Multimedia Services : Architecture Design

Group BeNine
TI2316 Context Project

Bryan van Wijk (bryanvanwijk, 4363329)
Dorian de Koning (tcmdekoning, 4348737)
Ege de Bruin (kedebruin, 4400240)
Jochem Lugtenburg (jlugtenburg, 4370805)
Naomi de Ridder (nderidder, 4383109)

June 10, 2016

Supervisor: Dr. Cynthia Liem
Software Aspect TA: Valentine Mairet
Context Aspect TA: Alessio Bazzica

Delft University of Technology
Faculty of EEMCS

Contents

1 Introduction

In this document the architectural design of the system which is developed during the context project Multimedia Services is described. The initial version of this document is created in the first weeks, but it is extended where necessary during the project.

1.1 Design goals

In the product vision, goals are described about the features in the system from the users point of view. Here the goals for the system design which are not directly visible to the users are described. These are as necessary for the success of the project as the goals in the product vision.

Performance

Performance is very important in the system, camera operators must be able to load fast a preset when the director asks for it. Otherwise the system will be useless because they could do it faster if they do it manually.

Reliability

The presets stored are expected to be the right presets as they are loaded again and they are not changed because of an error. The system must also be reliable as in it will be very unlikely that it will crash during a recording.

Portability

The system should be usable with different operating systems. The web interface should be runnable on different tablet brands. Also the system should be extend-able, so a change of camera type must be possible without throwing away the whole system, but with a simple change in the system.

Maintainability

It must be easy to change parts of the system, for example when the users want an additional feature. The possibility of defect must be as small as possible but when they are detected they must be easy to find and fix.

Usability

Working with the system takes place under time pressure during the recordings, so everything has to be work smoothly to make this process less stressful. Also, the system should require a low amount of training before use.

Robustness

The system must be capable of handling invalid inputs. This refers to the reliability, because crashes cannot be tolerated during a recording.

2 Software architecture views

2.1 Subsystem decomposition

The system is divided in two main parts. The front-end server, providing a tablet web interface, and the back-end server, proving the model. This decomposition makes it possible to change the front-end view of the system by for instance replacing it with a native app for other devices instead of a web interface. In Figure 1, a global overview of the system is given.

The front-end interface consists of an HTML, CSS and jQuery web interface and a NodeJS server, serving the web interface. The web interface communicates with the back-end and NodeJS server over the HTTP protocol, using HTTP requests. The NodeJS server provides all files necessary for the web interface to run, and configuration options. The web interface itself is responsible for the communication with the back-end server.

The back-end interface consists of a model written in Java. The back-end provides the central system and will perform all computations, control the camera's and handle communication with the presets that are stored in a database. Information from this database can be retrieved using SQL queries which will be sent to the database. Next, we will further describe the various subsections.

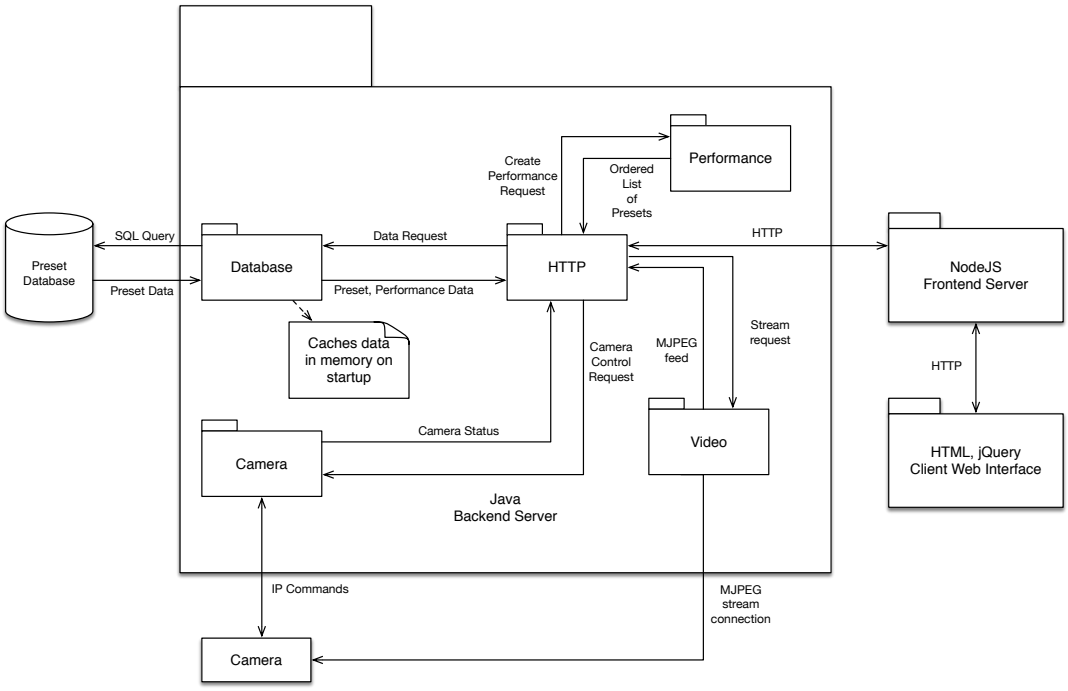


Figure 1: Global overview of the system

2.1.1 Java back-end

The Java back-end server is where all the main computations are done. The intelligent presets are calculated and corresponding commands are sent to the camera's. These commands are sent using the java camera module, implementing the camera communication for various camera types. If the client requests information about the camera's, e.g. the camera status, using HTTP requests, this will be processed by the HTTP module of the server. This module provides endpoints for controlling camera's, fetching preset information and general server status information. The HTTP server is made asynchronously, because that makes it possible to send multiple streams at once. When the program starts, SQL queries will be sent to the preset database, getting all the presets from the available cameras. When the HTTP module of the server requests data, the back-end will send the preset data back from the memory. Only getting the data from the database when the program starts ensures that de program keeps working if the database crashes. In the back-end, a configuration file with information about the camera's (like their IP address), can also be found which offers a layer of protection and can be easily used by the user.

Camera implementation

The communication with the camera's is done from the java back-end server. Every camera type has its own class which implements the abstract basic camera class. For every extra function a camera has, the camera implements the corresponding interface. The rest of system will treat every camera type as the same. The creation of camera objects is done using a CameraFactory for each camera type. The concrete factory has the responsibility of creating the right camera object. The responsibility of choosing which camerafactory to use is for the CameraFactoryProducer. The class diagram of this part of the system is displayed in Figure 2. The idea of using this pattern to implement the camera's is that this makes it easy to change to another camera type if necessary.

MJPEG Streaming

If a camera supports Motion JPEG video streams, the server connects to the streams. A MJPEG Stream is a sequence of JPEG frames, each containing a header and jpeg image. The server reads and parses the bytestream to find the header and image part of each frame and then distributes it. Optional is a form of compression in which each image frame is resized to be smaller, reducing the amount of data used by the client web interface.

2.1.2 NodeJS server

The NodeJS subsystem provides the resources needed for the clients view. It provides routes serving the HTML, JavaScript, jQuery and CSS files required to run the client. It can also provide the user with configuration for the web-interface such as the location of the back-end server to send requests to. The API can return information about the backend server like its server address and port in JSON format. All in all, the NodeJS front-end serves as a bridge between the client and the back-end. The choice for NodeJS could have been any platform. Since the software clearly distinguishes between the back-end and the front-end server, the front-end server can be replaced. If the client wishes to develop a native mobile application, they can throw away the Node Server and just let the application, or operating system speak to the back-end server.

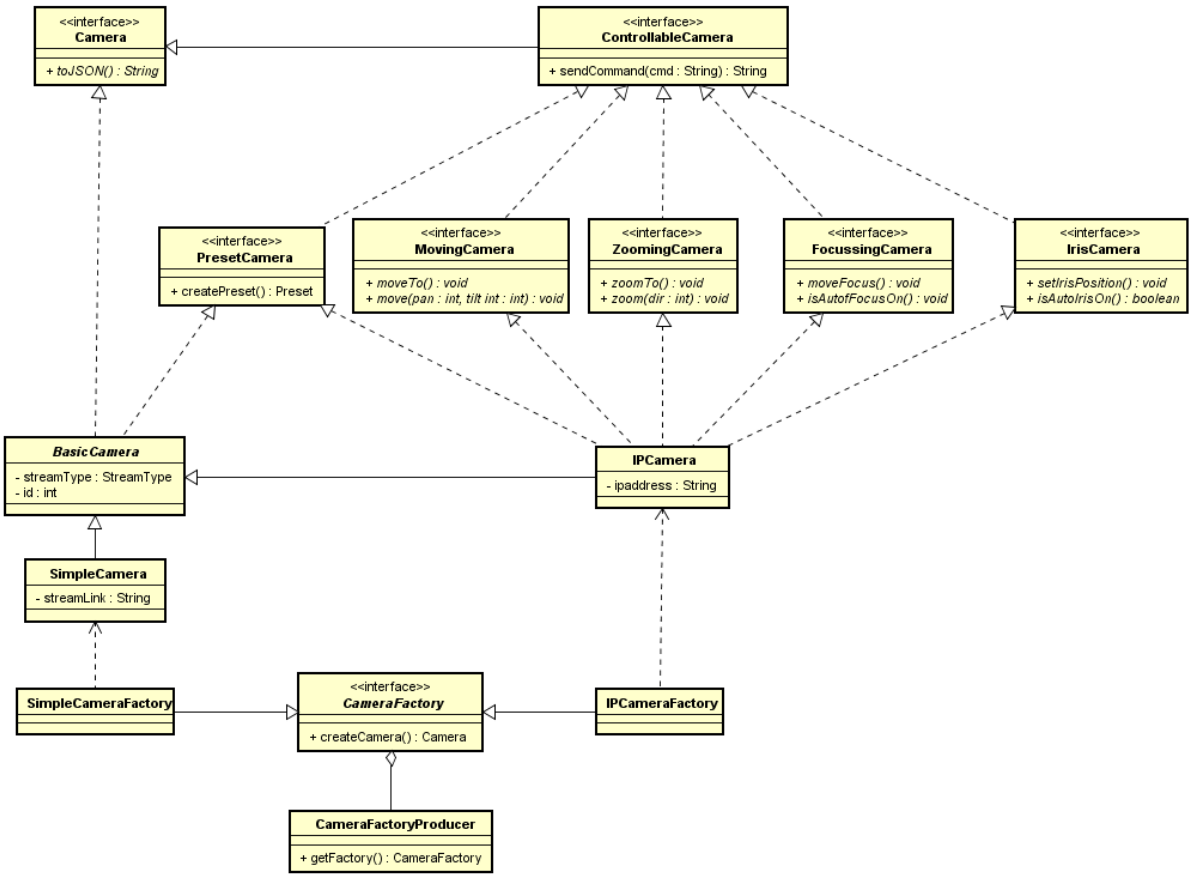


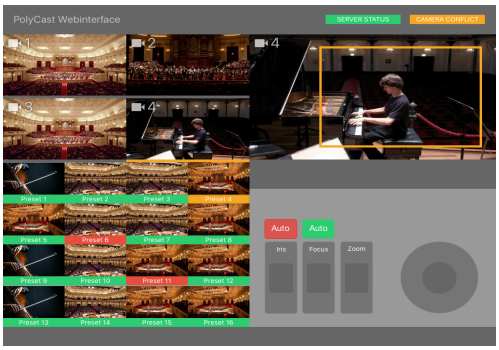
Figure 2: Camera UML Diagram

2.1.3 Clients view

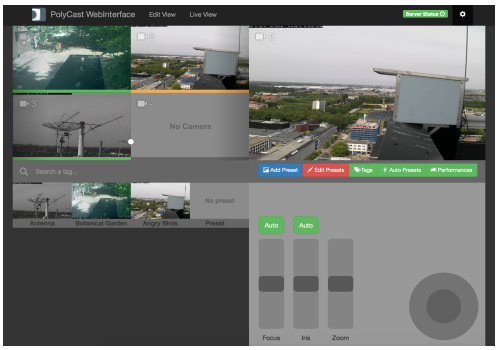
The only thing the user will see of the program, is the client view. The client view is a tablet (touch) controlled web interface primarily designed for Apple iPad, however it works on multiple tablet brands. The client view allows selection of camera's, selection of corresponding presets and control of a camera. It also provides the user with a live feed of all available cameras. This part is one of the most essential subsystems. If it fails, the user may not be able to use the system at all. It should be easy to use, and robust to failure. A global mockup of the user interface compared to the current state of the ui is given in Figure 3

2.2 Automatic preset creation

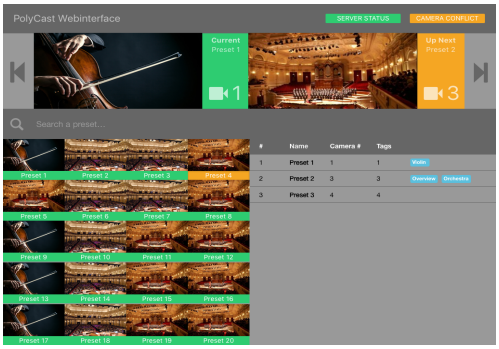
Creating presets can be a labour intensive task which can of course be automated. The client sends a request to the backend server to get a list of rectangles indicating of what parts of the current scene presets will be made. The locations of these so called sub views can be tweaked using parameters in the http request. If the user agrees, another request is



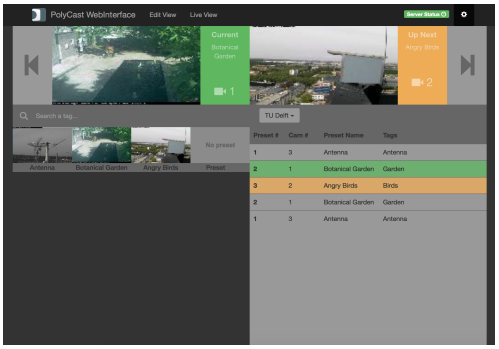
(a) Mock-Up Edit View Interface



(b) Current Edit View Interface



(c) Mock-Up Live View Interface



(d) Current Live View Interface

Figure 3: Mockup vs current user interface.

send to the backend server asking it to start creating the presets. The backend server uses an algorithm to convert the sub views to camera positions. These positions will be used to create presets with a corresponding preview image.

2.3 Queue of presets

PolyCast is currently working with scripts that state which presets follow after the current preset. A performance class creates this list of presets after which they are shown in the live view page of the web interface. In the frontend a list of the upcoming presets with their name, tag and camera number are shown as well as a view of the current and next preset. In the frontend the user can use a queue that they created as well as adapt it on the go.

2.4 Hardware/software mapping

The software can be used on every device that has a web browser. This makes it possible for the users to use any device they like for operating with the system. However, the web interface is mostly focused on fitting best on an iPad because this is what PolyCast is going to use. The camera's are able to receive IP commands from the java back-end. If in the future the camera's will be replaced, the only part of the system that also needs to be replaced is the camera module. This module takes care of the communication between the java back-end and the camera's.

2.5 Persistent data management

The data is stored in a database from which the back-end can retrieve the data using SQL queries. The database schema, is shown in Figure 4. In the camera table the camera's are stored with their id and mac-address. This way every camera can be uniquely identified. The presets are stored in the IPpreset or simplepreset table, depending on the preset type. These presets have a cameraID connecting to the camera the preset belongs to. All the presets are also stored in the preset table only showing the ID of the preset. These presets are connected with the tags where a preset can have multiple tags. There is also a queue table storing all the queues, with a presetList table combining the presets to the belonging queue.

The SQL-queries are all prepared in the code, preventing use of SQL injection from the user.

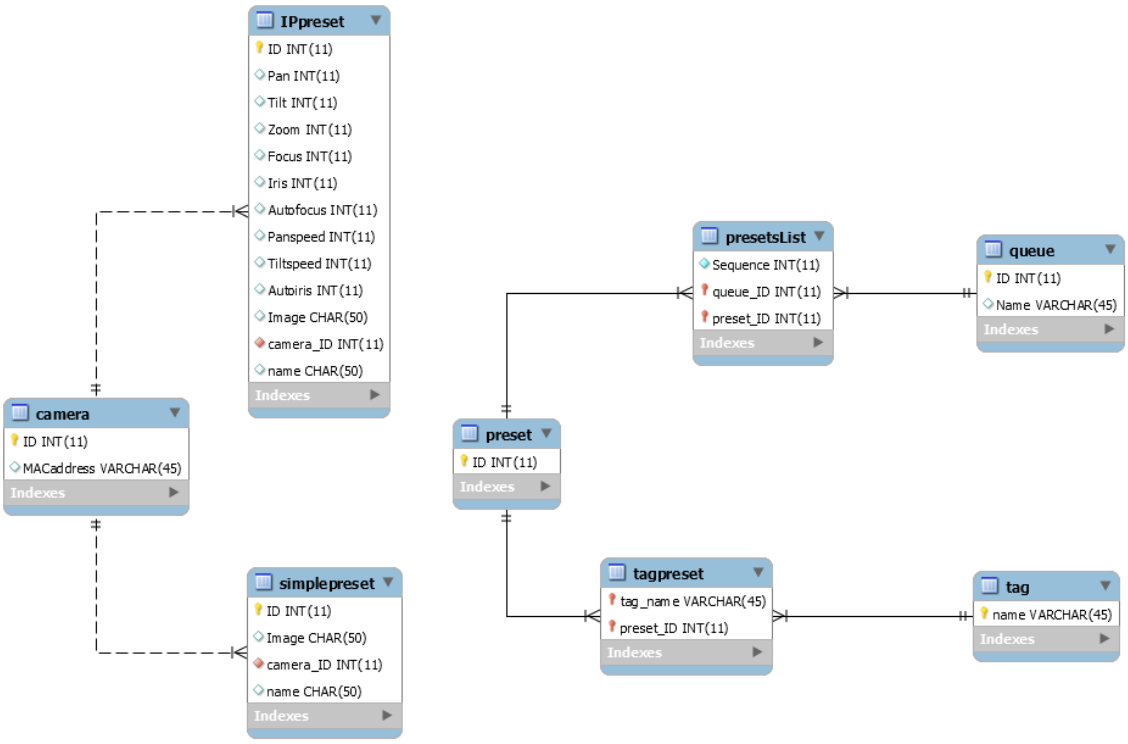


Figure 4: Database design

Glossary

CSS a style sheet language used for the presentation of an HTML document. 5

HTTP a protocol for exchanging information over a network. 5

HTTP requests messages send to a server to request some data. 5

IP commands commands send using the Internet protocol which are widely used to communicate between all kinds of systems. 8

Java a widely used object oriented programming language. 4

JavaScript programming language to perform HTML related tasks in the browser. 5

jQuery a JavaScript library that is used to manipulate HTML objects more easy. 5

NodeJS open source run-time environment for developing server-side web applications. 4

SQL a structured query language used for accessing databases. 5