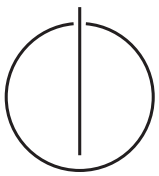
FAKULTÄT FÜR INFORMATIK

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

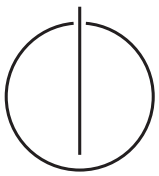
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

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

1.2. Distributed User Interfaces

1.3. Case Study

1.4. Structure of The Work

2. Literature Review

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, following 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. 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.

3.1.2. 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.

3.1.3. 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.

3.1.4. Task: What to Distribute?

One prespective 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, and recommended content rating.

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.

3.1.5. Distributed UI Scenarios

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. 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 application 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.

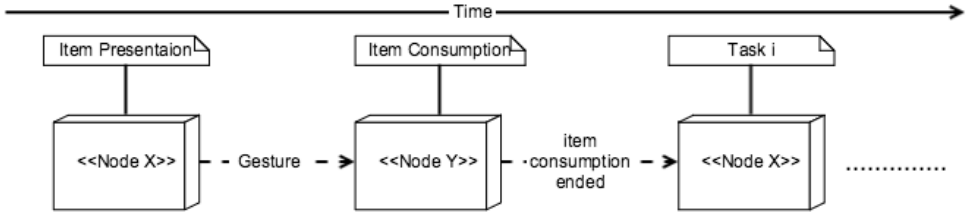


Figure 3.1.: Migrating Item Consumption from Presentation Node to Consumption Node.

Performing Parallel Activities

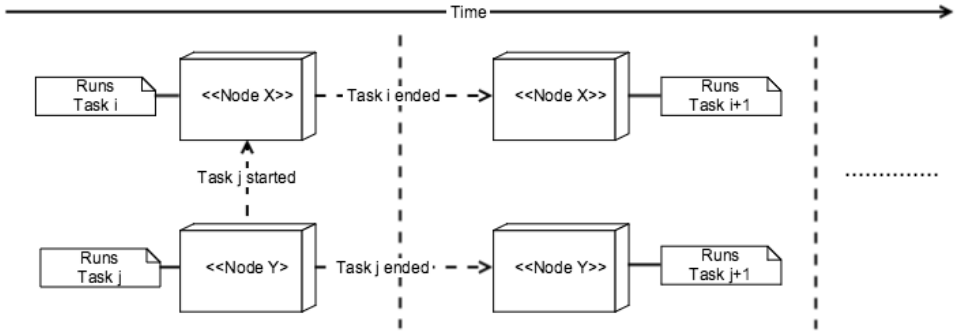


Figure 3.2.: Caption

In a distributed environment, presenting the user with different platforms through which the user could perform tasks simultaneously could be leverage to perform

parallel activities. In this case the distribution is achieved through distributing different interface components to be executed asynchronously (distribution in time). 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. This is a case of distribution in time. After the tasks end on their given node, execution of proceeding tasks could be carried out also in a distributed manner.

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 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. 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.

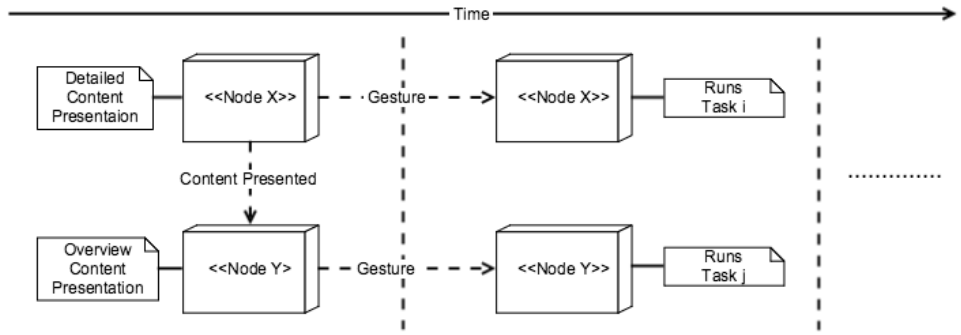


Figure 3.3.: Overview-Detail Presenations

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.

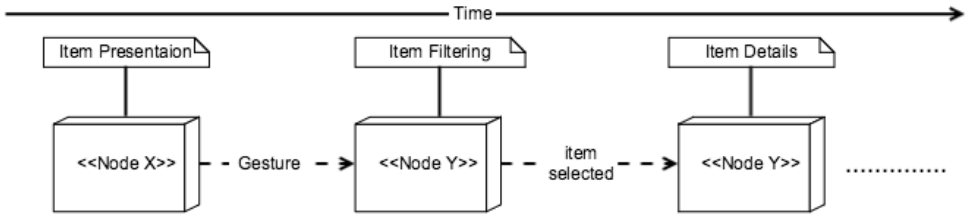


Figure 3.4.: Recommended Content Filtering

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.

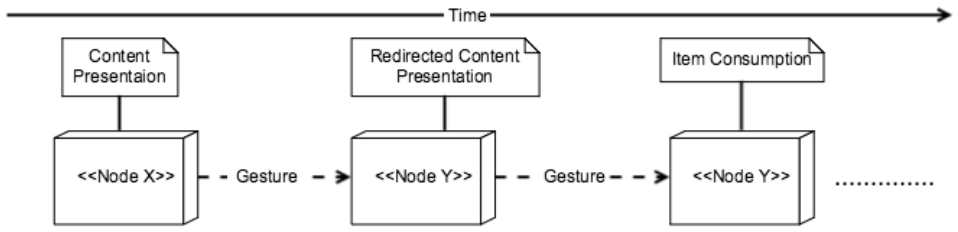


Figure 3.5.: Content Redirection

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. Distribute for Whom?: 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.

3.2.2. Platforms and Environment

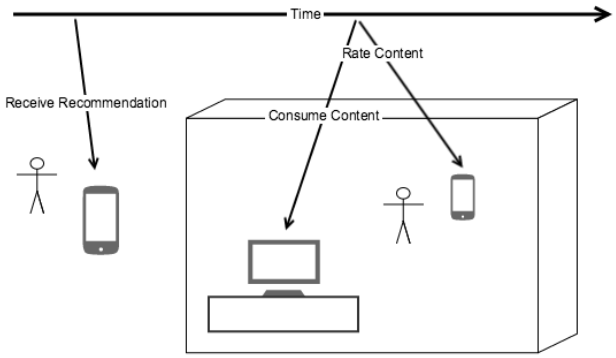


Figure 3.6.: Overall view of the system's environment

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 Tasks in a Distributed UI Environment

In this section, the concept behind the distribution of pre and post recommendation tasks is provided. The reason for which tasks or UI components are selected to be displayed on which display, as well as the description of other distribution dimensions, are given.

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. As shown in figure, after the user finishes the profile creation step on the SD, the recommended content is then displayed on both SD and LD displays.

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

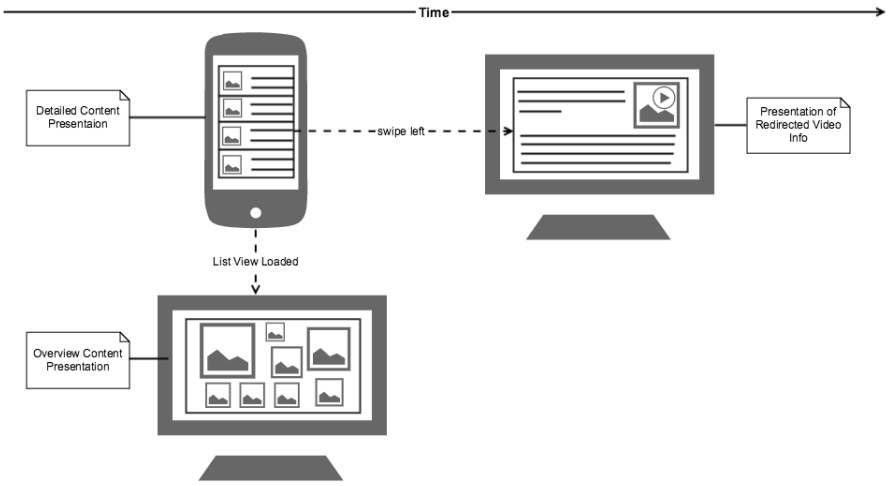


Figure 3.7.: Presentation of Recommendation Results in LD-SD Modes.

items that scored the highest for the user without details, however shown in different sizes to indicate the recommendation score. 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. Then, if more details is needed, the mobile device would provide more a detailed information about the video content. This concept UI distribution is what comes to be known as overview-detail coupling [cite].

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. Alternatively, as shown in figure 3.7 content redirection is also possible for viewing such information on the LD. Also by applying a simple swipe-right gesture on any of the video items in on the SD list, the video details content will be redirected to be displayed in parallel on the LD.

Recommended Item Consumption

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.

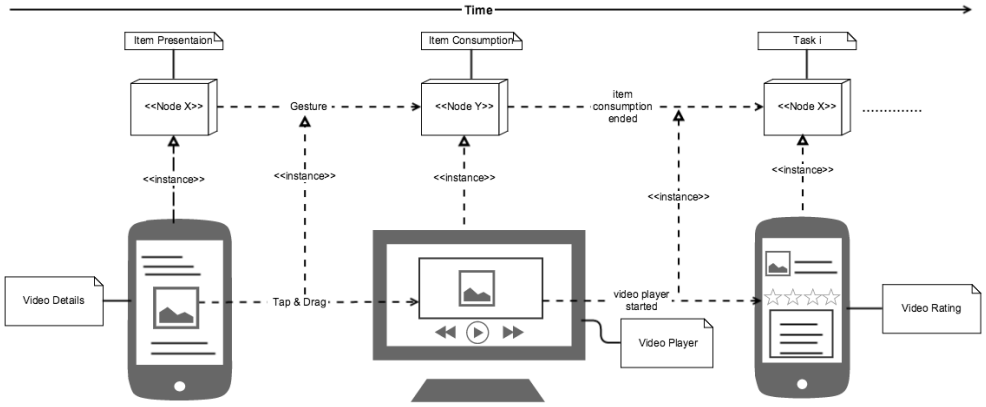


Figure 3.8.: Recommended Items Consumption.

Rating of Recommended Items

There are two different proposed options for the rating task. The first option is distributed in time; i.e. 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.
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

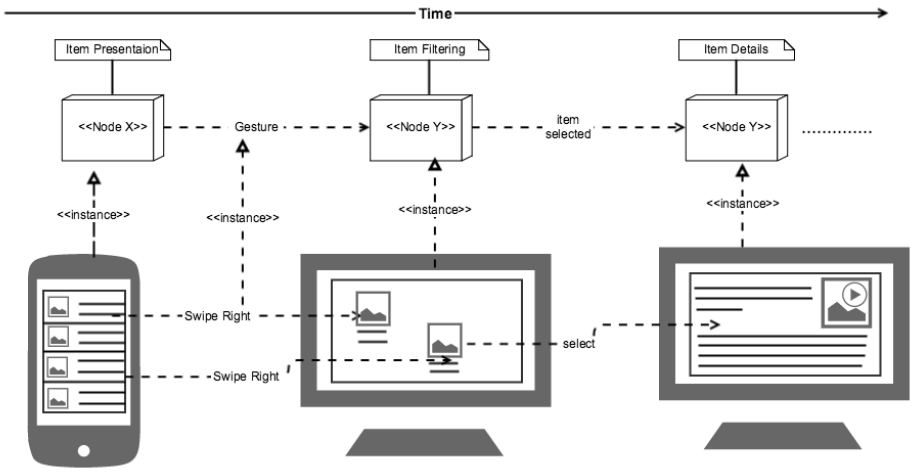


Figure 3.9.: Filtering of Recommended Items.

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.

3.2.4. Transferring Recommendation Lists

-Example of distribution initiated by user... (basically any type of content redirection)
-Select which components within a view (details view) are to be distributed.
-Readjust to old view after that -Save as a configuration -Rating: in parallel or after player?

3.2.5. Pre-Configuring the System

Meta UI. Time distribution. Enables system to load components differently at runtime. Select a configuration. Selecting which views to show on which platform.

4. Implementation

In the previous chapter, 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.
- The user should be able to select a video and get more detailed information on the selected video.

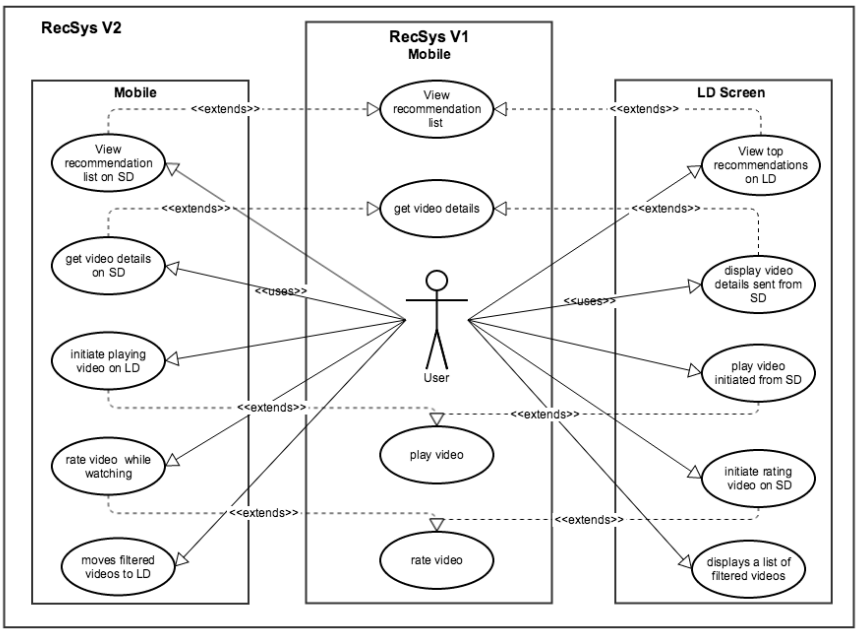


Figure 4.1.: Use Case Diagram of Prototype Versions 1 and 2.

- 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.

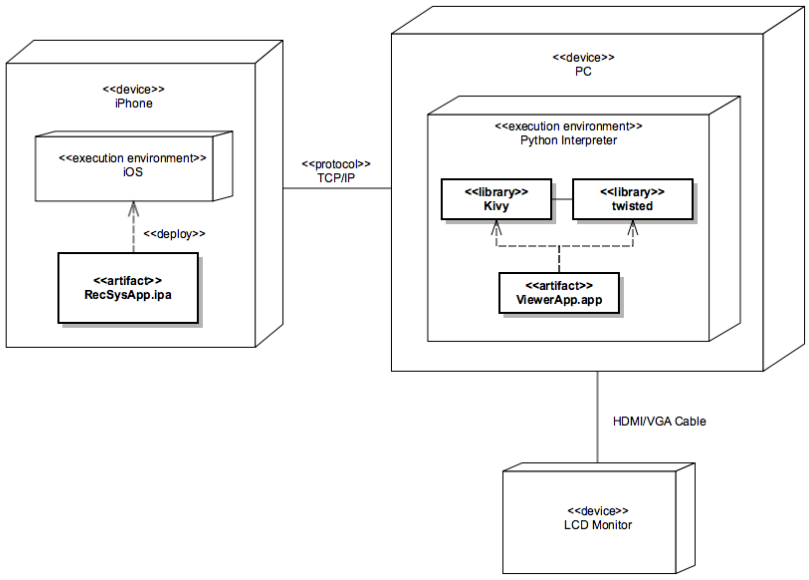


Figure 4.2.: Overview Architecture of The System's Components.

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

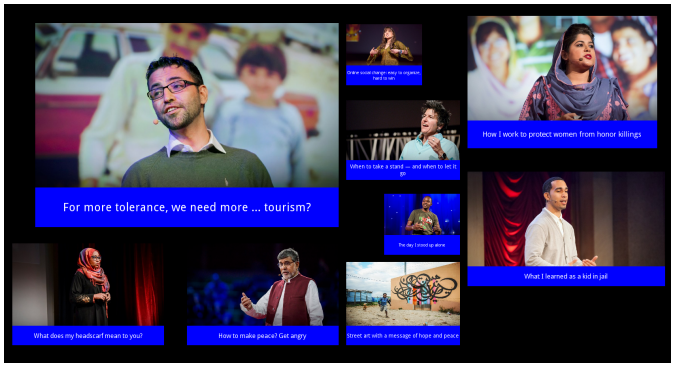
communication by passing light-weight messages that are sufficient for performing the tasks.
At this point, the desktop server enters the Ready State and awaits for receiving messages from iOS client.

4.3.2. Loading Video Data

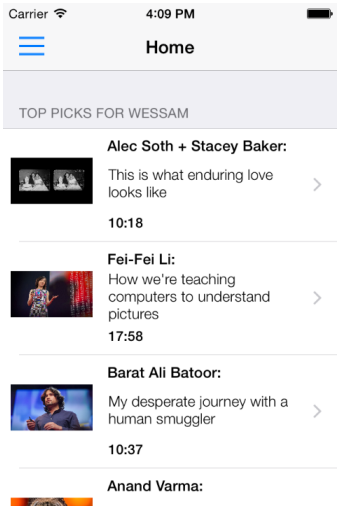
After the connection is established successfully and the protocol is initialised, loading of video data is done on both sides. On the iOS side, a data model class *Video* is created to hold video details. For each video object, the following attributes are loaded: id, title, description, speaker, duration, 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.



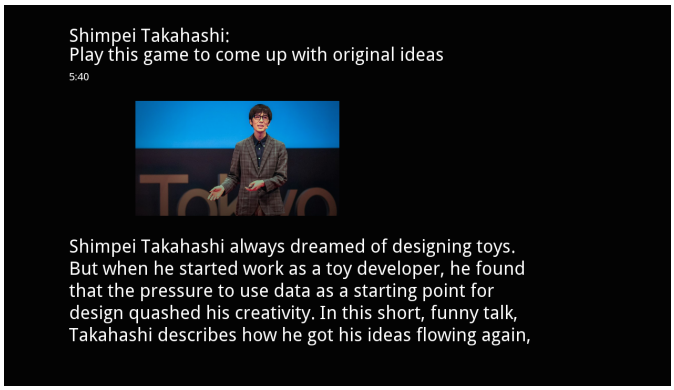
(b) Home view on SD/mobile display.

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

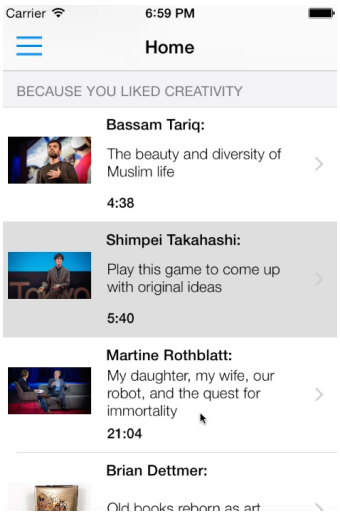
Grid layout with an overview presentation of recommended videos in different sizes which indicate the recommendation score. Figure 4.3 shows the different screens shown on each side.

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



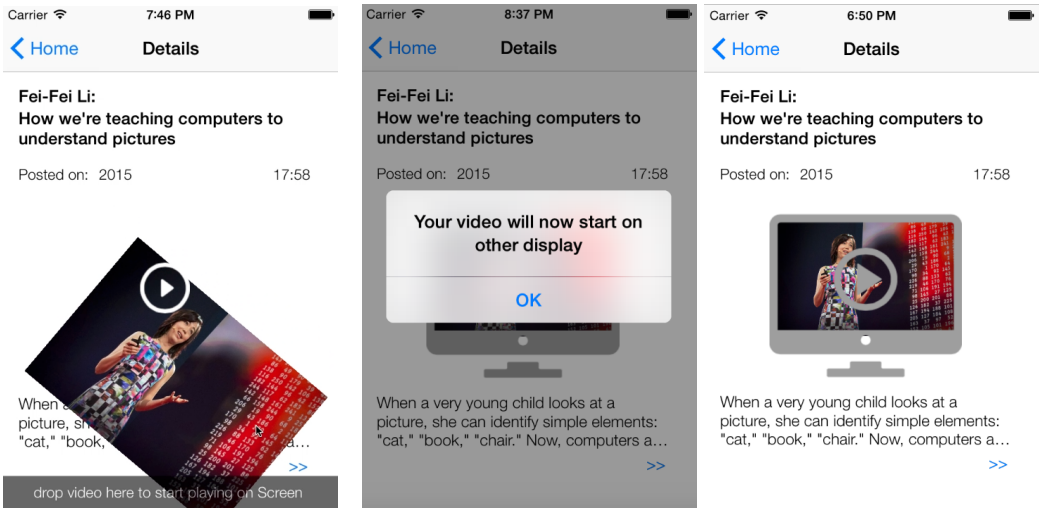
(b) Swipe right gesture on table view cell.

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

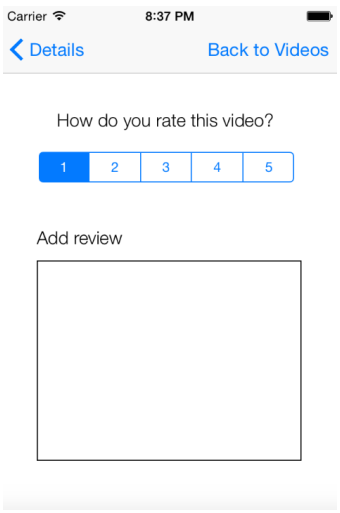
attached LD screen. For playing and controlling the video kivy.uix.videoplayer package is used which is built-in in the Kivy library.

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



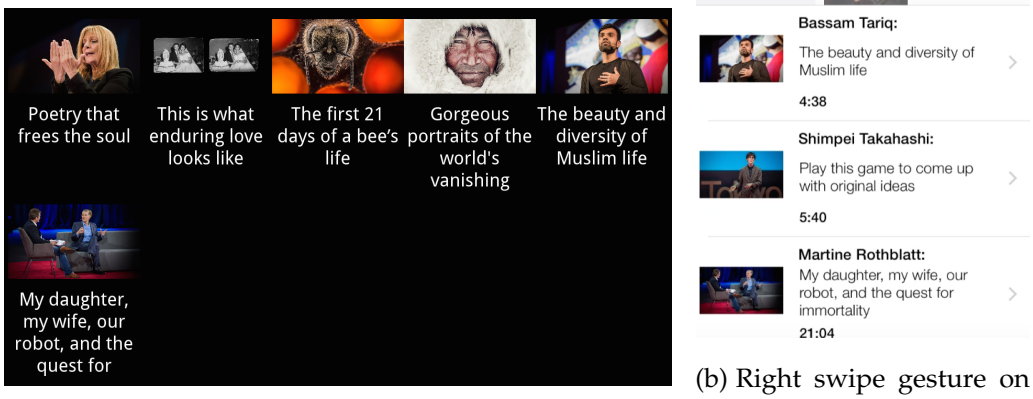
(b) Rating view started on SD on starting video player on LD

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

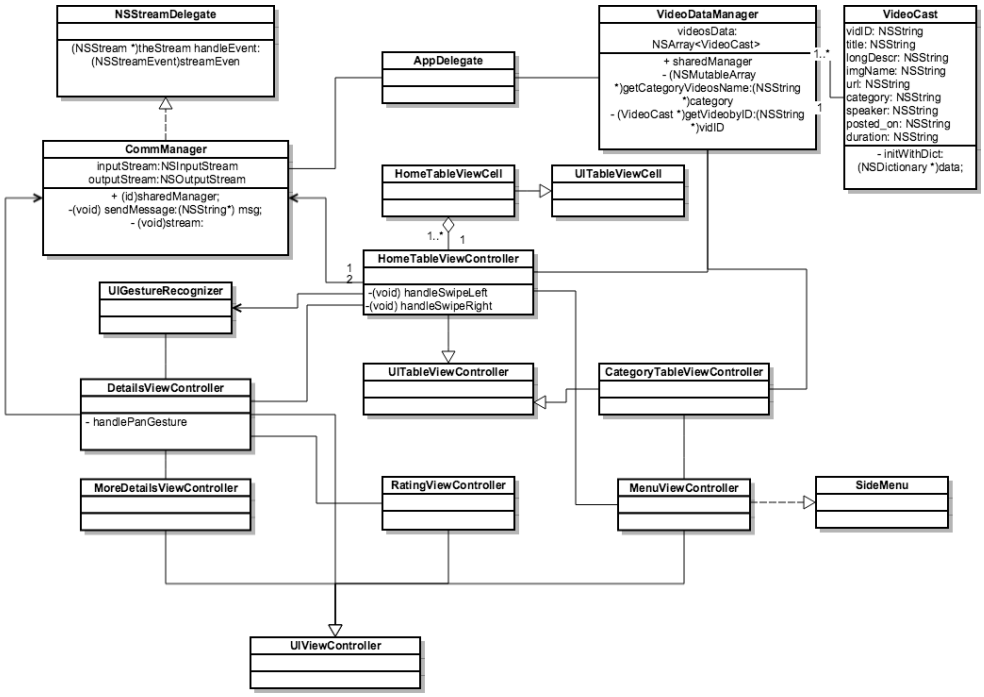
presented on the LD and SD.

5. Evaluation

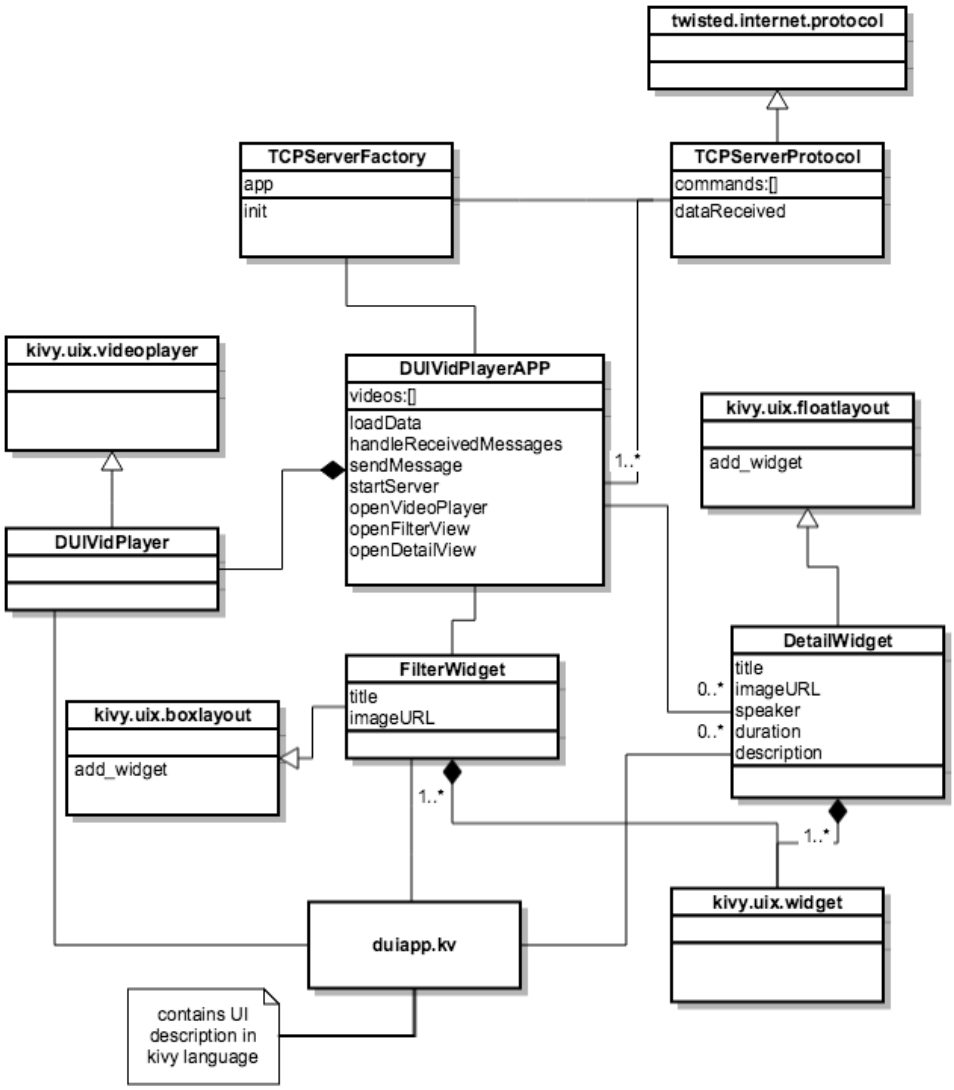
6. Conclusion And Future Work

A. Appendix: Class Diagrams

A.1. Class Diagram of Mobile Application



A.2. Class Diagram of LD Application



List of Figures

List of Tables