Final Report

Team Free Pizza, Multimedia Services

Name	*NetID*	*Studentnumber*
Tim Yue	tyue	4371925
Bart van Oort	bvanoort	4343255
Steven Meijer	stevenmeijer	4368061
Thomas Kolenbrander	tjkolenbrander	4353137
Etta Tabe Takang Kajikaw	eettatabe	4373758

Table of contents

1. Introduction

We have been contacted by PolyCast Productions B.V. to smoothen the workflow for them. The PolyCast team mostly records classical concerts. They currently use printed scripts and IP cameras in addition to manually controlled cameras. During live production, the cameras are put into their presets by the camera operators, in preparation for the next shot. The cameramen then control the camera manually to make the shot. After the recording, the recording is checked and is edited in some parts. Finally, the recording is done, but PolyCast is not fully satisfied. Some parts of this workflow seem to take up more time than desired and there are options lacking for flexibility during live production. The recording could be better if this workflow could be improved. To this end, PolyCast has presented us with a number of issues, which kept them from making an even better recording.

One of their problems is that the cameramen are spending a lot of time on loading presets. In the time that they are loading presets, they could be concentrating on controlling the current camera for their shot.

A second problem arises when the director wants to make adjustments during live production. Since scripts are printed, it is hard to make adjustments during live production, since scripts from the director and the cameramen are not synchronized. This greatly impacts their flexibility.

PolyCast have given us the freedom to resolve any of these issues and are open to any improvements in other sectors of the workflow. The only requirement put out is that our product must solve part of their issues.

2. Overview

Our software product is a camera control application, implemented in Java, which should smoothen the workflow for the people at PolyCast. The application uses JavaFX for its graphical user interface and is made in Java 8. Since we use VLCj in the user interface to show camera streams, it is required for the user to have a working VLC installation. This has to be the same bit version as their Java version. In addition to this, an internet connection needs to be available, so a database can be used to synchronise data between multiple instantiations of the application.

At the moment, scripts are printed and used during pre-production for rehearsals, and during live production to control the workflow at these stages of the production process. This is a problem for the people of PolyCast, because these scripts have to be manually updated for the whole team when abrupt changes are made during live production, leading to some shots not being taken perfectly or not taken at all during live production.
Our app handles the aforementioned problem by allowing users to digitally create a script, which can be accessed from our application. All changes or updates brought to the script at

any stage in the production process are automatically updated for every user at the time the changes are made.
Our product can also automatically load presets to the cameras during productions, giving the cameramen more time to focus on making better and more beautiful shots instead of manually loading new presets to a camera after the preceding preset has been executed during live-production.
Some other features have been implemented. A director can move through all the shots in the script during live production with one single button as requested by PolyCast. This is a handy feature for the director who can then easily move across the script during pre, post or live production, which is synchronised with the digital script of the cameramen, so they always know where they are in the script.

As of now, we have a working product which meets the needs of PolyCast. Our end product is able to access and control automated cameras by means of the camera's IP address. When connected to a camera using its IP address, the user can control the cameras using our application, which sends IP commands to the cameras with the necessary instructions. A live stream from the cameras can be displayed as well by our product, implemented using the VLCj API. The product should, as explained in the previous two paragraphs, be of great help to the people of PolyCast for an efficient workflow.

3. Software engineering

TFP Camera Control is written in Java 8 and has been designed along the idea of a model-view-controller design. This means that the product consists of three main components: the model, the view and the controller. The model component is discussed in section 3.1, whereas the view and controller components are discussed in section 3.2.

3.1 The Model

The model is the heart of the application. It consists of different classes, each with their respective responsibilities, modeled after the single responsibility principle [1]: a class should only be responsible for a single functionality of the software and this functionality should be encapsulated by that class. We also used several design patterns to improve the quality of our software. This section will explain where, how and why these design patterns were used.

The ‘Camera’ class was designed to represent a camera and keep track of the current cameras in the system. In order to actually set up a connection to and control a real camera, a ‘Camera’ object uses a ‘CameraConnection’ object, which handles all communication from the camera object to a real camera. That raised the problem that there are a lot of different kinds of real cameras, which operate in different ways, whether that be through a different type of communication altogether, or through differing messages within this type of communication. That is why ‘CameraConnection’ was implemented using the Strategy design pattern [2]: the ‘CameraConnection’ class is an abstract class, with the ‘LiveCameraConnection’ and ‘MockedCameraConnection’ classes extending this class. ‘LiveCameraConnection’ is an implementation that uses IP commands to connect to a

Panasonic AW-HE130 camera, whereas ‘MockedCameraConnection’ is a class that pretends as if it is a connection to a real camera, for when there is none. Using this pattern allows (other) programmers to easily extend the application for use with other cameras by creating another class that extends the ‘CameraConnection’ class and implements the required methods. Although the strategy pattern does prescribe to use an interface instead of an abstract class, we decided to use an abstract class for the reason that interfaces cannot define ‘protected’ methods. We decided that we should not make these ‘protected’ methods ‘public’, since they should not be called from outside the ‘Camera’ class, with the sole exception of the unit tests, as it may result in inconsistent settings in a camera object.

The ‘CameraConnection’ class also uses the Observer design pattern [3] to maintain synchronisation with the camera object more easily: it implements the ‘Observer’ interface, allowing it to observe a camera object, which extends the ‘Observable’ class and notifies its connection when there have been changes to the settings of the camera. The ‘CameraConnection’ class then decides what commands it should send to the actual camera in order for it to actually move to the specified location.

The same strategy design pattern used for ‘CameraConnection’ was also used for the ‘Preset’ class, which describes a preset for a camera. Since a ‘Preset’ object needs some basic information, such as an identifier and description, we used an abstract class here as well. This class has one abstract method that should be implemented to define the way the preset is applied to a camera. Currently there is one implementing class, namely ‘InstantPreset’, which as the name implies, applies a preset to the camera instantly. Using this design pattern allows programmers to define presets that apply presets in specific ways, for example over a certain amount of time.

Finally, we used a Singleton design pattern [4] to design the ‘ApplicationSettings’ class. This class keeps track of some general settings used throughout the application. Since these settings should be the same throughout the entire application and because there should be a single, static access point to these settings, we decided that it would be best to use this design pattern. It makes use of a so called ‘eagerly created instance’, which means that the only instance of the ‘ApplicationSettings’ class that exists, is created during the static initialisation of the ‘ApplicationSettings’ class, therefore making sure that this instance is created before it can even be requested.

3.2 The View and Controllers

The view consists of several FXML files that define what components the user interface consists of, where these should be placed and what some of their respective settings are. The controllers for the views are the classes that load these FXML files and programmatically control what exact data should be displayed in which area of the view. Together, they form the user interface, making use of the JavaFX framework.

We chose the JavaFX framework because it is the most recent, well maintained and most functional graphical user interface framework there is for Java. There is support for animations, as well as media players and can be styled with CSS. But most importantly, it supports the FXML documents that can be made using the JavaFX Scene Builder: a stand-alone application which allows its users to create these FXML files using an easy drag-and-drop system.

4. Functionalities

In order for our application to be useful, certain features had to be implemented. Three of these features will be discussed in this chapter.

4.1 Script creation

First and foremost, our application enables the user to create, edit, load and save a digital version of their script. A script is made by composing multiple shots in a row. A shot consists of multiple items, the most important ones being the camera that is supposed to make the shot, and the subject that the shot is focused on. By providing the opportunity to create a digital script, we remove the need for tons of printed out scripts that can not be modified as easily as this digital alternative. This script can be changed and shared instantly with everyone that needs access to it. A script can also be saved to and loaded from an XML file, which can then also manually be transferred to another computer.

While creating a script, the application checks if multiple shots in a row use the same camera. This is not allowed to happen, since that would mean that a single camera would have to take the first shot, move to its next shot while still live and take the next shot. PolyCast told us that a camera is not allowed to move to a different shot while it is live, since this will look strange.

4.2 Script synchronization

During live production every team member should see the exact same script and know what the current shot is. In order to achieve this, we implemented the synchronisation of the script with a MySQL database. This database stores the current script, the camera presets and the current shot. When the director decides to move to the next shot, he presses a button and everyone's interface shows the next shot. This is an improvement in the workflow because the communication about what shot is live is now fully handled by our application, avoiding human errors.

4.3 Automatic loading of presets

In order to improve the workflow even more, our application implements the automatic loading of presets. When a camera is finished with a shot, it will automatically load its next preset, meaning the camera automatically moves to the position it should be in, in order to take its next shot. As a result of this, the cameramen themselves do not have to load new presets to a camera each time a shot needs to be taken, since this is done automatically, therefore giving them more time to make beautiful shots during productions. The director controls the progression through the script by clicking a ‘next shot’ button. He or she also has the option to go to the next shot without loading any presets in the cameras. When loading is turned on again, all cameras are moved to their next preset.

5. Interaction design

The first part of our interaction design process was to visit PolyCast to get a view of who we would be dealing with and the status quo. During this meeting we discussed how they would like to be able to create digital scripts and that it should be easy to change the aforementioned script. This led us to propose a drag-and-drop interface, which they really seemed to like. We also designed the interfaces for live production with a flow model which showed that the cameramen and the director both need their own interface.

After weeks of work it was time to get some verification, we had to know if we were building the right interface. We did this by sending a screencast of our program including a questionnaire, which can be found in Appendix A. The feedback we received was quite detailed and we found it to be very useful. The most important information contained in the feedback was the fact that during live production we do not need to display the camera livestreams. PolyCast already has its own monitors in order to do that. This led to major changes to our live production interfaces. By removing the livestreams we had more space to display other aspects. We used this space by displaying the script, as requested by Polycast.

6. Evaluation

Our product is ready for release with all its functions and features, but what issues are covered and by which functions? What could be better? What are its weak points? In the coming subsections, we will evaluate the functions of the product, followed by a failure analysis.

6.1 Evaluation of functional modules

As mentioned above, the main wishes that PolyCast have is to give their cameramen more time to control the camera when taking shots and to have more flexibility during live production.

Flexibility during live production is reached by the combination of the functions mentioned in 4.1 and 4.2. Because of these functions, there is no need to print out paper scripts. Instead, the scripts are digital and synchronised through a database, allowing changes to be made on the fly, thus improving the flexibility.

Freeing up the time the cameramen spend on preset loading is achieved by the automatic loading of presets in a previously prepared script, as mentioned in section 4.3. Since the director is automatically loading the presets when going to the next shot, the cameramen can focus on making their shot.

6.2 Failure analysis

The features of our application resolve part of these issues, but the people from PolyCast did mention some additional things that seemed like a good addition to them.

For the script, we do not have a tag field, which they could use to tag a shot with ‘obstructed’, or ‘too crowded’ on the fly. Also, all shots in the digitally created scripts in our application are sequential, so no alternative shots can be set for one particular shot, which they could choose from during live production. Finally, although we have a ‘subject’ field, it needs to be filled in manually, although the members of PolyCast did mention that it would be nice to have several subjects to choose from.

Automatic preset loading is done when the director presses the next shot button. It is also possible to skip the loading of the preset. Right now, either none of the presets are loaded, or every next preset is loaded except the live camera’s. This can make skipping confusing to use, as it is not possible to skip only one preset. It would be better if there would be more control on which preset should not be loaded.

The UI is simple and effective, but it is so simple, that there is still more room for polishing. In the current state it is very much usable, but not so much appealing to the eye, which is always a plus for the user.

7. Outlook

Unfortunately our application is not perfect. There are still a few features for which we did not have enough time to implement. Those features will be discussed in this chapter, as they might be added in the future.

7.1 Preset image synchronization

When a preset is created, a picture of the current view of the camera is saved. This picture can be used to display what the preset will look like and can be used when the script is displayed. When one of these pictures is taken, it will only be stored on the local machine. When a user on a different machine wants to display this image, he simply can not do that since the image is not on his machine. That is where the synchronisation of preset images comes in. For this feature, we would use the Imgur API to upload the pictures, and the application would download the necessary images by itself since the URL would be stored in the database.

7.2 Android support

During live production every cameraman has his own Android tablet. In order to keep the workflow more or less the same, our application should be able to run on Android. Since our application uses Java and the model-view-controller design pattern we would only have to create new view and controller classes to support the Android Java runtime. Then again, since Android does support JavaFX applications, the controllers may not need as much change. The model does not have to be changed, making it easy to support the Android environment.

7.3 Dynamic presets

Since we use the strategy pattern for our presets, it would be easy to create new types of presets. One of these presets could be the so called ‘dynamic’ preset: a preset which contains a movement, “turn 15 degrees to the left in 5 seconds”, for example. Another option is that they could perform a movement and record that movement in a preset. This allows the cameramen to plan out all their movements in advance, which means there is more time for error correction during live production.

References

[1]: Martin, R. (2003). *Agile software development: Principles, patterns and practices* (pp. 95-98). Upper Saddle River, N.J.: Prentice Hall.

[2]: *Strategy Design Pattern*. (2016). *Sourcemaking.com*. Retrieved 16 June 2016, from https://sourcemaking.com/design_patterns/strategy

[3]: *Observer Design Pattern*. (2016). *Sourcemaking.com*. Retrieved 16 June 2016, from https://sourcemaking.com/design_patterns/observer

[4]: Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2000). *Design Patterns: Elements of Reusable Object-Oriented Software* (p. 127). Addison-Wesley.

Appendix A

Dear PolyCast,

As shown in the screencast, a simple and easy to use Graphical User Interface for camera control has been made by Team Free Pizza. In the beginning of the cast, we have the main menu in which we have an overview of the features which have been implemented by our Team. From top left to bottom right we have:

1. **view as director**
2. **view as cameraman**
3. **Edit Presets**
4. **Create New Script**
5. **Preview Shots**
6. **Edit Current Script**
7. **Load Script**

First and foremost, it will be of great interest to us to know what you think of the main menu. Are there features which you do not like to have and why? Or are there some features which are lacking? It is true that it is a very simple GUI but we made it this way so it may be easy to use by everyone without difficulties, with all features being very clear as to what they do.

Considering the fact that the screencast explains every feature more or less in detail we, as a team, are curious to know what you, as a user of our application thinks of the application. It is an application with features which are extendable meaning changes can still be brought to them as of now. For each of the above listed features, we would like to ask the following questions:

1. How self explanatory and clear are these features?
2. Are there some aspects lacking in any of the features? Maybe some handy things the director would like to see on his screen during live production which have been omitted, some cool feature you would like to have while creating a new script, etc.
3. Do the features contain too many details? Maybe some unnecessary aspects which need to be removed for better clarity.
4. What would you like to have added to the application in general as users of our product?
5. How do you expect the end/final product to look like?
6. Are there any major changes which should be made to the present product?
7. Are you happy with the product? Is it more like what you expected or far away from what was/is expected from us.

We also had some uncertainties on our side regarding the workflow:

1. Do the cameramen only use tablets or do they have their own PC?
2. Would a live interface for the cameramen be any help at all since they have their own interface already?
3. Do you always have an internet connection available?
4. How many PCs are usually used?
5. Do you know the static IP-addresses of the cameras before production?

Regarding questions 1 and 2, we were not sure if the live production UI would be helpful or usable for the cameramen and were thinking of just having a live view for the director. About question 3 and 4, we were wondering if a database is necessary so the script and presets are synced over several computers. Regarding question 5, we were wondering what the workflow is and should be like when connecting to the cameras.

Furthermore, we were having doubts on the preview screen and having simple movements like a constant zoom in or out. We were doubting the actual usefulness of this, so we would like some feedback on this.

We are almost at the end of context project and as such may not have sufficient time for implementation of very heavy or major changes. We also do not have any extra feature or point which distinguishes us from other groups. A mail was sent by Bart van Oort regarding this, and we have yet to receive a response on this matter so we could work on it.
Your feedback will go a long way to making our application more interactive and even better and as such, will be very much appreciated.

Kind Regards,
Team Free Pizza