The Virtual Splitter: An Authoring Tool For Web Applications In Multiscreen Environment

Mira Sarkis, Cyril Concolato, Jean-Claude Dufourd
Telecom ParisTech; Institut Mines-Telecom; CNRS LTCI
{sarkis, concolato, dufourd}@telecom-paristech.fr

ABSTRACT
This paper presents a new approach that exploits existing web applications and re-purposes them towards the multi-screen environment. For this purpose, a virtual splitting system is proposed. It consists of a generalized mapping technique that, contrary to previous works, automatically analyzes the application content based on a semantic or a visual criteria and prepares it for the splitting process. The system delivers two instrumented applications ready for distribution across devices. During runtime, a mirroring phase is proposed to maintain the functionality of the distributed application and to support a dynamic splitting process. Developed as a Chrome extension, our approach is validated on a youtube page and on a video application from Mozilla.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Distributed Applications; D.2.11 [Software Engineering]: [Software Architecture]

Keywords
Authoring, Web Application, Multiscreen, Split, Mapping

1. INTRODUCTION
Recently, intense research activity has been focused on multi-screen scenarios inside home environment where cooperation between heterogeneous devices is possible. In such a cooperative environment we define a "Multi-Screen Application" (MSA) as an application distributed across multiple connected devices, each having a screen, and designed to offer a convivial experience. Examples of MSA are: using a tablet to display additional information synchronized with a TV program, or using the interaction capabilities of smartphones (e.g., touch screen) jointly with the large screen and processing power of PC or TV to display media elements.

Multi-screen applications impose multiple challenges to application developers. First, they have to design an application that leverages the multi-screen environment and copes with the diversity of devices. Then, they have to determine how the application content will be distributed across devices based on their specific capabilities, to manage the distributed content and to maintain the synchronization and consistency of the distributed content. The use of web technologies helps reducing the complexity of these tasks and increasing the possibility of deploying a ubiquitous application. In this paper, the term application refers to a web-based application. Many applications were created before the development of the multi-screen concept. Many of them are actually made of different components that could benefit from being distributed across devices. However, few of them were designed in a modular manner that facilitates code distribution.

With such a motivation and focusing on challenges related to authoring MSAs, this paper aims to meet one principal objective: To propose a new approach that reuses existing single-screen applications and re-purposes them for the multi-screen environment. In contrast to existing work, our approach is based on analyzing, mapping and annotating the application contents automatically and intelligently following the author or user choices, on instrumenting the application code to handle the dynamicity of applications, on splitting the application into two sub-applications and on synchronizing the sub-applications while maintaining the overall application functionality.

This paper is organized as follows. Section 2 compares our MSA approach with related work. Section 3 describes the approach, including the mapping, splitting and mirroring phases. An implementation of the solution, an evaluation and a survey on its limitations are given in Section 4. Finally perspectives and conclusions are drawn in Section 5.

2. STATE OF THE ART
This section presents prior efforts related to the MSA development. A "WebSplitter" was proposed to split XML-based applications, based on a metadata file. This file determines which application portions, i.e., which XML elements can be seen on each user device. The splitter requires a middleware proxy that splits the application content into partial views and a client-side component, e.g., an applet, that receives data pushed by the server. The XML splitter architecture is centralized and requires a manual mapping for each XML element of the application.

In his research, Cheng proposes a virtual browser capable of separating the application logic from its rendering. The logic is kept within a virtual web page. Automatically the virtual browser splits the main DOM tree into multiple...
DOM trees and maps these trees to corresponding devices as denoted in a hint file that is manually created by the developer. Cross-device operations are executed in a centralized manner depending ultimately on the virtual browser.

Bassbouss et al. [2] outline how to enable traditional applications to become multi-screen-ready. The application is developed as a single-screen application and requires a multi-screen enabled browser. Based on metadata information provided manually by the developer, specific elements are assigned to a remote device while always being shown on the main device. In contrast to the work of Cheng, Bassbouss et al. present a master-slave application model where the master application contains the logic.

In contrast to [8] and [9], our system has a decentralized architecture and similar to [2]. It delivers master-slave applications. The common part for the three previous works is that applications are analyzed and then mapped manually by the author (via a hint file or metadata). In contrast, we propose a system that is capable of automatically analyzing the application and then intelligently mapping its elements, thus simplifying the author responsibilities to only determine the analysis criterion.

3. VIRTUAL SPLITTING SYSTEM

3.1 Overview

A web application consists mainly of HTML, CSS and JavaScript (JS) resources. HTML mark-up usually provides the structural elements forming the application content, a CSS stylesheet determines how these elements are rendered on the screen, and a script contains the application logic that defines application behaviour. These resources are tightly linked. Links can be expressed in terms of DOM node hierarchy, JS DOM tree navigation, CSS tree-based selectors, property inheritance, etc. In such a context, splitting an application will break some links and cause a failure in the application look and functionality.

In addition, an application presents two dynamic aspects that make the splitting approach more complicated. On one side, splitting a web application, in contrast to a static web page, requires support for its dynamism since elements are continuously modified, created, moved or removed during runtime. Supporting this dynamism is essential, otherwise the application functionality breaks. On the other side, automatic partitioning of the application JS code is a hard task since JS is a dynamic language characterized by high-order functions, closures, 'eval' function which dynamically evaluates a string expression, etc.

In this paper, we take care of the links and dynamicity of the application by focusing on splitting only the HTML document, rebuilding and maintaining broken links while keeping the JS code as a whole running on one device. The following subsections present a detailed description of the phases evolved in this approach as illustrated in Figure 1.

3.2 HTML Elements Mapping

The mapping phase is the first phase in our system. Its purpose is to determine which of the application elements map to the devices involved in the multi-screen experience. In any multiscreen scenario, at least two devices are cooperating. The literature refers to the smartTV as the first screen and assigns the expression 'companion screen' [1] or 'second screen' [8] to a device providing a means of interaction with the smartTV services. In this work, the 'principal device' is responsible of processing the main application logic, while the 'secondary device' receives processing results only if it is concerned.

As depicted in Figure 1, the mapping phase takes as input a query from the application author, when the application is pre-processed offline; or from the user, when the whole process of splitting the application is done at run-time. In our approach, we have envisaged several possible mapping techniques: techniques based on the analysis of HTML elements, their associated semantics and roles discussed in Section 3.2.1; or techniques based on the visual rendering of elements discussed in Section 3.2.2. In our architecture these techniques could be combined. For instance, mapping only interactive elements placed in a certain region of the screen to the secondary device. The input of the mapping phase is a query which can be either a simple query indicating which mapping technique should be used along with its specific parameters (such as element category or position); or a combined query using boolean logic. The output of the mapping phase is two lists of elements, one for each device.

3.2.1 Semantic Analysis

As a possible mapping criterion, we describe here a semantic based approach. It is fully automatic, possibly selected by either the author or the user. In the context of multi-screen applications, we analyzed the HTML5 elements defined in the standard to determine their roles and how they could be classified for the purpose of application splitting. We identified four relevant classes: interactive elements (i.e., a, area, button, datalist, form, input, keygen, textarea, nav, optgroup, option, output, select), multimedia elements (i.e., video, audio, source, track), and non-interactive, non-multimedia visual elements (i.e., caption, dialog, embed, figcaption, h1 to h6, hgroup, img, kbd, label, legend, object, p, progress) and other elements.

As part of the query parameters, the author indicates one or more class of elements to be moved to the secondary device depending on the characteristics of the available devices. For instance, if a smartTV is present, the system or the user...
may decide to move only multimedia elements on that device. As another example, if a touch screen is present, the interactive elements may be moved onto that device.

This mapping technique produces the lists of elements as follows. First, for each element of the application, the algorithm analyzes and compares it to the tag names present in the classes indicated in the author query. If the element falls within the indicated classes, it is added to the secondary device list. If there is no match and if the element does not belong to the ‘other’ class, it is added to the primary device list. Then, the algorithm detects any change in the element basic role by checking its attributes, mainly declarative event listeners (e.g., ‘onclick’) since they are responsible of making elements interactive. A non-interactive element which role is only displaying content i.e., an image, becomes interactive if it has an event listener that lets users interact with it. In addition, we try to exploit the semantic links that are created between HTML elements (e.g., ‘for’ attribute) by keeping these elements together in the same list.

### 3.2.2 Screen Region Analysis

Another mapping criterion that we investigated is a region-based approach. It is also a fully automatic mapping approach but this time based on the application visual rendering. It is selected during run-time by the end-user in her browser. Once the user selects a screen region, the tool detects elements within that rectangular region to produce the secondary device list. All other elements are placed in the primary device list.

### 3.3 Annotating Elements

As depicted in Figure 1 the second phase is the annotation. It prepares the application for the splitting phase and takes as input the list of elements produced by the previous mapping phase. As a result, the application is totally annotated: each element contains metadata information, in a ‘data-device’ attribute, reflecting its target device. The annotation algorithm starts by processing each DOM leaf node and setting the ‘data-device’ value to ‘device2’ for elements in the first list and ‘device1’ for elements in the second list. Each remaining leaf node (resp. parent node) in the DOM tree is annotated with the value of its siblings (resp. its children), if the annotation is the same across siblings (resp. children), or with the value ‘dev1&dev2’ if they differ, meaning that the element will need to be present on both devices.

### 3.4 Splitting Application Content

After the annotation phase, upon user request and during run-time, the splitting phase relies on the element metadata information to form two separated applications: a master and a slave application. From the original application, elements annotated with ‘device1’ or ‘dev1&dev2’ values are kept visible on the screen of the primary device. Elements annotated with ‘device2’ value are hidden on the primary device. This forms the master application. These hidden elements serves as a shortcut whenever the application main logic requires reading or modifying elements of the remote application on the secondary device, thus the term ‘virtual splitting’. Elements annotated with ‘device2’ values are extracted from the original application. Elements annotated with the ‘dev1&dev2’ value are cloned. These last two types of elements are imported to the new slave application running on the secondary device. On the master application, in addition to the retained original application logic, JS code is added in the instrumentation phase and aims at making the master application capable of working synchronously with its slave (see 3.5). This code also helps support the application dynamism and ensure a dynamic mapping, splitting at run-time. On the slave application, the JS code makes the application capable of collecting user interactions, redirecting them to the master, receiving and integrating changes made to its DOM tree.

### 3.5 Mirroring Application Contents

The virtual splitting phase described in Section 3.4 duplicates some content between the master and slave applications. The role of the mirroring phase is to ensure that the slave application has a DOM tree that is an accurate mirror of the hidden DOM tree in the master application. It is performed as follows: On the ‘primary device’, any dynamic change affecting elements of the ‘secondary device’ (e.g., node modification, removal or creation) is mirrored to that device through change messages. Upon receiving a message, the application running on the ‘secondary device’ updates its DOM tree and integrates this change. On the ‘secondary device’, any user interaction (e.g., clicks) is captured and propagated to the ‘principal device’ where the interaction handler is processed.

### 4. IMPLEMENTATION AND RESULTS

#### 4.1 Implementation

We decided to implement our virtual splitter as a Google Chrome extension, to enable on-the-fly instrumentation of the application and to ensure that the instrumentation code is loaded before the single-screen application loads on the primary device, without having to change the application itself. Master and slave applications are rendered as tabs in the browser and communication between them is done using the postMessage API.

As discussed in Section 3.5, to detect relevant changes in the DOM we use the Mutation Summary library which is based on the working draft of the Mutation Observer API. A Mutation-Summary object is configured to watch changes made to elements with ‘device2’ annotation. If any change happens to these elements, their descendants or attributes, the extension sends a message to the ‘secondary device’. The message contains a list of changes. Each change object consists of the type of change, the concerned node, its position in the DOM tree (i.e., parent and previous sibling node), the concerned attribute(s) and the new value of a text node. However, the Mutation Summary library suffers from some limitations. For instance, it cannot detect changes made to HTML elements using JS functions especially if they are not expressed on the DOM tree. To overcome this limitation, as well as for other purposes, we use the Monkey Patching technique, to extend the built-in functions with custom code. In particular: the ‘createElement’ function to detect the creation of new elements and to trigger dynamically the mapping, annotation and splitting of these elements; the ‘setAttribute’ function to update the Mutation Summary configuration, to enable the mirroring of newly created at-

---

2. W3C DOM [http://www.w3.org/TR/domcore]
tributes; and the ‘addEventListener’ function to overcome
the limitation of the Mutation Summary library, and to re-
place an event handler triggered on the slave application to
a forward call to the master application.

4.2 Results and Discussion
We tested our system on different applications from sim-
ple static pages to dynamic applications, in particular on:
a semantic video application relying on the Popcorn and
JQuery libraries and showing various information. We first
used the region-based mapping. On the video application,
we separated the video and flick’r images from the additional
information as Figure 2 shows. This experiment verified the
performance of the mirroring phase by maintaining a reli-
able mirror and the synchronization between both master
and slave applications. In addition, no compatibility issues
were reported between our instrumented code and the JS
libraries. We then used the semantic mapping to split a
YouTube page and to separate the interactive class of ele-
ments from the other classes (i.e., non-interactive and multi-
media). As a result, we got the video running on the master
with all the comments of users while all buttons, anchors,
guide container that proposes additional videos to watch
later are moved to the secondary device.

Based on this, we identified a few areas of improvement on
which we plan to work in the future. This includes mainly
the re-organization of application layout based on devices
screen characteristics, solving the relative URLs problem
that causes the application to break and implies security issues,
overcoming the limitation of blob elements (such as canvas)
that have no HTML representatives for their content, but
they are usually controlled via JS code.

5. PERSPECTIVES AND CONCLUSIONS
In the multi-screen context, this paper proposed a sys-
tem to transform existing applications from single-screen to
multi-screen applications. The system consisted of an au-
tomatic, intelligent and extensible mapping phase that ana-
alyzed the application semantically, visually or a combination
of these two, an annotation and a virtual splitting phase that
resulted in master-slave applications. A mirroring phase en-
sured the correct functionality and synchronization between
both parts. We validated our system on two existing appli-
cations: YouTube and semantic-video, and we verified the
correct content mapping, synchronization and application func-
nality. As a future step, we aim at implementing our tool in
the COLTRAM multi-screen platform and extending this system with a context driven splitting technique that
collects information concerning involved devices and creates
dynamically adaptive mapping criteria.

6. REFERENCES
[1] S. Basapur, H. Mandalia, S. Chaysinh, Y. Lee,
N. Venkitaraman, and C. Matcalf. FANFEEDS:
Evaluation of socially generated information feed on
second screen as a tv show companion. In EuroITV ’12
Proceedings of the 10th European Conference on
and Y. Miyazaki. Towards a multi-screen application model
for the web. In IEEE 37th Annual Computer Software
and Applications Conference Workshops, pages
528–533, Japan, July 2013.
web applications. In Proceedings of the Workshop on
Multi-device App Middleware, Montreal, Quebec,
December 2012.
and S. Steglich. An open platform for multiscreen
services. In In the 11th European Interactive TV
conference EuroITV, Como, Italy, 2013.
unified xml framework for multi-device collaborative
Conference on Computer Supported Cooperative Work,
live streaming traffic transfer for multi-screen services.
In IEEE International Conference on Information
dynamic, third-party code customizations in javascript
[8] E. Teckleves, L. Cruickshank, A. Hill, K. Kondo, and
R. Whitham. Interacting with digital media at home via
a second screen. In 9th IEEE International Symposium