Distributable Interface Design for Web Applications

Gianni Fenu(✉) and Lucio Davide Spano

Dipartimento di Matematica e Informatica,
University of Cagliari, Via Ospedale 72, 09124 Cagliari, Italy
{fenu,davide.spano}@unica.it

Abstract. The increasing number of devices available for each person allows to create unconventional interfaces that coordinate more than one device for supporting the interaction. In this paper, we introduce a framework for designing distributable web applications, which supports moving and sharing the different parts of a user interface across different devices. We depict the architectural solution and we introduce a set of distribution patterns. In addition, we describe a concrete application of the framework for a distributable video player application.

Keywords: Distributed interfaces · Web applications · User Interface Engineering · Development tools

1 Introduction

The wide availability of different types of devices, both stationary and mobile, is opening the opportunity for creating applications that go beyond a single device. And this is not only limited to providing the same application in different versions (e.g. one for desktop environments and one for mobiles), but also to unconventional interfaces that coordinate more than one device for the same user interface (UI).

As highlighted in [1], people is more and more developing a multi-screen behaviour, which results in both the sequential and simultaneous usage of different screens at the same time. Therefore, creating applications that exploit such new interaction habits effectively is an opportunity and a challenge for the HCI community. On the one hand, applications able to exploit different devices at once may be able to create experiences that go beyond the simple sum between the capabilities of each considered device. On the other hand, the device coordination creates different technical challenges, and poor solutions may affect the overall usability of the interface.

In this paper, we propose a framework for developing web applications allowing the interface to be distributed across the device we use everyday, for creating a personal interactive space going beyond the single device. We first introduce a distribution example scenario, identifying the users' needs, then we describe the architecture of the proposed solution, and finally we discuss the implementation of a first prototype supporting the proposed scenario.

2 Related Work

Starting from the first applications of the ubiquitous computing concept, different techniques related to distributing parts of the user interface (UI) in different devices started to take place in research work: the user does not own a single device, but she is empowered with different computing platforms that are pervasive in the environment. The different efforts aimed for instance to the migration of the entire UI state from a device to another [2], or for the adaptation and configuration of a UI according to the actual device that renders it [3].

Demeure et al. [4] created a reference model for examining a distributed interface according to 4 dimensions: the computation (which part is distributed), communication (when the UI is distributed), coordination (who is distributed) and configuration (from which device to which device the distribution is operated). Such work opened the space for creating different engineering solutions and models for supporting the distribution.

In particular, the model-based approaches for user interfaces community produced different models supporting the distribution of user interfaces. In [5], the authors exploit a XML format for defining how an interactive application can be distributed across different dimensions: end user, display device, computing platform, and physical environment. Frosini and Paternò [6] introduce a framework and the associated runtime support for supporting dynamically the distribution across different devices, with a peer-to-peer architecture.

Another field where the distribution of user interface gained the attention of the research community is the creation of shared spaces. A first example is the collaboration in museum environments [7], where distributed interfaces were implemented for supporting the collaboration of museum visitors for solving didactic games through mobile devices.

The second example is related to the information visualization. VIGO [8] supports the distributed interaction in a multi-surface environment through four components: Views, Instruments, Governors and Objects. Hugin [9] ia a graphical framework for mixed-presence collaboration settings. In such environment, the information visualization application is shared between different tabletops, which should be coordinated over the network for both controlling the data and making the users aware of each other.

In this paper, we shift the emphasis from a self-contained model or architecture to a lightweight framework that exploits the usual structure of a web application for building the support for distributable interfaces. In this way, it would be possible to adapt existing applications for a distributed setting.

3 Example Scenario

In order to explain the framework concepts with a concrete example, we consider a simple on-demand video streaming application. We detail the envisioned interaction through a small scenario.

Robert is just back home from work and he decides to watch the last episode of his favourite TV series, Game of Phones, on DistrFlix. He just started watching *the third episode on his laptop, when his wife Sarah interrupts him. She is a fan of Game of Phones too, and she would like to see the episode together with Robert. Therefore, Robert moves the video from the laptop to his Smart TV, while the information on the episode and the playback buttons are transferred to his smartphone, in order to be easily controlled from the sofa. While they are watching the episode, Robert receives different phone calls: one from his mother, one from his boss and one from the call center of his previous phone company, advertising discounts if he accepts to be their customer again. Sarah is annoyed since Robert never pauses the episode before answering the call, so she ask him to share the video control buttons on her smartwatch.*

According to the scenario description, we have four involved devices: a laptop, a TV, a smartphone and a smartwatch. All these devices allow to control the access to the same application during the same session. During the interaction, the interface assumes three configurations across the four devices, which we summarize in Fig. 1. In the first one, the user accesses all the application functionalities on the laptop. In the second configuration, the UI is splitted between the TV (video) and the smartphone (playback controls and additional information). In the last one, the smartwatch and the smartphone provide a redundant control on the video playback.

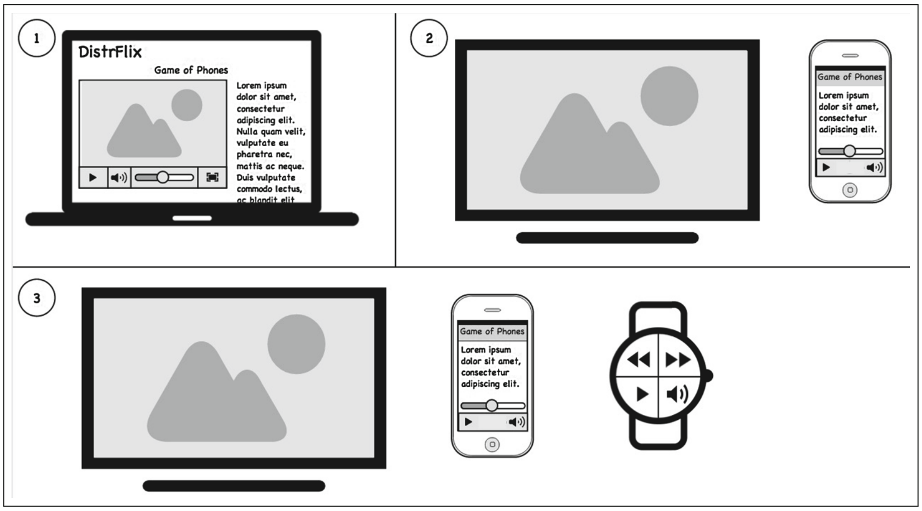

1
DistrFlix
Game of Phones
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit
2
Game of Phones
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
3
Game of Phones
Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Fig. 1. Scenario interface mockup.

In Sects. 2, 5 and 6 we describe a framework for supporting the distribution of the UI components on the different devices and their state management. In Sect. 7 we show a concrete application of the framework for solving the situation described in the scenario.

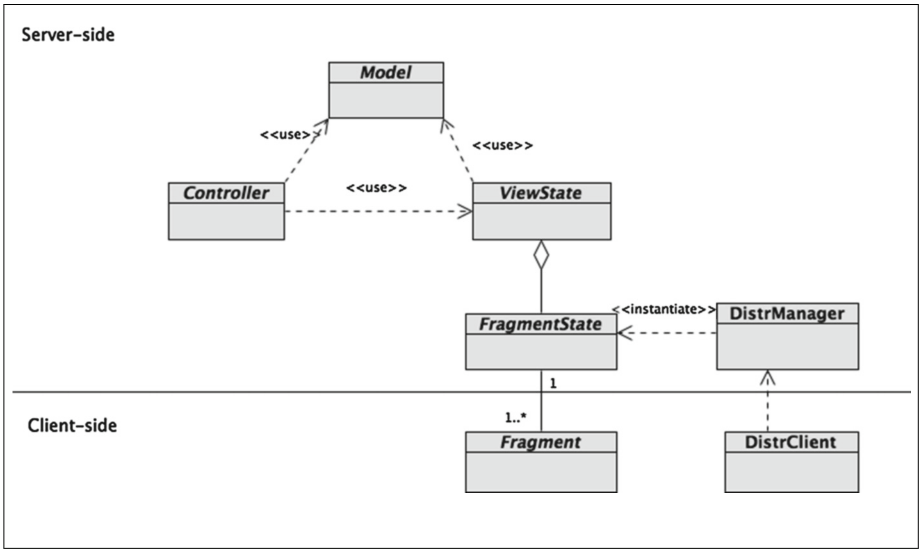


Fig. 2. Distribution framework reference architecture

4 Overall Architecture

The framework organisation is based on two main concepts. The first one is the definition of the UI parts that can be distributed across different devices. We started from the techniques for creating responsive layouts: the content of a single page is organised in a set of macro groups (that we call UI *Fragments*), which are positioned inside a grid. How such grid is displayed on a particular device screen (e.g. its rows and columns) depends on its width. In addition, each fragment has different associated styles according to viewport size. We push further this idea in order to support the distribution of web interfaces: the developer defines rules not only for positioning the fragments, but also for distributing them among different devices.

The second concept that grounds the distribution framework is the definition and the synchronization of the UI state, which must be shared among all the devices involved in the distribution process. We propose an abstraction mechanism for separating the UI element distribution aspect from the UI state management. Such mechanism is depicted in Fig. 2, which applies the separation concept inside the Model View Controller (MVC) pattern [10], since it is one of the most applied for organizing web applications. However, the separation concept we present in this paper is independent from the MVC, and it may be applied also to other variants (e.g. Presentation Model [11], Model View Presenter [12] or Model-View View Model [13]).

The component we modified for supporting the UI distribution is the *View*. The framework provides a view state abstraction (the *ViewState* class) for coordinating the dynamic changes to the state of a fragment. More precisely, the framework considers the rendering of a fragment on a particular device as a view on the UI state, which is instead the considered as the model of the fragment. The changes made by the user through a particular view modifies the values saved on the model (the UI state). They should be in turn reflected on the other fragments, which are rendered on different devices.

We detail both aspects in the following sections.

5 Fragment Management

A *Fragment* is a UI part that groups together interface elements that are logically related to each other. In our scenario, for instance, we have three different fragments: one related to the video, one related to the playback controls and one related to the TV show information. In order to establish how a UI can be distributed across two or more devices, the developer assigns a set of fragments to each of them. A fragment may be rendered in different ways according to the device type. However, it keeps the semantics of the UI controls: for instance a drop down list for a desktop interface may be rendered as a list view on mobile device, but the semantics of the action it supports (and therefore the reaction of the application back-end) is the selection of a value. The different rendering is supported through both CSS rules and/or different HTML code generation at server side.

The same fragment can be assigned to more than one client device. Therefore, its internal state must be synchronized across all the devices. Such synchronization is required at two levels: the UI and the application state. At the application level, the framework relies on the model component of the MVC pattern, which is shared by different views. In our case, such views are spread among the different devices. At the UI element level, the framework splits the state management in two parts. Such concept is similar to the distinction of abstract and concrete interactors in model-based approaches for user interfaces (i.e. as reported in [14]).

- The UI element *semantics*, which is independent from the actual widget exploited for supporting the user. A widget can support the *selection* or the *editing* of value (text, number, date etc.), it can *trigger* a specific command, or it can *display* information. This part of the state is maintained into the *ViewState* (see Fig. 1). The controller component of the MVC pattern accesses the *ViewState* for reading or updating the UI values, without relying on any information about their actual rendering.
- The UI element *implementation*, which depends on both the device type (e.g. laptop or smartphone) and the current state of the UI distribution among the devices. For instance, the control over the video stream position in our example can be implemented through a slider on the laptop and the smartphone, while on the smartwatch we decided to support it through a set of buttons. The widgets have semantically the same purpose (controlling the stream position value), but they are implemented differently. Therefore, there is a part of the state that depends on the actual implementation, which is maintained into the *FragmentState*.

At the code level, the framework provides the support for such coordination through a specific javascript object, which has a bidirectional binding with the UI elements and the server side part. This means that the fragment behaviour specifies (1) which fragment state variables change when the user interacts with a UI element, and (2) how the UI elements should be modified for reacting to a change on the fragment state object.

A fragment state object can be configured for containing different sets of variables according to the specific fragment. In addition, it includes different functions for getting and setting the variable values. In these functions, we encapsulated the synchronization protocol: the object manages a web socket communication with the application server for receiving and sending updates from and to other devices. The developer is not in charge of any synchronization, and he can specify the behaviour code as if the UI state was handled locally. The only requirement is managing it through a fragment state object extension.

The server side code reacts to the changes of all fragment states. The framework provides facilities for managing application and device level sessions. The application sessions are related to a single user that access an application through different devices. Therefore, an application session contains many devices sessions.

6 Distribution Management

Besides the classes for managing the view component state in the MVC pattern displayed in Fig. 1, the framework contains also the classes for managing the association between the fragments and the devices.

The first class, the *DistrClient* provides the user support for dynamically adding or removing a device from the distribution set. The user has a device management page where she can add all her personal devices. After this operation, the user can request to add one of her personal devices to the distribution set for each web application supporting the distribution. Optionally, the application developers may also request to specify which fragments they want to include on the different devices, providing a user-friendly name and description for them (e.g. playback controls).

Once the user requests to activate (or deactivate) the distribution towards a particular device, the *DistrManager* evaluates distribution strategy (in form of a set of rules coded by developers) and it creates (or destroys) the fragments related to the selected device, which in turn will receive an update message for displaying them.

In a given device, a fragment can be rendered in different modes: supporting the input, the output or both of them. In the input mode, the fragment contains only the interface elements that are devoted to collecting values from the user and those that depend on them (e.g. field labels etc.). In the output mode, the fragment renders only elements that have no interaction capabilities (texts, images, videos etc.). The mixed mode shows both types of interface elements, as usually happens in web interfaces. Such configuration allows developers to dynamically distribute fragments specifying an association between fragments and devices.

The mode is considered an attribute of such association by the fragment and its rendering is controlled accordingly.

Different patterns exist for defining such association. In our framework we considered the well-known CARE properties for the defining multimodal user interfaces [15]:

– *Complementarity:* two fragments are complementary if they are assigned to two different devices, but none of them is able to complete their corresponding task without the other. For instance, the video and the playback controls when split between the TV and the smartphone are complementary: the video is useless without the controls and the vice-versa.
– *Assignment:* a given fragment is assigned to only one device. In our scenario, the video fragment is assigned to the TV after the first device change.
– *Redundancy:* the same fragment is displayed in two (or more) devices and its functionalities should be confirmed in all of them for being executed.
– *Equivalence:* a fragment is displayed on more than one device. The user can activate its functionalities from anyone of them. In our scenario, the playback controls may be activated from both the smartphone and the smartwatch.

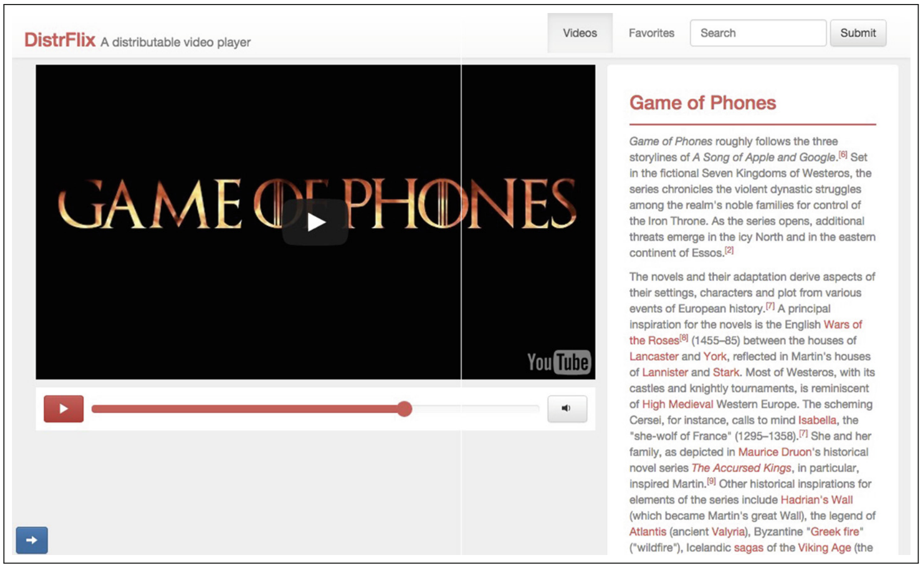
Developers may both select a single specific pattern for distributing the UI across the devices or require the users to select one of the patterns for a subset of fragments. In this case, the *DistrClient* allows two select among two or more distribution patterns, providing a user-friendly description for each choice.

7 Scenario Support

In this section we provide some technical details on the implementation of a Java prototype supporting the scenario described in Sect. 3 through the proposed framework. The starting point is the usual desktop interface for a video player, as shown in Fig. 3, which consists of three fragments: the larger part of the screen is dedicated to the video, with the controls displayed below it. The right bar contains the plot description.

The current state of this simple UI contains: the current position in the video stream, the player state (playing/paused), the volume level and the reading position in the description sidebar. All these variables are maintained in the *ViewState*, and they are valid independently from the UI elements we selected for interacting with the user. The *FragmentState* performs the mapping between such values and the UI elements in both directions. For instance, if the user changes the position of the slider knob for rewinding the video, the fragment state receives the new position of the knob and maps it to the correct time, notifying the *ViewState* object.

In our scenario, at a certain point Robert decides to continue watching the video on the TV, controlling its state from his smartphone. In order to do this, Robert needs a way for requesting the distribution, accessing the functionalities of the distribution client (*DistrClient*). The framework provides a reusable JSP which can be simply included into all the web application that require the support for the distribution.



Game of Phones

Game of Phones roughly follows the three storylines of *A Song of Apple and Google*.[6] Set in the fictional Seven Kingdoms of Westeros, the series chronicles the violent dynastic struggles among the realm's noble families for control of the Iron Throne. As the series opens, additional threats emerge in the icy North and in the eastern continent of Essos.[2]

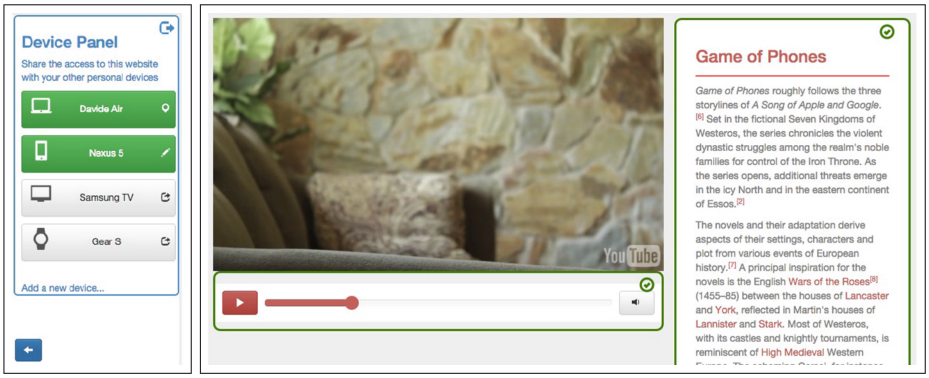
The novels and their adaptation derive aspects of their settings, characters and plot from various events of European history.[7] A principal inspiration for the novels is the English Wars of the Roses[8] (1455–85) between the houses of Lancaster and York, reflected in Martin's houses of Lannister and Stark. Most of Westeros, with its castles and knightly tournaments, is reminiscent of High Medieval Western Europe. The scheming Cersei, for instance, calls to mind Isabella, the "she-wolf of France" (1295–1358).[7] She and her family, as depicted in Maurice Druon's historical novel series *The Accursed Kings*, in particular, inspired Martin.[9] Other historical inspirations for elements of the series include Hadrian's Wall (which became Martin's great Wall), the legend of Atlantis (ancient Valyria), Byzantine "Greek fire" ("wildfire"), Icelandic sagas of the Viking Age (the

Fig. 3. Laptop interface for the distributable video player.

In our example, we chose to insert a small arrow button at the bottom of the page (with a fixed positioning) that allows to access the distribution client UI. The button is visible in Fig. 3 in the bottom-left corner. Once pressed, the application shows the sidebar in Fig. 4 (left part), which displays the list of user's personal devices (after an authentication step). The panel contains a button for each one of them. In the example, we have four devices: the laptop (Davide Air), the smartphone (Nexus 5), the TV (Samsung TV) and the smartwatch (Gear S). Through these buttons, the user can select or deselect each device for the distribution: the green devices are currently active (they contain at least a fragment), while the other ones are not in use with the current web application.

In order to distribute the interface, the user presses the button of an inactive device. If the application has a configurable distribution policy for that kind of device (as in our example for smartphones), the user is requested to select the fragments to be included in the new device through the interface shown in Fig. 4 (right part): he simply clicks on a content, which is highlighted with a green box and a check icon on the top right corner. The user can de-select them clicking on the highlighted area again. In our scenario, we suppose that the user selects the smartphone, clicking on the video control and description fragments.

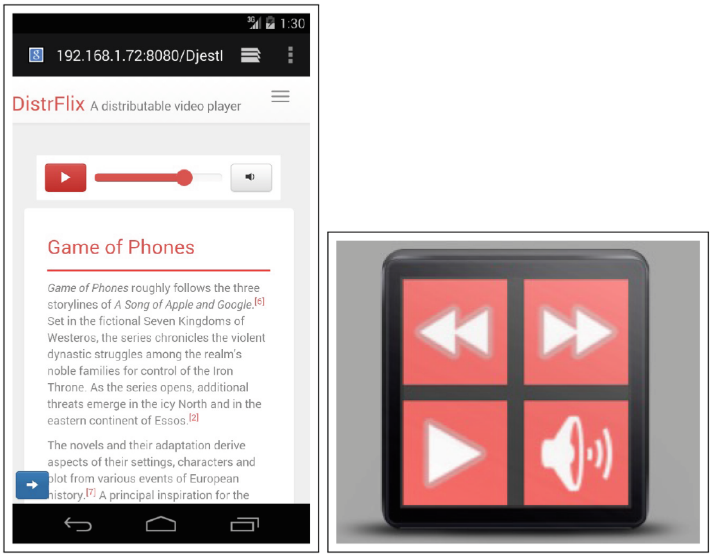
Once the selection is completed, the laptop *DistrClient* sends a request to the *DistrManager*, which in turn creates the new instances for the *FragmentState*, which registers for receiving updates from the *ViewState*. After that, the *Distr-Manager* notifies the smartphone *DistrClient*, which loads the web application interface. We implemented this push protocol in the prototype through a browser tab, which remains open as long as the device participates to the UI distribution



Game of Phones roughly follows the three storylines of *A Song of Apple and Google*.[6] Set in the fictional Seven Kingdoms of Westeros, the series chronicles the violent dynastic struggles among the realm's noble families for control of the Iron Throne. As the series opens, additional threats emerge in the icy North and in the eastern continent of Essos.[2]

The novels and their adaptation derive aspects of their settings, characters and plot from various events of European history.[7] A principal inspiration for the novels is the English Wars of the Roses[8] (1455–85) between the houses of Lancaster and York, reflected in Martin's houses of Lannister and Stark. Most of Westeros, with its castles and knightly tournaments, is reminiscent of High Medieval Western

Fig. 4. Distribution client (left part) and fragment selection interface (right part).



Game of Phones roughly follows the three storylines of *A Song of Apple and Google*.[6] Set in the fictional Seven Kingdoms of Westeros, the series chronicles the violent dynastic struggles among the realm's noble families for control of the Iron Throne. As the series opens, additional threats emerge in the icy North and in the eastern continent of Essos.[2]

The novels and their adaptation derive aspects of their settings, characters and plot from various events of European history.[7] A principal inspiration for the

Fig. 5. The web application UI distributed on a smartphone (left part) and a smartwatch (right part).

and receives the updates through a websocket. However, this step should be supported by the browser or by a dedicated application, in order to provide the user with a reliable and trustworthy mechanism. Finally, the smartphone shows the interface in Fig. 5 (left part).

Robert follows the same process for managing the devices when he wants to distribute the interface from the laptop to the TV (this time he selects only the video) and from the smartphone to the smartwatch. The interesting part in the final configuration is the equivalent video playback control from two devices. The application needs to coordinate the values entered by Robert through the smartphone and by Sarah from the smartwatch. The framework supports such combination as follows: when, for instance, Sarah presses the rewind button, the video control *Fragment* on the smartphone sends the update to the associated *FragmentState*, which writes the new value (e.g. the previous position minus five seconds) in the *ViewState*. The *ViewState* notifies the value change to all the other registered *FragmentState*s. One of them is associated to the phone that, again through a websocket, sends the notification to its associated *Fragment*, which is updated accordingly.

In addition, as shown in Fig. 5, the two interfaces provide different UI elements for the same action (same semantics, different implementation): the slider and the rewind and forward buttons allow the user to edit the same value (the video position) but the interaction is supported in different ways for different devices.

Finally, it is worth pointing out that the prototype implementation for the smartwatch interface is a native Android Wear application, which needs the support of an handheld for the network communication. The device limitations still does not allow to create web interfaces in such kind of devices, however we think that it will be possible soon.

8 Conclusion and Future Work

In this paper we introduced a framework for supporting the distribution of web-based user interfaces across different devices. We introduced the architecture of a device coordination solution that allows to separate the interface management aspect for the application logic. In addition, we described how it is possible to isolate the process of UI switching and splitting through a dedicated software component. Finally, we presented a distributed video player application as sample for the proposed framework.

In future work, we aim to provide a software library containing the reusable components for developing this type of applications. Our scope is to support the distribution as a specific aspect of UI development, which can be addressed injecting specific code even into existing applications. Moreover, we aim to merge the research in distribution with the solutions for developing multimodal interfaces, in order to go beyond the simple screen interaction for distributed UIs.

Acknoledgments. Lucio Davide Spano gratefully acknowledges Sardinia Regional Government for the financial support (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007–2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1 Avviso di chiamata per il finanziamento di Assegni di Ricerca.

References

1. Google: The new multi-screen world: Understanding cross-platform consumer behavior. Technical report (2012). Retrieved from: https://ssl.gstatic.com/think/docs/the-new-multi-screen-world-study_research-studies.pdf. Accessed 08 October 2014

2. Bandelloni, R., Paternò, F.: Flexible interface migration. In: Proceedings of the 9th International Conference on Intelligent User Interfaces, IUI 2004, pp. 148–155. ACM, New York (2004)
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interact. Comput. **15**(3), 289–308 (2003)
4. Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J.: The 4C reference model for distributed user interfaces. In: Fourth International Conference on Autonomic and Autonomous Systems, ICAS 2008, pp. 61–69, March 2008
5. Melchior, J., Vanderdonckt, J., Van Roy, P.: A model-based approach for distributed user interfaces. In: Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2011, pp. 11–20. ACM, New York (2011)
6. Frosini, L., Paternò, F.: User interface distribution in multi-device and multi-user environments with dynamically migrating engines. In: Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2014, pp. 55–64. ACM, New York (2014)
7. Ghiani, G., Patern, F., Santoro, C., Spano, L.: A location-aware guide based on active rfids in multi-device environments. In: Lopez Jaquero, V., Montero Simarro, F., Molina Masso, J.P., Vanderdonckt, J. (eds.) Computer-Aided Design of User Interfaces VI, pp. 59–70. Springer, London (2009)
8. Klokmose, C.N., Beaudouin-Lafon, M.: Vigo: Instrumental interaction in multi-surface environments. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2009, pp. 869–878. ACM, New York (2009)
9. Kim, K., Javed, W., Williams, C., Elmqvist, N., Irani, P.: Hugin: A framework for awareness and coordination in mixed-presence collaborative information visualization. In: ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, pp. 231–240. ACM, New York (2010)
10. Krasner, G.E., Pope, S.T., et al.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. J. Object Oriented Prog. **1**(3), 26–49 (1988)
11. Fowler, M.: Presentation model Retrieved from: http://martinfowler.com/eaaDev/PresentationModel.html. Accessed 08 October 2014
12. Potel, M.: Mvp: Model-view-presenter the taligent programming model for c++ and java (1996). Retrieved from: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf. Accessed 08 February 2015
13. Smith, J.: Wpf apps with the model-view-viewmodel design pattern. Retrieved from: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx. Accessed 08 October 2014
14. Paternò, F., Santoro, C., Spano, L.D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput. Human Interact. **16**(4), 19:1–19:30 (2009)
15. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.: Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In: Proceedings of INTERACT, vol. 95, pp. 115–120 (1995)