study is based on comparing user feedback on using two different versions of the prototype: a distributed version and non-distributed one. This section explains the difference between each version through describing the different use cases of each version.

4.1.1. Non-Distributed Vs. Distributed Prototype Versions

The first steps of designing the prototype was to think of the different versions in terms of user and system functionalities that would be available through each version. The non-distributed version of the application is deployed as an iOS mobile application only. The distributed version is distributed along 2 different platforms: the iOS mobile phone and a LD screen attached to a PC. The iOS application in both versions are fundamentally similar except for the use cases that were selected for distribution. Figure shows the different use cases in the distributed and non distributed versions.

4.1.2. Use Cases and Functionalities

As shown in figure 4.1, the main use cases for the first non-distributed versions of the prototype are depicted by the inner rectangle. They are described as follows:

- The user should be able to view a list of all recommended videos. The list should be categorized with respect to the video topic or related topics.

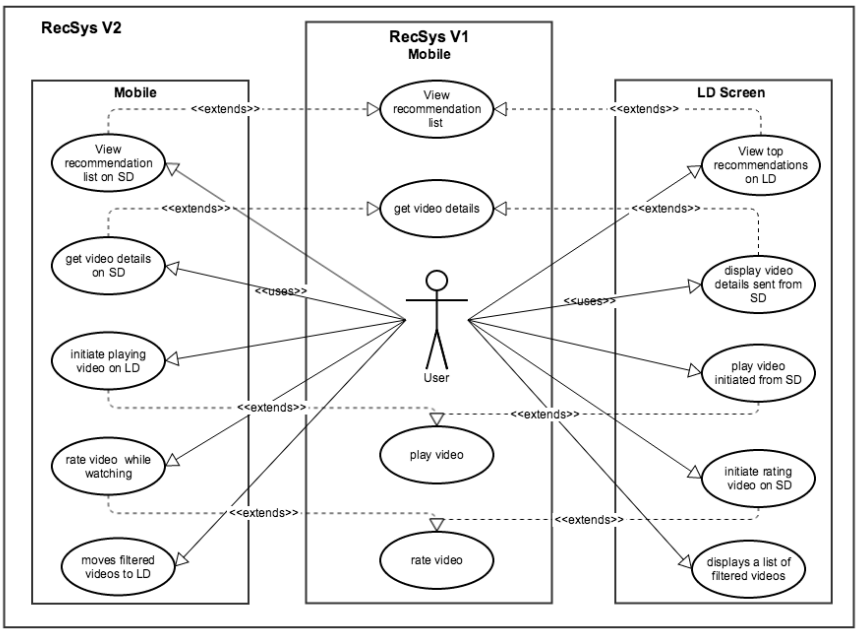


Figure 4.1.: Use Case Diagram of Prototype Versions 1 and 2.

- The user should be able to select a video and get more detailed information on the selected video.
- The user should be able to play the selected video on his/her mobile phone.
- The user should be able to rate the played video after watching.

The use cases of the distributed version of the prototype are given in figure 4.1 by the outer rectangle and grouped by the mobile and the LD components of the systems. The functionalities are thought of as special cases, or extensions, of the first version:

- As opposed to getting a list of recommended videos on the mobile, the user should be able to get both a list recommend videos on the mobile phone (SD) as well as an overview of the top recommended video items on the LD. Both views should be presented in parallel on triggered by the other.

- Getting the video details should be available on the SD. Additionally, the user should be able to transfer the details of the video to be displayed on the LD using a simple gesture.
- The video consumption/playing is distributed between the two displays. The user should be able to select the video on the mobile phone and with a simple gesture, should be able to trigger playing the video on the LD.
- Once the video starts playing on the LD, the user should be prompted with the rating view on the SD. Both functionaries (playing and rating) could be done in parallel.
- The user should also be able to filter his/her choices of videos. This use case is also distributed along both devices. The user should be able to apply a gesture on the selected video on the SD and get this video to be transferred to the LD. By the end of this process, the user should be able to be presented of all filtered videos on the LD.

4.2. Overview System Architecture

The user is presented with a mobile device and an LD screen through which the interaction with the system is possible. Figure 4.2 shows a deployment diagram of the different system components. The system is composed of two separate applications running on two different platforms with a communication layer between them: an iOS application running on an iPhone mobile device, and a python application running on PC connected to an LD screen. A number of artifacts are needed to run both applications, as depicted by figure 4.2: An iPhone mobile device with the .ipa of the mobile app installed on it, and on the LD side, a desktop application, with the Python runtime environment installed, should have the .py UI application installed on it. The

PC is connected through an HDMI cable to an LD screen. Both the mobile device and the PC should be connected to the same network for communication.

This section provides implementation details of each application as well as the layer that channels communication between them.

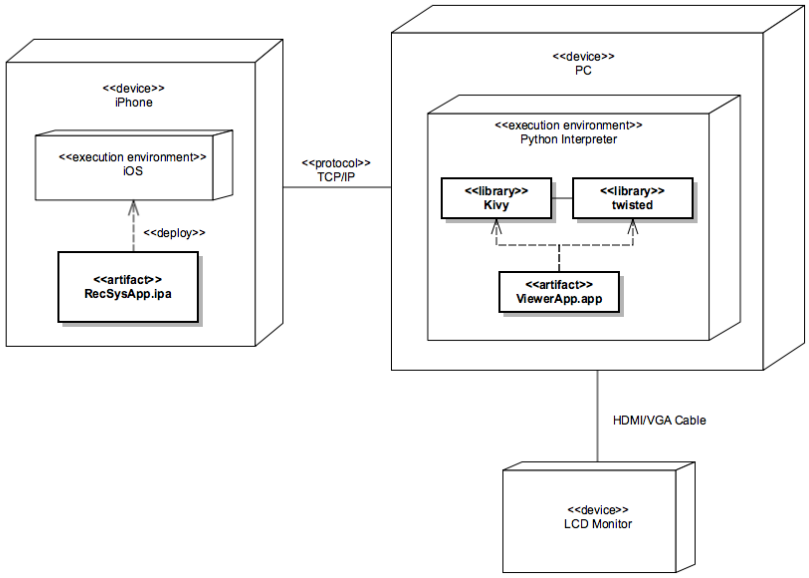


Figure 4.2.: Overview Architecture of The System's Components.

4.2.1. Mobile Application

The first step of implementation, after deciding on the set of functionalities that should be present in each version as described in the previous section, was to start with creating a storyboard for the first non-distributed version of the application. As mentioned, the mobile application is implemented as a native iOS application running on iPhone mobile device. The app was written in Objective C, and uses Apple's iOS SDK 8.4 and basic frameworks for the implementation of most of the functionalities, except for some third party APIs that will be mentioned later. Xcode version 6.4 was used in

implementing and preparing the distribution version of the app.
The second distributed version of the app did not include much variation when it comes to most of the views and navigation available in version one. Therefore, the storyboard was not changed to a great extend in the second version. The main variation in the distributed version was included on the functionalities implementation which will be discussed in later sections.

4.2.2. LD Screen Application

The design of the LD application started by investigating the different technologies available that support the functionalities we needed to implement as a desktop application with a graphical UI. What needed to be supported was means to build a graphical UI that enables user interaction, as well as the capability of playing and controlling video files. Axiomatically, easy means for communicating with the iPhone app that would allow for implementing the UI distribution is a key feature. At first, implementing a Mac OS X desktop application seemed as the best option. However, as we aimed for a more potable option that also support rapid prototype development. Hence, Python based application was rather thought of for being highly portable and light weight for development for this special purpose prototype. Hence, for the implementation of the desktop application, we use the Kivy; an open-source Python based library that was developed with the creation of innovative user interfaces and rapid development as its primary edge. Kivy provides the tools needed to build graphical UI within its basic library, as well as libraries needed for video playback. The graphics engine is built over OpenGL ES 2, using a fast graphics pipeline. It also supports third party libraries that would be easily integrated for network communication with the iOS application as described in the following section. Kivy's portability makes it possible to run the developed application on any platform: Windows, Linux, OS X, iOS Android, which provides a length of variety on the selected platform for deployment.

4.2.3. Communication Layer

The development of applications that use distributed UI components along different platforms is not made possible through special frameworks or tools. Therefore, it is up to every application developer to build a solution that makes the distribution requirements achievable. In our model, we rely on a message passing protocol between the iOS mobile application and the desktop application. This message passing work as commands between the two applications to execute certain tasks (e.g. play video, show details, etc ...). This communication needs to be light-weight and quick to prevent any latency. One of the design goals is to provide seamless distribution of the UI between the two displays, such that the user would be able to carry out the tasks as efficiently as if the task was not distributed in the first place. Hence, the network communication between the iOS application and the Kivy desktop application is chosen to done through socket communication over TCP/IP. The desktop application, besides running its main functionalities, also runs as a custom TCP server that listens to connections on a given TCP socket. The client in this scenario is the iOS application which initiates connection once the app is running on a specific port and IP address. Using TCP sockets for communication makes the implementation of the communication layer independent on a web server, which also means a freedom to choose the implementation language for the server. Moreover, using TCP sockets provides light-weight means for building a lean and efficient protocol for communication through which we could send exact messages that needs to be sent between the applications.

To build the network protocol and the custom server on the desktop side Twisted is used. Twisted is a Python networking framework that makes network communication easier than using Python's basic networking library. Kivy also comes with support for Twisted.

On the iOS side, the CFStream API is used to establish a socket connection and, with the stream object created as a result, send data to and receive data from the custom

python server on the desktop side.

4.3. Implementation of Distributed Scenarios

This section provides details of the implementation of the high-level functionalities of the distributed version of the prototype through the explanation of how the different described scenarios were implemented. It is worth noting that the implementation of such scenarios were taking place iteratively; some the mobile application functionalities were implemented and then modified when new requirements were added as more scenarios were added or changed. Also, the implementation of the server desktop side was done in parallel to the implementation of the iOS side in a continuous integration manner in order to ensure both sides integrate correctly. It is also worth mentioning that the data gathered for this prototype was done using a python script that scrapped TED.com for video cast files' urls and information including detailed description, speakers, dates, images and duration. A total of 73 video casts information was gathered from different topics. Data was not saved in a database, however, was duplicated as XML-based property list files on both the iOS and desktop sides. For the sake of the prototype, fast access of data was needed, and since the data fits more a dictionary representation and has no relational aspect, creating a separate database was thought of as redundant. Also duplication of property list resource on both sides is through to be more efficient than having to access a shared resource despite of the update overhead that would be necessary if any of the copies were to be edited.
Appendix A provide a high-level class diagram for both iOS and desktop applications for details.

4.3.1. Establishing Connection

Initialising TCP sockets communication between the iOS app and desktop app is the first step. Since the desktop app works also as the server, this step has to start by calling Twisted reactor on the server side to start its main loop and wait for connections from the iOS client. The specific IP address and port for communication are hardcoded in the application for simplicity, so that no pairing mechanism would need to be implemented. Once Twisted is started and added successfully to the main loop of the Kivy application, the desktop app enters the Waiting for Connection state. Figure shows the welcome screen that indicates that the server is running and waiting for connection.

On the iOS app side, a Communication Manager singleton class is initialised in the App Delegate once the application finishes launching. In the Communication Manager init method, NSStreams (input and output) are created after a connection with the server is requested at the given IP address and port which are also hardcoded with the same values as the desktop application. Once a connection is made successfully (given that the server is already running), Communication Manager initialises the messaging protocol with the following messages:

- ***open:home*** for opening home view on LD.
- ***play:<videoID>*** for playing a video with id *videoID* on LD.
- ***detail:<videoID>*** for redirecting details of video with id *videoID* to LD.
- ***filter:<videoID>*** for redirecting video with id *videoID* for filtering on LD.
- ***rate:<videoID>*** for rating video with id *videoID* on SD.

As shown, the message payload consists of a command part and a value part separated with a colon. In case video information is needed to be redirected, only the video id is sent and not the actual content of the video. This way, we minimise the overhead of

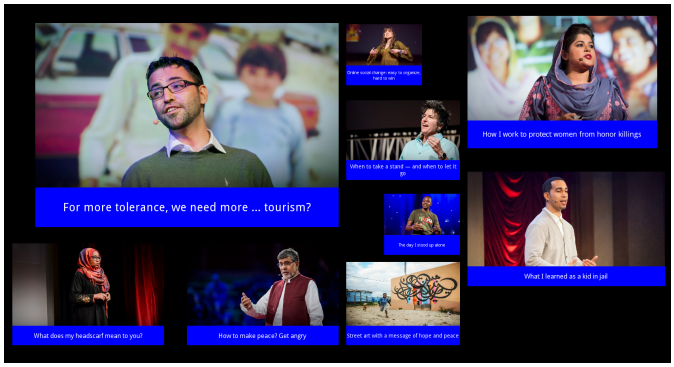
communication by passing light-weight messages that are sufficient for performing the tasks.
At this point, the desktop server enters the Ready State and awaits for receiving messages from iOS client.

4.3.2. Loading Video Data

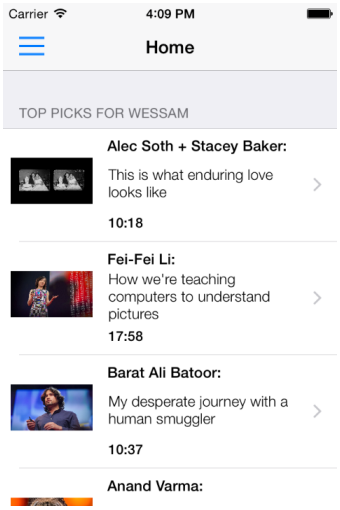
After the connection is established successfully and the protocol is initialised, loading of video data is done on both sides. On the iOS side, a data model class *Video* is created to hold video details. For each video object, the following attributes are loaded: id, title, description, speaker, duration, year, and topic. Data is loaded in memory once from the property list by a Data Manager as Video objects. On the desktop side, the property list is also loaded in a read-only structure.

4.3.3. Presentation of Recommended Videos

The first distributed scenario that user is presented with is the presentation of a Home view which holds the recommended video items. On the iOS side, the videos are presented as a TableView list, which also acts as a master view to a details view which is loaded on clicking one of the items. Fine granularity of detail is presented to the user on the iOS side. The list is divided into sections showing different categorisation of the recommended content titled with explanations as such: *Top Picks for <User>*, *Because You Liked <Category>*, *Top Rated in <Category>*, *You Might Also Like*, etc... This type of explanation is hardcode and not based on actual recommendation logic but was added to mimic real recommendations. Beside the Home view, a side menu enables the user to get a list of all available categories for further details. Selecting one of the categories would also open list view similar to that of Home however specific to that category.
Simultaneously, on loading the home view on iOS, ***open:home*** is sent to the sever to load the home view on the desktop side. On receiving this message, the server loads a



(a) Home view on LD display.



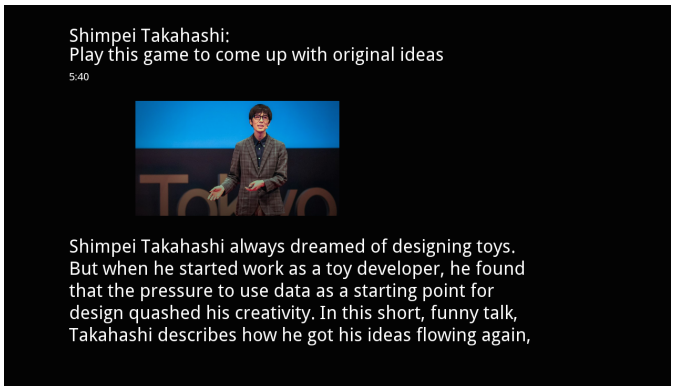
(b) Home view on SD/mobile display.

Figure 4.3.: Presentation of Recommended Videos on SD and LD displays

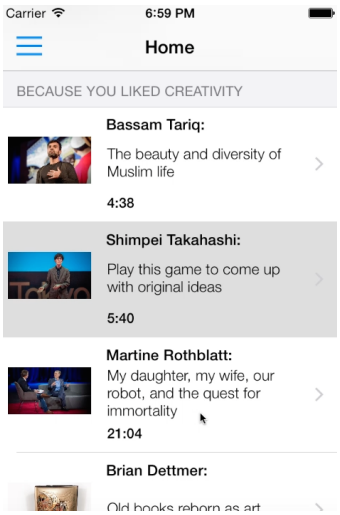
Grid layout with an overview presentation of recommended videos in different sizes which indicate the recommendation score. Figure 4.3 shows the different screens shown on each side.

4.3.4. Presentation of Video Details

Getting details of a video is started on the iOS side by clicking on a given video in the Home list which presents a details view. The details viewew is similar in both the distributed and undistributed versions of the prototype. Moreover, viewing the details of the video on the LD is possible by performing a left swap gesture on the tableview cell of the video item on the iOS side. After performing the gesture, Communication Manager sends ***detail:<videoID>*** with the specific video id. The desktop server receives and parses the sent message. For the command ***detail***, a layout is created, loaded, and filled with the content of the video details whole id was sent as the value part of the



(a) Showing video details on LD



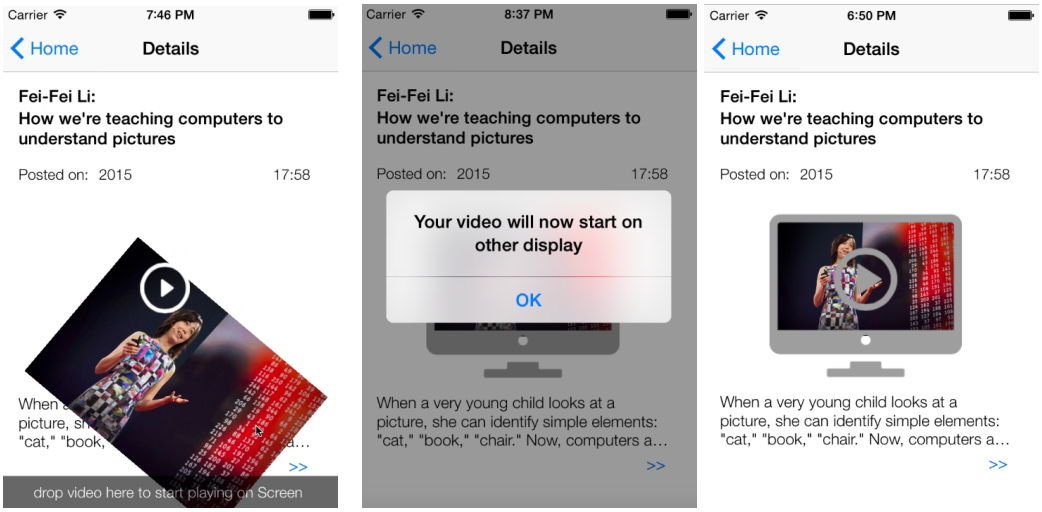
(b) Swipe right gesture on table view cell.

Figure 4.4.: Presentation of Video Details

parsed message. After this operation is complete, the user is able to view details of the videos on both the SD and LD. Figure 4.4 depicts this scenario.

4.3.5. Playing a Video

The distributed scenario of playing a video (figure 4.5, 4.6a) is one that also starts at the iOS side. The user select a video from the list and then is directed to a details page. Inside the details page, besides view the video details, to play a video on the LD, the user simply taps and drags (panning) the image of the video presented in the centre of the screen towards the bottom of the screen. Figure shows a sample of this action. The panning gesture triggers sending a message to the desktop server ***play:<videoID>***, which send the play command and the selected video id to the server. The server receives and parses the message and loads an instance of a video player initialized with the video url. The video player starts streaming the content and displaying it on the



(a) Pan gesture on video image

(b) Notification alert to play transfer video to LD.

(c) Indication that the video has started on LD

Figure 4.5.: Initiating video playing on LD

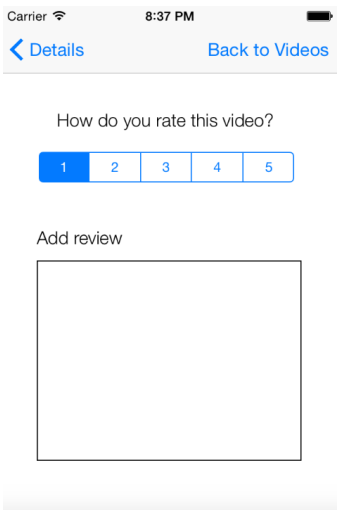
attached LD screen. For playing and controlling the video kivy.uix.videoplayer package is used which is built-in in the Kivy library.

The non-distributed version of the prototype presents the user with a play button on top of the video image in the details page. On clicking the button, the video player is loaded as a modal view on top of the details view, where control and playing of the video is made possible on the mobile device.

4.3.6. Rating a Video

In the distributed version of playing a video, starting the rating scenario is done once the player is started on the LD. The desktop sever sends a message ***rate:<videoID>*** to the iOS app to display the rating view. Hence, the user could perform both tasks (rating and playing) in parallel. Figure 4.6b shows the rating screen on the iOS side. The same screen is shown in the non-distributed version however it is prompted after

(a) Video player started on LD



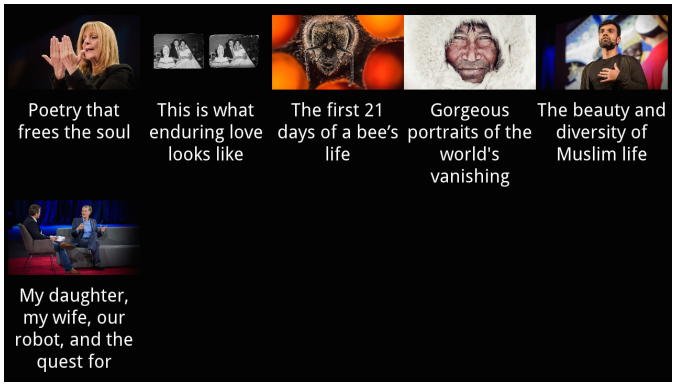
(b) Rating view started on SD on starting video player on LD

Figure 4.6.: Rating recommended videos on SD while playing a video on LD

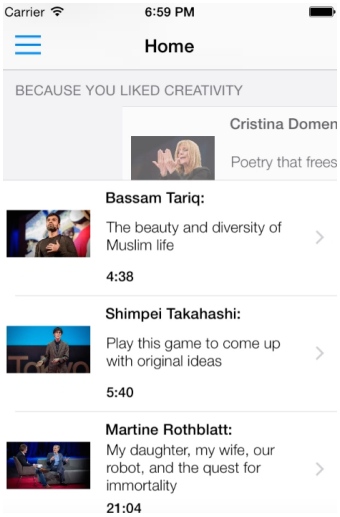
video playing on the SD ends or is stopped by the user.

4.3.7. Filtering Recommendations

Optionally, the user is able to filter his/her choice of videos before deciding on which videos to play. Filtering is done by transferring the selected video from the recommendation list on iOS side to the LD application by performing a right swipe gesture on the video item table view cell. The swipe gesture triggers sending a message ***filter:<videoID>*** from the iOS side to the desktop server side. When the message is received on the server side, the filter page is created as a grid layout and each transferred video is added as a new widget to the layout. Only the video image and title are shown on the LD. The user is also able to display more information for the page by clicking on the video image on the LD side. Figure 4.7 shows the filtering scenario as



(a) Filtering screen on LD



(b) Right swipe gesture on table view cell on SD.

Figure 4.7.: Filtering Recommendations

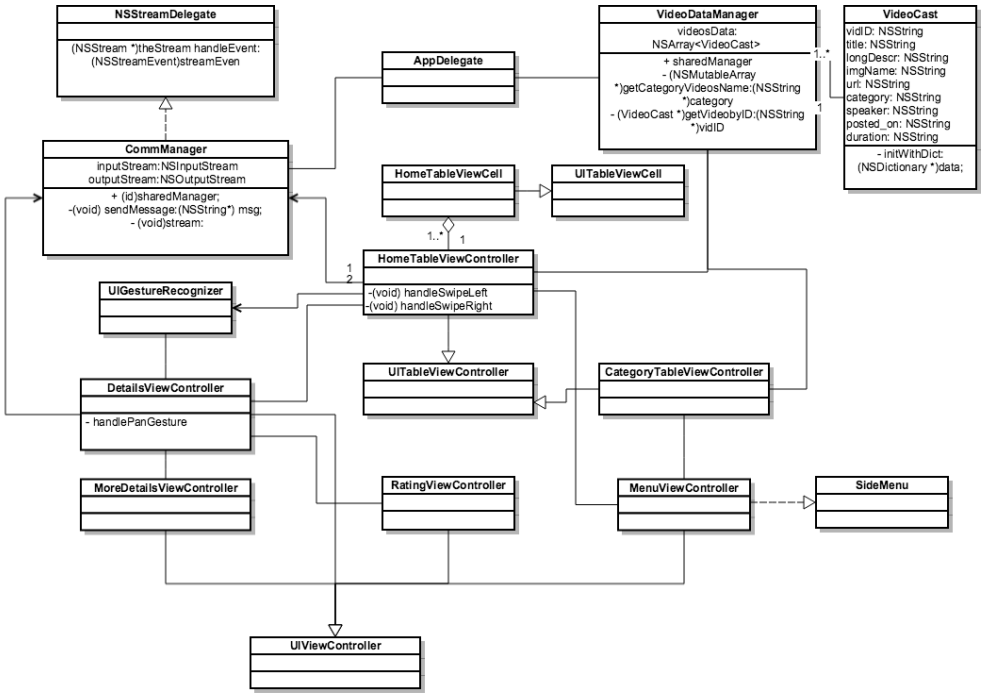
presented on the LD and SD.

5. Evaluation

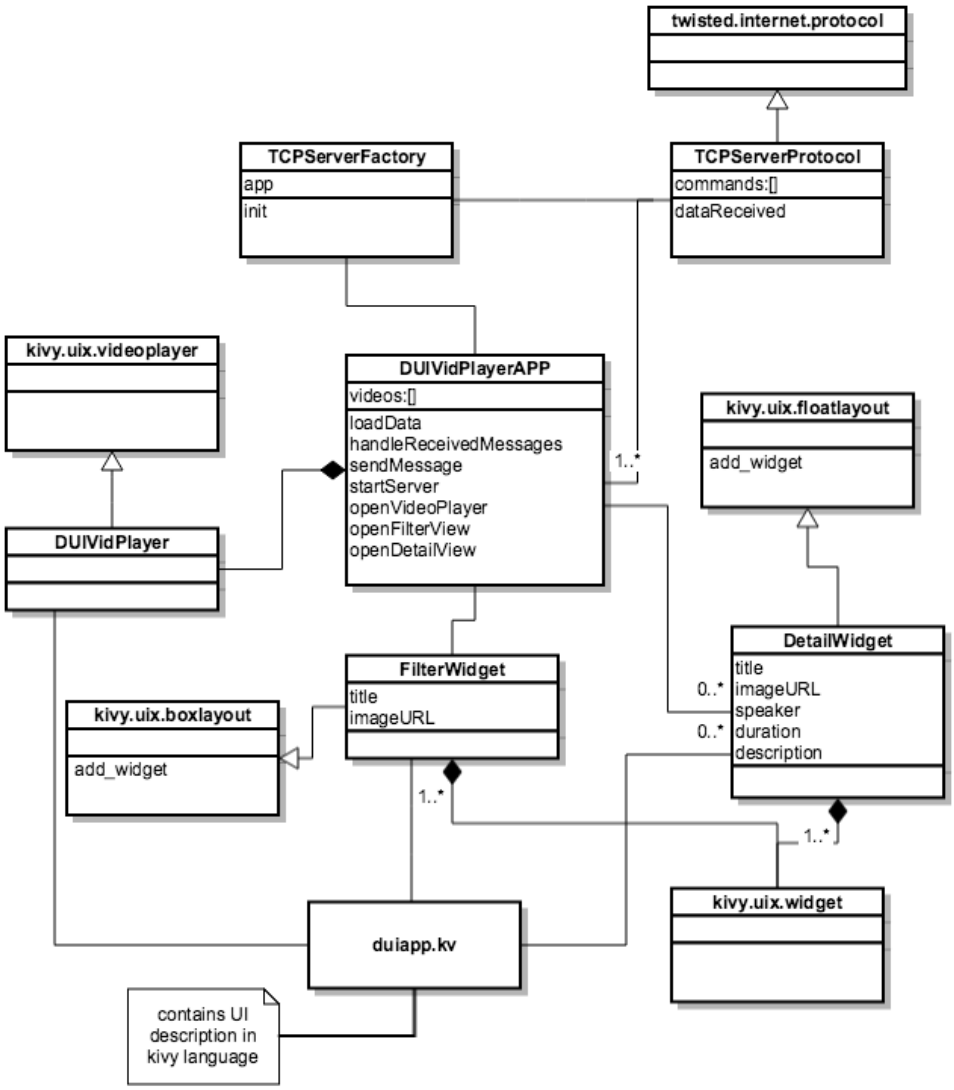
6. Conclusion And Future Work

A. Appendix: Class Diagrams

A.1. Class Diagram of Mobile Application



A.2. Class Diagram of LD Application



List of Figures

List of Tables

Bibliography

[1] Marco Blumendorf, Dirk Roscher, and Sahin Albayrak. Distributed user interfaces for smart environments: Characteristics and challenges. In *Distributed User Interfaces CHI 2011 Workshop, University of Castilla-La Mancha, Spain*, pages 25–28, 2011.

[2] Nicholas Chen, Francois Guimbretiere, and Abigail Sellen. Distributed user interface for a multi-tablet active reading system. *DUI 2011*, page 73, 2011.

[3] Alexandre Demeure, Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, Vincent Ganneau, and Jean Vanderdonckt. The 4c reference model for distributed user interfaces. In *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*, pages 61–69. IEEE, 2008.

[4] Niklas Elmqvist. Distributed user interfaces: State of the art. In *Distributed User Interfaces*, pages 1–12. Springer, 2011.

[5] Anders Fröberg, Henrik Eriksson, and Erik Berglund. A model for dui development. *DUI 2011*, pages 49–52, 2011.

[6] Hans-Christian Jetter, Michael Zöllner, Jens Gerken, and Harald Reiterer. Design and implementation of post-wimp distributed user interfaces with zoil. *International Journal of Human-Computer Interaction*, 28(11):737–747, 2012.

[7] Nima Kaviani, Matthias Finke, Rodger Lea, and Sidney Fels. Dual displays:

towards an interaction model and associated design guidelines. *DUI 2011*, page 69, 2011.

[8] JJ Lòpez-Espin, JA Gallud, E Lazcorreta, A Peñalver, and F Botella. A formal view of distributed user interfaces. In *Distributed User Interfaces CHI 2011 Workshop, University of Castilla-La Mancha, Spain*, pages 97–100, 2011.

[9] Marco Manca and Fabio Paternò. Distributing user interfaces with maria. *DUI 2011*, pages 93–96, 2011.

[10] J Melchior, D Grolaux, J Vanderdonckt, and P Van Roy. A toolkit for peer-to-peer distributed user interfaces: Concepts. *Implementation, and Applications*, 69:15–17.

[11] Jérémie Melchior. Distributed user interfaces in space and time. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 311–314. ACM, 2011.

[12] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. Distribution primitives for distributed user interfaces. In *Distributed User Interfaces*, pages 23–31. Springer, 2011.

[13] Thomas Seifried, Hans-Christian Jetter, Michael Haller, and Harald Reiterer. *Lessons learned from the design and implementation of distributed post-WIMP user interfaces*. Springer, 2011.

[14] Jean Vanderdonckt et al. Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments. *Proc. of XI Interacción*, 20, 2010.