Camera integration with UPnP to openHAB

Blanc Antoine, Law Christopher
RICM4

Table of contents

Introduction

OpenHAB is an open source application which allows devices to be controlled through a dedicated interface. In order to be up to date without limits, it is maintained by a community of users and is mainly coded with Java which allows a great compatibility with a lot of devices.

For this project, our main goals is to contribute to openHAB and to implement a motion detection with the provided cameras.

Tools

2 cameras from D-Link model that both support UPnP protocol:

. DCS-5222L

Features: pan and tilt

Output: MPEG for video and JPEG for image

. DCS-932L

Output: MPEG for video and JPEG for image

openHAB application which provides the interface and a design to produce our smarthome.

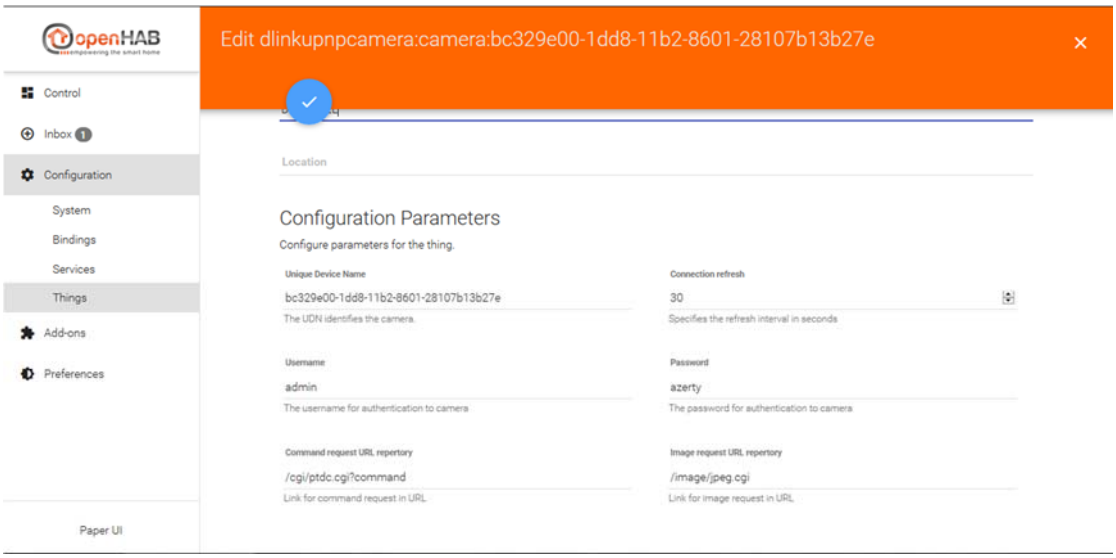
Eclipse smarthome framework which contains a lot of features and modules to design a Smart Home solution according to our needs.

D-Link UPnP camera binding

In order to contribute to openHAB with a new binding, we decided at first to use the code made by the last year team. However, when contributing to an open source, it's needed to follow a coding guideline which is in our case the Java coding guideline. Moreover, it's also needed to follow the design given by openHAB. That's why, we decided to recreate a new binding, with the help of the last year project. We used some codes from their binding but we changed a lot of parts to make an easier contribution and fit to the openHAB design.

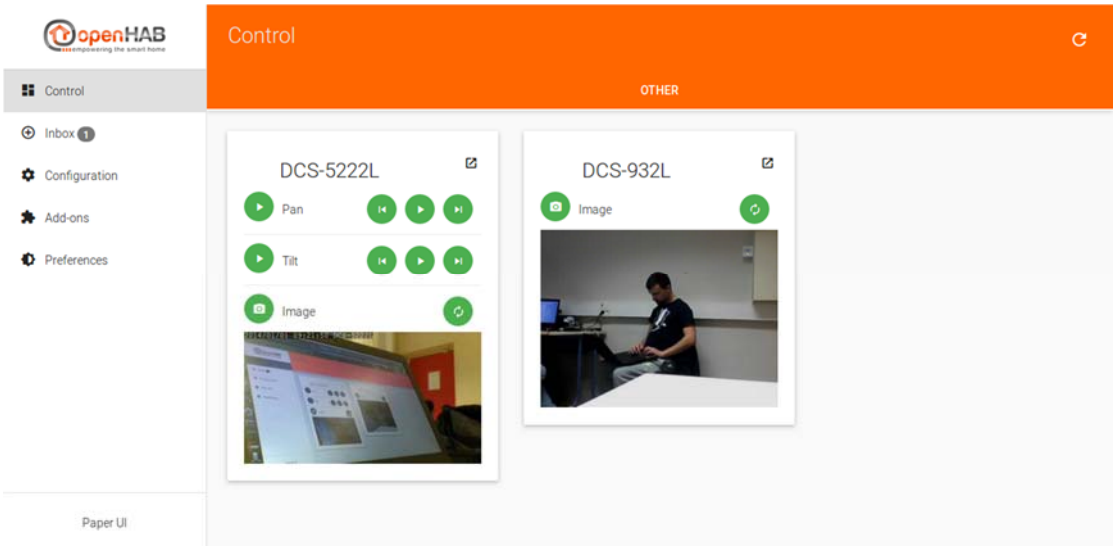
In order to recreate a binding, we generated a skeleton provided by openHAB through the IDE Eclipse. Once created, it's time to read a lot of documentations and examples from other bindings. Indeed, even if reading documentations is a start, we need to see how it works when it's working. So we tried using the demo provided by openHAB when it's installed and the binding created last year.

To implement the discovery, we used the Eclipse smarthome documentation and the other bindings. Once the cameras discovered, they are added to openHAB as a Thing, which represents an entity. Then, a Thing can be added to the inbox order to configure their parameters. In our case, a camera needs to configure parameters such as the username or the password. The other parameters are defined by default because they can be in common with the parameters of other devices. Besides, the UDN (Unique Device Name) is provided when it is discovered on the local network.



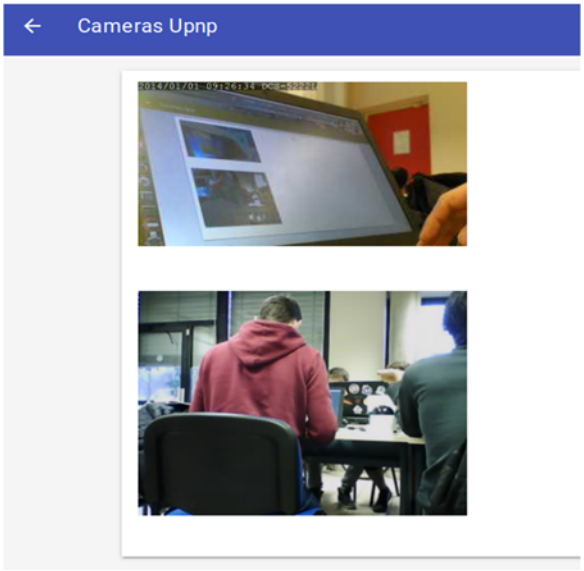
Configuration panel for the camera

When they are configured, the cameras can be controlled in the control panel in the Paper UI interface. Cameras with a pan and tilt control have buttons to control them remotely. Finally, each camera provides a static image that can be refresh. It is automatically refreshed according to a refresh parameter set in the camera configuration. In addition, during the refresh, we also check if the camera is alive and update the camera status.



openHAB interface with cameras control

Video from cameras can be displayed on the basic UI when the sitemap is defined with a webview.



Videos displayed from the cameras on the Basic UI interface

Once the binding was done, it was time to make a pull request to the GitHub repository.

Motion detection

Controlling remotely a camera is interesting, but make it detect motion is better. That's why, we decided to implement a motion detection for the cameras. In order to make it more relevant, we mainly used the camera model with PT (pan – tilt) control which is the DCS-5222L model. However, it still works with any other camera.

In order to implement the motion detection, we mainly used a binding of openCV in Java. openCV is a library of functions which are commonly used in image processing and real time computer vision.

To work, motion detection needs to be divided in several steps. We chose to implement the algorithm of subtraction as it is a common solution for motion detection. First of all, a video stream is needed as a support for the detection. Then each image is processed and compared with previous image. During this step, the image is switch to gray scaled image and filtered by a Gaussian filter to make some blurs and reduce the noise.

After this, we define a threshold to convert the colors to only white and black which is called binarization. Finally, we detect the outlines from the image and create rectangles to surround these outlines and make visible the areas with motion.

Once we implemented the algorithm for the motion detection, we needed to make use of it. Therefore, we chose to implement the camera tracking along with the motion detection. When a motion is detected, we had 2 choices: the camera follows the biggest area of motion or it considers all the detected motions which are still filtered in order to avoid noise in the detection.

For our demonstration, we decided to take the second choice because the first one can give random outputs if we have several people on the screen.

Encountered problems

During the project, we face several issues concerning the cameras. The first one was that in order to discover these cameras through the UPnP protocol, we couldn't use a network that doesn't allow multicast. That's is why we had to use our personal local network to get an access to the cameras with UPnP.

The other one is the configuration of cameras because they can only be connected to the network with a software that only supports Windows and Mac. However, as a Linux user, a tool like Wine can be used to install Windows supported software.

Then, we encountered problems with the installation of openHAB. For instance, we used at first openHAB through the terminal with Karaf. However, even if it was a complete version, setting up the different parameters and installing personal binding wasn't easy. Moreover, to test the binding, it was needed to create a jar every time something was changed. That's why, we decided to install openHAB through Eclipse which was at the end more convenient because new code could be tested right away. However, installing openHAB with Eclipse isn't clean because it always come with some errors in the build. Therefore, we needed to fix these errors before trying anything new.

Concerning openHAB, it's not easy to handle the coding way at first because the documentation isn't clear and divided. However it can be explained by the fact that they recently switched to openHAB2 which still use the first version but with some improvement and the use of Eclipse smarthome.

Nevertheless, after several weeks of understanding, the coding way became more natural.

Improvements

Currently, the motion detection only activate a tracking with cameras which support PTZ (pan, tilt, zoom) controls. However, to go further, we can implement a monitoring system where every time a motion is detected, we send an e-mail to the user.

Another improvement is the integration of the motion detection into openHAB but problems with the library integration have to be dealt with. Then, currently, even if our motion detection is working we can still improve it notably on the precision of the detection but it will need more complex algorithms.

Finally, we can still improve the binding as it is our first binding for openHAB.

Conclusion

Making this project wasn't easy but we learned a lot about open source project and how they work. Trying to contribute to an open source isn't easy as it requires to learn how to code in the same way as the project we want to contribute and read a lot of documentation. However, participate to an open-source community give a rewarding feeling as if we achieved something. We learned a lot thanks to the help of the community and contributing to other open source project might be interesting.