VRIJE UNIVERSITEIT BRUSSEL

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in de Toegepaste Informatica

# CastJS: A Web-based Context-aware Ambient Media Player System

**JOERI BOONEN**

**Academic year 2016 - 2017**

Promoter: Prof. Dr. Beat Signer
Advisor: Prof. Dr. Beat Signer
Science and Bio-engineering Sciences

VRIJE UNIVERSITEIT BRUSSEL

Masterproef ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Toegepaste Informatica

# CastJS: A Web-based Context-aware Ambient Media Player System

**JOERI BOONEN**

**Academiejaar 2016 - 2017**

Promotor: Prof. Dr. Beat Signer
Begeleider: Prof. Dr. Beat Signer
Wetenschappen en Bio-ingenieurswetenschappen

# Abstract

Today's digital signage systems have a number of shortcomings that we will address in this thesis. Each vendor provides a system that is only compatible with their hardware. Therefore the software is not portable to other platforms which binds a customer entirely to their signage vendor. Secondly, the signage systems are very closed; they support a number of fixed media types and cannot be expanded to introduce other types of media. A third problem in existing digital signage solutions is that it is often difficult to display media content that is interesting to everyone in the audience. The signage administrator creates a playlist with media objects (of built-in types) which is then played by media player software in a configured sequence. This process is repeated until the administrator manipulates the content. We evaluated a number of systems, both academic and commercial, that automatically change the displayed content based on contextual information. We found that they also have a common shortcoming; they are all built for a specific purpose and can only be used in a limited number of use cases.

We address these shortcomings by developing our own media player system, called CastJS, that is completely web-based (including the media player software), which makes it fully portable. Our system supports both static and context-based content, which can be influenced by the system administrator as well as the users. Users have their profile in which they indicate their interests and the types of content they wish to see on nearby displays. Interest keywords are semantically disambiguated with the help of DBPedia. New media types may be introduced in the system by means of plugins that generate media objects. The system renders the generated plugin output in the media player software.

Arbitrary context information may be submitted to the system by any type of sensor. Our software will update the context of linked media player groups and generate media items that are interesting to the audience that is present in the current context. The built-in rule engine drives the selection of appropriate content; support for external rule engines is equally provided. Media players in the environment may interact with each other so that the content will be distributed among multiple displays, hence interesting items are played for everyone in the audience. Displays may even split in separate zones so that multiple items may be played simultaneously by the same player. Media players may also play predefined content that the administrator designs in the content management system.

The features that we have provided — hardware-independence, configurable context-awareness and extensibility with plugins — make CastJS an open system that may be extended and configured for any use case.

# Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

# Acknowledgements

I would first like to thank my thesis promoter, professor Beat Signer of the WISE research group at Vrije Universiteit Brussel. Professor Signer is a man that radiates great passion for computer science, which has inspired me to contact him for promoting this thesis in the first place. I thank him for giving me the necessary inspiration to expand my ideas into a thesis-worthy concept, guiding me in writing the software and this text and providing constructive feedback on my progress. I equally thank the researchers at the WISE lab for their feedback on my intermediate thesis presentations.

Next, I thank my employer EASI[1], and especially Christophe Verhaeghe, Johan Tournet, Jean-Michel Block and Geert Van de Steen. EASI has given me an enormous amount of flexibility in terms of time management and work planning, which has helped me tremendously in completing not only this thesis, but the four years that I spent at Vrije Universiteit Brussel as a working student.

Last, but certainly not least, I thank my wife Sanne Adriaens for supporting me throughout the complete programme, helping me through some stressful times and coping with having a husband who combines a full-time job with a four-year academic education. I know that it has not been easy, but I am confident that our efforts will positively impact our future.

[1] `http://www.easi.net`

# Contents

# 1 Introduction

Digital signage is a fast-growing concept around the globe. Companies are rapidly replacing their traditional signage with a digital version. For an example, we can look at McDonald's. In the early days, the hamburger menu in the restaurants was displayed on fixed billboards made out of paper or carton, or slightly more adaptable wooden billboards in which you could slide panels with information. Of course, the carton boards would have to be reproduced completely when the menu changes. The wooden billboard has a fixed number of slots, which is an issue when the number of choices in the menu grows. Nowadays, the menu is displayed on television screens. These displays still have a fixed size, but the content can be adapted completely so that the menu and temporary promotions can be fit perfectly on the available screen estate. The content can be adapted in a content management system and pushed to the displays. All this is made possible by digital signage. Other use cases of digital signage include timetables in a train station, advertisement billboards in retail stores and information bulletin boards in companies.

With a worldwide revenue of more than 17 billion USD in 2016, the digital signage market is still expected to grow in the coming years. Information Handling Services[1] estimates the revenue to surpass the 20 billion USD mark

[1] `https://www.ihs.com`

in 2018, as illustrated in Figure 1.1. According to the MarketsandMarkets[2] forecast, the digital signage market is expected to reach USD 27.34 billion by 2022[3]. Market leaders in digital signage systems include Scala, Cisco, Intel, Samsung and Philips.

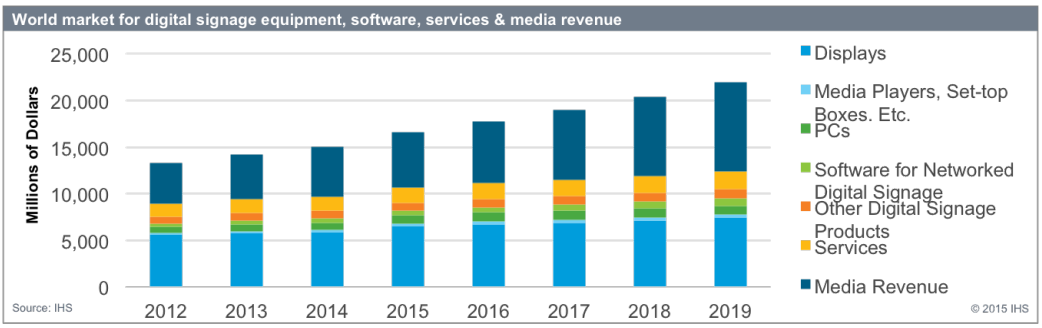



Figure 1.1: Digital signage market revenue forecast[4]

The content that is pushed to displays via a content management system can be classified as dynamic, as the platform administrator can dynamically adapt playlists and push updated content at any time. Some systems also include a form of dynamic content, like a graph that shows the trend of a specific share on the stock market that is updated periodically. However, the digital signage system can be made even more dynamic by adapting the content based on the context of the environment where it is displayed.

There has been some research in the past about making a digital signage system context-aware. Also commercial companies exist that include context-based content in their system, like DigitalDM. In this thesis, we will evaluate existing systems and research on ambient digital signage and develop a context-aware ambient media player system.

## 1.1 Problem Statement

Digital signage systems are often not designed to be compatible with devices from other brands. For example in the case of Philips, the signage software only works with specific types of Philips televisions. This is also the case with other hardware vendors. Scala[5] has another approach to digital signage, as

[2] `http://www.marketsandmarkets.com`

[3] `http://www.marketsandmarkets.com/Market-Reports/digital-signage-market-513.html`

[4] `http://news.ihsmarkit.com/press-release/technology/digital-signage-revenue-grow-10-percent-2015-ihs-says`

[5] `http://scala.com`

they do not focus on selling screens, but software and media players. However one still needs a platform that supports the Scala software to be able to run the system.

Both approaches entail a certain form of customer lock-in, to be able to generate more revenue from multiple aspects of the digital signage solution. Therefore, today's digital signage systems lack the flexibility of being platform-independent. In addition, the content offered in these systems is also fairly static. Administrators can create playlists with a fixed list of built-in media types to be displayed on a screen. The administrator needs to take the intended audience into account while creating content; however it is almost impossible to create a playlist that will be of equal interest to everybody in the audience.

A number of context-aware digital signage systems exists that aim to automatically adapt the content that is displayed, based on the interests of the audience and the current context in which the audience and display are situated. However, we found that these systems are built for their own specific purpose, lacking flexibility in creating custom types of content. The systems we found also use a fixed set of parameters in their context model, again lacking the option to create contexts that fit into any use case.

## 1.2 Objectives and Research Questions

The objective of this thesis is to create a digital signage solution that is completely platform independent. This will avoid customer lock-in by vendors and provide more flexibility to customers in terms of choosing the hardware they wish to work with. The platform should be web-based and run natively in a web browser. The software should have generic interfaces so that new media types can be introduced by means of custom plugins.

Other than software customisability, also the playlist content should be customised to the people that are near a specific screen. Therefore it should be possible to add media content with general parameters so that it can adapt to person-specific requirements and properties. The system should be able to collect any sort of context parameters, contributed by any type of sensor.

While developing the proposed system, we plan to answer the following questions:

- To what extent is it possible to create a digital signage platform that is purely web-based? What are the advantages and drawbacks?
- Is it possible to create a digital signage system that is usable in any possible use case?
- How accurately can the media content be personalised to the audience in a specific area?

## 1.3 Research Methodology

We used the Design Science Research Methodology for Information Systems Research [28] to structure our work. This methodology proposes six activities that should be completed:

1. Problem identification and motivation: Chapter 1 introduces and motivates our research.
2. Define the objectives for a solution: Based on our literature study in Chapter 2, we defined the objectives for our solution.
3. Design and development: Chapters 3, 4 and 5 explain the design and implementation of our solution.
4. Demonstration: We demonstrated our solution by means of a proof of concept implementation in the WISE lab, as well as a live scenario for a user evaluation.
5. Evaluation: We evaluated our solution in comparison with other systems in Chapter 6. Furthermore, we have conducted a user evaluation based on a live scenario followed by a questionnaire.
6. Communication: This thesis contains extensive documentation about the problem and our solution. We will present our work during a thesis defence.

## 1.4 Contributions

We make a number of contributions that we have not encountered in the literature:

- Web-based digital signage: A digital signage solution that is completely hardware-independent as a result of the web-based implementation.
- Interests disambiguation based on semantic web technologies: We use the public SPARQL endpoint of DBPedia to disambiguate user interests by scanning the available semantic resources.
- Multiple-media-player support: While playing context-aware media content, multiple media players in the same media player group will cooperate to display media content and take full advantage of the available displays in a specific area.
- Plugins: Anyone may introduce new media types into the system by developing a custom plugin that is executed by the content management system.

## 1.5 Thesis Structure

We start with a review of the literature related to digital signage, context-aware systems and useful technologies and an evaluation of various context-aware digital signage systems in Chapter 2. Then we explain our solution for the problem in Chapter 3, followed by the technical implementation details in Chapter 4. Chapter 5 explains how the system may be used. Finally, we end with conclusions and future work in Chapter 6.

# 2 Related Work

## 2.1 Digital Signage

Multiple handbooks exist on digital signage, written by experts in the field. Schaeffler [31] gives an overview of content types that can be used in playlists on a digital display. Table 2.1 illustrates these different types. Any content is always either real-time, near-real-time of non-real-time. Table 2.2 explains these properties.

| **Content Type** | |
|---|---|
| Stills | Photos or artwork |
| Animations | Moving pictures |
| Video | Video content in digital signage is not equal to regular television in households |
| Data | News or stock information |
| Audio | Not very common in digital signage, audio must match the visual content |
| Miscellaneous | Text, logos and other objects |

Table 2.1: Content types in digital signage

Lundström [25] goes more in depth on the technical side of a digital signage system and the creation of playlists. The author explains about

transitions between content, like fading out or sliding. Playlists should also be able to be scheduled and it should be possible to specify how long a specific item needs to be visible. Also the concept of channels is introduced. They are similar to regular television channels, but they contain playlists for digital signage.

| **Content Property** | |
|---|---|
| Real-time | News broadcasts or live video streams |
| Near-real-time | Stock tickers, data that is stored briefly and is updated in a regular interval |
| Non-real-time | Content that is stored and is available for long term use, like photos |

Table 2.2: Content properties in digital signage

Kelsen [19] provides some details about the video media type. Videos have aspect ratios like 4:3 or 16:9. Depending on the aspect ratio of the screen that will display the video, the media may need some resizing. The different resizing methods are explained in Table 2.3. If we were to divide the screen into multiple zones or containers, we should try to size the container to the aspect ratio of any videos that are added to that container in the playlist.

| **Method** | |
|---|---|
| Letterboxing | Fitting a 16:9 video on a 4:3 screen by resizing the video to match the width of the screen |
| Pan and Scan | Fitting a 16:9 video on a 4:3 screen by cropping the video to the exact screen size |
| Pillarboxing | Fitting a 4:3 video on a 16:9 screen by matching the height of the screen |
| Stretching | Stretching a 4:3 video to 16:9 size |
| Zooming | Zooming in on a 4:3 video to match the height of the 16:9 screen |

Table 2.3: Content resizing methods in digital signage

Kim & Kim [20] explain the implementation of a digital signage platform. The system uses an Update Checker functionality on the media players; they are constantly polling the signage server to check if new content is available. We wish to avoid this behaviour in our system, as other technologies currently exist that allow us to omit any polling overhead. The authors also link a

schedule to a specific media player. However, the schedules could be created separately and then linked to multiple players.

Dupin & Adolph [13] take a next step in digital signage systems by describing biometric recognition of consumers as a hot topic. Tracking of heat paths can show a consumer's movement in a store. Gaze tracking can be used to identify the area where a consumer looks at most. This information can be priceless to marketeers. The authors also mention the SMIL standard, which is created and maintained by the World Wide Web Consortium (W3C), for describing playlists.

## 2.2 Context-aware Digital Signage

Context-aware systems that show personalised information to the audience are a next step in digital signage [12]. To gain a better understanding of context-aware digital signage, we must elaborate on the notion of context. A context is a multidimensional construct that describes our spatial-temporal environment [17, 18]. However, context data may be incomplete or imprecise due to sensor limitations and the data of different sensors must be carefully fused, as multiple sensors may provide data for overlapping dimensions [11]. Context-aware systems then use this context information to adapt their own behaviour, without requiring explicit input from the user. Colman et al. [10] differentiate between context-aware systems and adaptive systems. Context-aware systems can adapt their context of use or task, while adaptive systems aim to keep their behaviour in line with a specific goal such as the value of a monitored variable. The system that we propose in this thesis has both properties, but leans more towards a context-aware system, as the adaptiveness lies only within the media player software.

Alegre, Augusto & Clark [1] describe the required steps for handling context information; acquisition of sensor input, modelling of the context, reasoning about the context information and dissemination of the context and conclusions. These steps are reflected by the conceptual framework for designing context-aware systems in Figure 2.1, proposed by Baldauf, Dustdar & Rosenberg [4]. The bottom layer contains the sensors that contribute information. This information is retrieved in a raw form by layer two and passed to layer three for processing and adding the information into a context model. The current state of the context is stored by layer four and is then accessed by the application in layer five, which will adapt its behaviour based on the retrieved information. The authors also formally classify the different types of sensors as physical, virtual or logical. Physical sensors are the most used type; they are actual hardware that capture physical data, like

cameras. Virtual sensors are software applications and services, for example a calendar. Logical sensors combine both physical and virtual sensors to provide information that was already subjected to a low-level pre-processing.

| | |
|---|---|
| Layer 5 | Application |
| Layer 4 | Storage |
| Layer 3 | Preprocessing |
| Layer 2 | Raw data retrieval |
| Layer 1 | Sensors |

Figure 2.1: Layered model of a conceptual framework for context-aware systems (based on [1])

A number of context-aware digital signage systems exist, mostly in an academic context, but commercial systems exist as well. In this section, we will compare a selection of these systems. We will not focus on usability or maintainability, but evaluate the core features of the systems in terms of adaptability. Adaptability may be approached on different levels. First, we determine for which use cases a system may be suited and which means are provided to adapt the system when a use case is expanded or changed. Second, we evaluate how a system may adapt itself in different situations.

We have selected the following criteria to structure our evaluation:

- Context Awareness: Which properties does the system include in the context model?
- Configurability: How deeply can an administrator or end user configure the system and influence the content that will be displayed in the media players?
- Content: Which types of content can the system handle?
- Portability: On which hardware, operating system or framework does the media player software depend?

### 2.2.1 AwareNews

AwareNews [6] is a research project by the Institute for Information Systems at ETH Zurich. They propose a system for reactive information environments, called SOPHIE. SOPHIE's architecture consists of sensor inputs to monitor the environment, user interface components, a context engine that

processes the sensor inputs, a content publishing service and a database. Figure 2.2 illustrates this architecture.

AwareNews is a concrete instance of the SOPHIE platform, where the database contains different news articles. Specific news articles are displayed in the mother language and within the interest profiles of persons within the current context. The database therefore contains profiles for different users including their interests and preferred language.

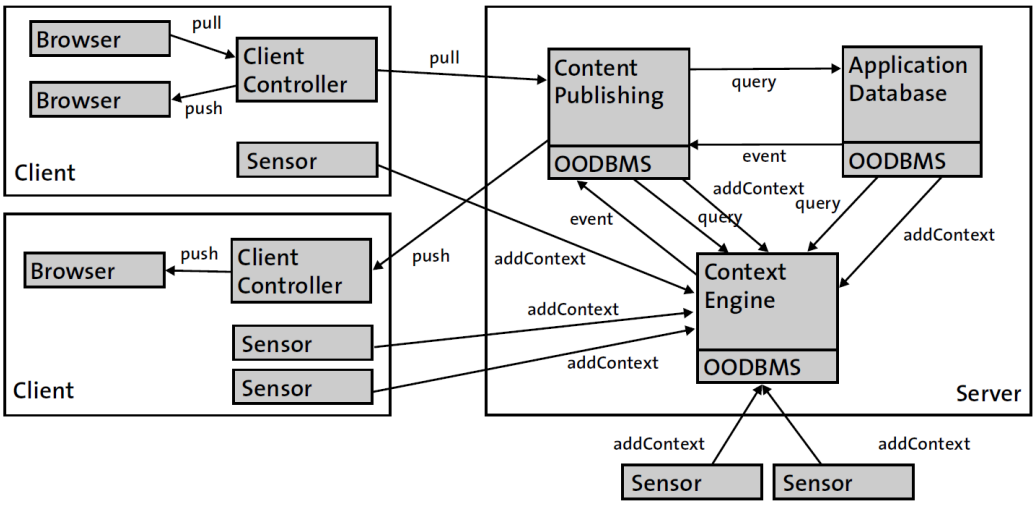



Figure 2.2: SOPHIE architecture [6]

The context entails the identity of all present users. These identities are reported to the server by cameras. End users can specify in their profile which subjects they find interesting by adding them as keywords. The generated media can only be played one by one in fullscreen with a fixed interval of 30 seconds. The content is mostly text-based with basic images. Therefore, AwareNews is not able to play multi-media content like videos. The AwareNews media player runs in a web browser and is therefore fully portable.

AwareNews is able to play four types of information:

- News: Headlines with a short description of the article.
- Events: Information about events like conferences and meetings.
- Activities: Information about a person's activities, like ongoing research.
- Memos: A Memo is a free-text media type and can contain any kind of information like reminders and announcements.

### 2.2.2 Context-aware Hospital Information System

The Context-aware Hospital Information System (CHIS) [16] was originally designed as an application for hand-held clients. However, the authors have thought about use cases for public displays. CHIS is a context-aware system that shows hospital related information to the hospital's staff.

The context entails the identity of the present hospital staff, including their location and function. This information is reported by the users' hand-held devices. The configuration of the system is based on the function of each user and not on an individual user basis. CHIS plays hospital-related information like x-ray results and patient records. The system can also show a floor map that indicates the locations of colleagues and the location of patients whose status has recently been updated. We were unable to find out which framework was used to develop the system, hence we cannot evaluate the portability.

### 2.2.3 PriCal

PriCal [32] is a context-aware calendar display system with a strong focus on privacy. The context entails the identity of all present users. A collection of infrared sensors detects how many users, if any, are near a specific screen. The nearest wireless access point is then monitored to determine which user's mobile device maintains the strongest WiFi signal with that access point, based on the Received Signal Strength Indicator (RSSI) value of the connections. Users have a profile in which they can configure their privacy settings and create rules indicating when specific types of calendar items may or may not be displayed. The system is specifically designed to only display calendar items in a paper-based wall calendar layout. The display agent has been implemented on a Raspberry Pi[1], but the used development framework was not specified, hence we cannot evaluate the portability.

### 2.2.4 DigitalDM iDetect

DigitalDM is a commercial company that delivers interactive digital solutions to retail stores, like digital mirrors and interactive changing room screens. In this section, we are evaluating the AssistMe Virtual Mannequin product[2] in combination with the optional iDetect module.

[1] `https://www.raspberrypi.org`
[2] `http://www.theendlessaisle.com/virtual-mannequin`
[3] Retrieved from `http://www.theendlessaisle.com`


Figure 2.3: DigitalDM Virtual Mannequin[3]

The Virtual Mannequin is a pre-recorded virtual person who is able to give information about retail products. An implementation of the product in a shop window is shown in Figure 2.3. By default, the displays operate in a static way. A playlist is then looped infinitely. The iDetect module is able to do a headcount in a specific area and determine the majority of present persons of the same gender. Hence, this is the contextual information that is provided to the system. For each item, the administrator can indicate for which gender this item is relevant. The iDetect module will use this information to display the correct items in a specific gender-based context. The content is then adapted based on the largest number of people of the same gender that are present. For example, if there are more women than men in a clothing store, the promotional content on the displays will be mostly about women's clothes.

The Virtual Mannequin can be programmed to play pre-recorded messages and show pictures and videos of the retail items. The Virtual Mannequin itself does not have to be used, as the system may also be used as a regular digital signage system. To operate the Virtual Mannequin, specific DigitalDM hardware is required. This hardware is sold with the AssistMe software in one package.

### 2.2.5 Conclusion

Table 2.4 contains an overview of our evaluation of the previous systems, based on the proposed criteria. For the purpose of completeness, we have also added traditional digital signage systems in the comparison.

| System | Context Awareness | Configurability | Content | Portability |
|---|---|---|---|---|
| Traditional signage | None | Media, playlists and schedules | Fixed content types | Vendor-specific hardware |
| AwareNews | Present users | User profiles, interests | News, events, activities and memos | Web-based, fully portable |
| CHIS | Present users | None | Hospital information | Unknown |
| PriCal | Present users | User profiles, privacy settings | Calendars | Unknown |
| DigitalDM iDetect | Number of users, gender | Static playlists, gender-related media | Images, videos, virtual mannequin | Vendor-specific hardware |

Table 2.4: Evaluation of context-aware digital signage systems

From the systems that we discussed in the previous sections, we conclude that all of them have been designed with a specific goal and therefore also offer specific content and a specific configuration for these content types. For instance, PriCal specifically displays calendar items and CHIS is designed for displaying medical information. In terms of context awareness, the parameters that are included in the context model are also fixed. The system can only handle these predefined parameters. In commercial digital signage systems, we see that each vendor has their own specific hardware that is required to run the system. This is no different with the system of DigitalDM and thus the customer is completely dependent on this vendor's hardware. In AwareNews, the media player was implemented in a web browser, which means that the media player can run on any underlying operating system or hardware.

To improve the evaluated criteria, we need a system that is more open and usable in any use case. We cannot foresee all possible use cases directly in the system, but the adaptability must be improved so that it is feasible to extend the system with new media types and configuration possibilities. To avoid the hardware dependence that we noticed in commercial systems, we will adopt the approach of AwareNews by designing a web-based media player and provide a way to structure playlists that a web browser can play. In order to improve the adaptability in terms of context-awareness, we need user profiles with accurate interests management, as well as the means to provide context information. The following technical sections explain technologies that aid us in designing this system.

## 2.3 Technologies

### 2.3.1 SMIL

#### The SMIL Standard

The Synchronized Multimedia Integration Language (SMIL) is an XML-based standard by the W3C to describe a playlist or schedule in a single- or multi-zone screen layout. It supports different media types, scheduling and transitions. Note that the SMIL workgroup was discontinued in April 2012 and the last version of the standard, SMIL 3.0 [8], was published in December 2008.

#### SMIL Structure

A SMIL document has a basic structure with a `head` and `body` element within a SMIL `root` element as illustrated in Listing 2.1.

```
<smil>
  <head>
    <!--head content-->
  </head>
  <body>
    <!--body content-->
  </body>
</smil>
```

Listing 2.1: SMIL document structure

The `head` element contains information about the screen layout. This can, for example, be in the form of a `layout` element containing a `root-layout`

element with an absolute size in pixels with two `region` elements of a relative size. Listing 2.2 provides an example of a `layout` element.

```
<layout>
  <root-layout id="SMIL-" width="1280" height="720"/>
  <region id="top-container" width="100%" height="50%"/>
  <region id="bottom-container" width="100%" top="50%"/>
</layout>
```

Listing 2.2: SMIL `layout` element

In the `body` element, the actual media content is specified. Table 2.5 provides the SMIL elements that can be used to reference content.

| **Element** | |
|---|---|
| `ref` | A generic media element, the player that loads the SMIL file will try to play this media element if it has the means to play that content type |
| `animation` | Animated vector graphics |
| `audio` | An audio clip |
| `img` | A still image |
| `text` | An external text reference |
| `textstream` | A text document that includes timing information for time-dependent rendering of portions of the document content |
| `video` | A video clip |

Table 2.5: SMIL media reference elements

Any `media` element should contain at least three attributes, namely `src`, `type` and `region`. These attributes are explained in Table 2.6. Elements may also contain arbitrary `param` elements for specifying custom parameters to pass to the player software.

| **Attribute** | |
|---|---|
| `src` | The URI of the media element |
| `type` | The content type of the object that is referred to by the `src` attribute |
| `region` | The ID of any `region` element that is specified in the `head` of the SMIL document |

Table 2.6: SMIL media attributes

Of course, `media` elements can contain much more attributes to specify, for example, duration and size. The example in Listing 2.3 is a detailed representation of a video. This video will be played starting from 5 seconds into the video for a duration of 30 seconds. This process will be repeated twice.

```
<video
  src="promo.mp4"
  type="video/mp4"
  region="top-container"
  begin="5s"
  dur="30s"
  repeat="2"
  alt="Promo video of our company"
  title="Company Promo"
/>
```

Listing 2.3: SMIL `video` element

Media elements can be played in a sequence or in parallel, using the `seq` and `par` elements respectively. For example, Listing 2.4 represents a sequence with two groups of elements that are played in parallel.

```
<seq>
  <par>
    <video src="promo.mp4" type="video/mp4"
           region="top-container"/>
    <img src="logo.jpg" type="image/jpg"
         region="bottom-container"/>
  </par>
  <par>
    <video src="xmas-video.mp4" type="video/mp4"
           region="top-container"/>
    <img src="snow.gif" type="image/gif"
         region="bottom-container"/>
  </par>
</seq>
```

Listing 2.4: SMIL `seq` and `par` elements

### Support for SMIL

SMIL is supported by a number of well-known media players like QuickTime[4] and RealPlayer[5]. Another example is AMBULANT[6], a player that is developed specifically for SMIL. SMIL also found its way to the world of digital signage, where it was introduced as an open standard[7]. However, as we designed a web-based application, we avoided the use of any media player software or browser extensions. When looking at web browsers, support for SMIL is actually quite poor. Microsoft does not have plans to add support for SMIL in Internet Explorer 11 or in the Edge browser[8]. Google intended to deprecate SMIL in the Chrome browser in favour of CSS and web animations in April 2015[9], however, they revisited this intent in August 2016[10] as they believe that CSS is not ready yet to completely take over all SMIL functionality. On the other hand, proposals will be made to already remove support for some parts of SMIL that are not widely used. In browsers that use the Gecko 2.0 rendering engine, SMIL is still supported for animating SVG elements[11].

## 2.3.2 DBPedia

DBPedia[12] is a project by the universities of Leipzig and Mannheim [23]. Using the Resource Description Framework[13] (RDF), DBPedia provides a semantically annotated version of the complete content of Wikipedia. Each Wikipedia page is described as a resource in the DBPedia ontology.

DBPedia provides a SPARQL[14] endpoint to access roughly three billion semantic relations between objects [7]. As we are building an ambient system in which a user's interests are important, we can use the semantics of DBPedia to disambiguate keywords and relate to actual DBPedia resources.

[5] http://www.real.com/realplayer
[6] http://ambulantplayer.org
[7] http://www.a-smil.org
[8] https://developer.microsoft.com/en-us/microsoft-edge/platform/status/svgsmilanimation/?filter=f3e0000bf&search=smil
[9] https://groups.google.com/a/chromium.org/d/msg/blink-dev/5o0yiO440LM/59rZqirUQNwJ
[10] https://groups.google.com/a/chromium.org/d/msg/blink-dev/5o0yiO440LM/YGEJBsjUAwAJ
[11] https://developer.mozilla.org/en-US/docs/Web/SVG/SVG_animation_with_SMIL
[12] http://wiki.dbpedia.org
[13] https://www.w3.org/RDF
[14] https://www.w3.org/TR/sparql11-overview

### 2.3.3 Person Identification

There are numerous ways to identify a person, like using any form of ID cards, RFID chips or fingerprints. However, if any of these approaches were to be used in a context-aware system, a user would have to explicitly make themselves known or would have to make sure that they are carrying their means of identification. Ideally the user should not be concerned with this. Facial recognition or any other form of biometrics recognition can be used to identify users without depending on additional hardware that must be carried by the user.

#### Low-cost Open Source Solution

Computer vision software has proven to be an effective asset in intelligent digital signage [5] when provided with good accuracy [15]. OpenBR[15] is an open-source biometric recognition library. The library can be used to recognise faces, estimate age, evaluate gender and perform other biometric evaluations [21]. OpenBR itself uses another library called OpenCV[16] for detecting faces in an image. OpenCV is an open-source computer vision library that can be configured to recognise specific types of objects or patterns in an image, including human faces [29].

Finally, The Motion daemon[17] is a software motion detector that can be configured to record images. These images could then be fed to OpenBR to recognise and identify any familiar faces. Thanks to the open source software we discussed in this section, the only hardware we require is a simple webcam, which keeps the cost of the solution fairly low.

#### Commercial Solution

The Microsoft Kinect[18] is a motion sensing input device that was originally developed for the Xbox, Microsoft's gaming console, but was quickly provided with development APIs for the Windows platform as well[19]. The Kinect provides a very rich functionality for a relatively low price.

The sensor is able to locate up to six people in a room and track their faces and skeletons. It provides different input streams like colour, heat, infrared and depth images and it can generate 3D meshes of objects and people. The abilities of the Kinect have been explored to identify persons, not only

[15] `http://openbiometrics.org`
[16] `http://opencv.org`
[17] `http://www.lavrsen.dk/foswiki/bin/view/Motion/WebHome`
[18] `https://developer.microsoft.com/en-us/windows/kinect`
[19] `https://developer.microsoft.com/en-us/windows/kinect/develop`

through face detection, but also skeleton and gait analysis [2, 3, 30, 33]. Although the Kinect is a very feature-rich device that can locate faces, it has no means of identifying these faces. Therefore this solution would still require a library like OpenBR to actually link faces to identities. Once the identity information is acquired, the Kinect can link that information to the face and track the face until the person leaves the picture. Therefore a person has to be identified only once. In the open source solution, we have to identify everyone in the picture repeatedly.

# 3 Solution

## 3.1 Requirements

The main purpose of our solution is to address the missing functionalities that we discovered in our evaluation of existing systems in Section 2.2. First, we must design a media player that is completely hardware-independent. We decided to adopt the approach of AwareNews, which uses a web-based media player that runs in a web browser. AwareNews was developed in 2005. Web technologies have definitely improved since then, which will help us to implement a media player that is not limited to textual or basic visual content, unlike AwareNews. The media player should be usable in a static way to play predefined playlists, as well as play content that is generated based on context information.

To send content to a media player, we need a way to structure playlists. We already discussed the SMIL standard in Section 2.3.1, which we used to inspire our own data structure. The administrator must be able to cast content to the player instantaneously, avoiding the need for the media player to regularly check for new content. Our system must be able to process any context parameter, so that it is not limited to a fixed set of variables. It must be feasible to introduce new context parameters and configure actions that must be executed based on their values.

The system must be usable for any use case. Even if the required content types for a specific use case are not yet available, it should be possible to extend the functionality of the system to support new types of content. It must be possible to describe the configuration that is accepted by media of a specific type in a structured manner.

## 3.2 Web-based Media Player

### 3.2.1 HTML5

HTML5 is the latest version of the Hypertext Markup Language by the W3C. This standard has the possibility to play different media types natively without the help of any browser plugins like Adobe Flash[1]. The language also provides a number of JavaScript APIs to provide extra functionality to a webpage. In combination with CSS3, different styling can be applied to any element in a HTML document. The combination of HTML5, JavaScript and CSS3 provides the necessary means to develop a media player that runs natively in any recent web browser that supports these technologies.

#### Tags

The audio and video tags in HTML5 allow to play some audio and video formats natively in the web browser. In our media player, we focussed especially on the video[2] tag. The video tag supports three different video formats, including MP4, WebM and OGG. However, as illustrated in Table 3.1, only the MP4 format is supported in all main browsers.

| **Browser** | **MP4** | **WebM** | **OGG** |
|---|---|---|---|
| Internet Explorer | Yes | No | No |
| Chrome | Yes | Yes | Yes |
| Firefox | Yes | Yes | Yes |
| Safari | Yes | No | No |
| Opera | Yes | Yes | Yes |

Table 3.1: Browser support for video formats in HTML5

[1] `http://www.adobe.com/nl/software/flash/about/`
[2] `http://www.w3schools.com/html/html5_video.asp`

#### APIs

We found that some HTML5 APIs provide essential functionality to the media player. We use the WebSocket API to maintain a permanent connection to the content management server. This allows the server to push content to the player, without the player having to poll the server frequently to search for new content. The server also knows whether a specific player is online or not, based on the fact that it has an active WebSocket connection.

HTML5 also provides the Fullscreen API, which allows a website to toggle fullscreen mode in the web browser. This API would in theory enable us to start a web browser in fullscreen, which is ideally the case if the browser is used as a media player. Unfortunately, we are unable to use the API for our purpose, as we explain in Section 6.2.4.

### 3.2.2 Media Types

#### Native Media Types

Our media player supports a number of native media types such as `image`, `video`, `youtube`, `vimeo` and `webpage`. These media types are played by a dedicated function within the media player code. They provide rather static content and should be used in static playlists that are manually pushed to media players.

#### Custom Media Types as Plugins

Next to the native media types, arbitrary custom media types can be integrated into the system as plugins. These plugins can be designed to deliver both static or dynamic content to the media players.

A description of the plugin in JSON Schema should be provided to the system, as we will explain in Section 3.4. Furthermore, the plugin is responsible for its own output in HTML format. The media player ensures that the output is loaded in a container on the screen. Section 3.6 explains how plugins can be used to provide personalised content for users that are close to a specific screen.

### 3.2.3 Player Modes

#### Static Mode

When a media player is started, it always initialises in static mode. The media player will request its default playlist from the content management

server. Section 4.2.1 explains how the playlist is processed and played. Other static playlists may be pushed to the media player by the administrator.

#### Context-aware Mode

Media players may also play content that is generated by a content management server, based on context information. The media players are divided into media player groups, to which the context information may be related. The context of a player group is updated by various sensors in a physical area, going from cameras to temperature sensors, infrared sensors or any other sensor that can make a worthy contribution. Section 3.7 explains how the context is managed. A media player may participate in only one group that is related to a specific context. Hence, a context may be related to multiple media player groups whose members do not overlap. Furthermore, not all player groups must be used in context-aware mode, as this mode may be disabled for groups that are only created for administrative purposes.

When a new context contribution is received from a sensor for a media player group whose members are currently in static mode, one master player is elected from the members of the group. This master player is responsible for managing the generated context-based media and for instructing the other players in the group — the slave players — which content they should play. In Section 4.2.2 we will explain the inner workings of a media player in context-aware mode in a master or slave role.

## 3.3 Structuring Playlists and Screen Layouts

### 3.3.1 JavaScript Object Notation for Playlists

As explained in Section 3.2, our media player solution is based on HTML5 and JavaScript. As JavaScript natively supports the JavaScript Object Notation (JSON) and does not have any native XML parsing capabilities, we have opted to create the new data structure in the JSON format, but inspired by SMIL. Although support for SMIL in web browsers is far from ideal, the language itself is very rich. Therefore we used the philosophy behind SMIL in our own data structure for defining playlists and layouts, but not in an XML-based language.

The data structure supports the notions of containers or sections on a screen and has a way of relating media objects to the different containers. It supports any type of media, native or custom. A playlist must always refer to at least one container. This container will then be stretched to fit the display. Otherwise, a playlists may refer to multiple containers, which will

all span a different section of the display. Figure 3.1 provides an example of a playlist with only one fullscreen container (left) and a playlist with three containers of different sizes (right).

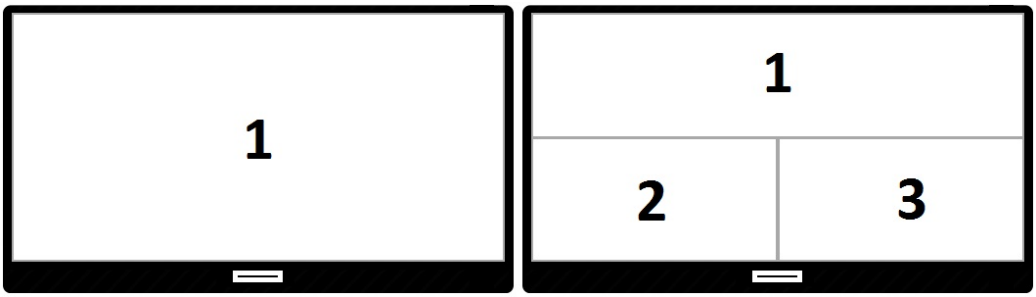



Figure 3.1: Containers as zones on a display

In our application, we make a difference between properties and options for media types. Properties are configured when a media object is added to the system and they are valid for all use cases of the object. Options are set when a media object is added to a playlist. For example, an image object should always have a source, hence that is a property. However, when the image is added to a first playlist, the image might have to be stretched to fit the container, while in another playlist it should not be stretched. Therefore, stretching the image is an option. Furthermore, the structure supports nesting of both playlists and containers.

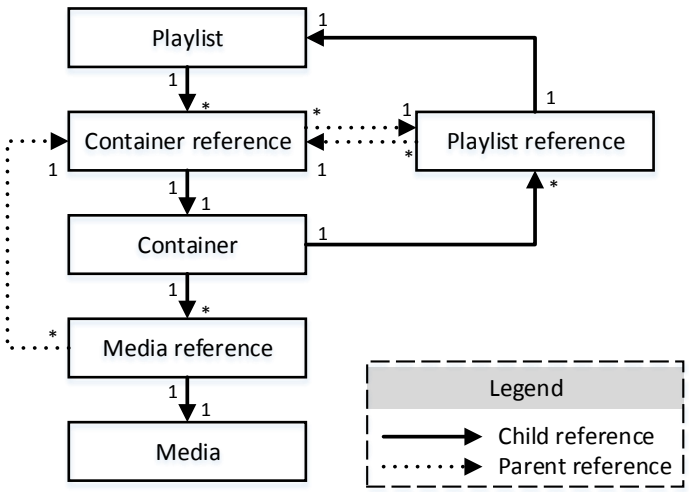



Figure 3.2: Playlist data structure model

Figure 3.2 explains the model of our playlist data structure. A playlist contains one or more container references; these have no added value at the moment, as we link containers directly to a playlist. However, if we wish to save containers separately in the future so that they may be referred to by multiple playlists, then the data structure is already foreseen to deal with this behaviour. Moreover, the dimensions of the container are added to the container reference instead of the container itself. This way, different playlists could include the same container with the same content, but with a different size on the display. Every container holds at least one reference to either a media object or another playlist that will be included as a sub-playlist. Media references hold the options for this instance of a media object. The media object itself contains the properties of a media item. There is a similar behaviour for playlist references. In the following sections, the different objects are described more in detail in JSON Schema[3].

### Playlist

The `playlist` object, described in Listing 3.1, can be considered as a SMIL `par` element combined with the `layout` element. It contains references to `container` objects that will be displayed on the screen simultaneously. Playlist objects contain a `type` property that specifies that this object is indeed a playlist, a `name` property and an array of references to containers.

```
"playlist": {
  "id": "/playlist",
  "type": "object",
  "properties": {
    "type": {
      "enum": ["playlist"]
    },
    "name": {"type": "string"},
    "containers": {
      "type": "array",
      "items": {"$ref": "/containerRef"}
    }
  },
  "required": ["type", "name", "containers"]
}
```

Listing 3.1: `Playlist` object description in JSON Schema

[3] http://json-schema.org

Any `container` can refer to a `playlist` object by means of a `playlistRef` object, described in Listing 3.2. In this reference object, values for the `repeat` and `repeat_infinite` properties are specified, to indicate how many times the playlist should be repeated within the container. As multiple containers can refer to the same playlist, they will use separate `playlistRef` objects to customise the repetition behaviour. Moreover, the `playlistRef` object contains some bookkeeping properties to refer to the parent container of the playlist (`parent`) and keep track of the progress when the media is played inside its containers (`containers_done`).

```
"playlistRef": {
  "id": "/playlistRef",
  "type": "object",
  "properties": {
    "parent": {"$ref": "/containerRef"},
    "repeat": {"type": "number"},
    "repeated": {"type": "number"},
    "repeat_infinite": {"type": "boolean"},
    "src": {"$ref": "/playlist"},
    "containers_done": {
      "type": "array",
      "items": {"$ref": "/containerRef"}
    }
  },
  "required": ["src"]
}
```

Listing 3.2: `PlaylistRef` object description in JSON Schema

### Container

The `container` object in Listing 3.3 can be considered as a SMIL `seq` element combined with the `region` element. It contains a `src` property, which is an array of references to `media` and `playlist` objects. The objects in the `src` array will be played sequentially.

A playlist can refer to a container by means of a `containerRef` object, described in Listing 3.4. This object contains the same `repeat_infinite` and `repeat` properties as a `playlistRef` object, but also contains the relative `width` and `height` of the container on the screen. Bookkeeping parameters for the container allow to keep track of the current media item or playlist that is being played within the container at a given time. The `containerRef` element also refers to an actual HTML element in the media player and saves

the absolute dimensions of that element on the current screen resolution. The `parent` property refers back to the playlist that refers to the container and is used by the media player to notify this playlist when the container has finished playing its content.

```
"container": {
  "id": "/container",
  "type": "object",
  "properties": {
    "src": {
      "type": "array",
      "items": { "enum": [ {"$ref": "/playlistRef"},
                           {"$ref": "/mediaRef"}     ] }
    }
  },
  "required": ["src"]
}
```

Listing 3.3: `Container` object description in JSON Schema

```
"containerRef": {
  "id": "/containerRef",
  "type": "object",
  "properties": {
    "parent": { "anyOf": [ {"$ref": "/playlistRef"},
                  { "type": "null" } //Root ]         },
    "repeat": {"type": "boolean"},
    "repeated": {"type": "number"},
    "repeat_infinite": {"type": "boolean"},
    "width": {"type": "number", "minimum": 0, "maximum":
        100},
    "height": {"type": "number", "minimum": 0, "maximum"
       : 100},
    "element": {"type": "string"},
    "element_width": {"type": "number"},
    "element_height": {"type": "number"},
    "src_cur": {"type": "number"},
    "src": {"$ref": "/container"}
  },
  "required": ["src", "width", "height"]
}
```

Listing 3.4: `ContainerRef` object description in JSON Schema

### Media

The `media` object corresponds to a SMIL `ref` element including all aliases, depending on the `type` property of the object. Listing 3.5 describes this object. The system supports a number of default media types including `video`, `image`, `youtube`, `vimeo` and `webpage`. Furthermore, custom media types can be created by means of plugins. The object must have a unique `name` and must refer to the location of the actual media file. This reference should be a HTTP URI. Based on the needs of a specific custom media type, additional properties can be added.

```
"media": {
  "id": "/media",
  "type": "object",
  "properties": {
    "type": {
      "anyOf": [
        {"enum": ["video", "youtube", "vimeo", "webpage"
            , "text", "image"]},
        {"type": "string"}
        ]
    },
    "name": {"type": "string"},
    "src": {"type": "string"}
  },
  "required": ["type", "name", "src"]
}
```

Listing 3.5: `Media` object description in JSON Schema

Containers can refer to a `media` object by means of a `mediaRef` object, as Listing 3.6 shows. This object can obtain some properties that are natively recognised by the system as they relate to at least one of the native media types. For example, the `dur` and `dur_infinite` properties are used to specify the duration of an object, while `stretch` indicates whether the contents should be stretched to fit the parent container regardless of the aspect ratio. Similar to other reference objects, `mediaRef` includes bookkeeping properties to keep track of the media's repetition during playlist execution. Along with the media object, the reference can also be expanded with additional properties. The `parent` property is used to refer back to the parent container of the media object; this allows the media object to access this container directly and instruct it to play the next media item in its sequence.

```
"mediaRef": {
  "id": "/mediaRef",
  "type": "object",
  "properties": {
    "parent": {"$ref": "/containerRef"},
    "repeat": {"type": "number"},
    "repeated": {"type": "number"},
    "repeat_infinite": {"type": "boolean"},
    "dur": {"type": "number"},
    "dur_infinite": {"type": "boolean"},
    "stretch": {"type": "boolean"},
    "src": {"$ref": "/media"}
  },
  "required": ["src"]
}
```

Listing 3.6: `MediaRef` object description in JSON Schema

### 3.3.2 Drawbacks of the JSON Structure

The JSON data structure that is presented to the media player can be considered as a tree of containers, playlists and media objects. However, when a media object is finished playing, the media player should access the parent container of this object to continue with the next media item or mark the container as done. Therefore, the media object has a reference to their parent container. In theory, this reference could be set on the server before the playlist is sent to the media player. However, it is impossible to stringify[4] circular references in a JSON object. Therefore, the stringified JSON object is sent to the media player as a tree, after which the media player must perform a pointer swizzling action on the data structure to set all parent references. In Section 4.2.1 we explain how the media player performs this process.

## 3.4 Plugins

Plugins can extend the content management system to provide extra content types to the users. Plugins have specific requirements, which will be explained in this section.

[4]To convert a JSON object to textual representation

### 3.4.1 Plugin Description Requirements

Each plugin must provide a description of its structure and requirements. At least a unique `name` and `alias` are required for the plugin. The system will add an `admin_enabled` boolean to indicate whether the administrator has enabled this plugin to be used. The `name` identifies a specific plugin, whereas the `alias` is used in the user interface.

#### Plugins in a Static Playlist

First of all, plugins can be used in a static context, where a media object of a custom type is added to a playlist. The plugin will require some information to be able to output the intended content. Therefore, the plugin must describe its supported properties and options for a static media item in JSON Schema as indicated Listing 3.7. Section 4.1.1 explains how we use this information to automatically generate HTML forms to add new media items to the system.

```
"property_or_option": {
  "type": "object",
  "properties": {
    "title": {"type": "string"},
    "description": {"type": "string"},
    "type": {
      "enum": ["string", "number", "integer", "boolean",
          "array", "object"]
    },
    "default": {"type": {"$ref": "#/properties/type"}},
    "required": {"type": "boolean"},
  },
  "required": ["title", "description", "type", "default"
      , "required"]
}
```

Listing 3.7: Media properties and options description in JSON Schema

Some options have a special function, as they are processed by the system instead of the plugin. Table 3.2 specifies these options. Especially the `dur` and `dur_infinite` options are important. If none of both are specified, the plugin is responsible for notifying the player when it has finished playing. Section 4.2.1 explains the implementation of this possibility in our media player.

| Option | Description |
|---|---|
| `dur` | An integer specifying how long the media should be played before continuing to the next media item |
| `dur_infinite` | A boolean indicating if the media should be played for an infinite amount of time until another playlist is pushed to the player |
| `repeat` | An integer specifying how many times the media should be repeated |
| `repeat_infinite` | A boolean indicating if the media should repeat infinitely until another playlist is pushed to the player |

Table 3.2: Special options for plugins

### Plugins in a Context-based Playlist

To enable the system and plugins to provide personalised content, the plugins may need additional information from the users or the system administrator. The implementation of the plugins that we mention in this section will be explained in Section 4.4. Firstly, this information can be specific to a plugin. For example the SVN plugin, requires a list of SVN repositories that the user is interested in, accompanied by credentials for authentication to the SVN server. This information must be completed by the user for this specific plugin. The plugin must describe the required information in JSON Schema in the `user_config` property of the plugin description. The structure of this description has the same requirements as the options and properties for static content. If the plugin requires any configuration from the administrator on a system-wide level, the `admin_config` property should be provided with the same structure requirements.

Secondly, plugins may require some fields from the user profile. The Traffic plugin for instance, can retrieve the home address of a user to calculate the time required to travel home. The user profile is provided to all plugins by default and does not require a specific configuration.

Thirdly, some plugins may generate their output based on what the user is interested in. The News plugin takes the interests of the user into account to show relevant news items. As we will explain in Section 3.6.3, the user maintains their own interests in their profile.

Not only the News plugin may benefit from user interests. Also the Weather plugin could show the weather for any physical locations that a user has listed in their profile. In this case, the Weather plugin should be able to filter out all physical locations. To achieve this, we use the semantic web to identify the type of every interest. The plugin can describe objects and

the relations that a valid interest should have to at least one of these objects in the `interest_filter` property. This description is actually a SPARQL query in JSON form. Section 3.6 explains how interests are managed. The format is specific to the parser that we use in our implementation.

Next to discovering interests dynamically, a plugin may only be able to use a set of fixed interests. The Distrowatch[5] plugin can only provide information about Linux distributions that are known by the Distrowatch website. Therefore, the plugin description should include a list of the Linux distributions that the plugin is familiar with. The `interests` property can contain an object that has relevant interests as properties. These interests should preferably contain a DBPedia resource URI. Lastly, the `all_interests` property is a boolean which may be used to indicate whether all possible interests are valid for the plugin, as is the case for the News plugin. Based on the requirements that we explained in this section, the description of a media type must be provided as demonstrated in Listing 3.8.

```
"plugin": {
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "alias": {"type": "string"},
    "properties": {"type": "object"},
    "options": {"type": "object"},
    "user_config": {"type": "object"},
    "admin_config": {"type": "object"},
  "interests_filter": {"type": "object"},
  "interests": {"type": "object"},
  "all_interests": {"type": "boolean"},
  },
  "required": ["name", "alias"]
}
```

Listing 3.8: Plugin description in JSON Schema

When a new plugin is introduced to the system, the system will scan all interests and indicate whether they are a valid candidate for this plugin, based on the `interest_filter`, `interests` and `all_interests` properties. On the other hand, when a new interest is added by a user, the system scans all plugins to indicate for which plugins this interest might be a candidate.

[5] http://www.distrowatch.org

### Plugin Description Variables

The system supports two variables that may be included in the description of a plugin. Variable names are always encapsulated in dollar signs. The `$CREDENTIALS$` variable can be used in the `properties` and `options`, as well as the `user_config` and `admin_config` objects. Whenever any of these objects is used in the system, the variable is replaced with an array containing the references to all saved credentials for the administrator or the active user. The `$INTEREST$` variable may be used in the `interests_filter` object. Whenever the filter is used to assess the validity of an interest as a candidate for this plugin, the DBPedia resource URI for the interest is injected in the SPARQL query wherever this variable is present.

## 3.4.2 Plugin Functionality Requirements

Next to a description as described in the previous section, the plugin must also provide a file that contains the functionality of the plugin. This file must provide at least two mandatory functions, called `play()` and `generate()`. The `play()` function takes a `mediaRef` object as parameter, containing the configuration of a specific instance of the plugin. The function should then generate the output for the media player, based on the configuration. The function is responsible for sending the HTML output back to the media player. The `generate()` function generates media objects based on the information for a specific user. It takes the user configuration, admin configuration, interest candidates, context and user profile as parameters and returns an array containing the `mediaRef` objects that are related to that user. The resulting media objects must contain an `_id` property that identifies the content of the media object. This identifier is used by the system to filter out duplicate media objects that are related to more than one user and would provide the same output.

It is up to the plugin to decide what makes a media object unique. For example: Suppose we have a plugin that shows the latest revision of an SVN repository and another plugin that shows the number of unread e-mails for a user. Both plugins would probably need a server address and user credentials to collect the appropriate data. In the case of the SVN plugin, it is the address of the repository that makes a media item unique, regardless of the credentials that are used to authenticate, as the output will always be the same. The situation is different for the unread mails. The server address may be the same for all users, but logging in with different credentials will give access to different mailboxes. Therefore a media item is uniquely identified by the user name rather than the server address.

## 3.5 General Purpose Content

Using a content management system, the administrator may add media content and create playlists. The playlists may be pushed to media players in static mode and set as the default playlist for arbitrary players. Section 4.3.5 explains how playlists may be created using the Playlist Designer. It is the responsibility of the administrator to add content that is relevant to the location and audience of the displays that will play the media.

## 3.6 Personalised Content

### 3.6.1 Managing User Interests Using the Sematic Web

Users are allowed to enter keywords that describe what they are interested in. However, to fully understand the meaning of these keywords for a specific user, we must know the proper semantics behind this keyword to avoid ambiguity.

DBPedia[6] provides a SPARQL[7] endpoint to access roughly three billion semantic relations between objects. Furthermore, the information in DBPedia is extracted from Wikipedia[8]. This allows us to disambiguate any keywords that the user specifies as interests, based on the disambiguation pages of Wikipedia.

First of all, any disambiguation page of Wikipedia is also a resource in DBPedia. Therefore we can first check if a disambiguation page exists for a specific keyword and retrieve all linked resources. This can be achieved with the SPARQL query in Listing 3.9. The query selects all distinct resources that are referred to by another resource using the wikiPageDisambiguates relation of the DBPedia ontology. Afterwards, the results are filtered to find the proper disambiguation resource based on the entered keyword. Then we filter out all remaining resources that do not have a label that contains the keyword. In the last filter, we remove all resources that do not belong to the default Wikipedia name space, like Category and Help pages.

For example, the user may be interested in Antwerp. More specifically, the user is interested in the city of Antwerp, located in Flanders, Belgium. However, Antwerp is also a province in Belgium. There are also locations named Antwerp in New York, Ohio or Victoria. Furthermore, the football club of Antwerp in Belgium is also referred to as Antwerp. Luckily,

[6] `http://wiki.dbpedia.org`
[7] `https://www.w3.org/TR/sparql11-overview/`
[8] `https://www.wikipedia.org`

Wikipedia provides a disambiguation page for this keyword[9]. When launching the SPARQL query with this keyword, we get a result set of 11 possible resources. The user can then specify the intended meaning of their keyword.

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?resource, ?disamb
WHERE {
 ?disamb rdfs:label ?label.
 ?disamb <http://dbpedia.org/ontology/
    wikiPageDisambiguates> ?resource.
 ?resource rdfs:label ?rlabel.
 filter (bif:lower(str(?label)) in (\'$INTEREST$\'))
 filter (bif:contains(?rlabel, \'\"$INTEREST$\"\'))
 filter (!regex(bif:lower(str(?resource)),
         "\\\\/.*\\\\S:"))
}
```

Listing 3.9: SPARQL query to retrieve Wikipedia disambiguation pages on DBPedia for an interest keyword

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?resource
WHERE {
 GRAPH <http://dbpedia.org>
 {
  ?resource rdfs:label ?label.
  filter (bif:lower(str(?label)) in (\'$INTEREST$\')
  && !regex(bif:lower(str(?resource)), "\\\\/.*\\\\S:"))
 }
}
```

Listing 3.10: SPARQL query to retrieve different resources on DBPedia for an interest keyword

If the query result is emtpy, a second query is launched, which is clarified in Listing 3.10. The second SPARQL query retrieves all resources that have a label that is an exact match with the entered keyword. Of course, the same final filter applies to remove resources from special Wikipedia name spaces. Suppose that a user enters the keyword "Leffe". Leffe is a beer that is produced in Belgium, but it is also a town in Lombardy, Italy. There is

[9] https://en.wikipedia.org/wiki/Antwerp_(disambiguation)

no disambiguation page for this keyword, so the first query will return an empty result. The second query will then return two possible resources.

In theory, we could just retrieve the different resources with only the second query. However, the first query is much faster. Therefore, if multiple resources exist and Wikipedia has a disambiguation page for them, then we can enumerate the different possibilities in a fast manner with this query. Secondly, if the first query does not return any result, it is unlikely that the result set of the second query will be very large. Then it is safe to launch this query so that it can complete within the timeout that is configured on DBPedia. Otherwise, if many resources are possible, the second query may not finish in time.

### 3.6.2 Administrator

The administrator is able to influence the content that a user will see. First of all, the administrator has to enable the plugins in the system that should be available to users. They also complete the admin configuration for each plugin, if required. Second, the administrator may create user templates. These templates contain default interests that are inherited by each user that references it. Therefore, the administrator decides which types of media are certainly not available to users and which interests are linked to each user by default.

### 3.6.3 User Profiles

Each user has a profile in the content management system. In this profile, they may enable plugins and complete the required user configuration, as well as add interests. The user can only enable plugins that have first been enabled by the administrator. Arbitrary interests may be added to the user profile, but will always be combined with the interests from the linked user template.

### 3.6.4 Playlist Generation with the Ambilist Object

The playlist for context-aware mode has to be generated on the fly, as users can enter or leave the detection zone at any moment. The main goal of the ambient player is that there is at least one interesting item on the screen for each detected person.

The system always remembers the last context that was reported by a sensor for a specific screen. This context contains the users that are in the detection zone. When a new context update is provided, this update is

compared to the previous context for that player. If one or more new users are detected, the playlist should be expanded for these newcomers. If any users have left, all media items that are only related to the leavers have to be deleted.

We call the dynamic playlist that is generated by the system an `ambilist`. The ambilist is a JSON object with four properties called `users`, `media`, `combinations` and `players`. All first three properties contain a lookup object. The media object has all media identifiers as properties, containing the `mediaRef` object for the corresponding ID. The `users` object contains the user IDs as properties containing the respective user profiles. The `combinations` property also contains an object with the possible combinations of user IDs as properties. Each combination is an object with four properties:

- The `members` property contains an array with the user IDs of the members of this combination. These IDs are used to generate the ID for the combination, being a concatenation of all member IDs in alphabetical order.

- The `media` property contains an array of media IDs that refer to properties of the media lookup object of the ambilist. All referred items are related to all members of the combination.

- The `media_playing` property contains the media that is being played for this combination at a specific moment in time.

- The `media_done` property keeps track of the media that has already been played for this combination of users.

If $c_n$ is the number of combinations that can be generated out of $n$ users, then $c_n$ is the summation of all $k$-combinations of those $n$ users where $k$ equals 1 to $n$ [22].

$$c_n = \sum_{k=1}^{n} \binom{n}{k} = \sum_{k=1}^{n} \frac{n!}{k!(n-k)!}$$

Whenever a newcomer is added to the ambilist, the combinations of this user with the other users are calculated. All media objects are linked to the combination of all users they are related to. The players object contains a `master` property and a `slaves` property. The `master` property is a string containing the player ID of the master player for the media player group. The `slaves` property contains an array of the player IDs for all slave players. Based on the four properties explained above, the basic structure for the ambilist object is as shown in Listing 3.11.

```
"ambilist": {
  "users": {},
  "media": {},
  "combinations": {},
  "players": {
    "master": "",
    "slaves": []
  }
}
```

Listing 3.11: Ambilist data structure

### 3.6.5 Playing Context-based Media

The master player will evenly divide the content to be played over all member players. It keeps track of which player is playing which media at any given moment. The master notifies the slave players of the media that they should play. When a media object is finished playing on a slave player, the slave notifies the master that it has finished playing a certain object. The master will then decide which object the slave should play next, if any, as there may be no media available.

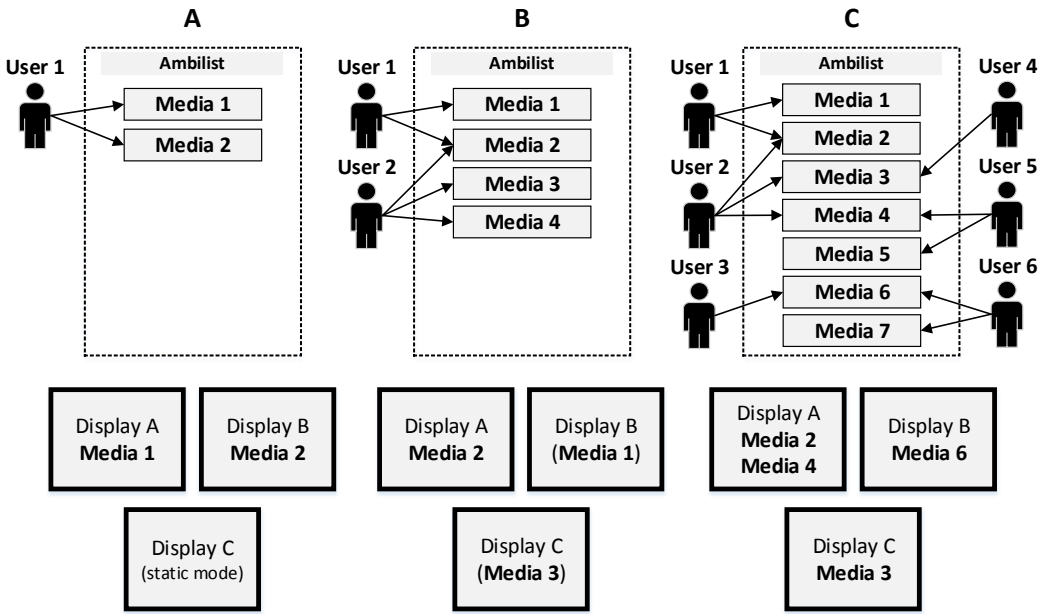



Figure 3.3: Possible situations A (left), B (middle) and C (right) during content distribution in context-aware mode

We notice three possible situations while trying to satisfy the rule of playing at least one interesting media item for each user, as we illustrate in Figure 3.3:

- There are less media items in the ambilist than there are media players in the group. In this case, some players will stay in static mode, as there is not enough ambient media to distribute. Situation A illustrates this possibility where display C remains in static mode as there are only two media items in the ambilist.

- There are equally many or more media items than there are media players or there are equally many users than media players. This means that all players in the group may enter context-aware mode. If we play a media item for each individual user, then each media player can play an item in fullscreen. However, to satisfy the rule, we will not always need one object for each user, as media can be related to multiple users at the same time, in which case less players are needed. In this situation, the remaining media players are given a random media object to play, selected from the objects that have not been played yet. In situation B, the master will play Media 2 first, as this action will satisfy the rule for the highest number of users. Some other media objects — Media 1 and Media 3 — are sent to the slave players. This action is not required to satisfy the rule, but does allow us to play even more interesting content for the users simultaneously.

- There are equally many or more media items than there are media players and there are more users than media players. Here we have the same possibilities as with an equal number of users and media players, however there is one more situation possible. The algorithm can be in a state where more media items are needed than there are media players to satisfy the rule. Here we must split one or more screens into multiple zones, as we explain in Section 3.6.6. Situation C shows that six users are present and there is enough media to distribute between all displays. However, due to the lack of similar interests between the present users, at least four media objects must be played to satisfy the rule. As there are only three displays, one display must be split in two zones. The master will start playing Media 2 and then distributes Media 6 and Media 3 to Display B and Display C respectively. As there is no media playing yet for User 5 at that point, Media 4 must also be played. All displays are now playing an equal number of media items, therefore the master will first split themselves in two zones to play Media 2 and Media 4 simultaneously.

### 3.6.6 Content Sizing

The personal content must be properly proportioned on the screen. If the combination of all present users has at least one related media object and there is only one media player in the group, then this object may be played in fullscreen. However, if multiple media items are needed to have at least one interesting item for each user and there are less media players than media objects, screens must be dynamically divided in multiple zones. We decided to keep this division in a square matrix as much as possible, as this method also keeps te aspect ratio of the screen for most of the zones. Figure 3.4 shows how the algorithm would arrange one to eight containers on one single display.

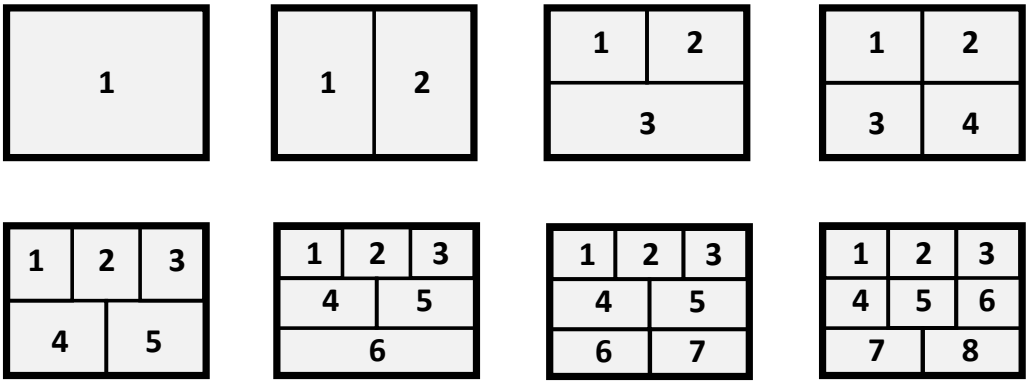



Figure 3.4: Sizing one to eight containers on a display

If $m$ is the number of media items or zones that have to be created, then the number of rows in the matrix $n_{rows}$ is calculated as follows:

$$n_{rows} = round(\sqrt{m})$$

Therefore, if $r_k$ is the k$th$ row of the zones matrix and $1 \leqslant k \leqslant n_{rows}$, then the number of zones for each row is defined by the following recursive formula:

$$r_1 = \left\lceil \frac{m}{n_{rows}} \right\rceil$$

$$r_k = \left\lceil \frac{m - \left( \sum_{l=1}^{k-1} r_l \right)}{n_{rows} - k + 1} \right\rceil$$

The dimensions $height_{z_k}$ and $width_{z_k}$ of each zone $z$ in row $k$ are therefore the following:

$$height_{z_k} = \frac{height_{screen}}{n_{rows}}$$

$$width_{z_k} = \frac{width_{screen}}{r_k}$$

### 3.6.7 Timeout and Freshness

Each media player group is provided with a `timeout` and `freshness` parameter to improve the system's performance. The `timeout` parameter indicates how many seconds a user should not be reported by any sensor anymore before they are removed from the context. For example, when a simple webcam is used to detect users, then this camera takes images in a specified interval. If a user steps out of the detection zone for a brief moment, right at the time that the image is taken, then this user should not immediately be removed from the context. Therefore, the `timeout` parameter may be used to counteract false positives from sensors with a high error rate. Depending on the quality of the sensor, this parameter may be adapted.

The user recognition sensor posts an update to the server every $x$ seconds. If the posted context contribution is unchanged compared to the previous post, then the ambilist for the related player group is not updated. However, if the system were to recalculate the ambilist for the present users, the plugins may output other media objects that are interesting for the present users, based on other context properties that have changed in the meantime or other information that is available. However, we do not wish to recalculate the ambilist with every post of the sensor, as this impacts the system's performance. Therefore we added the `freshness` parameter. This parameter indicates in which interval the ambilist should be recalculated for users that are present for a minimal amount of time.

## 3.7 Context Generation

The context of a media player group can consist of any properties that are required in the use case of the system. The only context property that is required for context-aware mode to be triggered, is the `users` property. Whenever a context is provided that includes one or more present users, the server will instruct all related media players to switch from static mode to context-aware mode. Of course, the users requirement may be omitted by

creating a dummy user profile and having one of the sensors report this user as being permanently present.

The system manages the `users` property in a different way than the other properties. Other properties are entirely overwritten by the last posted value of any sensor. The user objects in the `users` property are managed individually, `first` and `last` properties are added to each user to keep track of when they were detected for the first and last time, regardless of the sensor that contributed the information. Listing 3.12 provides a sample context object of a media player group. A sensor may post information to the system in a JSON object by including its `sensor_id` and the information that it wishes to contribute. If the sensor detects users, then it must provide these present users in the `users` context property. Listing 3.13 provides a sample context contribution of a sensor that detects present users.

```
"context" : {
  //users parameter mandatory
  "users" : [
    {
      "_id" : "585833f7024cbb239a99b872",
      "first" : 1491140540,
      "last" : 1491140540
    },
    {
      "_id" : "5873a5baefb5352e526670ac",
      "first" : 1491140435,
      "last" : 1491140510
    }
  ],
  //other parameters optional
  "local_time" : 1491140540,
  "local_temperature" : 21.3
}
```

Listing 3.12: Sample context of a media player group

```
{
  "sensor_id": "58bae110f768af03f3c81c38",
  "users": [  {"_id": "585833f7024cbb239a99b872"},
              {"_id": "5873a5baefb5352e526670ac"}   ]
}
```

Listing 3.13: Sample context contribution of a sensor that detects present users

## 3.8 Rule Engine

The system includes a basic rule engine that may be used to disable or exclusively enable certain plugins, based on a set of rules.

### 3.8.1 Rule Templates

Custom rule templates may be created so that any possible type of rule is supported. A rule template is a JSON object with specific properties. In appendix D we have added a skeleton that may be used to create templates along with a number of concrete examples. A rule template must provide the following properties:

- Title: A title for the rule template.
- Description: A description of the rule template.
- Input: An object describing the required user input when creating a rule from this template. The `operator` property is always required and contains an enumeration of possible operators for the rule. Each operator in the enumeration is an object containing an operator name and value. Other input is optional.
- Context: A description of the properties that this rule requires from the context object. For every property, a sensor must be present that provides the necessary values.
- Evaluate: An object that contains a property for every operator, named after the operator value. Each property contains a function that takes in the context and user input and returns a boolean that indicates whether the rule with that operator was satisfied.

### 3.8.2 Rulesets and Actions

Based on the rule templates, users may create rulesets including various rules based on any existing templates. They may decide whether a ruleset is processed with an `AND` or `OR` logic. When a ruleset is satisfied, the actions for that ruleset are executed. This happens every time when an ambilist is (re)generated for a player group that is included in one of those actions. An action consists of disabling or exclusively enabling one or more plugins for one or more player groups. If a plugin is both enabled and disabled for the same player group, due to overlapping actions, the disable operation always has precedence.

### 3.8.3 Use Cases for Rules

The template-based rule engine gives the administrator the possibility to create any kind of rule. A type of rule that would be used often, is a time-based rule. For example, the Traffic plugin that we describe in 4.4.2 can show a user the estimated travel time from their current location to their home when enabled in context-aware mode. If this plugin were to be used on a display that is located at the reception of a company, then it would be pointless to show that information to a person that has just arrived at the company in the morning. Instead, this information would be most useful when the user is leaving the building at the end of the company's business hours. A time-based rule could make sure that this plugin is only enabled at the right time. Of course this logic may also be implemented in the plugin itself, without the need for rules. However, the rule engine gives the administrator extra flexibility.

Another application for using rules, is a user-based template. This type of rule could require the user to enter a list of users that should (not) be present in the same context. It could then parse the `users` property of the context to evaluate the rule. When a user has enabled a plugin in their profile that may show some sensitive information that may not be exposed to other users, this rule could aid in protecting their privacy.

### 3.8.4 External Rule Engines

The rule engine that we have developed for our tests is rather basic compared to other engines that already exist, like Drools[10]. In Drools, the administrator would be able to configure more advanced rules, like specific sequences of events, based on the history of context information. Therefore, we have added a wrapper module that allows to integrate an external rule engine into the system. Section 4.3.8 explains more about the implementation of rule engines.

## 3.9 CastJS Architecture

The components and requirements that we explained in the previous sections yield our system architecture for CastJS, represented in Figure 3.5 (see Appendix A for a full page version). Our content management system contains different modules. First, there is the user interface, that both the administrator and end users use to configure the system. Section 4.3 explains the

[10] `https://www.drools.org`

implementation of this interface. The user is able to add interests to their profile. The interests manager provides functionalities to disambiguate these interests, by using the public SPARQL endpoint of DBPedia. Section 3.6.1 clarifies this process. The plugin manager module is used in multiple ways. It provides information about the requirements of a specific plugin, which is used in multiple locations in the user interface, as we will explain in Section 4.3. On the other hand, this module is also used by the playlist generator and the media players, as it executes functionality of the plugins that output media items and actual content for the displays. In Section 3.4 we explained more about the use of plugins.

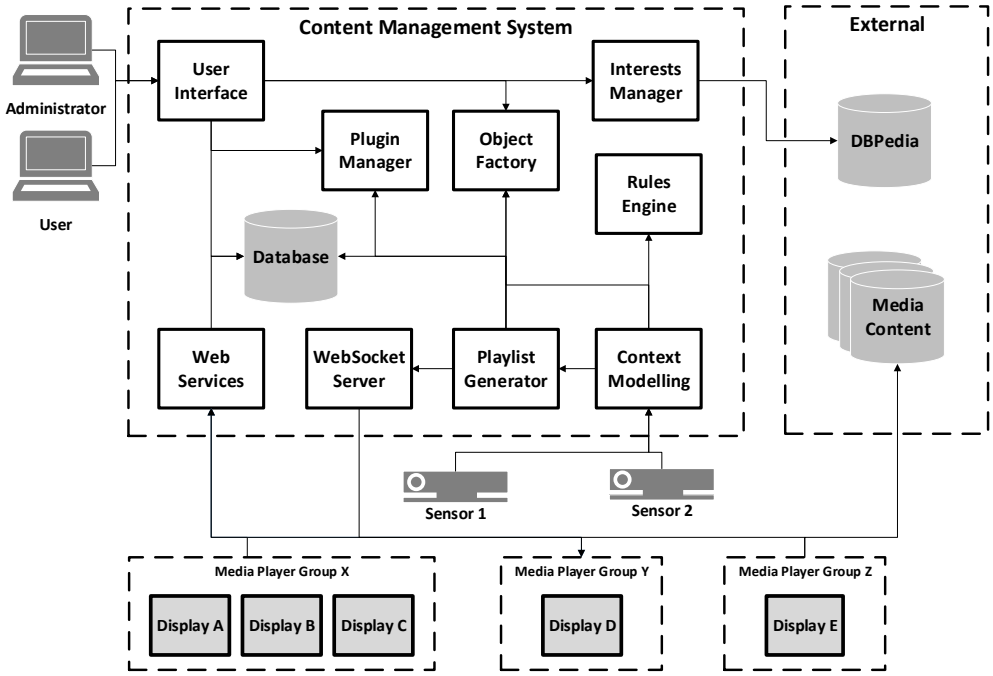



Figure 3.5: CastJS architecture

The context modelling engine receives input from various sensors and updates the context for linked player groups, as we explained in Section 3.7. The rule engine, explained in Section 3.8 is also consulted in this process. The retrieved information is handed over to the playlist generator, which consults the plugin manager to generate context-based media.

Section 3.6 provides information about generating context-based content. When a playlist is generated, a WebSocket connection is used to push the structure to the relevant media player groups. The media players then re-

trieve actual media content from external sources and from web services in the content management system that access the internal database and the plugin manager. The architecture of our internal database is available in Appendix B.

# 4 Implementation

## 4.1 Used Technologies

For the implementation of CastJS, we used HTML5, CSS and JavaScript on the client side. We continued to use JavaScript on the server side with NodeJS[1] and ExpressJS[2]. NodeJS is a server side Javascript runtime environment that uses the Google V8 JavaScript Engine for interpreting the JavaScript code. For the database, we use MongoDB[3], which allows us to easily save and retrieve JavaScript objects in JSON. Our web server was running on an Ubuntu[4] operating system.

We used several external libraries and scripts that already implement logic and features that we planned on integrating in our solution. Some of the plugins also use an external web service to retrieve content to generate media.

[1] https://nodejs.org/
[2] http://expressjs.com/
[3] https://www.mongodb.com/
[4] https://www.ubuntu.com/

### 4.1.1 JSONForm

We would like to especially mention the JSONForm[5] library, as it plays an important role in our content management system. JSONForm is capable of generating HTML forms based on a description of the required fields in JSON Schema. Therefore, this library allows us to easily generate forms to enter properties, options, user configuration and admin configuration for plugins, as explained in Section 3.4.

## 4.2 Media Player

### 4.2.1 Static Media Player Algorithm

#### Playlist Pre-processing

As explained in Section 3.3.2, the media player must first do a pre-processing of the received data structure to set the circular references to parent elements. Three functions are responsible for traversing the playlist. The first function that is executed, is `setParents()`. This function receives the reference object to the main container as parameter. The `sources` array of the main container only contains one object, which is the actual playlist.

The `setParents()` function will start an iterative process by calling the second function called `setParentsMediaLoop()`. This function receives a media or playlist reference object as first parameter and the reference object to the parent container as second parameter. The function will set the parent reference of the media reference object to the container reference. If the media object is a playlist, which is of course the case in the first iteration, the code will iteratively process all child containers of this playlist by calling the `setParentsContainersLoop()` function.

The `setParentsContainersLoop()` function receives a container reference as first parameter and a playlist reference object as second parameter. The function will set the playlist reference object as parent for the container reference. The function will then iteratively process every source in the `sources` array of the container using `setParentsMediaLoop()`. Listing 4.1 shows the implementation of these functions. When the pre-processing of the playlist has finished, the media player can start playing the media content.

[5] `https://github.com/joshfire/jsonform`

```
function setParents(main_container){
  setParentsMediaLoop(main_container.src.src[0],
     main_container);
}
function setParentsContainersLoop(container, parent){
  container.parent = parent;
  for (var i=0; i<container.src.src.length; i++){
    setParentsMediaLoop(container.src.src[i], container)
       ;
  }
}
function setParentsMediaLoop(media, parent){
  media.parent = parent;
  if (media.src.type === "playlist"){
    for (var i=0; i<media.src.containers.length; i++){
      setParentsContainersLoop(media.src.containers[i],
         media);
    }
  }
}
```

Listing 4.1: Media player functions for setting parent references

### Playing Media Content

The `playMedia()` function in Listing 4.2 is executed to start the media player process. The main playlist reference object is passed to this function as a parameter. Of course we do not need to pass the main container separately, as the playlist has a circular reference to it. The function will first create a new `div` element with equal size to the current viewport of the web browser. If the browser is started in fullscreen mode, this will equal the size of the screen. The main container saves a reference to this `div` element in its `element` property, after which the element is added to the `body` element in the DOM tree. After the container has been added to the screen, the media player will start processing the sources of the container object. The `playPlaylist()` function in Listing 4.2 will receive the playlist reference object as a parameter. If the function receives a playlist of which none of the child containers have finished playing their sources, then a new `div` element will be created for each child container. This `div` will receive an absolute size, based on the absolute size of its parent container and the relative size of the container. For each container, the `playContainer()` function is executed. If all child containers of the received playlist are done playing, then the `donePlaylist()`

function is executed. Otherwise, if the number of finished containers is not zero nor the total number of child containers, then this means that some containers are still playing and the `playPlaylist()` function will take no action.

```
function playMedia(playlist) {
  var width = $(window).width();
  var height = $(window).height();

  playlist.parent.element = $('<div />', {
    width: width + 'px',
    height: height + 'px',
  });
  playlist.parent.element_width = width;
  playlist.parent.element_height = height;
  $('body').html('');
  playlist.parent.element.prependTo($('body'));
  playPlaylist(playlist);
}
```

Listing 4.2: Media player `playMedia()` function

```
function playPlaylist(playlist){
  var container = playlist.parent;
  if (playlist.containers_done.length === 0){
   for (var i=0; i<playlist.src.containers.length; i++){
     var width = (playlist.parent.element_width *
            (playlist.src.containers[i].width / 100));
     var height = (playlist.parent.element_height *
            (playlist.src.containers[i].height / 100));
     playlist.src.containers[i].element = $('<div />',
     {width: width + 'px', height: height + 'px'});
     playlist.src.containers[i].element.appendTo(
        playlist.parent.element);
     playlist.src.containers[i].element_width = width;
     playlist.src.containers[i].element_height = height;
     playContainer(playlist.src.containers[i]);
   }
  } else if (playlist.containers_done.length >= playlist
     .src.containers.length){
    donePlaylist(playlist);
  }
}
```

Listing 4.3: Media player `playPlaylist()` function

The playContainer() function is executed for every separate container. This function reads the src_cur property of the container reference object, indicating the index of the last played media object within the sources array of the container. If this counter has not yet reached the total number of media objects, then the media player can play the next item by calling the playSource() function. Otherwise, the doneContainer() function is executed, which resets the source counter and marks the container as done. Then the playPlaylist() function is executed for the playlist of which this container is a child. Listing 4.4 shows the implementation of the container-related functions.

```
function playContainer(container){
  var i = container.src_cur;
  if (i >= container.src.src.length){
    doneContainer(container);
  } else {
    container.element.html('');
    playSource(container);
  }
}

function doneContainer(container){
  container.src_cur = 0;
  if (container.repeat_infinite == "true"){
    playContainer(container);
  } else {
    if (container.repeat > 0 && container.repeat >
       container.repeated){
      container.repeated++;
      playContainer(container);
    } else {
      container.parent.containers_done.push(container);
      playPlaylist(container.parent);
    }
  }
}
```

Listing 4.4: Media player playContainer() and doneContainer() functions

The donePlaylist() function from Listing 4.5 is executed following the playPlaylist() function if all child containers of a playlist are marked as done. This function resets the bookkeeping properties of the playlist. If the parent of the current playlist object is the root container, then the media

player has finished playing every media object in the data structure and it can restart the process. Otherwise, the playlist is a sub-playlist and its parent container must continue with the next object in its source array by executing `playContainer()`.

```
function donePlaylist(playlist){
  playlist.containers_done = [];
  playlist.parent.element.html('');

  if (playlist.repeat_infinite == "true"){
    playPlaylist(playlist);
  } else {
    if (playlist.repeat > 0 && playlist.repeat >
       playlist.repeated){
      playlist.repeated++;
      playPlaylist(playlist);
    } else {
      if (playlist.parent.parent === null){
        playPlaylist(playlist);
      } else {
        playContainer(playlist.parent);
      }
    }
  }
}
```

Listing 4.5: Media player `donePlaylist()` function

The `playSource()` function is the core function of the media player and can be found in Listing 4.6. Based on the type of the resource, this function will forward the resource to the appropriate player function. If the source is a playlist, the `playPlaylist()` function is executed. If the source is of a built-in media type, the player will execute a specific function for this resource. The media functions are explained in Appenix C. Otherwise, the media item is based on a plugin, in which case the `playPlugin` function is called. Due to the length of the code, we have added this function to Appendix C.6.

The `playPlugin()` function makes sure that the output of a specific plugin is loaded in an `iframe` element. As explained in Section 3.4.1, the plugin must provide a description of its required properties and options. During execution, the plugin must receive the parameters for a specific instance in a playlist. It is mandatory that this information is provided to the server from within the `iframe` element, so that the `iframe` can render the plugin output. Therefore, the `playPlugin()` function generates a dummy `form` el-

ement in the `iframe`, containing the parameters for the plugin. This form is then automatically submitted to the server via a HTTP POST request. In the HTTP response, the `iframe` receives the HTML output and any other files from the plugin.

```
function playSource(container){
  var source = container.src.src[container.src_cur];
  if (source.repeat_infinite == "true"){
    //Do nothing
  } else {
    if (source.repeat > 0 && source.repeat > source.
       repeated){
      source.repeated++;
    } else {
      container.src_cur++;
    }
  }
  source.parent.element.html('');

  switch (source.src.type){
    case "playlist":
      playPlaylist(source);
      break;
    case "video":
      playVideo(source);
      break;
    case "image":
      playImage(source);
      break;
    //... cases for other built-in media types ...
    default:
      playPlugin(source);
      break;
    }
}
```

Listing 4.6: Media player `playSource()` function

If a numeric or infinite duration is defined in the options for this instance of the plugin, then the media player will keep track of this duration. Otherwise, the plugin should let the media player know when it is finished, by executing a jQuery `click` event on its `body` element. The player will catch this event and proceed with the next media items.

### 4.2.2 Context-based Media Player Algorithm

When the context for a media player group is altered, the server generates dynamic content in an ambilist and pushes this ambilist to the master player of the media player group. This is achieved through the WebSocket connection of the media player to the content management server. If the master player is currently operating in static mode, it switches from static mode to context-aware mode and creates a main container with dimensions equal to the viewport of the web browser.

#### Bookkeeping Variables

The media player uses some global variables to keep track of its current state:

- `ambilist`: The current ambilist.
- `master`: The ID of the elected master player for the media player group.
- `n_media`: The number of media items in the ambilist.
- `users_to_play`: An array containing the users that do not have any interesting content on any screen yet.
- `users_playing`: An array containing the users for whom one or more interesting media objects are playing.
- `media_playing`: An array containing the media objects that are currently playing on a specific media player.
- `players`: A lookup object containing all media players with their role — master or slave — and the media they are currently playing.
- `players_pq`: A priority queue implemented as an array that contains references to indices in the `players` object. The priority queue is resorted after every change so that the player with the least number of objects playing is sorted first. If multiple players are playing the same number of media items, the master player is prioritised. The `compare()` function in Listing 4.7 is responsible for sorting the queue in the correct order.

```
function compare(a, b){
  if(players[a].media_playing.length < players[b].
     media_playing.length){
    return -1;
  } else {
    if (players[b].media_playing.length < players[a].
       media_playing.length){
      return 1;
    } else {
      if (players[a].master === true){
        return -1;
      } else if (players[b].master === true){
        return 1;
      } else {
        return 0;
      }
    }
  }
}
```

Listing 4.7: Media player `compare()` function for the players priority queue

### WebSocket Messages

The media players communicate with the content management server and with each other through WebSockets. We have implemented various messages that can be exchanged, as explained in Table 4.1. Any information must be encapsulated in a JSON object with at least one `message` property.

| Message | Details |
|---|---|
| ambilist | This message contains a new ambilist |
| ambistop | The content management server notifies the master player to exit context-aware mode and go back to static mode |
| ambislaveplay | The master player sends a media item to a slave player |
| ambislavedone | The slave player notifies the master player that it has finished playing a certain media item |
| ambislavereset | The master player notifies the slave player to exit context-aware mode and go back to static mode |
| ambislavestop | The master player notifies the slave player to stop playing a specific media item immediately |

Table 4.1: WebSocket messages for context-aware mode

### Ambilist Pre-processing

When the master player receives a new ambilist through an `ambilist` message, it is first processed and compared to the currently playing ambilist. The `startAmbiPlayer()` function performs the following actions:

- For all combinations in the current ambilist, check which media items are done playing. If a media item still exists in the new ambilist, mark it as done in the combination that the item is related to.
- For all combinations in the current ambilist, check which media items are currently playing. If a media item still exists in the new ambilist, mark it as playing in the combination that the item is related to. If the media item does not exist anymore, immediately instruct the media player to cease playing this item.
- Check for which users there is already a media object playing. For a new user, there may already be playing an object due to similar interests with a user that was already present. In this case we indicate that this user does not require a new item to be played by adding them to `users_playing`. Otherwise they are added to `users_to_play`.
- Remove all users that are no longer present from the bookkeeping arrays `users_playing` and `users_to_play`.
- Populate the `players` object and the `players_pq` priority queue. As stated in Section 3.6.5, not every media player may be added, based on the number of available media items.

### Playing Media Content

The `playAmbi()` function is responsible for scanning the `users_to_play` array and find media for the users. First, we create a list of all possible combinations that contain at least one user that is found in `users_to_play`. These combinations are ordered in a way that combinations that only contain users in `users_to_play` are prioritised, sorted by the number of users. For example, if we have users {A, B, C} in the ambilist and users {A, B} in `users_to_play`, then the combinations we will consider are {{A, B}, {A}, {B}, {A, B, C}, {A, C}, {B, C}} in that order.

The first combination that still contains unplayed media is used to play the next media item. The members of this combination are removed from `users_to_play` and added to `users_playing`. If there are still users without

an interesting media item on the screen, this process is repeated with the remaining users.

If there is no more unplayed media available in any combination that involves the remaining users, the combinations with these users are reset. This means that all media is moved from `media_done` back to `media`. If `users_to_play` is empty, the `playAmbi()` function will peek at the first media player in `players_pq`, this is the media player with the least number of media objects playing. If this player is not playing any media at all, a random media item is selected and pushed to the player. This process is also repeated until the first player in the priority queue is playing at least one item. If there is no more media available, then all combinations are reset.

For each media item that is to be played, the `playAmbiMedia()` function from Listing 4.8 is invoked, passing the ID of that media item. This function moves the media object to the `media_playing` property of the related combination. The priority queue serves the media player with the least playing objects. If this player is the master player, the media item will be played locally using the `playAmbiPlugin()` function, otherwise the player is a slave in which case `playSlaveAmbiPlugin()` is triggered to send the item to this player. The bookkeeping for this player is updated and the priority queue is resorted.

The `playSlaveAmbiPlugin()` function requires the media that is to be played and the slave player that should play this media object. It will create an `ambislaveplay` message that is sent to the WebSocket connection of the slave media player. If the slave player is not yet in context-aware mode, it will first create a new root container.

The slave media player then updates its `media_playing` array and executes the `playAmbiPlugin()` function. If it is the master player itself that should play the media, it would execute this function directly. The function requires the media to be played and the appropriate callback function. For a slave media player we require a different callback than for the master. The `playAmbiPlugin()` function is similar to the `playPlugin()` function explained in Section 4.2.1, except for the different callbacks. For a slave player, the callback is the `doneSlaveAmbiPlugin()` function, which updates the `media_playing` array and creates an `ambislavedone` message containing the ID of the media that has finished playing and the ID of the master player. The message is sent to the WebSocket of the master player. The master player then receives the message and executes the `doneAmbiPlugin()` function. When the master player has finished playing a media object, this function is executed directly.

```
function playAmbiMedia(media_id){
  var source = ambilist.media[media_id];
  ambilist.combinations[source.combination].media.splice
     (ambilist.combinations[source.combination].media.
     indexOf(media_id), 1);
  ambilist.combinations[source.combination].
     media_playing.push(media_id);
  var next_player_id = players_pq[0];
  var next_player = players[next_player_id];
  ambilist.media[media_id].player = next_player_id + '';
  if (next_player.master == true){
    media_playing.push(media_id);
    next_player.media_playing.push(media_id);
    players_pq.sort(compare);
    playAmbiPlugin(media_id, source, doneAmbiPlugin);
  } else {
    next_player.media_playing.push(media_id);
    players_pq.sort(compare);
    playSlaveAmbiPlugin(media_id, source, next_player);
  }
}
```

Listing 4.8: Media player `playAmbiMedia()` function

The `doneAmbiPlugin()` function updates the bookkeeping variables after a media object has been played, by taking the following actions:

- If the media was played by the master player, the media object still has to be deleted from the master's `media_playing` array.
- If the media object still exists in the ambilist — it may have been deleted in the mean time — we update the related combination and mark the media object as done.
- Check if there are any other media objects playing that are related to the users from the combination that contains this media object. Any user that does not have another interesting object playing are moved from `users_playing` to `users_to_play`.
- Find the media player that was playing this media object and remove the object from the player's currently playing media. The priority queue is resorted.
- Invoke the `playAmbi()` function to play the next media objects if required.

### Content Sizing

At some point, a media player may be playing more than one media item simultaneously. As explained in Section 3.6.6, the content will be automatically resized to fit the screen. The `screen_resize()` function in Listing 4.9 is called after every addition or removal of media items on a media player. The function properly resizes all media items that are currently playing on that player. In the case that a new media item is to be played, `screen_resize()` may be called right before the new item is added to the player. The function returns the dimensions of the last container, which does not exist yet at that point, so that the container for the new media item may be immediately created with the proper size.

```
function screen_resize(){
  var n = media_playing.length;
  var n_resized = 0;
  var rows = Math.round(Math.sqrt(n));
  var obj = {};

  for (var i=rows; i>0; i--){
    var cols = Math.ceil((n - n_resized) / i);
    var h = ambient_screen_size.height / rows;
    for (var j=0; j<cols; j++){
      var w = ambient_screen_size.width / cols;
      if (j == cols - 1){
       w = ambient_screen_size.width - ((cols - 1) * w);
      }
      if (n === n_resized + 1){
        // this is the last object, return values
        obj.width = w;
        obj.height = h;
      }
      var elem = media_playing[n_resized];
      $("#container_" + elem).width(w);
      $("#container_" + elem).height(h);
      n_resized++;
    }
  }
  return obj;
}
```

Listing 4.9: Media player `screen_resize()` function

#### Media Player Management

A media player may switch from context-aware mode back to static mode for one of the following reasons:

- The media player is a slave player and is no longer required by the master player as there is no longer enough content available for all players in the group. This may happen when, for example, a number of users has left the context.
- The media player is a slave player and was instructed by the master player to switch to static mode, because the context for the media player group is empty.
- The media player is the master player and switches back to static mode, because the context for the media player group is empty. A master player may only switch to static mode after it has first reset all active slave players by sending an `ambislavereset` message.

## 4.3 Content Management System

The content management system allows the administrator to add media, create static playlists, manage media players and define which plugins are available to users. End users may manage their profile and define their interests and relevant plugins.

### 4.3.1 Media Players

The administrator defines new media players in the content management system. The system requires a name, room, and address for the player. Figure 4.1 shows a list of media players in the content management system. When the media player is created, a unique PIN code for that player is generated. Afterwards, the administrator may run the web browser that will represent this media player and navigate to the media player URL, which is `http://server/player` where `server` represents the host name of the content management server. The first time, the administrator will be asked to enter the generated PIN code for that media player. When the code is entered correctly, a permanent cookie will be set in the web browser, containing the player ID. From that moment on, the player is recognised by the server by means of the information in the cookie. The media player code that is running in the browser, will start a WebSocket with the server, which the administrator can then use to cast playlists to the media player.

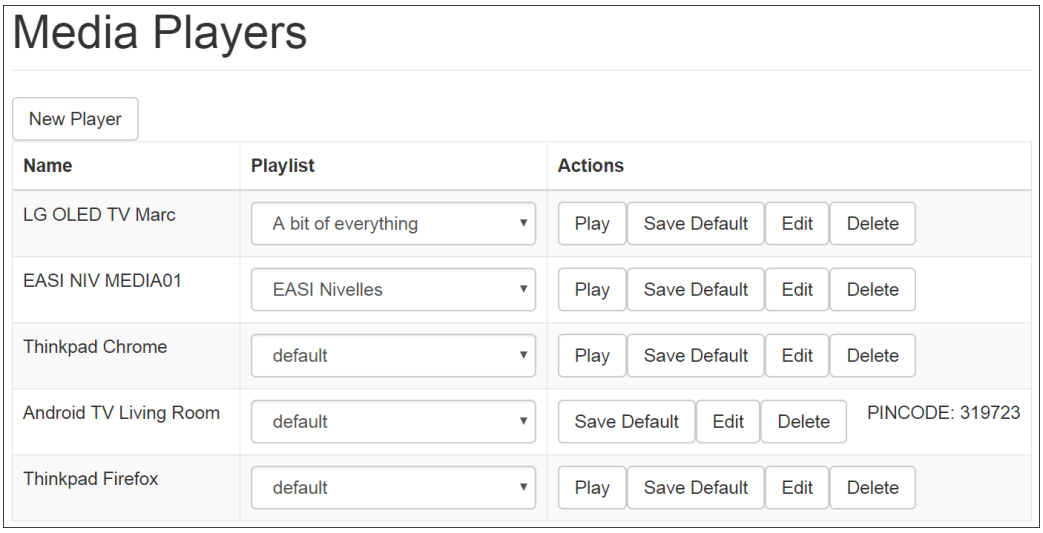



Figure 4.1: CMS: Media players

### 4.3.2 Media Player Groups

As stated in Section 3.2.3, media players may join media player groups for one of the following reasons:

- The media player should be part of a group of media players that will cooperate in context-aware mode.
- A media player group is created for administrative purposes, as playlists may also be pushed to all members of a media player group at once, instead of only individual media players.

As not every media player group is created for context-aware purposes, the administrator may choose to enable or disable context-aware mode for any group. The media player group definition also requires the appropriate `timeout` and `freshness` values that we explained in Section 3.6.7.

### 4.3.3 Sensors

Any device that can contribute dimensions to the context of arbitrary media player groups is considered a sensor. Sensors must also be registered in the system by the administrator, so that a unique ID is generated for each of them. The sensor must include their ID in any information it posts to the content management server. Section 3.7 specifies how a sensor may contribute information.

### 4.3.4 Media

The system administrator may create media objects that are to be included in static playlists. They must first choose the type of media item they wish to create, then we use JSONForm to generate a form that includes the necessary properties for a new media item of that type. Figure 4.2 shows a list of media items of different types that have been created in the content management server. Figure 4.3 represents two generated forms for media items of different types, namely Slideshow and SVN.

### 4.3.5 Playlists

Playlists are collections of media items that will be played in a specific order. The Playlist Designer may be used to create new playlists. First, at least one new container with relative dimensions must be added to the display. If the playlist should be played entirely fullscreen, a container with a height and width of 100% must be added. For each container, the administrator must specify the media items that should be played in that container. When a media item is added to a container, a new form is generated with JSONForm so that the options for that item may be entered. We implemented a drag-and-drop feature to be able to easily change the order in which the media items should be played.

Not only media items may be included in containers, but also other playlists can be added as sub-playlists. This way, a playlist with one fullscreen container may still split the screen in multiple containers at some point, if there is a multi-container sub-playlist included.

### 4.3.6 Credentials

In Section 3.4.1 we explained that some plugins may require credentials to be able to access some resource and retrieve information. Depending on the plugin, credentials may be entered on the administrator level and/or the user level. The SVN plugin that we implemented (see Section 4.4.1), requires user credentials for each SVN repository. However, if a read-only administrator account exists on the SVN server that has access to every repository, the plugin may be implemented to configure credentials on the administrator level. The administrator may save the credentials in the content management system and link them to plugins. This way, credentials may be linked to multiple plugins simultaneously, which facilitates password changes.

Media

New Media

| Name | Type | Actions |
|---|---|---|
| Adfinity | youtube | Edit Delete |
| Big Buck Bunny | video | Edit Delete |
| Blindspot Wallpaper | image | Edit Delete |
| CNN Top News | news | Edit Delete |
| EASI Blog | rssfeed | Edit Delete |
| Joeri's Art | slideshow | Edit Delete |
| Weather Tessenderlo | weather | Edit Delete |

Figure 4.2: CMS: Media

Media

Type: SVN

SVN Repo: https://wise.vub.ac.be/svn/myuser-thesi
The URL to the SVN Repo.

Name: SVN Thesis Joeri
A name to identify the media.

User: myuser
The user to connect to SVN.

Password: ••••••••
The password to connect to SVN.

Save

Media

Type: Slideshow

Name: Joeri's Art
The name to identify the media.

Album: joeri/art
The album to use for the slideshow.

Interval: 5
The time that a single picture is displayed.

Transition: fade
The transition method.

Save

Figure 4.3: CMS: Media forms

### 4.3.7 Plugins

The administrator may decide which plugins will be enabled in the system and made available to end users. When the administrator enables a certain plugin, the plugin may require admin configuration. Again, we use JSONForm to generate a form with the required fields.

### 4.3.8 Rules

The rule engine that we described in Section 3.8 may also be managed through the content management system. In the case of an external rule engine, it must be managed via an external interface. Arbitrary rulesets may be added by the administrator. At least one rule and one action are required for each set. First, the type of rule must be selected in a drop-down list, which is based on the existing rule templates that are created in the system. Then, JSONForm is used to generate a form to specify the required information for a rule of the selected type. Multiple rules of different types may be added. The administrator also selects the logic for this rule set, imposing evaluation with an `AND` or `OR` operator. Appendix H.2 shows a time-based ruleset that disables the Traffic plugin for one player group. When the ruleset is satisfied, all configured actions are executed. An action implies the enabling or disabling of a number of plugins for a number of player groups. The administrator selects the operator, plugins and player groups that should be included.

### 4.3.9 User Profiles

Each user has their own profile in which they configure personal information and preferences. See Appendix H.1 for an example of a user profile. The user must configure their basic information, being their full name, user name, password and address. Next, the user chooses which plugins they wish to enable. For example, if the user is never interested in any weather information or does not use a SVN server, they may decide to not enable the Weather and SVN plugins. If they would like to see news items, then they can enable the News plugin in their profile. When a plugin is enabled, it may require some user configuration. JSONForm generates a form based on the plugin description to allow the user to enter the required information.

In Section 3.6.1, we explained how user interests are managed and why it is important to disambiguate them. In the user profile, the user may specify their interests. When they start typing a keyword, a type-ahead section shows the keywords containing the same string that are already in

the system. The user may click any of the suggested keywords to add the interest to their personal profile. If the keyword is not found in the content management system, the user may add a new keyword. The system will then search DBPedia and display the resources that were found on DBPedia for that keyword. The user may then take one of the following actions, based on the results:

- If the keyword is ambiguous and multiple results are returned, the user should select the result that represents the meaning of their keyword, if any.
- If the keyword is not ambiguous and exactly one result is returned from DBPedia, the user may add that resource as an interest, if the retrieved resource indeed reflects the intended interest.
- If no resource exists for the keyword, or none of the retrieved resources are valid for the keyword, the user may choose to add only the keyword to their interests, without a link to a DBPedia resource.

The latter action implies that the interest will never be processed by any plugin that has their interest candidates described with the `interests_filter` property. As this property contains a SPARQL query that returns interest candidates based on semantic data, any keyword without an attached DBPedia resource cannot be evaluated.

Lastly, users may also specify credentials in their profile in the same way that administrators specify global credentials. These credentials may be referred to in user configurations of plugins that require them. For example, a user in a company may enter their central directory credentials and link them to various plugins that retrieve information from other servers within the company like a mail server. If the company has implemented a password policy that requires the users to change their password periodically, the credentials must be adapted only once in the profile.

### 4.3.10 User Templates

A user template is a generic user profile that only contains interests. The administrator may create such a template and link it to the user profiles to inherit the interests of the template automatically. This way, the administrator has some influence over the content that is generated for the users. For example, if the system is used in an IT consultancy company, a template may be created that already contains interests about the different technologies that the company specialises in.

## 4.4 Plugins

We have implemented a number of plugins for our content management system. A selection of those plugins will be described in more detail in this section. Our selection is based on the interest candidates description method and the user configuration requirements. Table 4.2 shows the different values for each selected plugin.

| Plugin | Interest candicates | User configuration |
|---|---|---|
| SVN | none | credentials |
| Traffic | none | basic |
| News | `all_interests` | basic |
| Weather | `interests_filter` | basic |
| Distrowatch | `interests` | none |

Table 4.2: Implemented plugins

### 4.4.1 SVN

The SVN plugin shows information about the last commit in an SVN repository. The output of this plugin is not based on user interests, but entirely on user configuration. The `user_config` property of the plugin description indicates that the user may provide a list of SVN repositories by providing the URL and appropriate credentials for each of them, as shown in Listing 4.10.

### 4.4.2 Traffic

The traffic plugin shows the route and traffic information from the current location of the user, being the location of the media player, to their home address. The plugin does not require any user configuration or interests. It only requires the address of the user, which is configured in their user profile. In static mode, the traffic media items require a source and destination address to show the traffic information for. The plugin is implemented using the Google Maps JavaScript API[6].

[6] `https://developers.google.com/maps/documentation/javascript`

```
"user_config" : {
  "repos" : {
    "title" : "Repositories",
    "description" : "Add the repositories you want to
       see updates for.",
    "type" : "array",
    "items" : {
      "type" : "object",
      "title" : "Repository",
      "properties" : {
        "repo" : {
          "title" : "Repository URL",
          "description" : "The URL of the repository.",
          "type" : "string",
          "default" : "",
          "required" : true
        },
        "credentials" : {
          "title" : "Credentials",
          "description" : "The credentials for accessing
             this repository.",
          "type" : "string",
          "enum" : [
            "$CREDENTIALS$"
          ],
          "required" : true
        }
      }
    },
    "required" : true
  }
}
```

Listing 4.10: SVN plugin user configuration description

### 4.4.3 News

The News plugin interacts with the RESTful web service of News API[7]. The API can return the top news or latest news of a specific news source, like CNN. Static media items require a specific news source to be selected to play the top news for. For context-aware mode, the user must complete the

[7] https://newsapi.org

user configuration for the plugin, which requires the user to specify which news sources may be accessed to search for relevant news. Any interest is a candidate for the plugin, therefore the `all_interests` property of the description is set to `true`.

The plugin processes all interests, including keywords without a DBPedia resource. We have searched for a semantic news source which would provide search results that are more accurate, but we have not found any. We did find the Semantic News Community Group[8], who describe themselves as "*A forum for exploring the intersection of W3C semantic technologies and news gathering, production, distribution and consumption*". However, they have no publications so far.

The plugin outputs a HTML document that displays the title and a short description of the article, followed by the editor, publisher and publication time. If a user is interested in reading the full article, they may scan the generated QR code that contains the URL to the article.

### 4.4.4 Weather

The Weather plugin retrieves weather information from the web service of OpenWeatherMap[9]. In static mode, a media item of this type simply requires a location to show the weather for. In context-aware mode, the plugin shows the weather for the home location and the interests of the user. The user configuration that the plugin requires, consists of two boolean parameters, one to indicate whether the plugin should show weather information for the home location and another to indicate the same for interests. Listing 4.11 shows the `user_config` property of the plugin description.

Any interest that is also a physical location is a candidate for the plugin. This type of requirement can only be made possible by using the semantics of the interests, which can be found in the DBPedia resources. The Weather plugin therefore uses the `interest_filter` property in its description, to differentiate between interests. It contains a SPARQL query in JSON format that the system may use to verify if a resource has a `lat` and `lng` property, implying that the interest is indeed a physical location. The resource of the interest that is to be verified is inserted in the place of the `$INTEREST$` variable. Appendix F shows the query in JSON format, as entered in the plugin description. In SPARQL, the query would be written as illustrated by Listing 4.12.

[8] `https://www.w3.org/community/semnews`
[9] `https://openweathermap.org`

```
"user_config" : {
  "home_location" : {
    "title" : "Home Location",
    "description" : "Show the weather for your home.",
    "type" : "boolean",
    "default" : false,
    "required" : true
  },
  "interest_location" : {
    "title" : "Interest Locations",
    "description" : "Show the weather for locations you
       are interested in.",
    "type" : "boolean",
    "default" : false,
    "required" : true
  }
}
```

Listing 4.11: Weather plugin user configuration description

```
SELECT DISTINCT ?location ?lat ?lng WHERE {
 ?location
   <http://www.w3.org/2000/01/rdf-schema#label> ?label ;
   <http://www.w3.org/2003/01/geo/wgs84\_pos#lat> ?lat ;
   <http://www.w3.org/2003/01/geo/wgs84\_pos#long> ?lng
      .
FILTER((STR(?location)) =
        "http://dbpedia.org/resource/resource_name")
} LIMIT 1
```

Listing 4.12: Weather plugin SPARQL query for defining interest candidates

We integrated the SPARQL.js[10] library in our system to parse the queries and interact with the SPARQL endpoint of DBPedia. For the front-end layout, we forked the Animated Weather App of Heather Tovey[11]. This weather widget is fully responsive and shows adapted animations based on various factors like temperature and wind speed.

[10] https://www.npmjs.com/package/sparqljs
[11] http://htovey.com/weather-app

### 4.4.5 Distrowatch

Distrowatch[12] is a website that shows news and the latest versions of various Linux distributions. Unlike the Weather plugin, Distrowatch cannot provide information about any resource that happens to be a Linux distribution. It only knows about a limited subset of Linux systems. Therefore it is pointless to use the `interest_filter` property in the description of the plugin, instead we use the `interests` property as Listing 4.13 illustrates. In this property, we include an array that contains the Linux distributions that the plugin will support, defined as interests and already including the correct DBPedia resource. The system will also save these interests in its internal database so that they will appear in the type-ahead field when a user searches for a keyword while adding interests into their profile.

```
"interests" : {
  "debian" : {
    "name" : "debian",
    "resource" : "http://dbpedia.org/resource/Debian"
  },
  ...
  "centos" : {
    "name" : "centos",
    "resource" : "http://dbpedia.org/resource/CentOS"
  }
}
```

Listing 4.13: Distrowatch plugin interests description

## 4.5 Context Processing

When the server receives a context contribution from a sensor, the `context` module in CastJS takes the following steps to update the context of the relevant media player groups:

- Retrieve the media player groups that the sensor may provide updates for.
- If the context for that group was empty before, elect a master player.
- Adapt the current context of each media player group with the new values.

[12] `https://distrowatch.com`

- Add any new users to the `newcomers`, add users whose timeout interval has been exceeded to the `leavers`, add users whose refresh interval has been exceeded to both the `newcomers` and `leavers`.
- If any new parameter values were different from the current values or if there are any `newcomers` or `leavers`, reprocess the rulesets that include this group in the actions.
- Pass the `newcomers`, `leavers`, context, master player, player group, current `ambilist` and rule actions to the `generateAmbilist()` function of the `ambi_manager` module.

After the context has been processed, the `ambi_manager` module takes the following actions to generate content:

- Process the `leavers`. All media items in the current `ambilist` that are only related to `leavers` are deleted from the list. Any user combination that includes at least one of the `leavers` is merged with the combination of users that remain in the context. For example, if users A, B and C are in the previous context and user C is leaving, then the content of combinations {A, B, C}, {A, C} and {B, C} will be merged with {A, B}, {A} and {B} respectively. Combination {C} will be removed entirely.
- Process the `newcomers`. For each new user, select the plugins that are enabled in their profile and call the `generate()` function of each plugin. This process will yield interesting media items for the `newcomers`. Next, we create new user combinations that include the `newcomers`, based on the combinations that already exist. If any of the newly generated media seem to already exist in the `ambilist`, then this means that the item is interesting to users that were already present in the context. Therefore we must move the item from its current user combination to a new combination that also includes the new user. Completely new items are added to combinations that only contain `newcomers`.

After generating the new `ambilist`, we evaluate the contents of the `ambilist`. If the `ambilist` does not contain any media, this means that there are no users present in the context or that the system was unable to generate interesting content for the users. In this case, the system will send an `ambistop` message to the master player, which will instruct all players in the group to switch to static mode if this is not the case already. Otherwise, we will send the `ambilist` message to the master player, which will then further process the new list locally, as explained in Section 4.2.2.

# 5 Using CastJS

## 5.1 Configuring a Media Player

A number of steps must be taken to add a new player to the system:

1. Go to the 'Players' section of the content management system and click the 'New Player' button.

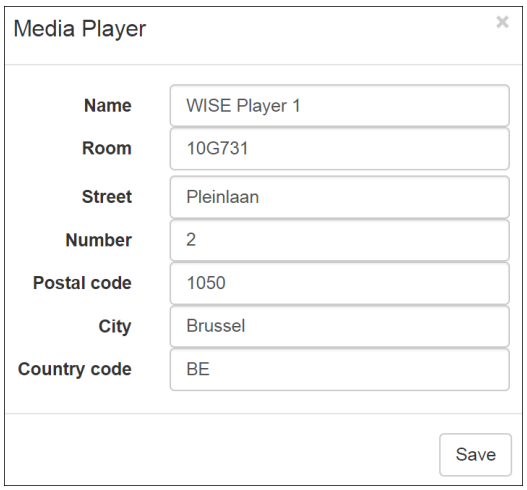



Figure 5.1: CMS: Form to register a new media player

2. As shown in Figure 5.1, fill in the name, room and address for the new player. Then save the configuration.

3. The new player is now added to the list of media players and a unique PIN code is displayed for this player on the far right of the page.

4. The new media player has no default playlist configured at this time. In the 'Players' section, open the drop-down list next to the name of the new player and select a playlist. Then click the 'Save Default' button to set the selected playlist as the default for this player.

5. Now open the web browser that will serve as this media player and navigate to the player URL of the content management system via `http://server/player`, where `server` represents the host name of the content management server. As this is the first time that this browser connects to the server, a PIN code will be requested. Enter the PIN code as found in the content management system and click 'OK'. The player will then immediately fetch its default playlist from the server and start playing. Notice that the PIN code for this player has now disappeared from the content management system and a 'Play' button is now visible.

In most cases, the administrator will want the web browser that acts as a media player to start automatically in fullscreen mode with the underlying operating system. The required configuration depends on the browser software and operating system. For example, to start Mozilla Firefox in fullscreen mode, an additional add-on like RKiosk[1] is required. For the Google Chromium browser, the following parameters may be used to start the browser in kiosk mode:

```
chromium-browser --kiosk --disable-session-crashed-bubble
--disable-infobars http://server/player
```

One must also make sure that the display of the used hardware does not enter sleep mode and that the screen saver is turned off. On a Linux operating system, we also recommend using the Unclutter[2] daemon to hide the mouse cursor when it has not moved for a number of seconds. Usually the cursor is located in the middle of the display when the system has just started.

[1] `https://addons.mozilla.org/nl/firefox/addon/r-kiosk`
[2] `https://unclutter.sourceforge.io`

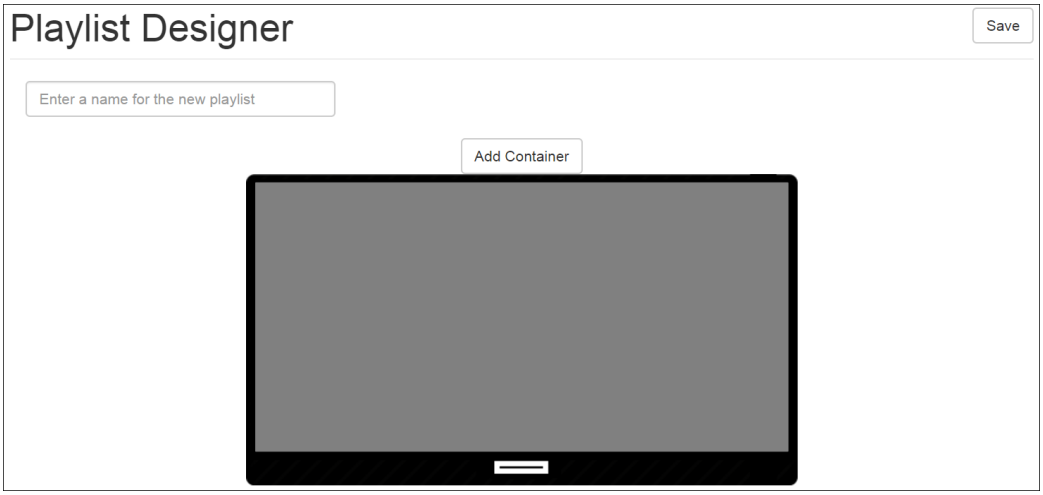



Figure 5.2: CMS: Playlist Designer with empty display

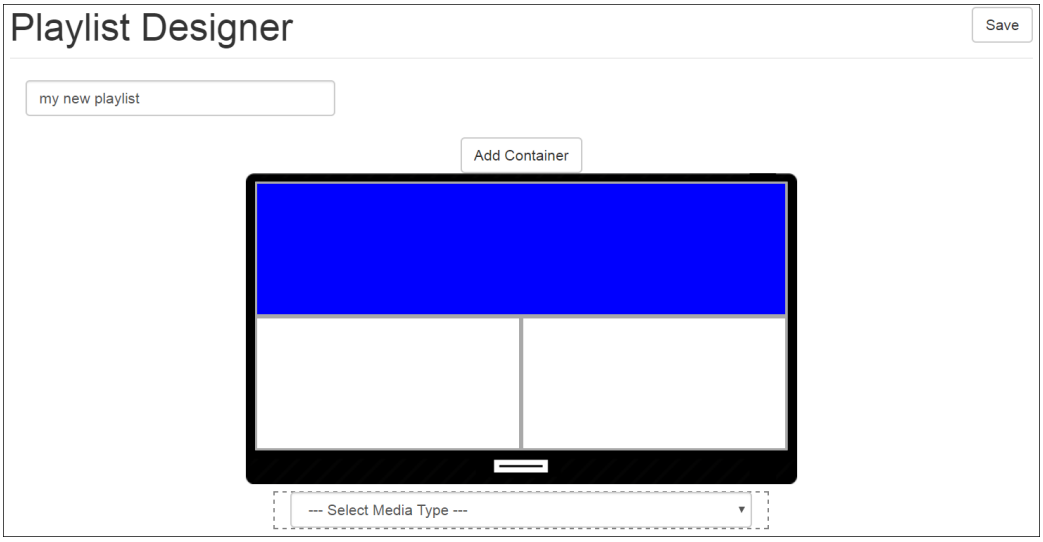



Figure 5.3: CMS: Playlist Designer with three containers, top container selected

## 5.2 Designing and Pushing a Playlist

To create a new playlist, the following steps must be taken:

1. Go to the 'Playlists' section of the content management system and click the 'New Playlist' button. This opens the Playlist Designer with an empty configuration, as shown in Figure 5.2.

2. Enter a name for the new playlist.

3. Add at least one container to the display, by clicking the 'Add Container' button and entering the relative size for each container. Figure 5.3 shows a display with three containers of sizes 100% x 50%, 50% x 50% and 50% x 50%. The top container is selected, meaning that we can now add media items to that container.

4. For the selected container, add the media items that should be played in that container. Select a media type from the drop-down list below the display. The system will then generate a form with a drop-down list of all media items of that type that exist in the system, as well as the possible options for that media type. Figure 5.4 shows a form for the SVN media type.

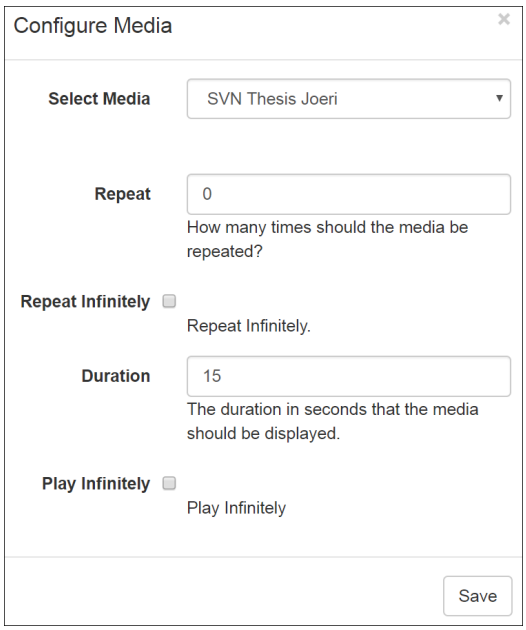



Figure 5.4: CMS: Adding a media item to a container in Playlist Designer

5. Repeat the previous step for all required media items in all containers. Figure 5.5 shows the top container that contains two media items at this point, an Image and an SVN item. The order of the media items may be altered by drag-and-drop.

6. Save the new playlist by clicking the 'Save' button. The playlist may now be pushed to a media player or player group on the 'Players' or 'Player Groups' sections of the content management system. Select the playlist in the drop-down list of the relevant player or group and click the 'Play' button. Appendix H.3 shows an example of what a playlist output might look like in the media player.

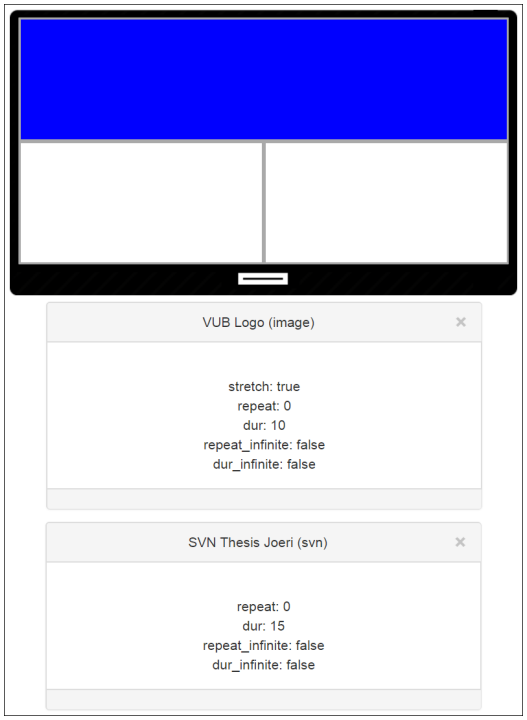



Figure 5.5: CMS: Playlist Designer with selected top container, bearing two media items

## 5.3 Adding a Plugin

A new plugin must be registered in the content management system before it can be used. There are two main steps that must be taken to achieve this; add the plugin description in JSON format into the database and copy the plugin

files to the correct locations. The plugin description must be added to the `mediatypes` collection in the MongoDB database. The plugin files may be copied to three distinct locations, as illustrated in Figure 5.6. As explained in Section 3.4.2, a plugin must implement a `play()` and `generate()` function. These functions must be added to a JavaScript file that carries the name of the plugin and the `js` extension. This file must be placed in a folder with the same name in the `plugins` directory of the content management system's NodeJS module. The HTML output of the plugin must be placed in an Embedded JavaScript file that also carries the name of the plugin and the `ejs` extension. Like the JavaScript file, it must be placed in a folder that is named after the plugin. This folder may be placed in the `plugincontent` directory under the `views` directory. Any external files that are referred to in the Embedded JS file, may be added in the `plugincontent` directory under the `public` directory, again grouped in a subfolder with the plugin name.

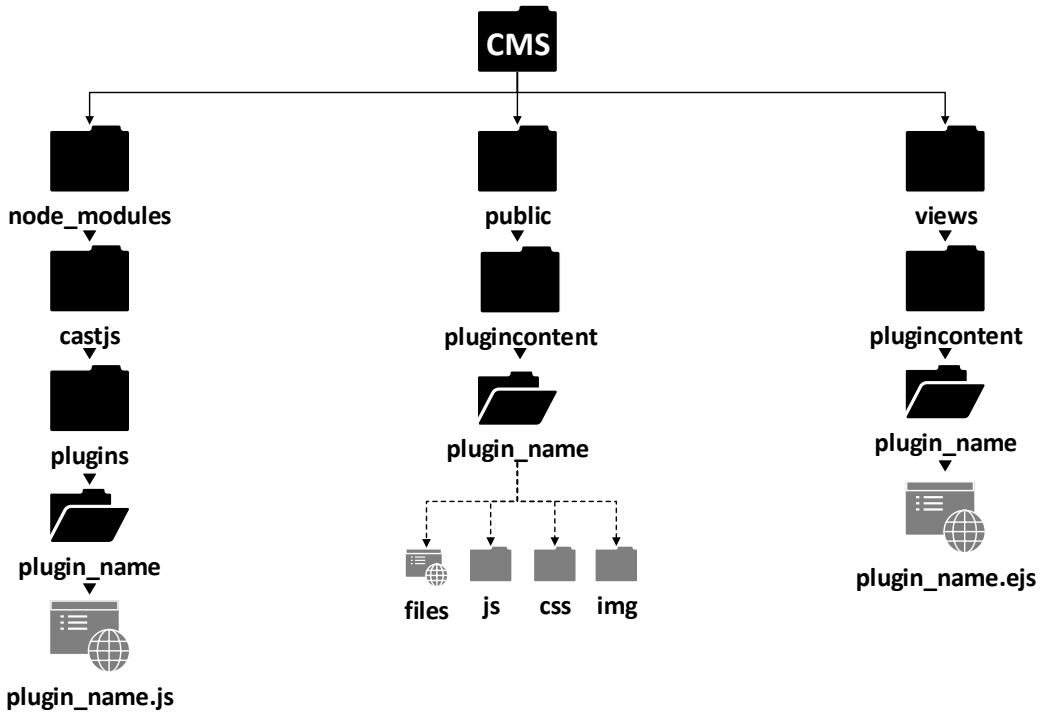



Figure 5.6: Folder structure for plugin files

When the plugin files and description are properly added, the administrator may navigate to the 'Plugins' section in the content management system, where the new plugin will be visible. Next to each plugin, a 'Scan Interests' button may clicked to start a scan of all interests that currently exist in the system which will evaluate the candidacy of each interest for that plugin.

## 5.4 Configuring a User Profile

Apart from basic personal information, users enable plugins and configure interests in their profile. See Appendix H.1 for an example of a user profile. To enable a plugin, the user may click the 'Enable' button next to the plugin name. If the plugin requires some specific configuration, a form will be displayed where the user can enter the required information. Plugin configuration may be edited by using the 'Edit' button and disabled with the 'Disable' button.

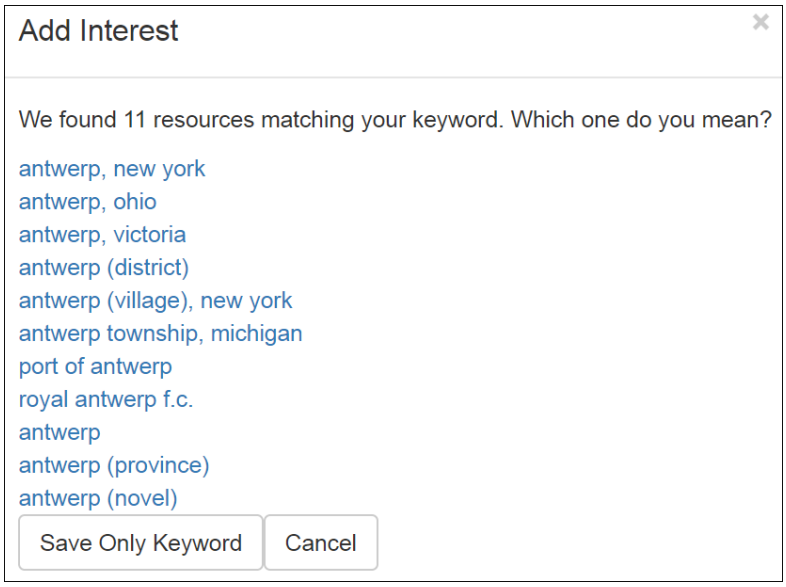



Figure 5.7: DBPedia search results for Antwerp

To add a new interest to the profile, the user may enter a keyword in the search field. If that interest already exists in the system, it will appear in a type-ahead list. The user may click a search result to add it to their interests. If the interest does not exist yet, the 'Add' button may be clicked. The system will then retrieve the possible resources from DBPedia. Figure 5.7 shows the search results for `antwerp`. The user may click the intended interest to add it to their profile. If the search results are empty or do not contain the correct resource, the 'Save Only Keyword' button may be clicked to only save the keyword without any semantics.

## 5.5 Creating Rules and Templates

Arbitrary rulesets may be created based on the rule templates that are defined in the system. These templates are defined in the `rule_engine.js` file. Appendix D contains a number of sample templates. External rule engines may be enabled in the `rule_engine_external.js` file.

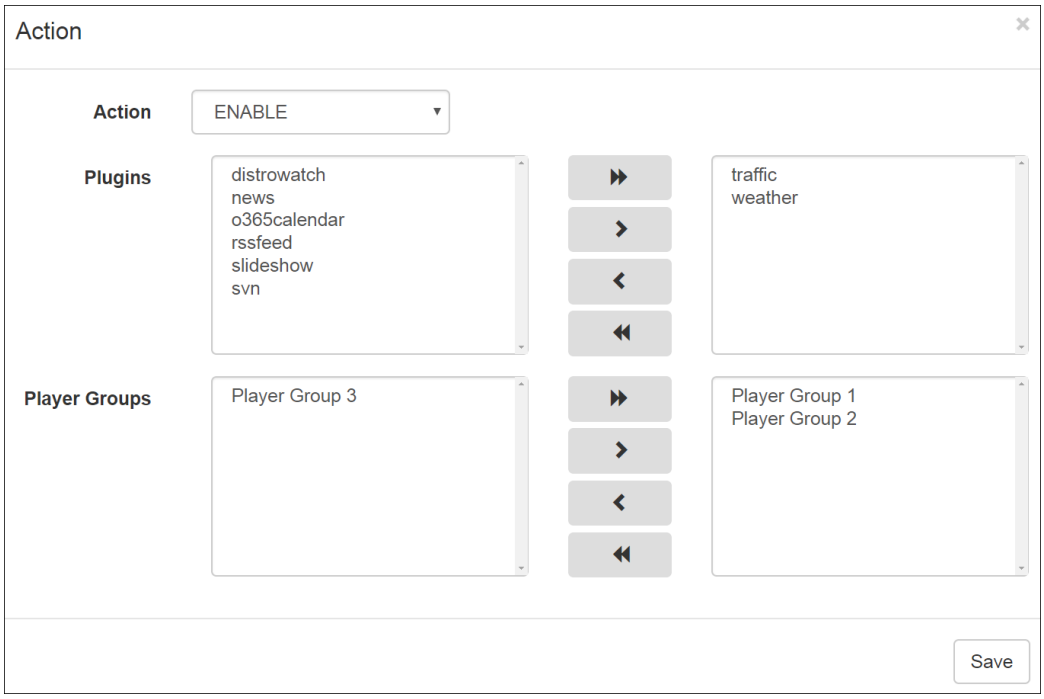



Figure 5.8: Ruleset sample action

To create a new ruleset, follow these steps:

1. Open the 'Rules' section in the content management system and click the 'New Ruleset' button.

2. Enter a name for the ruleset, choose whether the rules should be enabled or disabled and choose the required logic.

3. Click the 'New Rule' button to add a new rule. Select a rule type; rule types are loaded from the templates. Then, a form will be displayed where the operator and required parameters may be configured. Repeat this step for all rules that need to be a part of this ruleset.

4. Click the 'New Action' button to add an action. Select the action operator and the plugins and player groups that are included in this

action. Figure 5.8 shows a sample action which disables the Traffic and Weather plugins for two player groups. Repeat this step for all required actions.

5. Save the new ruleset.

# 6 Evaluation and Use Cases

## 6.1 System Evaluation

In Section 2.2 we discussed various context-aware digital signage systems based on context awareness, configurability, content and portability. In this section, we will evaluate our implemented system, CastJS.

### 6.1.1 Context Awareness

The context may contain any possible dimensions. These parameters may be used in rulesets to evaluate certain conditions. They are also passed to the plugins so that they may consider them while generating media. Therefore, the system provides any means necessary to generate tailored playlists for any purpose.

### 6.1.2 Configurability

The administrator may enable certain plugins in the system and create static playlists and rulesets. The user has a profile in which they may provide their interests and also manage plugins. The configurability of the media content itself is related to the implementation of the plugin. However, the system

does provide the necessary means for the plugin developer to create a highly configurable plugin.

### 6.1.3 Content

The system is able to play some trivial media types with built-in functions. Additionally, a framework for developing custom media types in the form of plugins is provided. Therefore, our system can theoretically play any type of media, as long as a plugin exists for it.

### 6.1.4 Portability

Our media player runs natively in any recent JavaScript-enabled web browser without extensions or add-ons and is therefore fully portable. Any device with a web browser installed is able to act as a media player.

## 6.2 Other Advantages and Drawbacks

Our media player is fully web-based and may run in any web browser, which makes the solution completely hardware and partially software independent. However, we must conclude that this approach also has some drawbacks.

### 6.2.1 Hardware Management

Although our solution is hardware-independent, a web browser does not natively have full access to the underlying operating system. Therefore, the manageability of the system is limited to the media player software itself. If the administrator wishes to perform tasks like rebooting the media player hardware, they must have other means to access the operating system remotely.

### 6.2.2 Hardware Requirements

Depending on the types of media that will be played, the hardware that runs the media player may require a minimum amount of resources. If the administrator creates a playlist with four containers that each play a high resolution video simultaneously, the hardware must be able to render those videos fluently.

We have tested our solution on a Raspberry Pi 3 Model B[1], as it is a small device that is easily connected to any display via HDMI and may be powered by one of the display's USB ports. In our tests, the Raspberry Pi handled any media types flawlessly, except for video resources, as it was incapable of rendering video without interrupting playback frequently. Further investigation indicated that no web browser stacks exist for the Raspberry Pi that offload graphics rendering on the Graphical Processing Unit (GPU) of the system. Instead, everything is processed by the Central Processing Unit (CPU) of the device.

### 6.2.3 Caching

All media content is downloaded into the web browser by the media player software. Thanks to the web browser cache, this content is only downloaded once, as long as it has not changed on the server side. Although the caching feature of the web browser is very convenient for the perfomance of our solution, our media player is unable to actively manage this cache. The maximum size of the cache and retention of the data must be configured in the browser settings, which are not accessible by the media player software. We conclude that the performance of our solution partially depends on the configuration of the web browser that runs the media player.

Another option to cache content on the media player, would be to save it directly to the file system of the underlying operating system. However, as JavaScript does not have file system access, a non-web-based media player software does have that advantage.

### 6.2.4 Fullscreen

The web browser that runs the media player should ideally be in fullscreen mode, so that any toolbars or menus of the browser software are invisible on the display. We have investigated the possibility to use the HTML5 Fullscreen API[2] to force the browser in fullscreen mode when the media player software is loading. However, fullscreen may only be triggered by an authentic user based event, like a mouse click or key press. Simulated events, by JQuery for instance, are not able to trigger fullscreen mode for security reasons. This method might be used by attackers to impersonate other websites for phishing attacks. Therefore, we cannot force the browser in fullscreen mode with our media player software. Instead, the web browser must be started

[1] `https://www.raspberrypi.org/products/raspberry-pi-3-model-b`
[2] `https://developer.mozilla.org/en-US/docs/Web/API/Fullscreen_API`

with an extra parameter or must be provided with an extension or add-on. For example, we used the RKiosk add-on[3] to start Mozilla Firefox web browsers in fullscreen.

### 6.2.5 External Webpages

CastJS has a native Webpage media type for adding external webpages as media items. A regular non-web-based digital signage system would use its integrated web browser for rendering that webpage, while using other applications for rendering images and video. As our media player is already web-based, we have to render external webpages inside an `iframe` tag. If a webpage has the `X-Frame-Options`[4] HTTP header configured, or the newer `Content-Security-Policy` header with the `frame-ancestors`[5] directive, then some additional configuration is required as these headers may deny the media player to embed the page. This entails to either configure these HTTP headers on the external website to allow the media player to embed the content or request the webpage via a proxy that strips these headers from the payload.

## 6.3 Privacy

Some plugins may expose information in context-aware mode that a user considers private. Our rule engine, explained in Section 3.8, aids in protecting privacy on a high level, by disabling plugins entirely based on certain conditions that may be privacy-related. Privacy protection on a per-media-item-level must be ensured by the plugin. For example, if one were to implement a plugin that shows calendar information, the design may be based on the PriCal system that we evaluated in Section 2.2.3. This plugin may then show or hide certain appointments based on the context information. Furthermore, plugins may implement the concept of ambient zones [27]. An advanced camera sensor, like the Microsoft Kinect, is able to detect the distance between a user and the sensor. Based on this information, a plugin may adapt the font size of certain information so that only users that are close to the screen are able to read it.

Privacy concerns are not only related to personal information that is displayed. Sensor data may also pose a privacy threat when not carefully handled. In our system, we use a camera to identify users. We delete the

[3] `https://addons.mozilla.org/nl/firefox/addon/r-kiosk`
[4] `https://tools.ietf.org/html/rfc7034`
[5] `https://www.w3.org/TR/CSP/#directive-frame-ancestors`

captured images as soon as the processing of them is completed, but other sensors may save them in another location. Smailagic & Kogan [34] also point out that the history of context information may be used to profile the behaviour of users, like tracking their location history. In our system, a malicious plugin may be able to do that.

## 6.4 Use Cases

### 6.4.1 WISE Lab

We have installed a proof of concept setup of CastJS at the WISE[6] lab of the Vrije Universiteit Brussel. The proof of concept consists of a content management server, two media players and one camera.

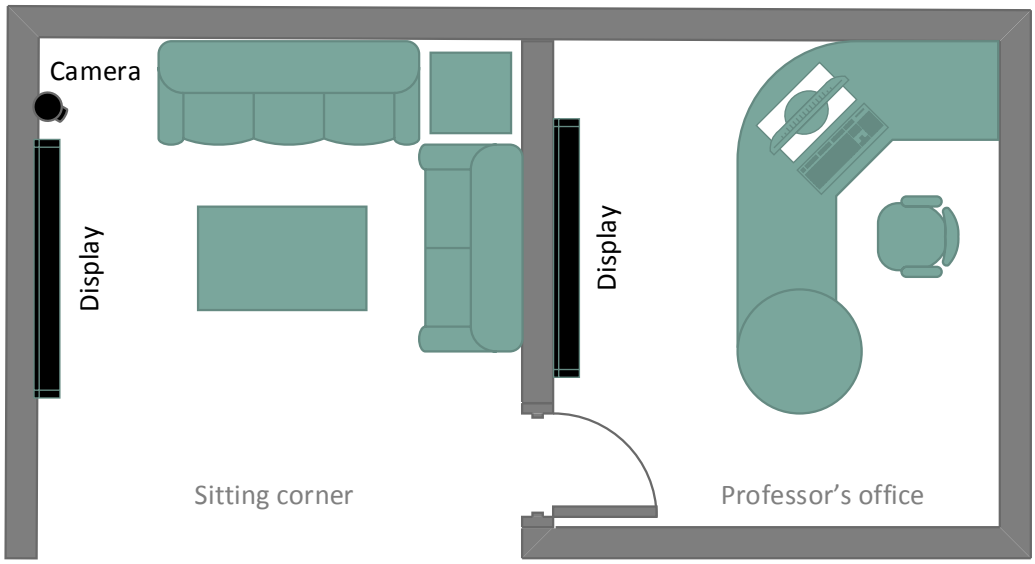



Figure 6.1: Proof of concept layout in the WISE lab

The device that we installed in the sitting corner of the lab is an Intel NUC[7] mini PC, which acts as both media player for the display and content management server. The camera is an ordinary HP HD 4310 webcam[8] that is connected via USB to the PC. We have installed an Ubuntu 16.04 operating system with the facial recognition software, NodeJS and MongoDB with CastJS and a Mozilla Firefox web browser with the RKiosk add-on. In the

[6]Web Information Systems Engineering

[8]`https://support.hp.com/gb-en/product/HP-HD-4310-Webcam/5273433/model/527343`

separate office, we could use a less powerful device, as it should only run a web browser that connects to the server on the Intel NUC. Therefore we have installed an instance of our media player on a Raspberry Pi 3 Model B.

We have created two player groups that each contain one display. The USB webcam provides context information to the content management server. We have added this camera as a sensor and linked it to the player group that contains the display in the sitting corner. Hence, only this display will be used for context-based content. The display in the separate office will only be used in static mode.

The goal of this setup is to display general news, pictures and other content on these screens. When members of the WISE lab are detected by the camera, the display in the sitting corner will switch to context-aware mode and show interesting content for these users, based on their profile.

### 6.4.2 Corporate

We have designed a fictive, more elaborate, yet simplified use case for CastJS. The setting for this case is a production plant. Figure 6.2 shows the floor plan of the building, see Appendix G for a full page version.

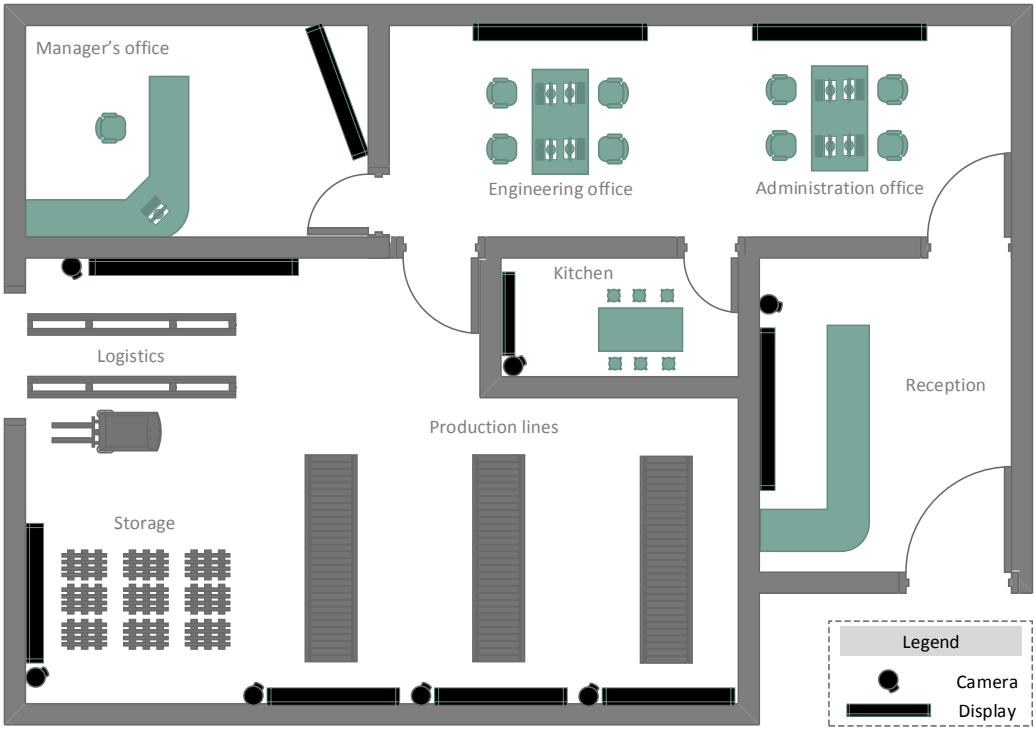



Figure 6.2: Production plant floor plan

The setup contains ten displays and seven camera sensors. The displays are placed in different areas of the plant:

- Reception: One display and camera are placed at the reception. General information, photos and videos of the company are displayed to inform any visitors that arrive. The camera detects employees of the plant, after which the display switches to context-aware mode. In the morning, the screen shows the employees' next meeting or task of the day. In the evening, traffic information is shown about the route to the home address of the user.

- Office: The office area is divided in two islands, one for the production engineers and one for the administration officers. Both teams have their own display near their island that shows a static playlist with information that is relevant to their function. For example, the administration officers see an overview of the stock and the ongoing and upcoming logistics operations, the engineers see an overview of the andon[9] information of the production lines and the operation state of all machines.

- Manager's office: The manager has their own display with more general information about the state of the production process and present employees. They may also see a number of live video feeds of the production hall.

- Kitchen: As the kitchen is normally for employees only, the screen is switched off if no users are detected by the camera. Therefore, this display operates only in context-aware mode. The administrator has configured a ruleset in the content management system, so that more recreational content is displayed, instead of business content. This includes news, weather, upcoming company events and new employees.

- Production hall: The production hall is divided in 3 sub-zones. First, there are the production lines. Each line has their own screen, which displays the andon for that line in numbers as well as a graphical representation. This allows the employees of the production line to constantly keep track of their own productivity. Each display is also accompanied by a camera. When this camera detects that an engineer is also present at the production line, the screen will split in two zones, adding the overview of all production lines as is displayed on the display in the engineering office. This way, the engineer can always dispose of

[9] https://en.wikipedia.org/wiki/Andon_(manufacturing)

a complete set of information, even when they are walking around in the production hall. The andon of the production line will go back to fullscreen when the engineer leaves the context.

The other zones in the production hall are storage and logistics. Here we use the same principle as for the production lines. On the display in the storage area, information about the stock is displayed. In the logistics area, we show the pick-up and delivery schedule. Also these displays are accompanied by a camera. When an administration officer is in the detection zone of any camera, the screen will be split in multiple containers to add overview information. Also engineers may be present at the logistics area to inspect any robots that may be present there. Then an extra container will be added on the display for the engineering overview.

## 6.5 User Evaluation

We have conducted a user evaluation with six participants; four male and two female between 28 and 56 years of age. All participants have received an introduction to CastJS, as well as the uses cases from Section 6.4, so that they are aware of how the system works and where it may be used. Next, we have set up a simple scenario with one display in an area that represented the break room of an office building. The display was playing a static playlist with random media. The participants were given the opportunity to create a profile in the content management system and add a selection of their personal interests. Each user had the News, Weather and Traffic plugin enabled. We then forced the media player in context-aware mode, by posting information about the present users to the context module of the content management system. We used a virtual sensor to facilitate our setup. The participants should then see content on the display that they find interesting.

After the scenario, the participants were asked to complete a questionnaire. We have based our questions on the categories of the Usefulness, Satisfaction, and Ease of use (USE) questionnaire [24] and extended them with questions about the accuracy of the provided media in relation to the interests of the users, as this is also an important aspect for context-aware and self-adaptive applications [14]. We have asked the participants to indicate their level of agreement with six statements on a seven point Likert scale, where one means strongly disagree and seven means strongly agree. A textual affirmation was requested for each score.

The statements that we used in the questionnaire and their average score by the participants are as follows:

1. (Accuracy) The information that was displayed was accurate. The provided media reflected the interests that I configured in my profile.
   Score: *4.8*

2. (Accuracy) I feel that my user profile provides enough configuration options to influence the accuracy of the displayed information.
   Score: *4.8*

3. (Ease of use) It was easy to locate the information that was generated for me on the screen estate.
   Score: *4.3*

4. (Usefulness) I see the system as a useful contribution to my everyday life.
   Score: *5.3*

5. (Satisfaction) I would install this system at my company.
   Score: *5.8*

6. (Satisfaction) I am globally satisfied with the system.
   Score: *5.2*

| **Statement** | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| **Participant 1** | 4 | 3 | 2 | 5 | 5 | 5 |
| **Participant 2** | 5 | 6 | 7 | 5 | 6 | 5 |
| **Participant 3** | 4 | 4 | 6 | 5 | 5 | 4 |
| **Participant 4** | 6 | 7 | 2 | 7 | 6 | 6 |
| **Participant 5** | 6 | 5 | 4 | 5 | 6 | 5 |
| **Participant 6** | 4 | 4 | 5 | 5 | 7 | 6 |

Table 6.1: Individual user evaluation scores per statement

The individual scores of each participant may be found in Table 6.1. Based on these scores and the extra feedback that we received, we conclude that the overall reaction towards the system is positive. The participants are able to locate the content that the system has generated for them, however, this is influenced by the number of containers on the screen. If there are more containers, it may take a bit more time for someone to locate their interesting content. Therefore, the user may also not have enough time to

read the presented information before the duration for that media object has expired. The duration for a media object is decided entirely by the plugin and is not based on the number of other media items that are displayed. A more thorough evaluation should be performed to determine how the duration of media items and the number of containers may be correlated. In this evaluation, most users agree that a maximum of four containers should be displayed on a single screen. There were mostly five or six containers on the display in our scenario, which may explain the relatively low score for ease of use. We also noticed that a higher number of containers generates more movement on the screen, as media items are added and removed. Some users experience this as slightly irritating. Again, a display with up to four containers is acceptable.

Some participants also pointed out that they would only like to include interests for a subset of the enabled plugins in their profile. For example, a user may be interested in any news about Antwerp, but not the weather in Antwerp. There are a number of ways to provide this possibility. Users could configure interests for each individual plugin, but this would generate a configuration overhead and undermine the system's capabilities for interest candidacy selection. On the other hand, we could let the user exclude some interests for specific plugins, either in the user profile or by adding an interactive component to the media player through which feedback may be provided. In the next section we will elaborate on this possibility and other future work.

## 6.6 Discussion and Future Work

### 6.6.1 Optimisation of the Current System

The current system provides a high level of functionality, however it lacks some features to make it a full-fledged digital signage system. For example, a playlist scheduler could be added for static playlists. This is a common feature in other digital signage systems, but did not have much added value for our initial project. However, our rule engine could be further expanded for this purpose, so that static playlists could be enabled or disabled in the actions for a ruleset.

Furthermore, other features that are not directly related to digital signage systems may be proposed. For instance, when a user adds interests to their profile, the system may propose other interests to the user, based on semantic relations to other resources in DBPedia. If a user adds Leffe — the beer — as an interest, together with Stella Artois and the city of Leuven,

then they may also be interested in AB Inbev, the brewery of these beers that is based in Leuven. Figure 6.3 shows the relations between these interests, indicating that many relations refer to AB Inbev. The graph was rendered using RelFinder[10]. Our user interface provides all required functionality for our purpose, but could be more user-friendly. Another study may be performed to optimise the interface, for easily creating, managing and displaying resources such as playlists, media items and user profiles.

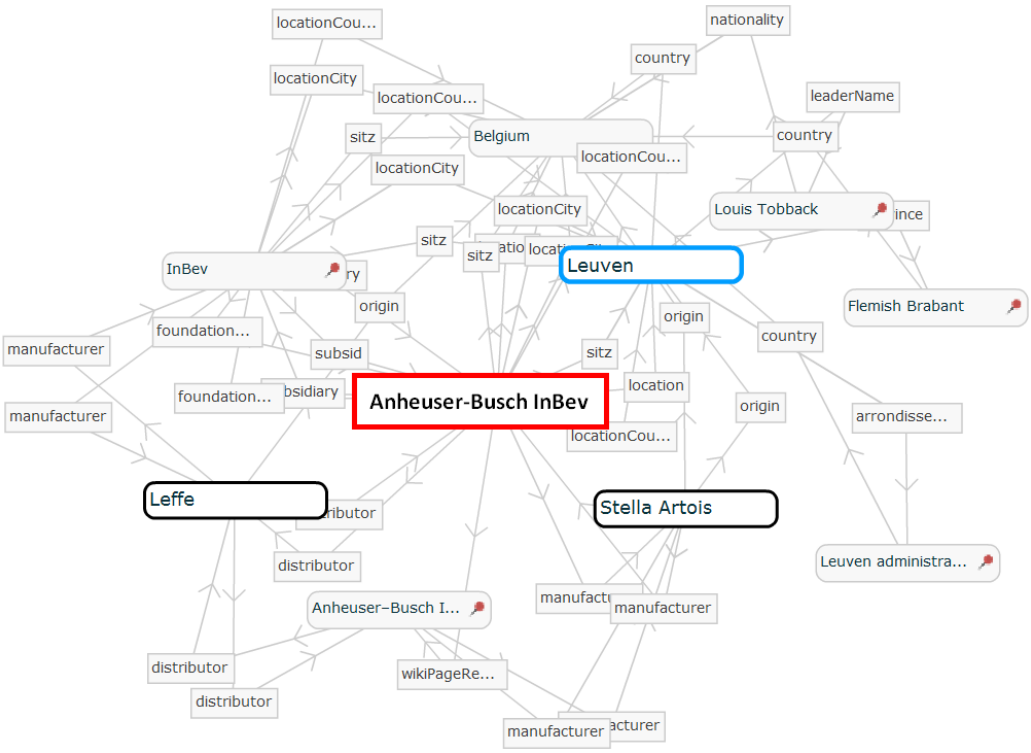



Figure 6.3: Semantic relations on DBPedia, simulated in RelFinder

### 6.6.2 Plugin Development

Every new plugin enriches the functionality of the overall system. We also see this concept in other types of software, like Nagios[11]. Nagios is a monitoring software that monitors the state of different aspects of IT infrastructure and other network-connected devices. For a server, it may check resources like the CPU, memory and disk space. However, Nagios is actually just the

[10] http://www.visualdataweb.org/relfinder.php
[11] https://www.nagios.org

scheduling engine, while plugins provide the actual functionality to carry out the different checks. Nagios Exchange[12] is a catalogue of plugins where developers may share their plugin code to monitor a specific type of device or resource. We see similar possibilities for our media player system. Developers may also write plugins for our system and provide them with high configurability. For example, an RSS plugin that parses the contents of an RSS feed may be developed in such a way that it is customisable through its `properties` and `options` descriptions for multiple use cases.

### 6.6.3 Interactive Components

Our system is able to adapt the displayed content to meet the interests of the present users, based on the current context. However, the users in that context are not able to actively control the content at a specific moment. Therefore, there are still some possibilities to be explored to also include interactive components into the system.

For example, a user has enabled various plugins like Weather and News in their profile. That user enters the context of a media player group so that this group switches to context-aware mode to show interesting content for this user. This content will probably contain news items and weather information, as the user has enabled these plugins in their profile. However, the media player decides which content will be played at which time and in which sequence. An interactive component would enable the user to force the system to play a specific type of content at a specific moment. This may be achieved with voice control for example. If a microphone were to be included as a sensor, the user may control the context and thus the media content with voice commands, like "show me the latest news" to force the player to show news items. These news items would still be based on the interests that the user configured in their profile. The system may even provide commands to overrule the user configuration, so that only news items of a specific source or subject are retrieved, regardless of the configuration in the profile.

Interactivity also provides the means for the user to give feedback on the generated media items. A user may indicate whether a media item is not interesting to them, for example by using hand gestures [9]. The content management system must then be expanded with a machine learning component to integrate this feedback in the generation of playlists.

[12] `https://exchange.nagios.org`

### 6.6.4 Elaborate User Study

We have particularly devoted our study to the technical side of the problem, complemented by a basic user evaluation. However, to thoroughly evaluate our system in terms of usability and accuracy of the generated context-based content, a more elaborate user study must be performed. This study may entail the setup of an environment like in one of the use cases of Section 6.4 in which we repetitively evaluate feedback of the users over a long period or use an audience measurement tool that is specifically designed for digital signage systems [26].

## 6.7 Conclusions

In this thesis, we have successfully implemented a web-based context-aware ambient media player system. The fully web-based system results in a number of advantages in terms of portability, but does have some drawbacks regarding management of the lower layers like hardware and the web browser that runs the media player. The software supports a multiple-media-player setup in which players may work together to display content that was derived from the current context, which takes full advantage of the screen estate in a specific area.

Arbitrary plugins may be developed and integrated into the system to enrich its features and the different types of content that it is able to handle. The developer may provide a description of the features and requirements of the plugin. This way, the system may be extended and configured to fit into any use case.

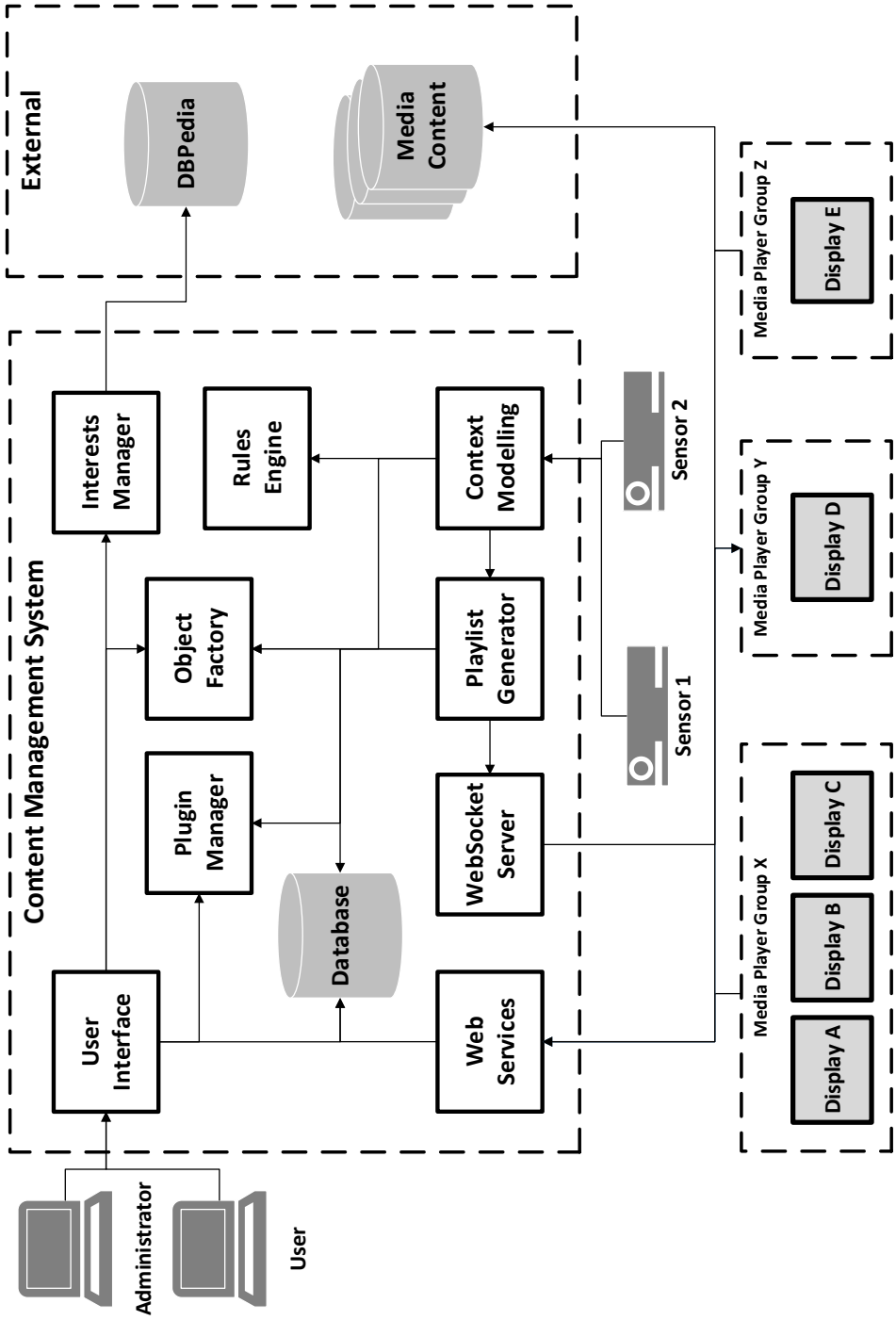
The system is able to generate content based on context information that is provided by sensors of any type. This context is parsed by the rule engine and the plugins. Therefore, the context may entail arbitrary dimensions, like present users, room temperature, local time and any other dimension that makes sense in the use case. The generated content is also based on user and administrator preferences. Users are able to influence the content that is displayed when they are present in a certain context. User profiles enable the user to configure their interests and the types of media that they wish to see. The accuracy of the generated media for a specific user heavily depends on the implementation of the plugins, as they are responsible for generating the context-based media. On the other hand, the disambiguation of interest keywords with the help of DBPedia also enhances the accuracy of the content; users are able to specify the exact meaning of a keyword and only relevant interest candidates are passed to the plugin when new content is required.

As we currently have no interactive feedback mechanism in the system, the accuracy cannot be further enhanced by the user on a per-media-item basis.

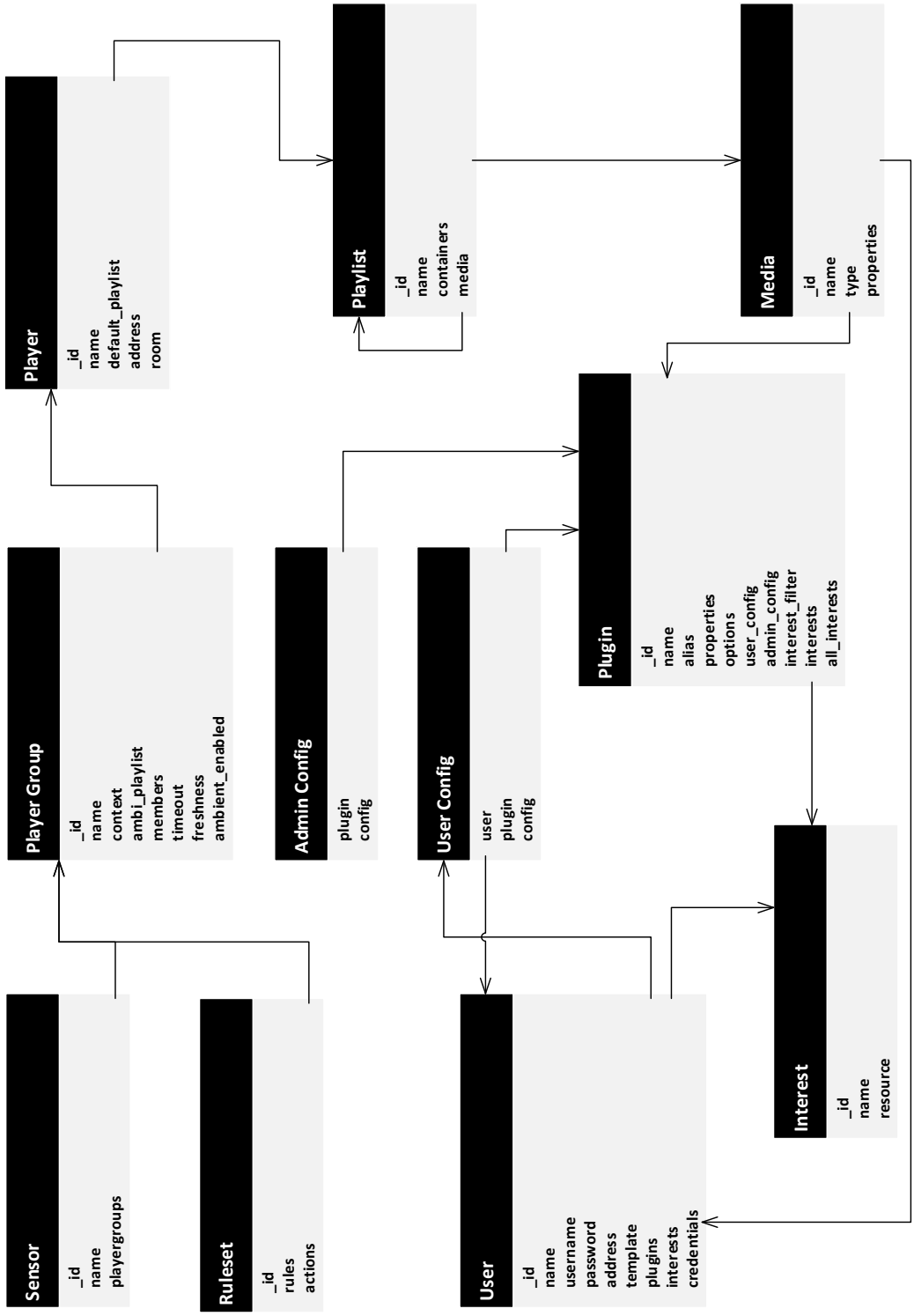
Finally we conclude that we have designed a generic digital signage system that may be configured for any use case and is able to provide context-based media. Its portability enables the user to freely choose the hardware that will be used. The implemented properties in terms of adaptability make CastJS truly unique.

# A

# System Architecture


External
DBPedia
Media Content
Content Management System
Interests Manager
Rules Engine
Context Modelling
Object Factory
Playlist Generator
Plugin Manager
WebSocket Server
Database
User Interface
Web Services
Sensor 2
Sensor 1
Media Player Group Z
Display E
Media Player Group Y
Display D
Media Player Group X
Display A
Display B
Display C
Administrator
User

# B
# Database Architecture


Player
_id
name
default_playlist
address
room
Playlist
_id
name
containers
media
Media
_id
name
type
properties
Plugin
_id
name
alias
properties
options
user_config
admin_config
interest_filter
interests
all_interests
Player Group
_id
name
context
ambi_playlist
members
timeout
freshness
ambient_enabled
Admin Config
plugin
config
User Config
user
plugin
config
Interest
_id
name
resource
Sensor
_id
name
playergroups
Ruleset
_id
rules
actions
User
_id
name
username
password
address
template
plugins
interests
credentials

# C

# Media Player Functions

## C.1 Image

```
function playImage(source) {
  var container = source.parent;
  var imgcontainer = document.createElement('span');
  imgcontainer.className = 'helper';
  var image = document.createElement('img');
  if (source.stretch){
    image.width=container.element_width;
    image.height=container.element_height;
  } else {
    image.className = 'keep_ratio';
  }
  image.src= source.src.src;
  image.onload= function(){
      setTimeout(function(){
        playContainer(source.parent);
      }, source.dur * 1000);
  };
  imgcontainer.appendChild(image);
  container.element.html(imgcontainer);
}
```

## C.2 Video

```
function playVideo(source){
  var container = source.parent;
  var video = document.createElement('video');
  video.width= container.element_width;
  video.height= container.element_height;
  video.src= source.src.src;
  video.type= 'video/mp4';
  video.controls= false;
  video.autoplay= 'autoplay';
  video.muted= true;
  video.onended= function(){
    playContainer(source.parent);
  };
  video.onplay = function(){
    //do nothing
  };
  container.element.html(video);
}
```

## C.3 Youtube

```
function playYoutube(source){
  var container = source.parent;
  var element = document.createElement('div');
  var myID = Math.random() + '';
  element.id = myID;
  container.element.html(element);
  var player;

  player = new YT.Player(myID, {
    height: container.element_height,
    width: container.element_width,
    videoId: source.src.src,
    playerVars: {
      controls: 0,
      fs: 0,
      iv_load_polic: 3,
      modestbranding: 1,
      showinfo: 0
    },
    events: {
      'onReady': onPlayerReady,
      'onStateChange': onPlayerStateChange,
      'onError': onPlayerError
    }
  });

  function onPlayerReady(event) {
    event.target.mute();
    event.target.playVideo();
  }
  function onPlayerStateChange(event) {
    if (event.data == YT.PlayerState.ENDED) {
      playContainer(source.parent);
    }
  }
  function onPlayerError(event) {
    playContainer(source.parent);
  }
}
```

## C.4 Vimeo

```
function playVimeo(source){
  var container = source.parent;
  var element = document.createElement('div');
  var myID = Math.random() + '';
  element.id = myID;
  container.element.html(element);
  var player;

  player = new Vimeo.Player(myID, {
    height: container.element_height,
    width: container.element_width,
    id: source.src.src,
    portrait: false,
    title: false,
    byline: false,
    autoplay: true,
    loop: false
  });
  player.setVolume(0);

  player.on('ended', function(data) {
    playContainer(source.parent);
  });
  player.on('error', function(data) {
    playContainer(source.parent);
  });
  player.play();
}
```

## C.5 Web Page

```
function playWebpage(source){
  var myID = Math.random() + '';
  var container = source.parent;
  var myiframe = document.createElement('iframe');
  myiframe.frameborder = 0;
  myiframe.src = source.src.src;
  myiframe.style = "overflow:hidden;height:100%;width
     :100%;";
  myiframe.height="100%" ;
  myiframe.width="100%";
  myiframe.id = 'iframe_' + myID;
  myiframe.name = 'iframe_' + myID;

  container.element.html(myiframe);

  setTimeout(function(){
    playContainer(source.parent);
  }, source.dur * 1000);
}
```

## C.6 Plugin

```
function playPlugin(source){
  var myID = Math.random() + '';
  var container = source.parent;
  var myiframe = document.createElement('iframe');
  myiframe.frameborder = 0;
  myiframe.style = "overflow:hidden;height:100%;width
     :100%;";
  myiframe.height="100%" ;
  myiframe.width="100%";
  myiframe.id = 'iframe_' + myID;
  var post_form = document.createElement('form');
  var post_field = document.createElement('input');
  post_form.action = '/plugins/' + source.src.type;
  post_form.target = 'iframe_' + myID;
  post_form.setAttribute('method',"post");
  post_form.id = 'form_' + myID;
  post_field.type = "hidden";
  post_field.name = "source_ref";
  post_field.value = JSON.stringify(source, function(
     key, value) {
    if( key == 'parent') { return '';}
    else {return value;}
  });
  container.element.html(myiframe);
  container.element.append(post_form);
  post_form.append(post_field);

  document.getElementById('form_' + myID).submit();
  if (typeof source.dur !== 'undefined' && source.dur >
     0){
    setTimeout(function(){
      playContainer(source.parent);
    }, source.dur * 1000);
  } else {
    var firstload = 1;
    myiframe.onload = function() {
      if (firstload === 1){
        firstload = 0;
        var content = $(this).contents();
        var thebody = content.find('body');
        thebody.on('click', function(e){
```

```
39            playContainer(source.parent);
40          });
41        }
42      };
43    }
44  }
```

# D

# Examples of Rule Templates

## D.1 Skeleton

```
"rule_template": {
  "title": "Rule title",
  "description": "Rule description",
  "input": {
    "operator": {
      "title": "Operator",
      "type": "string",
      "enum": [
        {"title": "Operator 1", "value": "op1"},
        {"title": "Operator 2", "value": "op2"}
      ],
      "required" : true,
      "default" : "op1"
    },
    "input_x": {
      "title": "Input title",
      "description": "Input description",
      "type": "string"
    }
  },
  "context": {
    "context_property_x": {
      "type": "object"
    }
  },
  "evaluate": {
    "op1":
      function(context, input){
        var res;
        // function body
        return res;
      },
    "op2":
      function(context, input){
        var res;
        // function body
        return res;
      }
  }
}
```

## D.2 Current Time Rule

```
"cur_time": {
  "title": "Current Time",
  "description": "Compare the current time.",
  "input": {
    "operator": {
      "title": "Operator",
      "description": "Choose an operator.",
      "enum": [
        {"title": "Current Time is Greater Than", "value": "gt"},
        {"title": "Current Time is Less Than", "value": "lt"}
      ],
      "required" : true,
      "default" : "gt"
    },
    "time": {
      "title": "Time",
      "description": "Time to compare to the current time.",
      "type": "string",
      "required" : true,
      "placeholder": "HH:mm:ss",
      "default" : ""
    }
  },
  "context": {
    "now": {
      "type": "string",
      "description": "Timestamp"
    }
  },
  "evaluate": {
    "gt":
      function(context, input){
        var now = '';
        if (typeof context.now !== 'undefined'){
          now = context.now; //timestamp
        } else {
          now = Math.floor(Date.now() / 1000);
        }
```

```
39          var time = input.time; //HH:mm:ss
40          var t1 = new Date();
41          var parts = time.split(":");
42          t1.setHours(parts[0],parts[1],parts[2],0);
43          var t2 = new Date(now*1000);
44
45          if (t2.getTime()>t1.getTime()){
46            return true;
47          } else {
48            return false;
49          }
50        },
51      "lt":
52        function(context, input){
53          var now = '';
54          if (typeof context.now !== 'undefined'){
55            now = context.now; //timestamp
56          } else {
57            now = Math.floor(Date.now() / 1000);
58          }
59          var time = input.time; //HH:mm:ss
60          var t1 = new Date();
61          var parts = time.split(":");
62          t1.setHours(parts[0],parts[1],parts[2],0);
63          var t2 = new Date(now*1000);
64
65          if (t2.getTime()<t1.getTime()){
66            return true;
67          } else {
68            return false;
69          }
70        }
71    }
72  }
```

## D.3 Present Users Rule

```
"together": {
  "title": "Together With",
  "description": "Check if users are (not) present together.",
  "input": {
    "operator": {
      "title": "Operator",
      "type": "string",
      "enum": [
        {"title": "All Are Together", "value": "twall"},
        {"title": "All Are Not Together", "value": "ntwall"}
      ],
      "required" : true,
      "default" : "twall"
    },
    "users": {
      "title": "Users",
      "description": "Present Users.",
      "type": "array",
      "items": [
        {
          "type": "string",
          "title": "User",
          "description": "User Name.",
          "default" : ""
        }
      ],
      "required" : true
    }
  },
  "context": {
    "user_names": {
      "type": "array"
    }
  },
  "evaluate": {
    "twall":
      function(context, input){
        var check_users = input.users;
        var present_users = context.user_names;
```

```
        var res = false;
        if (check_users.length > 0 && present_users.length > 0){
          res = true;
          for (var i=0; i<check_users.length; i++){
            if (present_users.indexOf(check_users[i]) < 0){
              res = false;
              break;
            }
          }
        }
        return res;
      },
    "ntwall":
      function(context, input){
        var check_users = input.users;
        var present_users = context.user_names;
        var res = false;
        if (check_users.length > 0 && present_users.length > 0){
          for (var i=0; i<check_users.length; i++){
            if (present_users.indexOf(check_users[i]) < 0){
              res = true;
              break;
            }
          }
        }
        return res;
      }
  }
}
```

## D.4 Always True Rule

```
"always_true": {
  "title": "Always True",
  "description": "This rule is always true.",
  "input": {
    "operator": {
      "title": "Operator",
      "type": "string",
      "enum": [
        {"title": "Is True", "value": "istrue"}
      ],
      "required" : true,
      "default" : "istrue"
    }
  },
  "context": {},
  "evaluate": {
    "istrue":
      function(context, input){
        return true;
      }
  }
}
```

# E

# Facial Recognition

## E.1 Recognition Script

```
#!/bin/bash

#set variables
sensor_id="58bae110f768af03f3c81c38"
context_folder="/var/context/"
people_folder="/var/context/people/"
motion_folder="/var/motion/"
context_url="http://x.x.x.x/context"
gallery_file="people.gal"
score_threshold=1.8
loop_interval=1
max_loop_counter=10
tv_timeout=3600

#bookkeeping vars
loop_counter=0
last_result='{}'
timestamp=$(date +%s)
last_face=$timestamp
last_motion=$timestamp
```

```

#load the faces
cd $context_folder
br -algorithm FaceRecognition -enrollAll -enroll
   $people_folder $gallery_file || true

#loop infinitely
while :
do
  #set variables
  time_now=$(date +%s)
  capture_file="capture_$time_now.jpg"
  matches_file_prefix="matches_$time_now_"
  matches_file_suffix=".csv"
  results=()
  name_results=()

  #capture image and do facial recognition
  motion_detected=($motion_folder*.jpg)

  if [ ${motion_detected[0]} != $motion_folder'*.jpg' ];
      then
    last_motion=$time_now
    motion_counter=0
    for motion in "${motion_detected[@]}"
    do
      if [ $motion_counter -lt 1 ]; then
        #cut out all the faces
        python face_detect.py $motion
           haarcascade_frontalface_default.xml || true
        faces_detected=(face-detected-*.jpg)

        if [ ${faces_detected[0]} !=
        'face-detected-*.jpg' ]; then
          for face in "${faces_detected[@]}"
          do
            highest_score=0.0
            user=''
            username=''
            matches_file=$matches_file_prefix$face
                         $matches_file_suffix


```

```
61            #recognize each face
62            br -algorithm FaceRecognition -compare $face
                $gallery_file $matches_file || true
63
64            #compare scores
65            INPUT=$matches_file
66            OLDIFS=$IFS
67            IFS=,
68            [ ! -f $INPUT ] && { echo "$INPUT file not
               found"; touch $INPUT; }
69            counter="0"
70            while read flname score
71            do
72              if [ $counter != "0" ]; then
73                if (( $(echo "$score > $highest_score" |
                   bc -l) )); then
74                  highest_score=$score
75                  user=${flname#$people_folder}
76                  username=${user#*_}
77                  user=${user%%_*}
78                  username=${username%%_*}
79                fi
80              fi
81              counter="1"
82            done < $INPUT
83            IFS=$OLDIFS
84            rm -f $matches_file || true
85
86            if (( $(echo "$highest_score >
               $score_threshold" |bc -l) )); then
87              #add to the results
88              results+=(${user})
89              name_results+=(${username})
90            fi
91          done
92        fi
93      fi
94
95      motion_counter=$((motion_counter+1))
96      rm -f $motion
97    done
98  fi
99
```

```
100
101   #add unique results in a JSON string
102   sorted_results=($(echo "${results[@]}" | tr ' ' '\n' |
        sort -u | tr '\n' ' '))
103   sorted_name_results=($(echo "${name_results[@]}" | tr
        ' ' '\n' | sort -u | tr '\n' ' '))
104   context_string="{\"sensor_id\": \"$sensor_id\", \"
        users\": ["
105   counter="0"
106   for var in "${sorted_results[@]}"
107   do
108     if [ $counter != "0" ]; then
109       context_string="$context_string, "
110     fi
111     context_string="$context_string{\"_id\": \"${var}\"}
          "
112     counter="1"
113   done
114   context_string="$context_string], \"user_names\": ["
115   names_counter="0"
116   for var in "${sorted_name_results[@]}"
117   do
118     if [ $names_counter != "0" ]; then
119       context_string="$context_string, "
120     fi
121     context_string="$context_string\"${var}\""
122     names_counter="1"
123   done
124   context_string="$context_string]}"
125
126   if [ ${#sorted_results[@]} -gt 0 ]; then
127     last_face=$time_now
128   fi
129
130   #turn on TV
131   timediff=$((time_now-last_face))
132   if [ $timediff -gt $tv_timeout ]; then
133     #Turn off screen
134     #do command...
135   else
136     #Turn on screen
137     #do command...
138   fi
```

```
139
140   #send the context to the server
141   do_post="0"
142   if [ "$loop_counter" -ge "$max_loop_counter" ]; then
143     do_post="1"
144   else
145     loop_counter=$((loop_counter+1))
146     if [ "$last_result" != "$context_string" ]; then
147       do_post="1"
148     fi
149   fi
150
151   if [ $do_post != "0" ]; then
152     loop_counter=0
153     curl -X POST \
154     -H "Content-type: application/json" \
155     -d "$context_string" \
156     "$context_url"
157   fi
158
159   last_result=$(echo $context_string)
160
161   #delete temporary files
162   rm -f face-detected-* || true
163   sleep $loop_interval
164 done
165 exit 0
```

## E.2 Enrollment Script

```bash
#!/bin/bash

#set variables
people_folder="/var/context/people/"
directions=("front" "up" "down" "left" "right")

#get variables
username=$1
userid=$2
capture_file_prefix=$userid"_"$username"_"
capture_file_suffix=".jpg"

#loop directions
echo "Enrolling $username with ID $userid."
for direction in "${directions[@]}"
do
  face_ok="0"
  while [ $face_ok == "0" ]
  do
    read -p "Taking $direction photo. Press <enter> to continue..."
    #capture image and find face
    capture_file="$capture_file_prefix$direction$capture_file_suffix"
    #take a picture with the webcam
    fswebcam -r 1280x720 --jpeg 100 -D 1 -S 2 $capture_file
    #cut out all the faces
    python face_detect.py $capture_file haarcascade_frontalface_default.xml
    faces_detected=(face-detected-*.jpg)
    echo $faces_detected

    if [ ${#faces_detected[@]} -eq 1 ] && [ ${faces_detected[0]} != 'face-detected-*.jpg' ]; then
      #we assume that we found the correct face
      face=${faces_detected[0]}
      cp $face $people_folder/$capture_file
      rm -f $face
```

```
35        echo "The $direction photo was saved successfully!
            "
36        face_ok="1"
37      else
38        #either no face or too many faces were found
39        nfaces=0
40        if [ ${#faces_detected[@]} -gt 1 ]; then
41          for face in "${faces_detected[@]}"
42          do
43            rm -f $face
44          done
45          nfaces=${#faces_detected[@]}
46        fi
47        echo "$nfaces faces detected, please try again."
48      fi
49      rm -f $capture_file
50    done
51  done
52  echo "Congratulations, you are now enrolled."
53  exit 0
```

# F

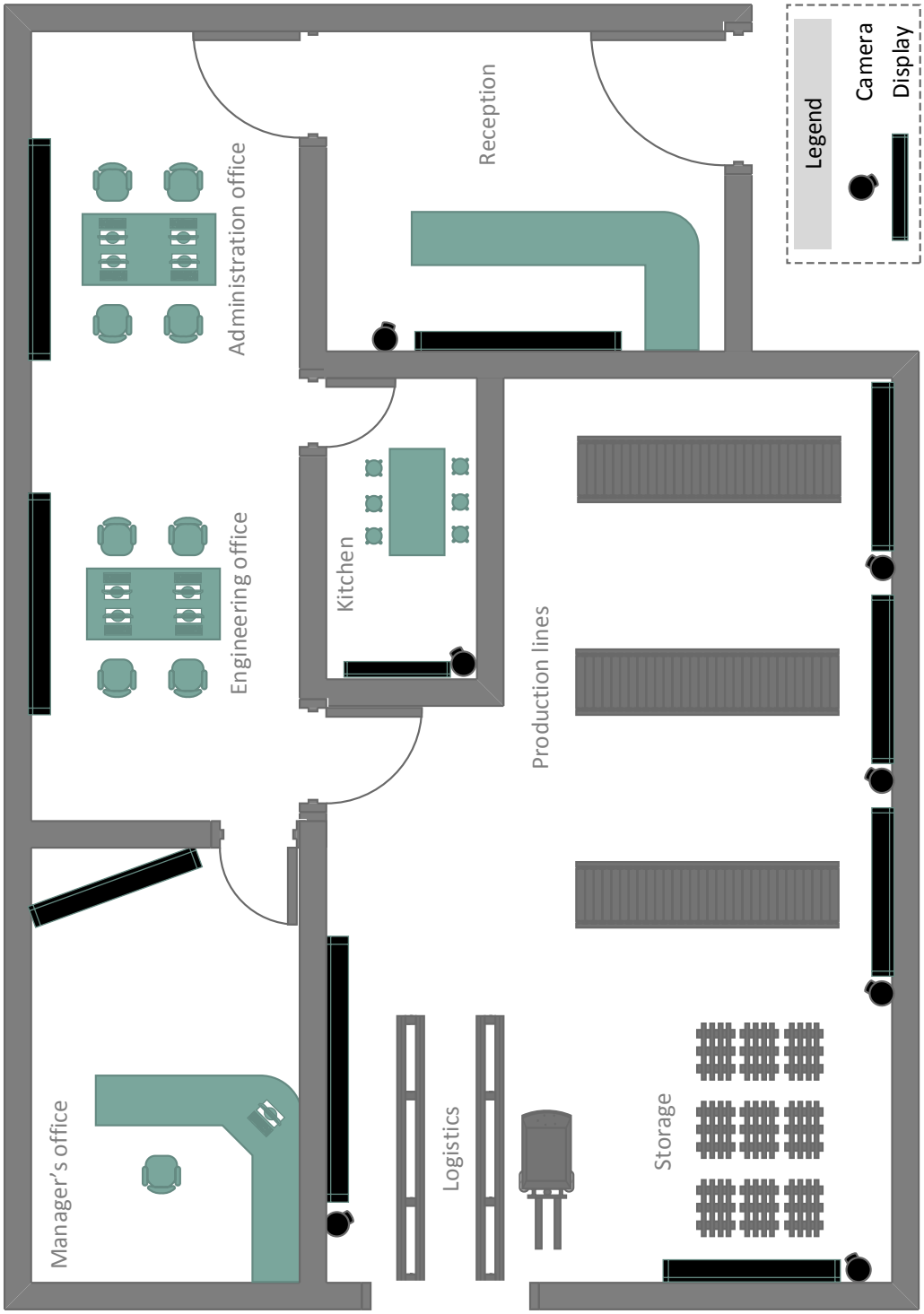
# Weather Plugin Interest Filter Query in JSON

```
"interest_filter" : {
  "queryType" : "SELECT",
  "distinct" : true,
  "variables" : [
    "?location",
    "?lat",
    "?lng"
  ],
  "where" : [
    {
      "type" : "bgp",
      "triples" : [
        {
          "subject" : "?location",
          "predicate" : "http://www.w3.org/2000/01/rdf-schema#label",
          "object" : "?label"
        },
        {
          "subject" : "?location",
```

```
20          "predicate" : "http://www.w3.org/2003/01/geo/
               wgs84_pos#lat",
21          "object" : "?lat"
22        },
23        {
24          "subject" : "?location",
25          "predicate" : "http://www.w3.org/2003/01/geo/
               wgs84_pos#long",
26          "object" : "?lng"
27        }
28      ]
29    },
30    {
31      "type" : "filter",
32      "expression" : {
33        "type" : "operation",
34        "operator" : "=",
35        "args" : [
36          {
37            "type" : "operation",
38            "operator" : "str",
39            "args" : [
40              "?location"
41            ]
42          },
43          "\"$INTEREST$\""
44        ]
45      }
46    }
47  ],
48  "type" : "query",
49  "limit" : 1
50 }
```

# G

# Production Plant Floor Plan for Corporate Use Case


Manager's office
Administration office
Engineering office
Logistics
Kitchen
Reception
Production lines
Storage
Legend
Camera
Display

# H

# CastJS Screen Captures

## H.1 User Profile

# User

| | |
|---|---|
| User Name | Joeri Boonen |
| User ID | joeriboonen |
| Password | |
| Confirm Password | |
| Template | VUB User |
| Street | mystreet |
| Number | 10 |
| Postal code | 1000 |
| City | Brussels |
| Country code | BE |

Save

**Plugins**

| | | |
|---|---|---|
| DistroWatch | Enable | |
| News | Edit | Disable |
| Office365 Calendar | Edit | Disable |
| SVN | Edit | Disable |
| Traffic | Edit | Disable |
| Weather | Edit | Disable |

**Interests**

Search for interests | Add

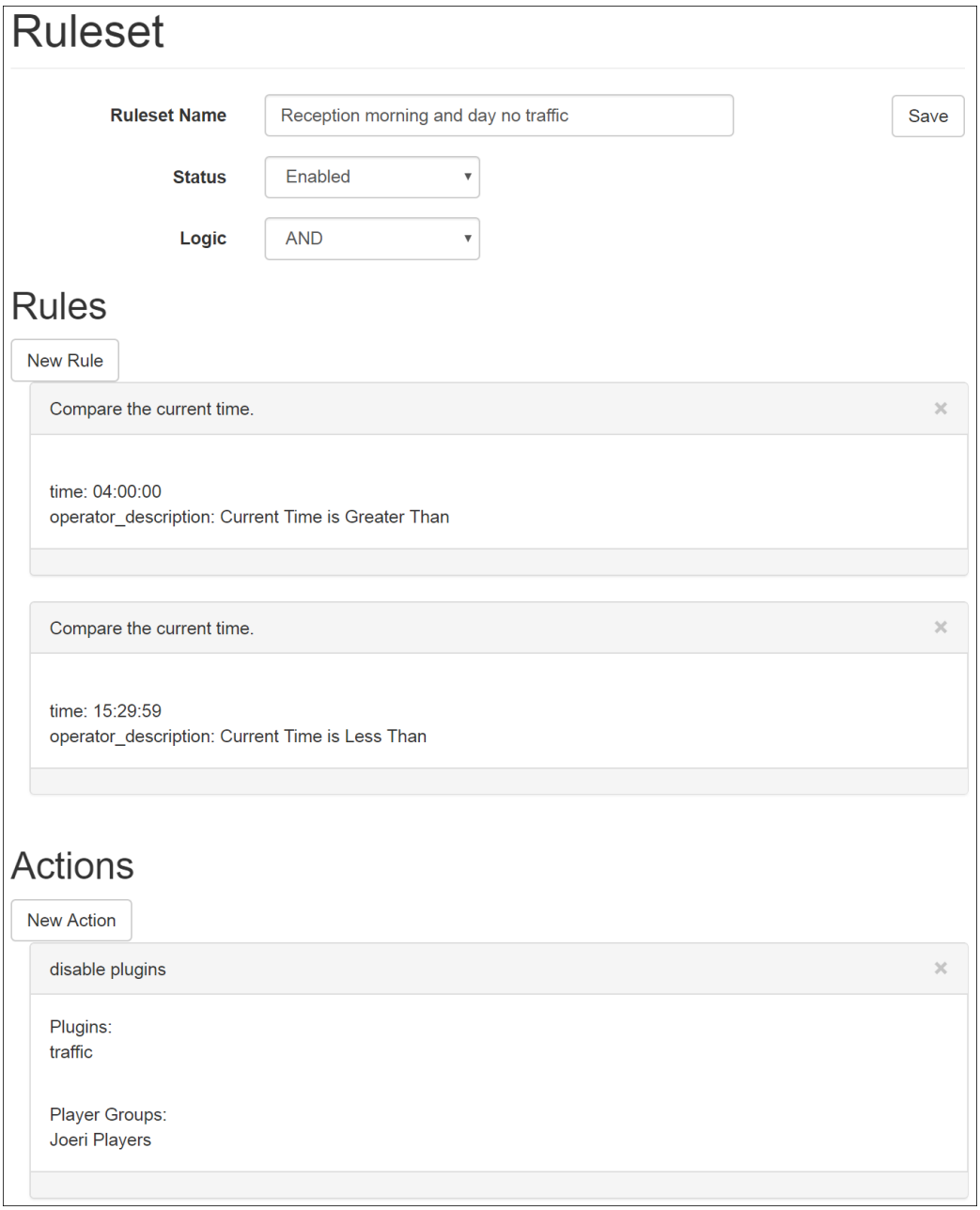
sense8 | mongodb | node js | parkway drive | raspbian | virtuix | google | facebook | ubuntu (operating system) | centos | linux mint | digital signage | raspberry pi | jommeke | game of thrones | jim carrey | jason statham | blindspot (tv series) | linux | netsky (musician) | ignite (band) | bamboo | leuven

**Credentials**

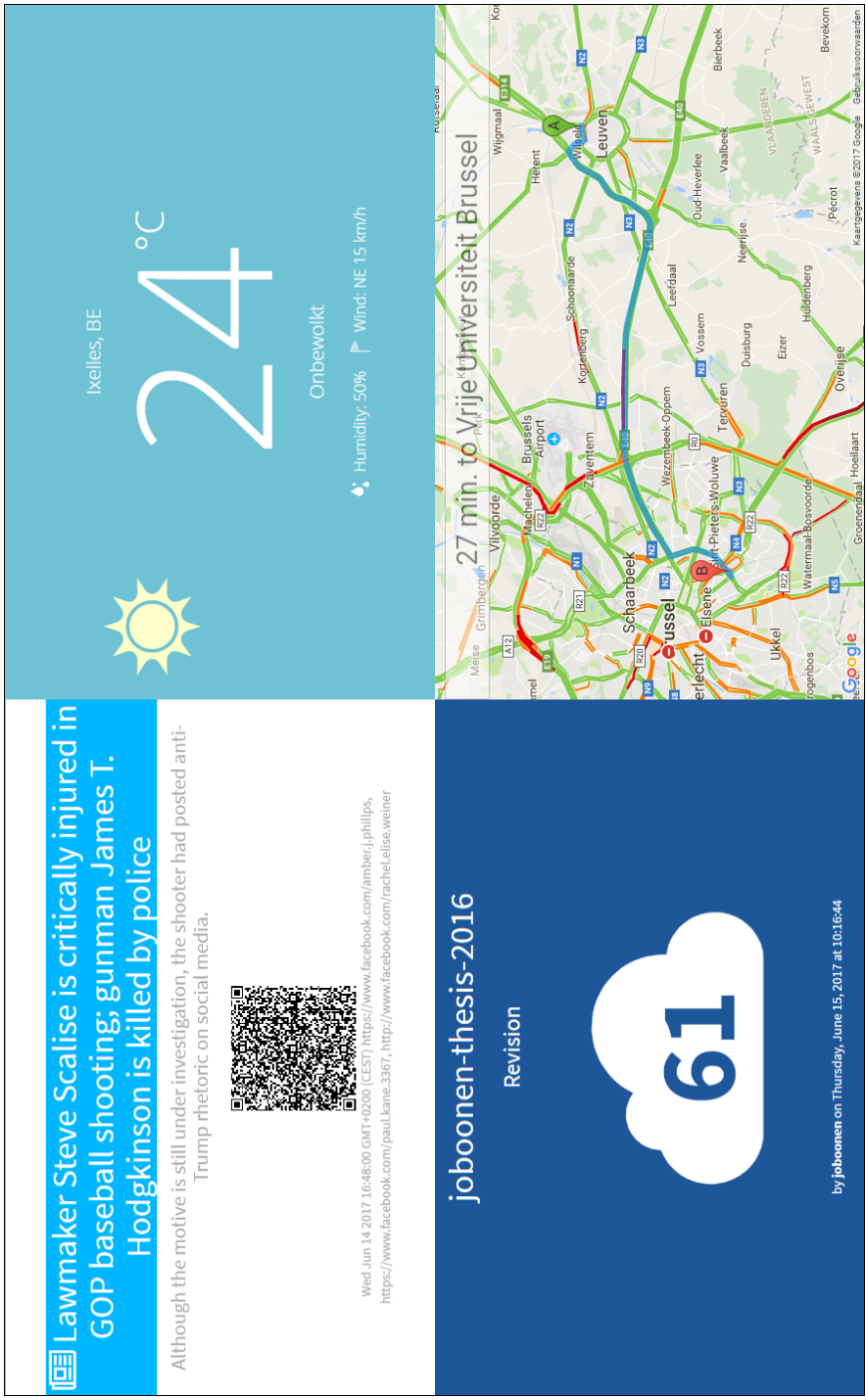
New Credentials

| Name | User | Actions |
|---|---|---|
| Active Directory Account | jboo | Edit Delete |
| SVN Server WISE | joboonen | Edit Delete |

## H.2 Ruleset


Ruleset
Ruleset Name
Reception morning and day no traffic
Save
Status
Enabled
Logic
AND
Rules
New Rule
Compare the current time.
time: 04:00:00
operator_description: Current Time is Greater Than
Compare the current time.
time: 15:29:59
operator_description: Current Time is Less Than
Actions
New Action
disable plugins
Plugins:
traffic
Player Groups:
Joeri Players

## H.3 Media Player with Four Containers



# Bibliography


[1] Unai Alegre, Juan Carlos Augusto, and Tony Clark. Engineering Context-aware Systems and Applications: A survey. *Journal of Systems and Software*, 117(C):55–83, July 2016.

[2] Virginia O. Andersson and Ricardo M. Araujo. Full Body Person Identification Using the Kinect Sensor. In *Proceedings of ICTAI 2014, IEEE International Conference on Tools with Artificial Intelligence*, pages 627–633, Limassol, Cyprus, November 2014.

[3] Virginia O. Andersson and Ricardo M. Araujo. Person Identification Using Anthropometric and Gait Data from Kinect Sensor. In *Proceedings of AAAI 2015, AAAI Conference on Artificial Intelligence*, pages 425–431, Austin, Texas, January 2015.

[4] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A Survey on Context-aware Systems. *International Journal of Ad Hoc Ubiquitous Computing*, 2(4):263–277, June 2007.

[5] Borut Batagelj, Robert Ravnik, and Franc Solina. Computer Vision and Digital Signage. In *Proceedings of ICMI 2008, International Conference on Multimodal Interfaces*, Chania, Greece, October 2008.

[6] Rudi Belotti, Corsin Decurtins, Michael Grossniklaus, and Moira C. Norrie. An Infrastructure for Reactive Information Environments. In *Proceedings of WISE 2005, Internal Conference on Web Information Systems Engineering*, pages 347–360, New York, USA, November 2005.

[7] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, September 2009.

[8] Dick Bulterman, Jack Jansen, Pablo Cesar, Sjoerd Mullender, Eric Hyche, Marisa DeMeglio, Julien Quint, Hiroshi Kawamura, Daniel Weck,

Xabiel Garcia Paneda, David Melendi, Samuel Cruz-Lara, Marcin Hanclik, Daniel F. Zucker, and Thierry Michel. Synchronized Multimedia Integration Language (SMIL 3.0). `https://www.w3.org/TR/SMIL3/`, December 2008.

[9] Qing Chen, François Malric, Yi Zhang, Muhammad Abid, Albino Cordeiro, Emil M. Petriu, and Nicolas D. Georganas. Interacting with Digital Signage Using Hand Gestures. In *Proceedings of ICIAR 2009, International Conference on Image Analysis and Recognition*, pages 347–358, Halifax, Canada, July 2009.

[10] Alan Colman, Mahmoud Hussein, Jun Han, and Malinda Kapuruge. Chapter 5: Context-aware and Adaptive Systems. In *Context in Computing: A Cross-Disciplinary Approach for Modeling the Real World*, pages 63–82. Springer Publishing Company, Incorporated, 2014.

[11] Waltenegus Dargie. *Context-aware Computing and Self-managing Systems*. Chapman & Hall/CRC, 2009.

[12] Nigel Davies, Sarah Clinch, and Florian Alt. *Pervasive Displays: Understanding the Future of Digital Signage*. Morgan & Claypool Publishers, 2014.

[13] Franck Dupin and Martin Adolph. Digital signage: The Right Information in All the Right Places. Technical report, ITU-T Technology Watch, Rennes, France, November 2011.

[14] Ali Farahani, Eslam Nazemi, Giacomo Cabri, and Alireza Rafizadeh. An Evaluation Method for Self-Adaptive Systems. In *Proceedings of SMC 2016, IEEE International Conference on Systems, Man, and Cybernetics*, pages 2814–2820, Budapest, Hungary, October 2016.

[15] Giovanni Maria Farinella, Giuseppe Farioli, Sebastiano Battiato, Salvo Leonardi, and Giovanni Gallo. Face Re-identification for Digital Signage Applications. In *Revised Selected Papers of VAAM 2014, International Workshop on Video Analytics for Audience Measurement*, pages 40–52, Stockholm, Sweden, August 2014.

[16] Jesus Favela, Marcela Rodriguez, Alfredo Preciado, and Victor M. Gonzalez. Integrating Context-aware Public Displays into a Mobile Hospital Information System. *Transactions on Information Technology in Biomedicine*, 8(3):279–286, September 2004.

[17] Geri Gay. *Context-aware Mobile Computing: Affordances of Space, Social Awareness, and Social Influence.* Morgan & Claypool Publishers, 2009.

[18] Mathieu Gross. Context-aware Computing: From Neuroscience to Mobile Devices. Technical report, Neuroscientific System Theory, Department of Electrical and Computer Engineering, Technical University of Munich, Munich, Germany, January 2016.

[19] Keith Kelsen. *Unleashing the Power of Digital Signage: Content Strategies for the 5th Screen.* Elsevier Focal Press, 2010.

[20] Woon-Yong Kim and SoonGohn Kim. The Design and Implementation of the Digital Signage Platform for Supporting Multi-Resources. In *Proceedings of SUComS 2015, International Conference on Security-enriched Urban Computing and Smart Grids*, pages 10–15, Porto, Portugal, June 2015.

[21] Joshua C. Klontz, Brendan F. Klare, Scott Klum, Anil K. Jain, and Mark J. Burge. Open Source Biometric Recognition. In *Proceedings of BTAS 2013, IEEE International Conference on Biometrics: Theory, Applications and Systems*, pages 1–8, Arlington, USA, September 2013.

[22] Donald L. Kreher and Douglas R. Stinson. *Combinatorial Algorithms.* CRC Press LLC, 1999.

[23] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 6(2):167–195, April 2015.

[24] Arnold M. Lund. Measuring Usability with the USE Questionnaire. *Usability Interface*, 8(2):3–6, January 2001.

[25] Lars-Ingemar Lundström. *Digital Signage Broadcasting: Content Management and Distribution Techniques.* Elsevier Focal Press, 2008.

[26] Jörg Müller and Antonio Krüger. MobiDiC: Context Adaptive Digital Signage with Coupons. In *Proceedings of AmI 2009, European Conference on Ambient Intelligence*, pages 24–33, Salzburg, Austria, November 2009.

[27] Kenton O'Hara, Mark Perry, Elizabeth Churchill, and Daniel Russel. *Public and Situated Displays: Social and Interactional Aspects of Shared Display Technologies*. Kluwer Academic Publishers, 2004.

[28] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, December 2007.

[29] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Realtime Computer Vision with OpenCV. *ACM Queue*, 10(4):40–56, April 2012.

[30] Arup K. Sadhu, Sriparna Saha, Amit Konar, and Ramadoss Janarthanan. Person Identification Using Kinect Sensor. In *Proceedings of CIEC 2014, International Conference on Control, Instrumentation, Energy and Communication*, pages 214–218, Kolkata, India, February 2014.

[31] Jimmy Schaeffler. *Digital Signage: Software, Networks, Advertising, and Displays*. Elsevier Focal Press, 2008.

[32] Florian Schaub, Bastian Könings, Peter Lang, Björn Wiedersheim, Christian Winkler, and Michael Weber. PriCal: Context-adaptive Privacy in Ambient Calendar Displays. In *Proceedings of UbiComp 2014, ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 499–510, Seattle, USA, September 2014.

[33] Aniruddha Sinha, Kingshuk Chakravarty, and Brojeshwar Bhowmick. Person Identification Using Skeleton Information from Kinect. In *Proceedings of ACHI 2013, International Conference on Advances in Computer-Human Interactions*, pages 101–108, Nice, France, March 2013.

[34] Asim Smailagic and David Kogan. Location Sensing and Privacy in a Context-aware Computing Environment. *IEEE Wireless Communications*, 9(5):10–17, October 2002.