Architecture Design, TI2806

Blazin and the Goons

Authors:

Shane Koppers,	4359062
Floris List,	4380258
Hidde Lycklama,	4397614
Joris Oudejans,	4393694
Jordy Weijgertse,	4247124

Contents

1 Introduction

In this document the design of the system being built will be discussed. The architecture of the system will be explained as well as split into smaller parts to explain further.

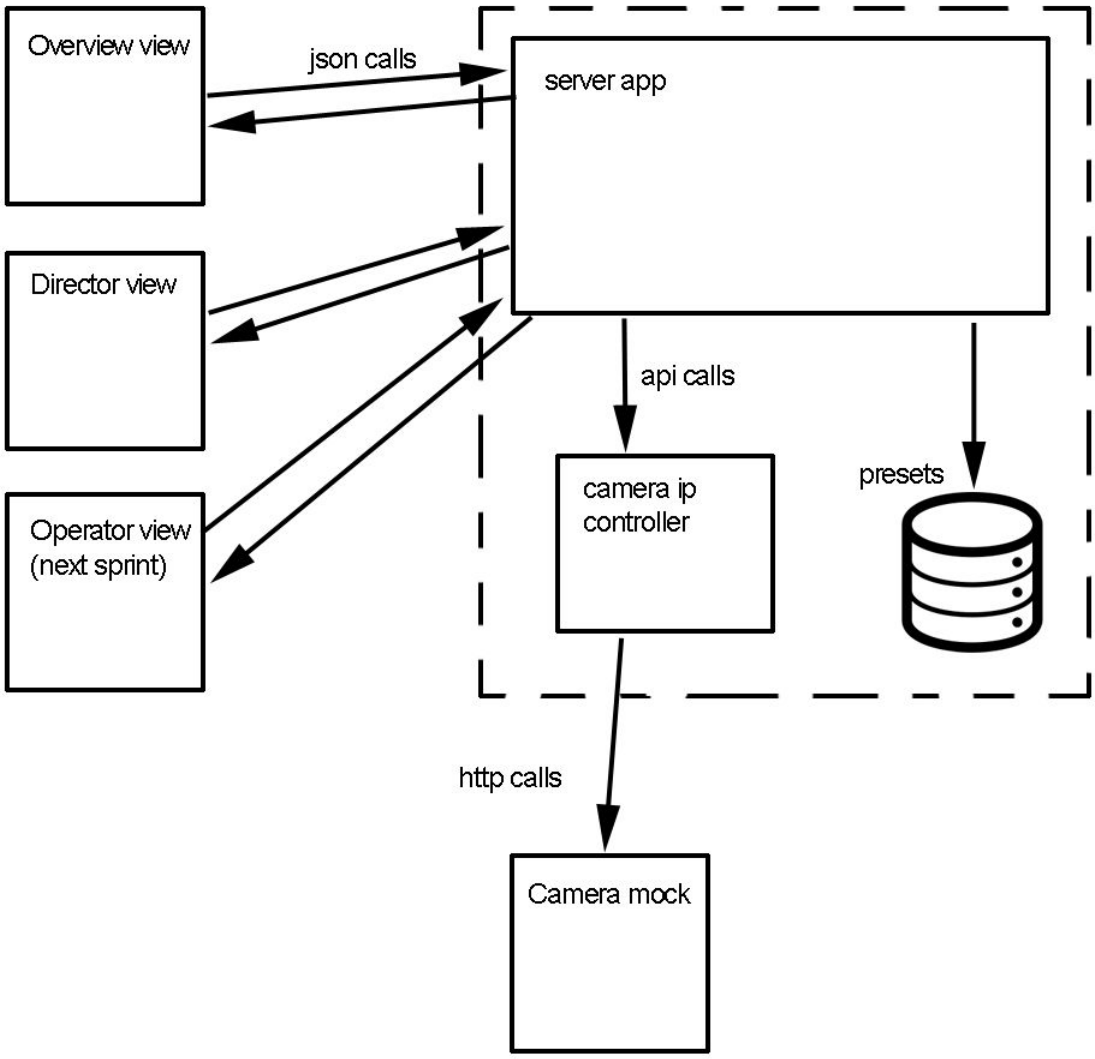
1.1 Design goals

- **Performance**
  Since we need to improve the workflow of the camera crew performance is of utmost importance. The system needs to work fast with as little delay as possible. Too much delay can lead to problems when cameras are not in position when they need to be.
- **Reliability**
  The system is being designed to lift pressure and workload off of the camera crew. If the cameras are not in the right preset at the right time or some other malfunction happens, the crew can't trust the system, especially not in a live environment. If they can't trust the system they also can't focus on the camera work.
- **Securability**
  If the server is not going to be local it should be unable for outsiders to go in the system and control the cameras or change the presets for obvious reasons.

2 Software architecture views

This chapter will explain the architecture of the system. The first paragraph will show the system as a set of subsystems and how they interact with each other. The second paragraph will explain the relation between the hardware and the software of the system. The third paragraph shows how the system manages its data. Lastly, the fourth paragraph explains how the system stays concurrent and prevents unwanted interactions.

2.1 Subsystem decomposition



The whole system is divided into subsystems:

- **Views**
  There are different types of views which each a different purpose. A manager view

for creating presets and timelines for live performance, an operator view for the camera operators and a director view which gives an overview of the different cameras.

- **Database**
  There is a database which holds the preset position the cameras are in. This database can also be extended with login information if the server won't be local. The script will also be stored in the database.
- **Camera ip controller**
  A camera ip controller which converts api calls the server app will send into ip calls the cameras can handle.
- **Server app**
  The server app here is the central part. It delivers the views to the client's computer, fetches presets from the database and converts the json calls to api calls for the camera ip controller.
- **Camera Mock**
  An extensive mock which will not only reply with the correct http messages, but will also act out its behaviour with the use of an image. Pan/tilt will move the camera on the x- and y-axis and zoom will enlarge the image.

2.2 Hardware/software mapping

Most of the hardware/software mapping is explained in the previous paragraph. A user can use a pc to connect to the server, which in turn will return one of the views. The user can then use those views to send calls to the server which will then handle those as explained in the previous paragraph.

2.3 Persistent data management

For local development, an in-memory database is used. This provides good flexibility and makes sure no bugs occur that arise from the (lack of the) state in the database, because every new run the database is refreshed.

There is an online version of the system which interacts with a Postgres database. This instance can be found at http://blazinandthegoons.herokuapp.com

2.4 Concurrency

The application has a lot of possible concurrency problems. This is because all views share the same database and running video process, concurrency can be an issue if not tackled rightly.

At this moment, concurrency is not yet an issue because the features that create them are not yet developed. When they are developed, extra, special tests to test concurrency will be made (using threads).

3 Glossary

View - A way the program represents itself through a user interface/web page.
Client - A remote browser that runs the app and is connected to the server.
Server - Mainly the brain of the structure, the server synchronizes the views on every client.