Emergent Architecture Design

Multimedia Services Team goto fail;
Alex Geenen 4210891
Bart de Jonge 4392256
Mark van der Ruit 4228723
Martijn Janssen 4316709
Menno Oudshoorn 4395751

April 29, 2016

Contents

1 Introduction

This document provides an overview of the software system that will be built during the Multimedia Services Context Project. The architecture of this system is explored by breaking it down into its subsystems, which as a whole, come together to form a solid software package.

1.1 Design Goals

Chief among the priorities within this project are the design goals:

1.1.1 Availability

The system will be built using an Agile process, and will utilize continuous integration to ensure that the software will always work. As such, the goal is to have a working product at the end of each development sprint, ready for use & testing by the client. This ”always ready” design discourages software anti-patterns and contributes to the overall health of the project. In addition, it allows the client to give feedback after every iteration, enabling changes to be made without increasing technical debt.

1.1.2 Modularity

In a digital audiovisual workflow there are many different users involved, each of whom have varying software needs. In order to meet all of them, the software system has to be flexible. This flexibility is best reflected in a modular architecture, as modularity encourages code and component reuse, which in turn increases the team's efficiency (and is also a software development best practice). Instead of designing and building three or four standalone systems, this principle allows the creation a blended system which in turn streamlines the workflow. In addition, changes can be made without fear of complex interdependencies.

2 Software Architecture Views

This section explains the architecture of the system. The overarching idea is that one program is used to plan and execute the camera script, which then sends commands to the remote IP cameras.

2.1 Subsystem decomposition

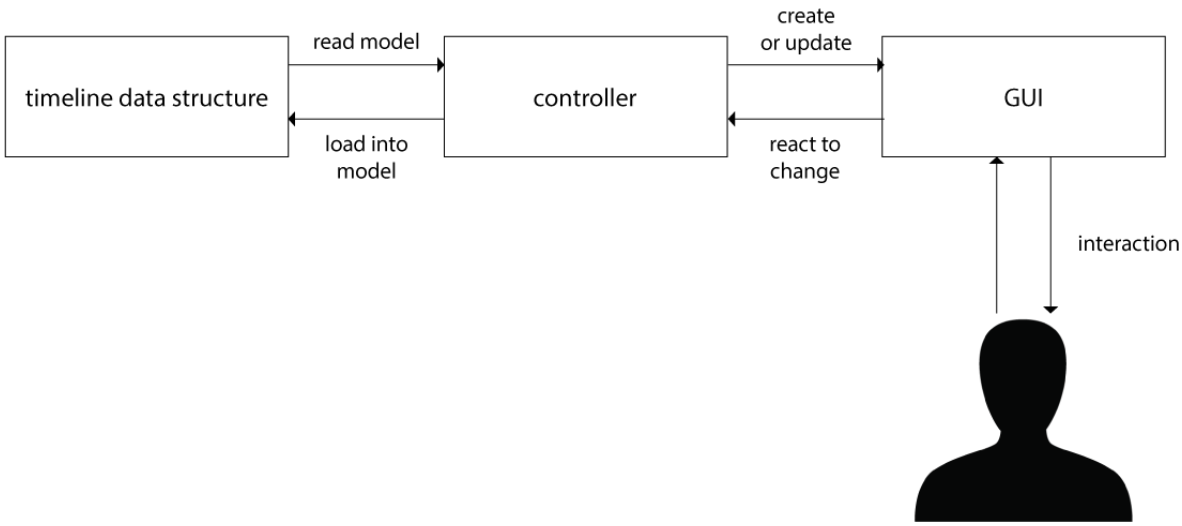

Figure 1: Architecture Decomposition

The architecture of the scripting software is divided up according to the Model View Controller design pattern. This pattern consists of a clear division of labor between the data model, the view that the user sees and interacts with, and the controller which ensures that the view reflects the model and vice versa.

- The model: The model consists of a data structure that reflects the state of timeline for the scripting view. The timeline data encompasses the shots both for the director's script and the individual cameras.

- The view: The view consists of a GUI which displays the current state of the script in a timeline format, and the buttons required to add or make changes to shots and otherwise manipulate the entire script

- The controllers: The controller reacts to changes in the view (e.g. resizing a shot), which then trigger logic that keep the model up to date (e.g. Making sure that the shot length is updated). These adjustments are not just one way. Based on actions in the view, other portions of the view may also change, allowing complex and powerful actions to be simplified. The controller is also responsible for initializing the part of the software that it "owns". This means that it is able to load persistent data into the model, and then subsequently create the corresponding view.

2.2 Hardware/Software Mapping

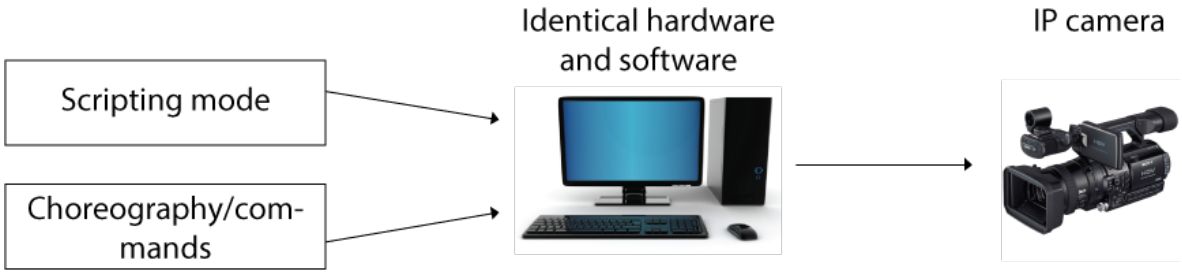

Figure 2: Hardware to Software Mapping

The hardware and software will be the same for both the director and the camera operator. The functionality of the software will differ through the use of different modes. There is one mode for the scripting, and one for camera choreography and coordination of the commands which will be sent to the IP cameras. When a recording is taking place the software will send commands from the computer to the proper cameras.

2.3 Persistent Data Management

Data persistence in the system is accomplished through the use of XML files. Each script is loaded in and saved as an XML file, recording information about the timeline which can then be used by the controller to restore the program state. It also facilitates the transfer of a project between computers if necessary (e.g. from the director's computer to the camera operator's computer).

3 Glossary

- GUI - A graphical user interface. What the user interacts with.