Digital Management of Multiple Advertising Displays

Arménio Baptista, Alina Trifan, António Neves
DETI/IEETA
Universidade de Aveiro
Aveiro, 3810-193

{armenio, alina.trifan, an}@ua.pt

***Abstract*—The technological boom that we have been experiencing in the last decade has impacted the retail sector in many ways. Captivating customers through smart advertising, engaging them in the retail process and enhancing their experience has been a long-time desideratum in this industry. Recent technology makes it possible to follow unprecedented approaches for achieving these goals. In this paper, we present a strategy based on a series of autonomous stations (either static, such as monitors, or mobile, such as autonomous robots) that can be used in any type of multimedia advertising across one or multiple entities. We present preliminary developments of this concept in an attempt to impact modern advertising. As a final product, this project aims to produce an autonomous system capable of displaying multimedia contents across several monitors.**

Keywords–Advertising; Cloud-based platform, Digital Contents Management; Multimedia.

I. INTRODUCTION

One of the keys to the success of the retail industry passes by advertising to the general public, mainly by using marketing strategies [1]. The technological evolution, more and more, has an important role in the marketing and the advertise of products [2]. In [3], experimental results show that sales in hypermarkets are enhanced when digital displays are used.

The main focus of this paper is to propose a solution to control the multimedia resources and display them using a unique platform which provides the management and manipulation over the resources in order to produce a final content to better fit the monitor associated to a terminal. This solution allows the control, maintenance, composition and division of the multimedia resources across the stations, displaying the information that the user selects into the different terminals. It also proposes a solution to control the permissions to the resources for each user logged in the platform. In order to handle the multimedia files uploaded by the users, the solution proposes the use of the FFmpeg library [4], which has compatibility with all major formats and codecs.

As it is a project in development, there are much aspects to improve and features to add. However, this paper exposes all the decisions made until date, starting with a system overview, following the management of the multimedia resources and the users and finally some preliminary results. This paper is structured in three other sections.

Section II overviews the methods that we followed in implementing this system. We present preliminary results in Section III and we assess the importance of this work and discuss future work directions in Section IV.

II. METHODOLOGY

The main goal of this project is to design a system capable to store, manipulate and manage multimedia contents uploaded by the users. This involves three major agents: the Web server, the control dashboard and the monitors, resulting in an architecture, as presented in Figure 1.

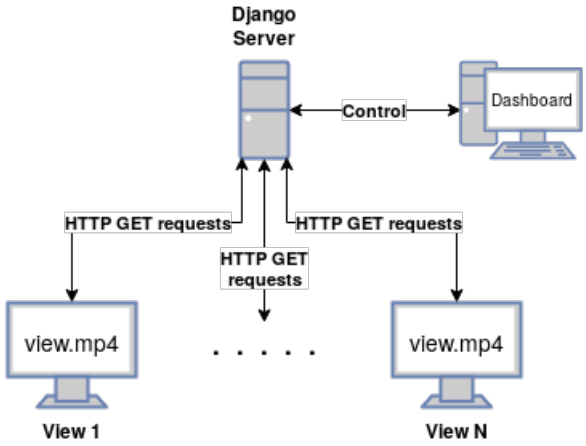

Figure 1. Proposed system architecture.

This architecture allows the Web server to control all the monitors (using HTTP requests to communicate) and expose a Web site so the users can manage them. Making the Web server a unique point of communication, the synchronization of the updates made by the users and the monitors is immediate.

A. Resources division

In order to split and organize the multimedia resources, a solution of three components is proposed, as described in Figure 2:

- **Contents:** base element of the multimedia resources, which is basically an image or video uploaded by the users.
- **Timelines:** set of Contents with a predefined sequence, much similar to a video composed of different contents. If the Content is an image, the user may define the duration of the image to be shown in the Timeline.
- **Views:** set of Timelines with a predefined sequence that is associated to a physical terminal. The final product to be displayed on the monitor and the only

 ISBN: 978-1-61208-634-7

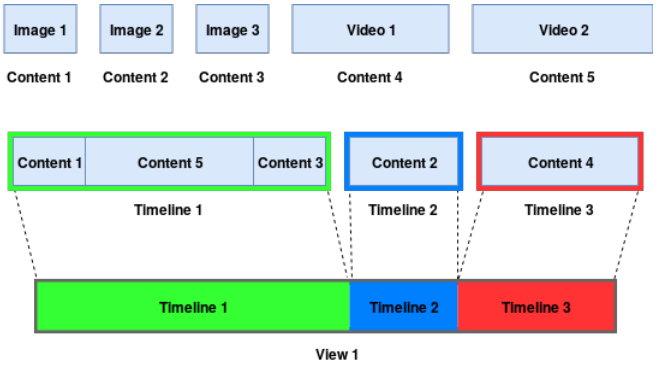

Figure 2. Resources composition.

way to create a View is with the connection of a terminal.

This division between the resources gives freedom to the users to create and dispose multimedia contents into any order and consequently display them. It also allows the users to reuse the Timelines in different Views without the need to recreate them again.

B. Users

In order to restrict the improper access of the users to the multimedia resources, a login system is proposed with different credentials and permission levels of the users. Therefore, before accessing the control dashboard, the system asks for the login credentials of the user. Django [5] has an user authentication system, which facilitates the implementation.

To control the users, the system has an *administrator superuser* that has access to a section in the dashboard that allows the management of the users (creation, edition, deletion and permissions). Then, the system has two types of users:

- **Administrator:** user responsible to manage the users access to the dashboard and their permissions for each resource. Additionally, the manager can add, edit and delete any resource, having no restrictions for his actions.
- **Regular User:** user with permissions predefined by the administrator and only with access to the resources predefined by those permissions.

C. Permissions

Given a scenario where the system is deployed in a shared environment between different users and the discretion of the resources needs to be maintained in order to block the modification of resources not owned by that user, a solution of permissions with two levels is proposed:

1) Resource level: The resource level has three different types of permissions: Contents, Timelines and Views. This permission defines what resources the user can create and edit and only the allowed ones will be displayed in the interface of the corresponding user.

2) Object level: The object level is basically what resources the user can access. By default, the user only has access to the resources he has created, although, the administrator can give access to a certain resource to a user.

The users with Timelines and Views permissions can view all the underlying resources in order to create a Timeline or View. To create a View, the user can use any of the underlying Timelines and to create a Timeline, the user can use any of the underlying Contents.

These blocking permissions allow, not only the creation of users to a specific task, but also provide the cooperation of users to a final multimedia product. As an example of specific tasks we can think of giving Content and Timeline permissions (Resource level) to a designer responsible to provide multimedia contents to the system or giving View permissions to a user responsible to the monitors inside of a specific building.

D. Website - control dashboard

In order to facilitate the control over the system, a website using the Django web framework was created. This dashboard gives control over four main resources: Contents, Timelines, Views and Users. Only the users with permissions over the resources can access and control the respective resource and only the administrator can edit these permissions and have access to the Users page. However, a regular user can edit his/her own profile information.

E. Image and video handling

Developing a system that handles multimedia contents needs the proper handling of the uploaded files. The system supports two types of files:

- **Image:** JPEG [6], PNG [7] among other image file formats supported by ffmpeg.
- **Video:** AVI and MP4 video file formats, among other video/audio containers supported by ffmpeg. All the video codecs supported by ffmpeg can be used (ex. H.264 [8], H.265 [9], just to name a few).

The upload of the files is made with the File Upload of Django and saved in a media directory of the server.

1) View creation: Upon the configuration of a View, if it has Timelines associated, the server begins the process to create the MP4 file associated to the view. This file is compressed using the H.264 standard [8], encoded with YUV420 at 25 frames per second, as all subjacent videos of the Timeline.

To better fit the resolution associated to the monitor, all the Contents are adapted to this resolution. In other words, when a View is configured with Timelines associated, the system goes through all the Contents associated to these Timelines and makes the changes needed to fit the screen resolution.

To create, manipulate and merge these files, the FFmpeg library was chosen since it has compatibility with all major video and image formats. The FFmpeg library adapts the Contents using mostly padding and resize transformations. When iterating over the frames of the Contents, the FFmpeg library resizes the frames which have different size from the resolution and applies padding to keep the Contents aspect ratio.

F. Connecting a monitor

The interaction between the different terminals (a single board like a Raspberry PI [10] and a monitor) and the Django server is made with HTTP requests from a terminal. In order

 ISBN: 978-1-61208-634-7

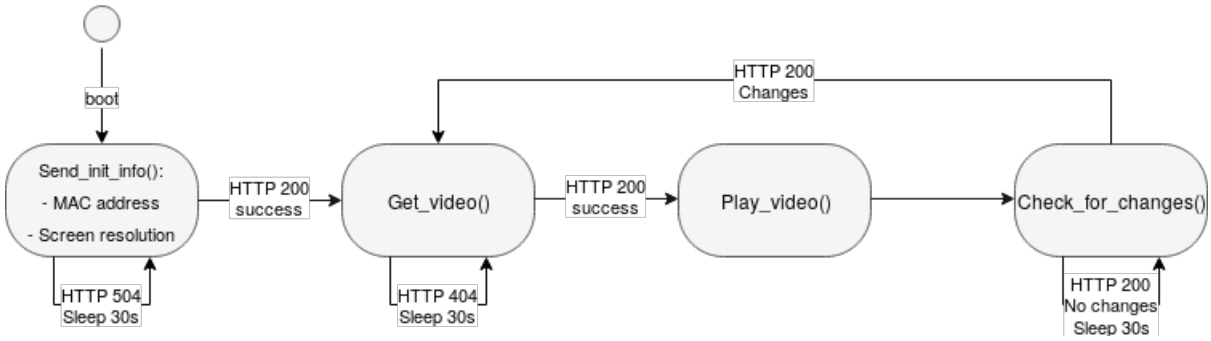

Figure 3. Monitor lifecycle.

to do this, a Python script was created and is responsible to control the terminal and its lifecycle (see Figure 3). When the terminal starts, the script runs on boot and goes through a series of steps in order to register in the server (if it is the first time connecting) and download the video.

Firstly, the terminal sends a HTTP POST request method with some information of his own:

- **MAC address:** this information allows the server to distinguish the different terminals and allows the user to know which View is associated to a terminal.
- **Screen resolution:** the resolution is used by the server to adapt the Contents associated with some View to its screen resolution.

The server, receiving this information, checks if the MAC address already exists and returns the path in the server to the video corresponding to that View.

Secondly, the terminal tries to download the video from the server with a HTTP GET request. If it fails (if the View wasnt configured yet or the server is down), the terminal waits a short time, thirty seconds by default, and tries to download it again. This loop is repeated until the video successfully downloads.

Finally, the video is played in loop and the terminal enters in a loop which keeps sending HTTP GET requests (with a time interval) to the server to check for changes in the video. This polling requests ensure if changes are detected, the video being displayed in the monitor is updated.

III. Results

While the work presented in this paper is a project in development, preliminary results regarding the current stage of development are presented in this section in order to confirm the effectiveness of the system.

A. Dashboard

As noticeable in Figure 5, the dashboard has a top Navigation bar, which has a Dropdown button so the user can edit his personal informations or logout, and a left Navigation bar with four possible navigations. This left Navigation bar only shows the possible navigations to which the user has access. When accessing a resource from the left Navigation bar, one of the pages presented in Figure 5 (except User) is displayed. This page allows the visualization over the existing Contents, Timelines or Views that the user has access. It also allows to create (except for Views as explained in Section II-F), edit or delete a resource and, if the user is the administrator, to edit the permissions (object level) of the users to that resource.

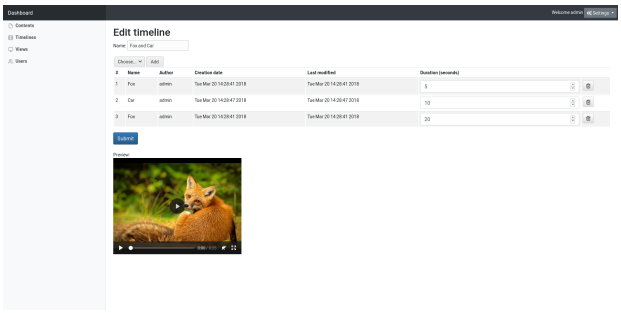

Figure 4. Edition of a created Timeline.

When editing a resource, a form is displayed so the user can edit the data associated to that resource, as shown in Figure 4. In the case of Contents, the name can be edited and a file upload button allows a new upload. However, when editing a Timeline or a View, a table with the Contents or Timelines associated to the resource, respectively, is displayed. This table allows the addition of objects from the dropdown button. Moreover, the objects from the table can be dragged and dropped into the intended order. When editing a Timeline, the table also has a duration input field for each image object. By filling in this field the user can specify the intended duration of each image that will be part of the timeline. In addition, a preview window displaying the video reproduced from the last Timeline submission is available.

B. Permissions

The platform that we propose supports user permissions at two distinct levels. On one hand, at the level of the resources and on the other hand, at the level of the objects. The administrator manages these permissions through the graphical interface as we will describe next.

1) Resource level: When creating a user, the administrator fills a form with some information associated to the user as in Figure 6. Note that this form has three Checkbox fields, each one for a resource (Content, Timeline and View). These Checkboxes are the permissions at resource level of the user being created and will define which resources the user will have access in the left Navigation bar.

 ISBN: 978-1-61208-634-7

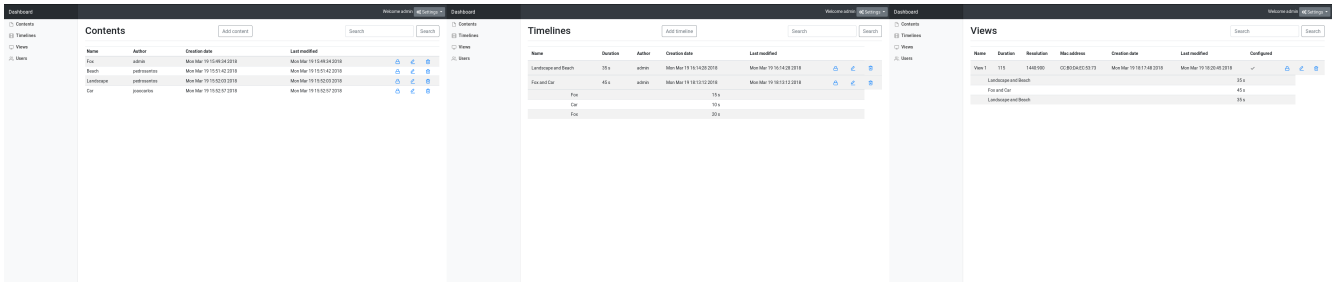

Figure 5. Visualization of Contents, Timelines and Views.

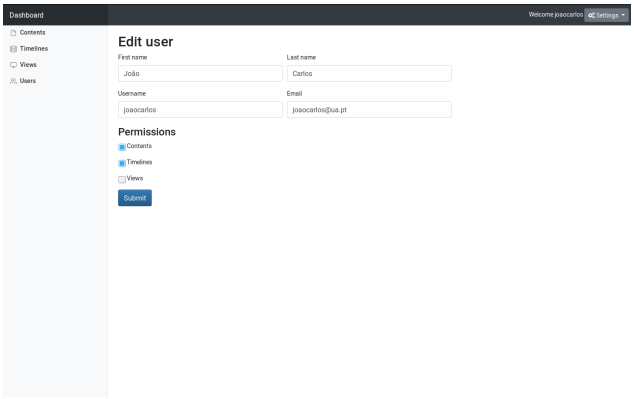

Figure 6. User edition with resource level permissions.

2) Object level: In Figure 5, there is a padlock for each object of the table. This padlock redirects to a page where the administrator edits the permissions over the respective object, as in Figure 7. This page lists all users with resource level permission over the resource of the object, so the administrator can grant access to that object.

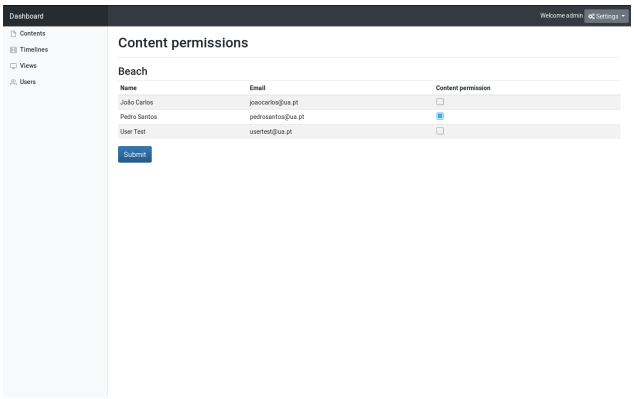

Figure 7. Modification of object level permissions of a Content.

C. Multimedia contents transformation

In Figure 8, it is possible to verify the transformations of FFmpeg. The figure shows five frames of a Timeline with three Contents (an image, a video and another image) with these resolutions:

- Image 1: 1000x665 px
- Video: 1280x720 px
- Image 2: 6000x1977 px

Before explaining the FFmpeg transformations to these 5 frames, it is important to refer 3 effects: letterboxing, pillarboxing and windowboxing [11]. Letterboxing consists in the transformation of frames with widescreen aspect ratio (16:9) to a standard-width video ratio (4:3) while preserving the frames original aspect ratio. This transformation consists of a padding transformation both on top and bottom of the frames.

On the contrary, the pillarboxing effect, consists in the transformation of a standard-width video format into a widescreen aspect ratio by applying padding into the frames both on left and right.

Windowboxing consists of the combination of both effects: letterboxing and pillarboxing. This is noticeable when the frames of a video are centered in the screen with a padding effect all around them. This happens when the resolution of the screen is bigger than the frames and no resize transformation is used.

Using FFmpeg with the arguments:

- ”scale” and ”force_original_aspect_ratio”
- ”pad”

makes it is possible to apply the intended transformations to the frames.

The ”scale” parameter allows to specify the scale resolution to apply into the frames, while using the ”force_original_aspect_ratio” to maintain the original aspect ratio of the images. This transformation will upscale the frames, if the resolution of the screen is bigger than the frames, and downscale, in the opposite situation. The ”pad” parameter allows to apply padding to the frames after the scale transformation. When the frames of the Contents have a different aspect ratio of the screen, the letterboxing and pillarboxing are perceptible.

Back to Figure 8, the example uses the FFmpeg to fit a monitor with a 1366x768 resolution which has a 1.78:1 aspect ratio. In Image 1 from Figure 8, as the resolution of the image is smaller than the screen, FFmpeg resizes the frames using an upscaling transformation to fit the screen. Although, as the aspect ratio of the image (1.5:1) is smaller than the screen (1.78:1), FFmpeg also applies a padding effect, resulting in a pillarboxing effect.

In the Video from Figure 8, FFmpeg resizes the frames using an upscaling transformation. In this case, the aspect ratio of the frames (1.78:1) and the screen (1.78:1) are equal, so

 ISBN: 978-1-61208-634-7

Figure 8. Expansion of a Timeline with five frames from three Contents (Image 1 (fox), Video (tree frames) and Image 2 (mountain landscape).

FFmpeg doesn't apply the padding effect and the frames fit perfectly the screen.

Image 2 from Figure 8 is exactly the opposite of Image 1. The resolution of the frames is much bigger than the screen and FFmpeg resizes the frames, but using a downscale transformation. As for the padding effect, the frames of the image (3:1) have a much bigger aspect ratio than the screen (1.78:1), so FFmpeg applies padding to the frames, resulting in a letterboxing effect.

With these FFmpeg arguments, the result will never reach a windowboxing effect, because the "scale" parameter will always try to resize the frames to fit the screen and the "pad" parameter will compensate the difference between the aspect ratios, either with letterboxing or pillarboxing effects. Using the FFmpeg upscaling transformation over the frames of the Contents may have influence in the final quality of the frames if the resolution of the image is much smaller than the screen, which forces the users to upload Contents with an appropriate resolution.

D. Final result

Figure 9. Monitor displaying a video using a Raspberry Pi to communicate with the system.

In Figure 9, we can see a real Full HD monitor (1920x1080 px of resolution and a 1.78:1 aspect ratio) with a Raspberry PI single board connected to the developed system displaying the final output, the video stream for that View.

IV. Conclusion

The main goal of this paper was to propose a solution to manage multiple multimedia contents in order to display them in multiple monitors. The system we have presented, as a whole, is operational and ready to manage the upload of multimedia contents and display them. It implies the existence of a computer to host the server (with proper image and video processing capability), monitors to display the contents, single boards to connect to the monitor (like a Raspberry Pi) and, of course, network connection between the server and the terminals.

A global overview over the results, highlights some features that we intend to improve, as future work. Such features are:

- The aspect of the dashboard that needs to be more appealing and intuitive to the user and display for example, snapshots of the Contents;
- A more advanced Timeline editor in order to give the user a greater control over the sequence of Contents;
- The capability to fetch information in real time about the monitor in order to show the status in the dashboard; Moreover, making it possible to run certain commands over the monitor, like turn on and turnoff would be a desired improvement of this work.

In the future, as the system improves, we intend to expand the supported formats of Contents, like supporting some presentation formats (PDF or PowerPoint), as well as audio files. This enhancement of the system would eventually imply a reformulation of Timelines creation in the dashboard. Another interesting feature that we plan to develop is the support for the interaction between the target audience and the system, either by a physical contact or even by voice control. This interaction would allow the user to pause and skip contents, for example.

While this is still a project in development, the preliminary results are optimistic and encourage us to improve the current solution by extending several of its features, as detailed.

Acknowledgment

This work was partially funded by FEDER (Programa Operacional Factores de Competitividade - COMPETE) and by National Funds through the FCT - Foundation for Science and Technology in the context of the project UID/CEC/00127/2013.

References

[1] A. Di Rienzo, F. Garzotto, P. Cremonesi, C. Frà, and M. Valla, "Towards a smart retail environment," in *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 2015, pp. 779–782.

[2] N. I. Bruce, B. Murthi, and R. C. Rao, "A dynamic model for digital advertising: The effects of creative format, message content, and targeting on engagement," *Journal of Marketing Research*, vol. 54, no. 2, pp. 202–218, 2017.

[3] A. L. Roggeveen, J. Nordfält, and D. Grewal, "Do digital displays enhance sales? role of retail format and message content," *Journal of Retailing*, vol. 92, no. 1, pp. 122–131, 2016.

[4] "FFmpeg," http://ffmpeg.org/, accessed: 2018-04-18.

[5] "Django," https://djangoproject.com, accessed: 2018-04-18.

[6] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

 ISBN: 978-1-61208-634-7

[7] C. Wilbur, “Png: The definitive guide,” *Journal of Computing in Higher Education*, vol. 12, no. 2, pp. 94–97, 2001.

[8] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the h. 264/avc video coding standard,” *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[9] V. Sze, M. Budagavi, and G. J. Sullivan, “High efficiency video coding (hevc),” *Integrated Circuit and Systems, Algorithms and Architectures. Springer*, vol. 39, p. 40, 2014.

[10] “Raspberry PI,” https://www.raspberrypi.org/, accessed: 2018-04-18.

[11] C. Poynton, *Digital video and HD: Algorithms and Interfaces*. Elsevier, 2012.

 ISBN: 978-1-61208-634-7