



# MediaMosa

## An introduction

**Version 1.0**

**Date 8 July 2009**

**SURFnet/Kennisnet Innovation Programme**

## Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>4</b>
1.1	MediaMosa in brief .....	4
1.2	What does this document provide? .....	5
<b>2</b>	<b>Functional structure.....</b>	<b>6</b>
2.1	Introduction.....	6
2.2	Design decisions .....	6
2.3	Architecture.....	7
2.4	Content organisation .....	9
<b>3</b>	<b>Front-end applications and actors.....</b>	<b>11</b>
3.1	Introduction.....	11
3.2	Applications .....	11
3.3	Actors.....	12
<b>4</b>	<b>Back end.....</b>	<b>15</b>
4.1	Authentication.....	16
4.2	Authorisation.....	16
4.3	Play Proxy.....	17
4.4	Search .....	19
4.5	Job.....	20
4.6	Transcoding.....	22
4.7	Upload .....	22
4.8	OAI-PMH .....	24
4.9	Media Management.....	25
4.10	Logging.....	26
4.11	Statistics .....	27
	<b>Glossary.....</b>	<b>28</b>

# 1 Introduction

## 1.1 MediaMosa in brief

MediaMosa is a high-quality, scalable, open source media distribution platform based on the Drupal CMS. It offers user-level and administrator-level access to (shared) data repositories, metadata databases, and transcoding and streaming servers. Any type of media content (audio, video, documents, etc.) can be managed in MediaMosa.

End users can search for, play, upload and transcode media. The media can be tagged with metadata, which can be searched using CQL queries.

MediaMosa is built based on a Service Oriented Architecture (SOA). Part of what this means is that MediaMosa is divided into a back end and a front end. The back end consists of a number of web services, components and data repositories. This is where all the tasks are carried out, such as transcoding, uploading and playback. The front end consists of applications used by end users and administrators to communicate with the back end, using Representational State Transfer (REST).

The major advantage of this software architecture is its flexibility. Components can be combined easily to implement web services that can be accessed using front-end applications. The division between front end and back end results in better performance because tasks are carried out on central servers.

The most important features of MediaMosa are:

- Provides a storage platform for any type of content
- Streams Flash, H.264 MPEG-4 and Windows Media
- Transcoding based on FFmpeg
- Flexible Metadata Element Sets

MediaMosa offers many web services, including the following:

- Video playback (PlayProxy HTML wrapper)
- Authentication (D-Bus for EGA)
- Authorisation (domain, realm, group or a combination)
- Upload (PUT, POST, FTP)
- Transcoding (converting media files from one format to another)
- Media Management
- Search function: Contextual Query Language (CQL), level 2
- OAI-PMH: DC, QDC, LOM, CZP
- Logging and statistics
- Automatically generates stills

## 1.2 What does this document provide?

This document presents a brief overview of the most important information on MediaMosa. It is intended as an introduction for:

- Administrators who will be implementing MediaMosa and may need to make changes.
- Developers of front-end applications.

The following topics are covered:

- The functional structure of the MediaMosa environment (including design decisions, architecture of the content organisation, applications, roles).
- Front-end applications.
- Back-end (MediaMosa components and how they interact).

## 2 Functional structure

### 2.1 Introduction

The MediaMosa system offers a platform for third-party development of content streaming applications. For example, MediaMosa offers education institutions the option of uploading and playing specific video materials.

MediaMosa can be accessed by various types of applications (such as end-user applications and provider applications) that users utilise to communicate with MediaMosa. Users can play content, search for content or upload new content, among other options.

The content can be enriched with basic metadata and special metadata. The special metadata can be used by users of the application that owns the content. The metadata are provided to external systems using standardised protocols, such as OAI or SRU/SRW.

The metadata for the content is stored in MediaMosa; the actual media file is put in the data repository.

### 2.2 Design decisions

An important UNIX principle formed the starting point for designing and developing MediaMosa: make sure that each component does one task very well and use a simple protocol to combine the separate components into a single entity.

Throughout the process, what was already available as open source was constantly kept in mind. Adapting the system to existing components was the preferred approach, rather than adapting existing components to the system.

MediaMosa has a Drupal CMS and is divided into components based on the services that the system offers, so expansion is always possible.

The REST protocol was chosen for communication with external applications and communication between MediaMosa components. In contrast to the SOAP protocol, REST is an open source protocol.

## 2.3 Architecture

MediaMosa was built based on a Service Oriented Architecture (SOA). It consists of five layers (see Figure 1). Layers 1 through 3 are the MediaMosa back end; layers 4 and 5 comprise the various front end applications that communicate with MediaMosa. These applications are discussed in Chapter 3. Layers only communicate with other layers directly adjacent to them (according to the layering principle). Communication between the front end and the back end takes place via REST invokes

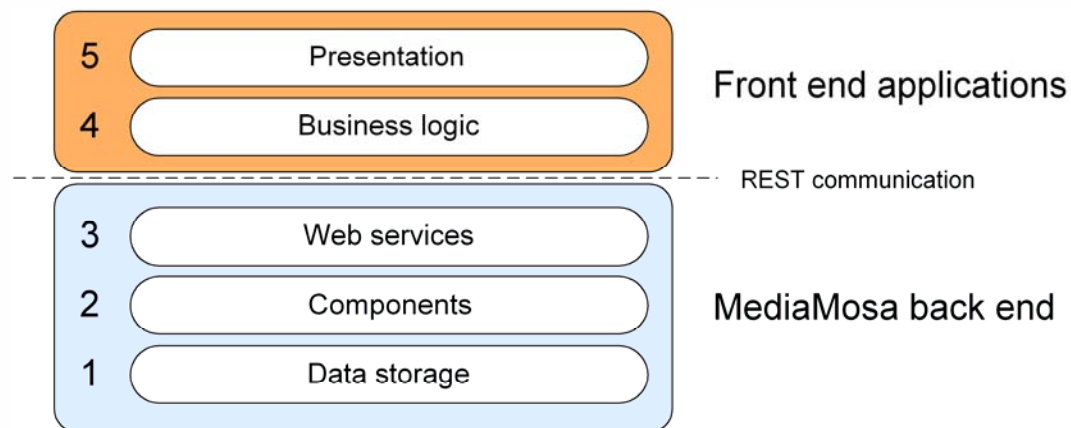


Figure 1

### 2.3.1 *Presentation (layer 5)*

End users communicate with the *Presentation* layer. This layer consists primarily of presentation in the form of a graphic user interface. It contains almost no functionalities, but it invokes functions from the *Business logic* layer.

Examples:

- Presenting a media file in a player.
- Screens for handling user input (such as search queries).

### 2.3.2 *Business logic (layer 4)*

This layer contains all the user functions of the end-user application, which are invoked from the *Presentation* layer. The *Business logic* layer then invokes services from the *Web services* layer.

Example:

- A function for storing a media file. The user is asked to provide a file name, format and metadata before saving the media file. The *Business logic* layer then invokes *Web services* to store the metadata and the media file.

### 2.3.3 *Web services (layer 3)*

MediaMosa communicates with the various applications through this third layer. The *Web services* layer contains building blocks that facilitate rapid realisation of the functions from the *Business logic* layer. These building blocks, the web services, invoke one or more components from the *Components* layer.

Examples:

- Saving a media file. When the *Business logic* layer has compiled the necessary data on a media file, the layer invokes the web service to save the media file.
- Protective mechanism. All play requests go through the *Play Proxy* web service. This service checks whether a user is authorised to play the media file, assisted by the *Authorisation* component from the *Components* layer.

### 2.3.4 *Components (layer 2)*

This layer contains the components that the web services use to handle requests from the applications.

Examples:

- Uploading a media file
- Streaming a media file
- Transcoding a media file

### 2.3.5 *Data storage (layer 1)*

This layer handles storage of the data in a data repository, making distinctions between the media files, the associated metadata and the authorisation data.

## 2.4 Content organisation

The central object in the implementation of MediaMosa is the *asset*. This can be considered a unit of content, usually a video. Figure 2 shows the relationships between the objects (*collection*, *still*, *media file* and *technical metadata*) and the asset.

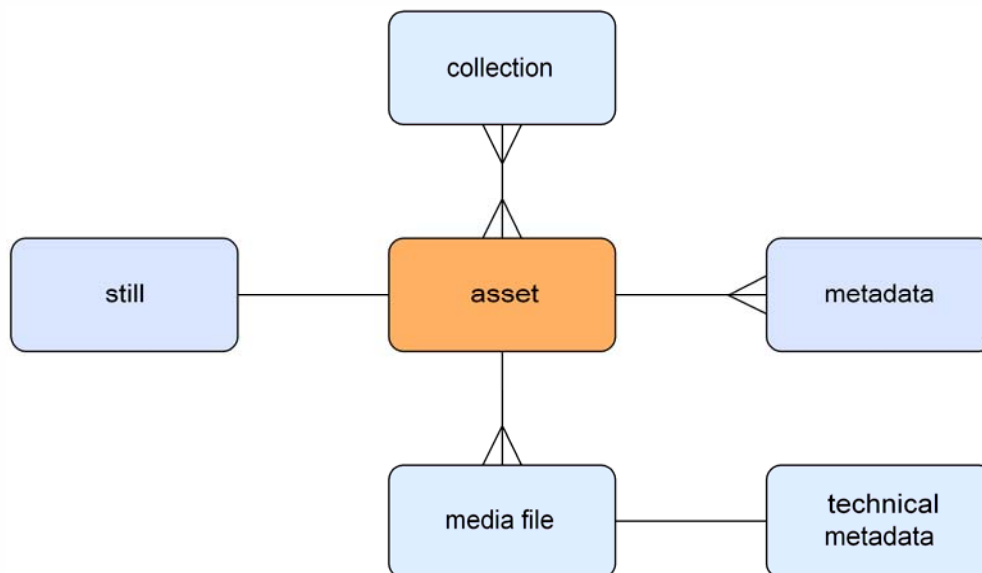


Figure 2

### 2.4.1 Asset

An asset is a media clip, for example a video about the origins of the earth. An asset mainly consists of media files (such as a Quicktime version and a WMV version of the media clip) and properties (metadata) that describe the asset. An asset could be limited to a specific fragment of a media file. An asset could contain stills. The asset could also be part of 0 or more collections.

An asset is 'owned by' an end-user application. The end-user application has the option of registering 'ownership' of an asset at the level of the individual end users. MediaMosa does not track information about ownership at the level of the individual user.

Only the owner and the MediaMosa administrators can manage an asset.

### 2.4.2 Media file

A media file is a file that can be stored and downloaded through MediaMosa. In MediaMosa, a media file is always part of one single asset. An asset can contain multiple media files. If it does, these files are versions of the same media clip in different file formats, bit rates or resolutions.

MediaMosa currently only supports media files with one video stream and one audio stream.

### 2.4.3 Collection

A collection consists of a group of assets. A collection has two properties: a title and a description.

A collection is 'owned by' an end-user application. The end-user application has the option of registering 'ownership' of a collection at the level of the individual end users. MediaMosa does not track information about ownership at the level of the individual user.

### 2.4.4 Still

A still is a non-moving image captured from a video file. A still is generated on the basis of a media file and stored separately in that media file's asset.

### 2.4.5 Metadata

The metadata records information about an asset's properties (such title, language, publisher). The metadata is used in search queries to find assets.

The technical metadata contains information about the individual media files (such as file format and resolution).

Metadata can also be added at the level of the end-user application, which is not managed by MediaMosa. These 'supplements' can be stored in the MediaMosa database in a *binary large object* (blob), a field which can contain any type of data (such as ZIP files or images).

## 3 Front-end applications and actors

### 3.1 Introduction

The MediaMosa back end is invoked via front-end applications (section 3.2). These applications are used by various actors (section 3.3).

### 3.2 Applications

The following applications have access to MediaMosa via REST invokes, for the purpose of carrying out various tasks.

#### 3.2.1 *End-user application (EGA)*

An end-user application or EGA gives an end user access to the facilities offered by MediaMosa. MediaMosa offers a repository for the various EGA's; access to the repository is, by default, restricted to that specific EGA.

EGA's can use MediaMosa's services to realise specific services for their users according to their own preferences. Assets and collections created by an EGA are therefore always owned by the EGA.

If desired, the EGA can also register the individual end users as owners of the assets and collections. The EGA is responsible for correct implementation of end-user registration.

#### 3.2.2 *White label EGA*

A white label EGA is a template that can be used to develop an EGA. A white label EGA allows you to start using MediaMosa quickly.

#### 3.2.3 *OAI harvester(s)*

A harvesting system is an application that requests metadata from MediaMosa and other data providers. Users can perform searches based on the metadata.

#### 3.2.4 *Provider application*

Content providers can offer collections and assets through provider applications (such as NIBG Teleblik) for use by multiple EGA's. These provider applications can only upload and manage content, not provide it to end users. Provider applications also make it possible to carry out mass operations that are not possible with EGA's.

#### 3.2.5 *Management application*

The MediaMosa component is managed through the management application. This includes:

- Managing servers (including the data repository) that can be used by MediaMosa.
- Managing EGA authorisations.
- Accessing MediaMosa usage statistics.
- Managing media (assets, collections, metadata, etc.).
- Managing transcoding profiles.

### 3.3 Actors

Figure 3 shows which main functions in MediaMosa are used by the various actors. These functions (referred to as components) are described in more detail in Chapter 4.

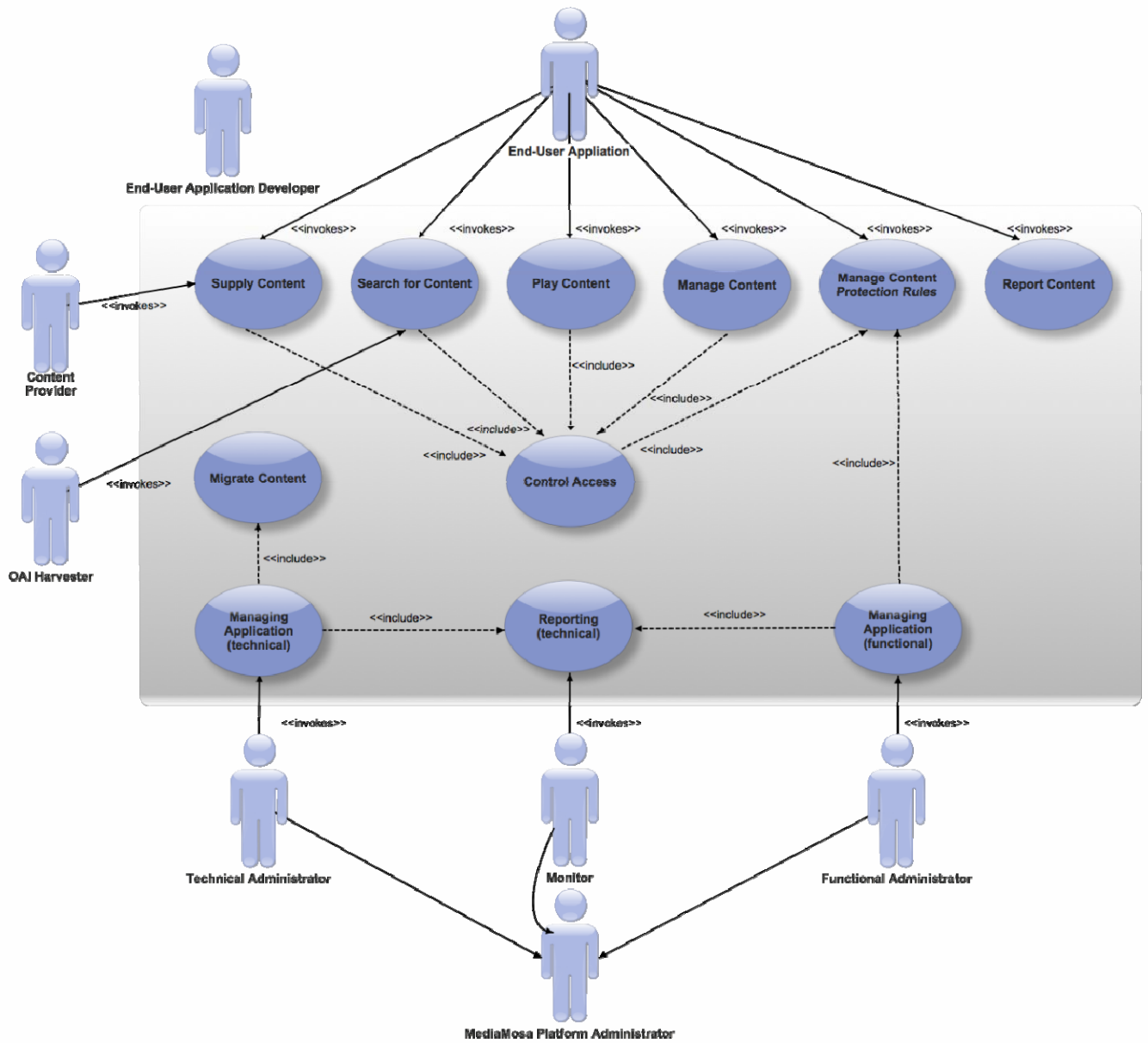


Figure 3

#### 3.3.1 The end user

End users can play, submit and manage media files and metadata through an end-user application (EGA). Except for the MediaMosa administrators, end users do not communicate directly with MediaMosa. The MediaMosa services are always accessed via an EGA.

### 3.3.2 *Content provider*

A content provider delivers new media files and metadata to MediaMosa on a one-time or periodic basis. Content is delivered through a provider application.

### 3.3.3 *OAI harvester developer*

The developer of the OAI harvester develops the application that requests metadata from MediaMosa.

### 3.3.4 *Administrator*

Administrators use the MediaMosa management application.

#### TECHNICAL ADMINISTRATOR

The technical administrator manages the MediaMosa platform and has access to the following functionalities:

- Managing access rights to the front-end applications.
- Managing the static configuration of the servers.
- Managing the static configuration of the mount points for the data repository.
- Managing availability of the web services.
- Managing the interpretation of the error messages.

#### FUNCTIONAL ADMINISTRATOR

A functional administrator has access to the following functionalities:

- Reporting content use.
- Reporting application use.
- Monitoring the status of web services and external systems.
- Monitoring consistency between the file system and the database.
- Managing availability of the web services.
- Managing interpretation of error messages.
- Managing access rights to EGA's.
- Managing root tables (such as media types, media formats, media extensions, transcoding profiles, etc.).
- Managing content.
- Configuring and managing quotas for users and groups.

Each organisation has its own functional administrators. They only have authorisation to change data for their own organisation.

## MONITOR

A monitor is an administrator who only has authorisation to view server configurations and various reports. He is not allowed to make any changes in MediaMosa.

### 3.3.5 *EGA developer*

The developer of an EGA does not communicate directly with MediaMosa. The EGA created and managed by the developer does.

### 3.3.6 *Data warehouse*

The data warehouse can request information on statistics from MediaMosa.

## 4 Back end

Figure 4 offers an overview of the components that comprise the back end of MediaMosa (inside the *MediaMosa* box). The diagram also includes components and applications that communicate with MediaMosa.

The diagram only displays the most important relationships between the components. For more information, see the component descriptions in this chapter.

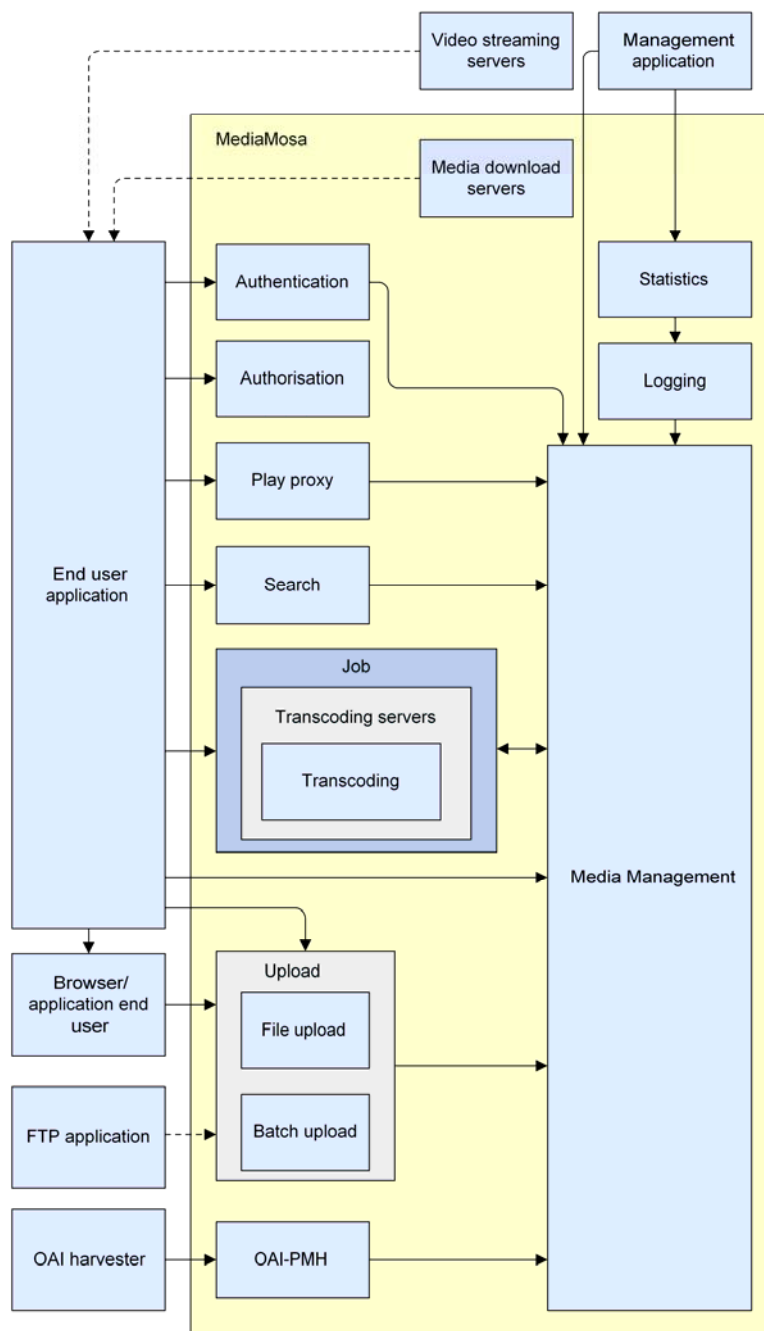


Figure 4

#### 4.1 Authentication

Before an EGA can use MediaMosa, the EGA has to provide authentication for MediaMosa. A REST invoke is used to authenticate the EGA.

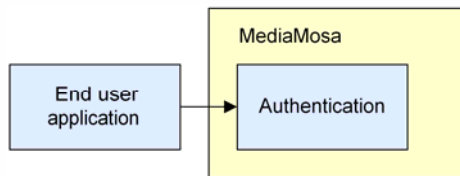


Figure 5

When the first request comes in from the authentication process, a cookie is sent back by the MediaMosa server (via the 'Set-Cookie' HTTP header string). The cookie contains a session ID, which also needs to be sent back as a cookie with every subsequent request (via the 'Cookie' HTTP header string). The EGA has access to MediaMosa for that session.

The EGA is authenticated using a challenge response protocol based on the D-Bus protocol. The client sends the D-Bus messages as HTTP POST data (variable 'dbus') and the server responds in XML (tag 'dbus').

#### 4.2 Authorisation

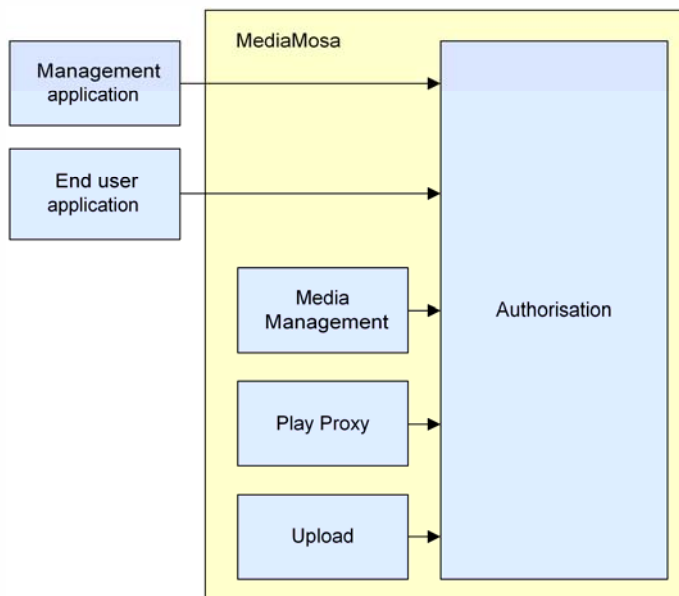


Figure 6

Content protection rules define which users, user groups and EGA's are authorised to access media files. The content protection rules can be set using the management application and the EGA. An EGA can only set content protection rules for the media files stored in MediaMosa by that EGA.

The rights are assigned at the level of the media file. One of the options this offers is to provide public access to a number of low-quality videos in an asset, providing access to a high-quality

version for a select group of users. Authorisation check takes place when the user requests video playback.

When a media file is uploaded, it is not automatically protected: all users can play the file. If one content protection rule is defined, the status changes to fully protected, except for the protection rules as defined.

Content protection rules can be used to set the authorisation check for playback to:

- **Domain**  
An Internet domain name, such as: only give access to *surfnet.nl*. All subdomains (such as *flex.surfnet.nl*) would also have access.
- **Realm**  
Such as *jan@xs4all.nl*, or *@surfnet.nl*. In the first case, the realm must be an exact match for the party requesting the media file. The second case is more flexible: it gives access to all users that match *\*@surfnet.nl*, as well as any subdomains (such as *piet@surfnet.nl*, *jan@flex.surfnet.nl*, *joris@flex1.z33.surfnet.nl*).
- **Users**  
These are MediaMosa users. The *user\_id* field is checked when playback is requested.
- **Groups**  
These are groups of MediaMosa users. The *group\_id* field is checked when playback is requested.
- **EGA's**  
This is a special variation on play rights. It is possible to assign play rights for the media file to other EGA's that are linked to MediaMosa. To make this possible, the other EGA has to know the application ID. The media file will then be available to the public in that EGA, unless realm or domain rights have been set. If realm or domain rights have been defined, the other EGA also has to match those rights in order to view the media file. User and group rights for other EGA's are ignored, since users and groups are unique to each EGA. A user or group from a different EGA can never own or have rights to a media file.

## 4.3 Play Proxy

### 4.3.1 Introduction

A video can be played using an *asset\_id* and a *mediafile\_id*. The video is not played in MediaMosa itself. MediaMosa sends URLs to video streaming servers, which play the videos.

There is a special play option for stills: if a still has been created for an asset, this invoke brings up a link to the still. There is also a REST invoke for playing a still.

### 4.3.2 Requesting play and/or download information for a video

The *Play Proxy* component handles the play request from the EGA and sends a plain URL, ASX file, object, download URL or HTML snippet that an end user can use to play the video (directly or indirectly).

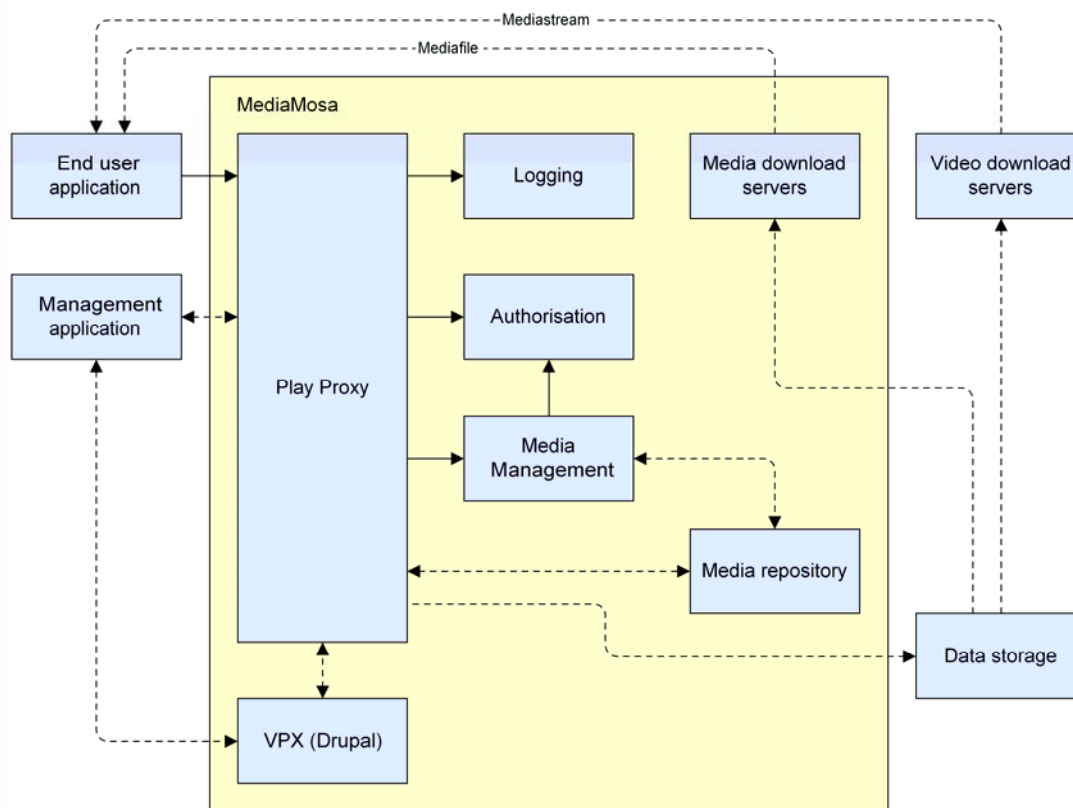


Figure 7

An EGA makes a REST invoke to submit a request to play a video. The request is received by the *Play Proxy* component. The request contains at least the information about the media file that needs to be played.

The request is processed as follows:

1. Retrieve the information on the asset for that media file via the *Media Management* component.
2. Check the authorisation based on the user and the content protection rules.
3. Create a ticket (in the MediaMosa database) and a symbolic link (in the data repository).
4. Generate the response. The response will contain a URL for the video streaming server that the end user can use to view the video. Depending on the request parameters, the response will also contain a URL to download the media from the download server. The URLs point to the symbolic links in the data repository, not directly to the actual media file. The streaming and/or download servers that are available are determined based on the information in the MediaMosa database.
5. Make an entry in the log.

#### 4.3.3 Requesting a still

A still can be requested using a REST invoke. An `asset_id` is provided as a parameter. The result is a URL for an image on a MediaMosa web server. The URL can easily be used in a `<img src=""/>` tag in the EGA, for example.

## 4.4 Search

The *Search* component handles EGA search queries.

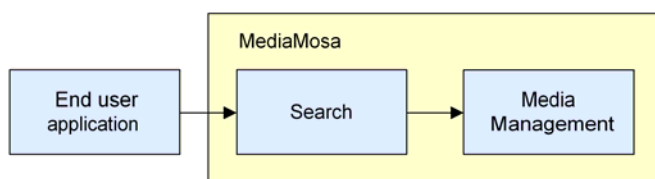


Figure 8

The search function uses CQL (Contextual Query Language), a standard for using a simple string syntax to search in different platforms or databases.

For MediaMosa, it represents an advanced search method. A CQL string is sent to MediaMosa via an extra parameter. In MediaMosa, the CQL string is converted to an SQL query, which is then applied to the database. Once the CQL string is defined, other search parameters cannot be used; they will produce an error message.

#### 4.4.1 Structure of the module

The MediaMosa CQL module is derived from a Drupal CQL module. As a result, it will also be possible to use CQL for other purposes in future. The Drupal module supports generic CQL level 2 code, v1.2. The MediaMosa module creates the MediaMosa CQL by extrapolating from the CQL class and placing the result in the MediaMosa context.

## 4.5 Job

### 4.5.1 Introduction

A job is considered to be: a task that MediaMosa does in the background, while the user does other things. A job mainly involves uploading and transcoding media files and generating stills. The *Job* component makes it possible to register jobs for asynchronous processing. The component also offers functions for registering the current status of specific jobs so an EGA can request an interim status report on the job.

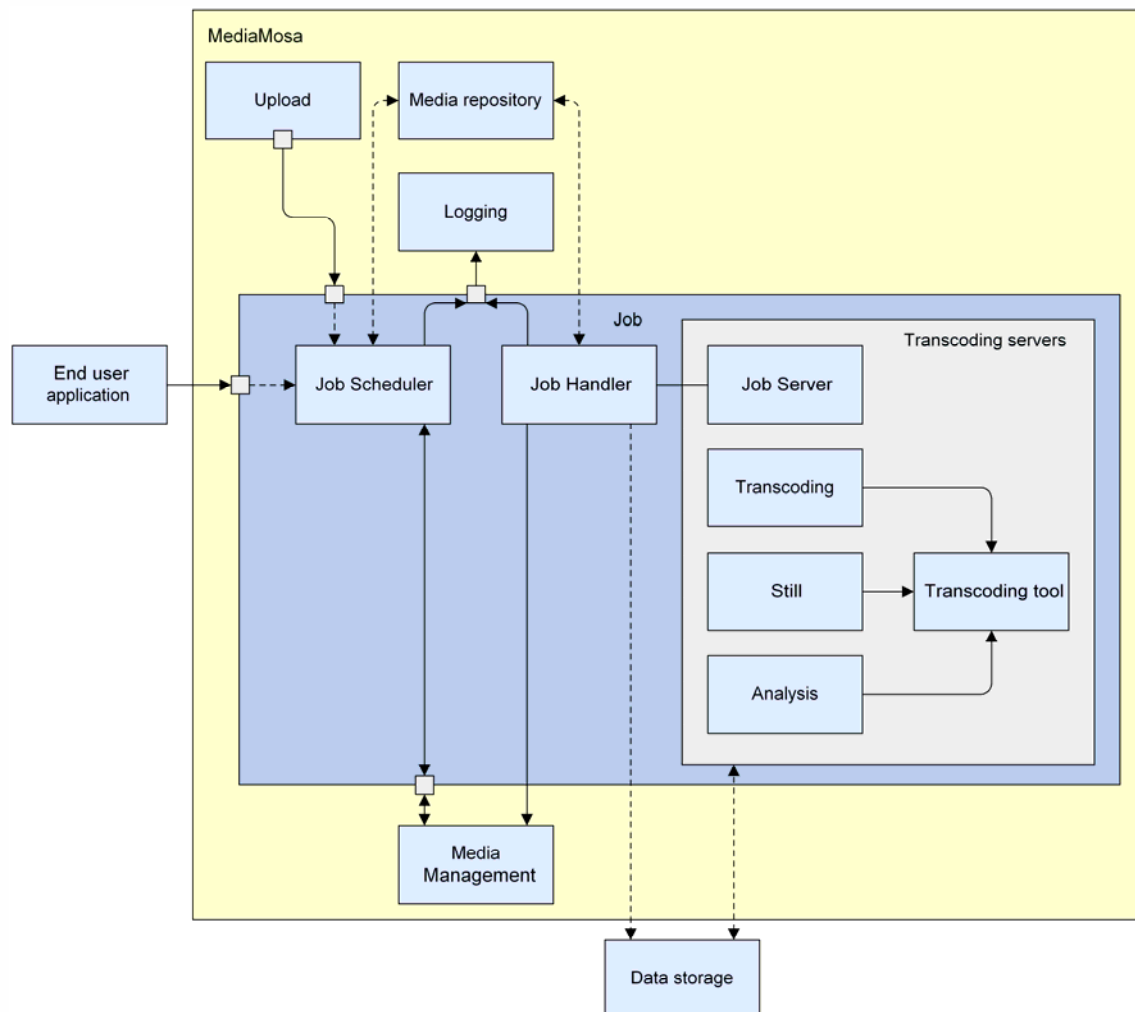


Figure 9

### 4.5.2 Subcomponents

#### JOB SCHEDULER

The Job Scheduler is the interface between the *Job* component and all the other components and the EGA's.

The Job Scheduler can be used to add new jobs or request information about a job's status. When a new job is added, it is saved to the *jobs* table in the *vpx\_data* database.

This component uses configuration settings that can be changed using the following modules in the management application:

- Web service management: availability of the *Jobs* web services.
- Transcoding profiles: available transcoding profiles.
- MediaMosa servers: available servers.

#### JOB HANDLER

The Job Handler handles communication with all the Job Servers. That means that jobs are assigned to Job Servers and that the job status for jobs that have already been assigned is retrieved from the correct server.

This component uses configuration settings that can be changed using the following modules in the management application:

- Web service management: availability of the *Jobs* web services.
- Transcoding profiles: available transcoding profiles.
- MediaMosa servers: available servers.

#### JOB SERVER

The Job Server carries out the job.

#### 4.5.3 Processing jobs

The EGA and the *Media Management* and *File Upload* components can create jobs in the Job Scheduler.

The Job Handler periodically checks which jobs have been created and what the status is of the jobs (transcoding, stills and analysis) that have already been sent to the Job Servers. The Job Handler sends jobs to the Job Server of a specific transcoding server once the media file has finishing uploading and the transcoding server has processing capacity (according to the Job Handler).

The Job Server checks periodically to see if a job needs to be started on the transcoding server. This job immediately records the result of the job and a status overview to a temporary location in the data repository. The interim status and final status of a job are registered by the Job Server.

The Job Handler periodically retrieves the status of the jobs from the Job Servers. This information is recorded in the MediaMosa database. An EGA can request the information via the Job Scheduler.

When a job is completed in full, the Job Handler updates the media data using the *Media Management* component. The media file is also moved from the temporary location to the definitive location in the data repository.

For a file upload, a job is created via the *Media Management* component and the Job Scheduler. The status of the upload job is only updated at the beginning and the end of the upload. The Job Handler does not start jobs for media files that have not yet uploaded completely.

#### 4.6 Transcoding

Transcoding means converting video files from one format to another, or changing the bit rate or resolution. MediaMosa supports all formats that can be transcoded by FFmpeg, including MPEG, WMV and Flash. MediaMosa can handle a wide range of parameters within these formats. Only original media files can be transcoded; media files that have been transcoded previously cannot be transcoded again.

When transcoding a media file, it is possible to specify which transcoding tool should be used (e.g. FFmpeg or Windows Media Encoder). A transcoding tool is not part of MediaMosa; it is a plug-in.

Transcoding can be done using transcoding profiles (*profile\_id*) that specify the transcoding tool bit rate, frame size, etc.

There are three transcoding options:

- If no *profile\_id* is provided, transcoding is launched based on a default profile. Other parameters are ignored.
- If a *profile\_id* is provided, transcoding will take place based on that profile. Other parameters are ignored.
- If a transcoding tool is specified, a *file\_extension* is also expected. In that case, a command is optional.

#### 4.7 Upload

The upload functionality in MediaMosa is split into two components:

- The *Batch Upload* component allows registered MediaMosa users to upload multiple files to MediaMosa simultaneously using FTP (section 4.7.1).
- EGA's and EGA end users can use the *File Upload* component to upload files. This is supported using the following methods:
  - File Upload via a POST request: one media file at a time is offered with the relevant metadata (section 4.7.2).
  - File Upload via a PUT request: one file (a media file or an XML file with metadata) or several files are offers by the end user (section 4.7.3) or the EGA (section 4.7.4).

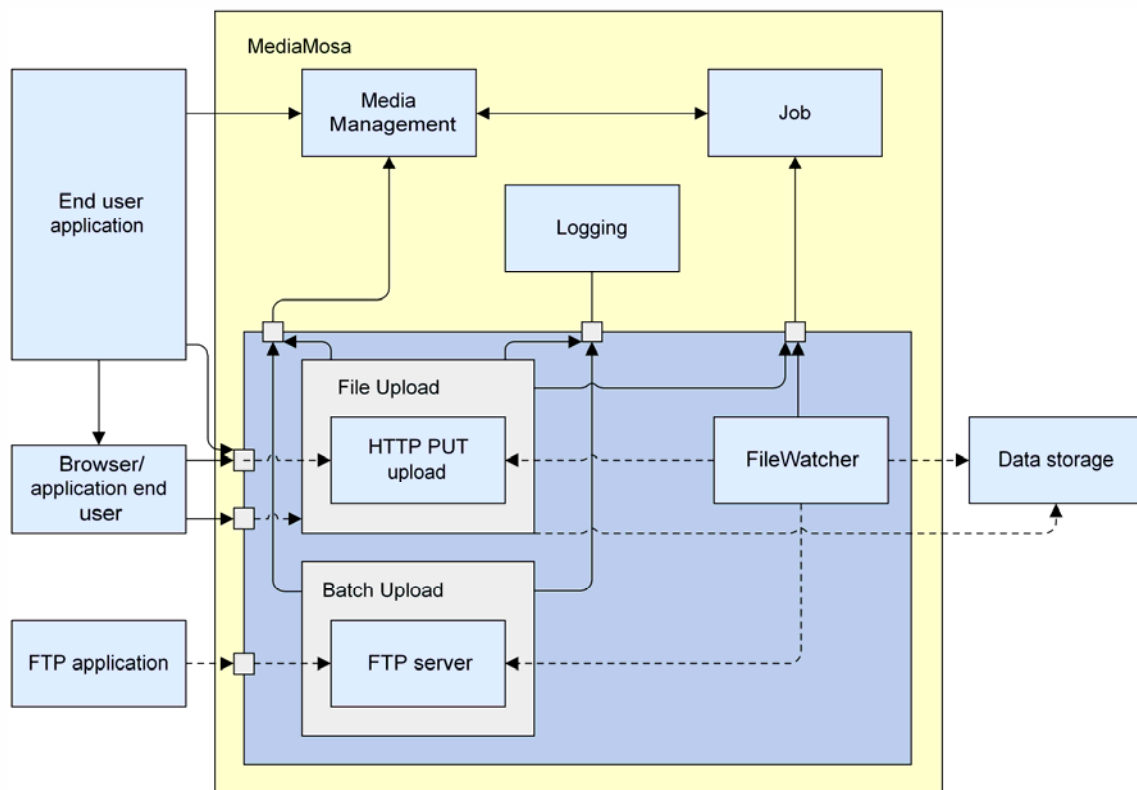


Figure 10

#### 4.7.1 Batch Upload via FTP

This method is for uploading a larger file and/or uploading several files at once. The upload is handled as follows:

1. The end user submits a request for an FTP account through the EGA. The EGA can create an FTP account via MediaMosa.
2. The end user can use this account to upload files to his own folder on the FTP server. The end user also has to make sure that a matching XML file with metadata is uploaded via FTP.
3. Once an upload is complete, the *FileWatcher* component creates a job via the *Job* component to determine the technical metadata for the media file.

#### 4.7.2 File Upload via POST request

This method is for uploading a file no larger than a few hundred MB. The upload is handled as follows:

1. The end user receives an upload ticket via the EGA.
2. The EGA shows the end user a web form. In this web form, the user can select the media file (for example a file on the local hard drive) and enter the metadata for the media file. The web form also includes the ticket number.
3. The web form is posted directly from the end user's web browser to MediaMosa (not through the EGA).

4. The *File Upload* component uses the ticket number in the POST request to check if it is a valid File Upload POST request.
5. If the metadata has been provided correctly, the media file will be moved straight to the data repository and the metadata will be recorded using the *Media Management* component. A job is also created using the *Job* component to determine the technical metadata for the media file.

#### 4.7.3 *File Upload via PUT request (upload by end user)*

This method is for uploading a larger file and/or uploading several files at once. The upload is done by the end user and is handled as follows:

1. The end user receives an upload ticket via the EGA.
2. The EGA gives the end user access to an upload application (for example a Java application).
3. The upload application sends a PUT request to MediaMosa for each media file that needs to be uploaded.
4. An XML file with metadata is sent to MediaMosa using a PUT request. The URL of each PUT request includes the ticket number.
5. The files that are received by the *File Upload* component are saved to a temporary location.
6. Once an upload is complete, the *FileWatcher* component creates a *Job* via the *Job* component to determine the technical metadata for the media file.

#### 4.7.4 *File Upload via PUT request (upload by EGA)*

This method is for uploading larger files and/or uploading several files at once. The files are uploaded to MediaMosa by the EGA. If applicable, the end user will first upload the media files to the EGA. The upload is handled as follows:

1. The EGA requests an upload ticket from MediaMosa.
2. Each media file is uploaded to MediaMosa via a PUT request.
3. A *Media Management* REST invoke sends the metadata to MediaMosa.

## 4.8 OAI-PMH

The OAI-PMH component allows OAI harvesters to retrieve the metadata from the MediaMosa media files using the OAI-PMH protocol.

The component has been implemented as an independent web service running as a Java application on a servlet container, such as Tomcat. This implementation was chosen so the OAI provider could run on an independent machine without problems, and so it would not jeopardise the performance of other services. OAI does not use any other components in MediaMosa or vice versa.

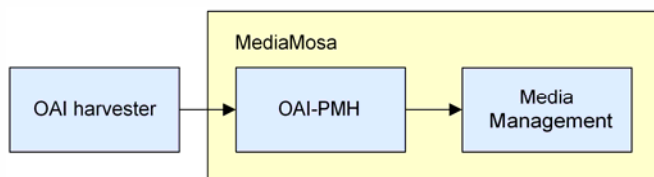


Figure 11

#### 4.9 Media Management

The *Media Management* component offers various basic functions for manipulating the assets, collections and media files, such as:

- Uploading files
- Adding metadata
- Deleting files
- Requesting overviews
- Changing ownership

A number of basic functions (such as creating a file upload ticket via HTTP PUT request) can be accessed directly by an EGA using REST invokes. However, the MediaMosa component also has a number of complex REST invokes (for example to handle a play request) that are based on the basic functions in the *Media Management* component.

The *Media Management* component interacts with nearly all the other components MediaMosa. See also the general component overview on page **Error! Bookmark not defined.**

#### 4.10 Logging

The *Logging* component offers functions for recording log entries. Configuration of which information should be logged is possible using the management application.

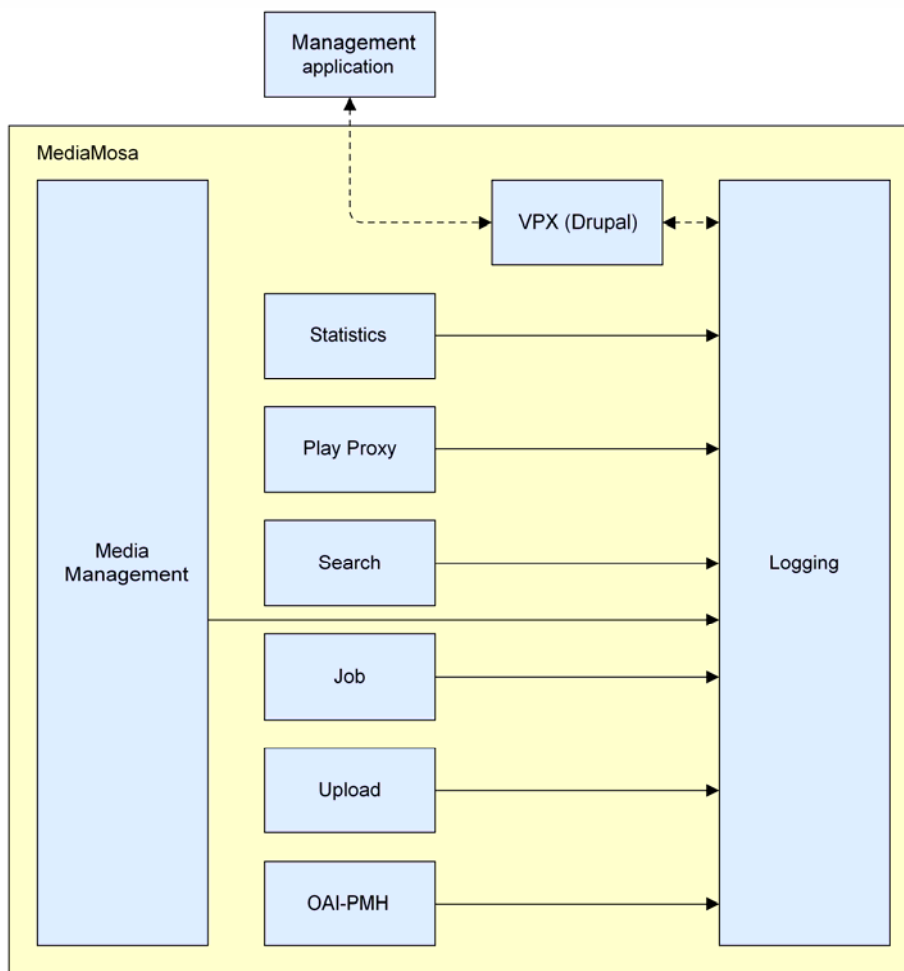


Figure 12

#### 4.11 Statistics

The following statistical overviews can be consulted via the management application:

- Latest media files
- Most popular media
- Most popular collections
- Upload of media files per month
- Disk space per container with media files
- Requested streams and media files
- Search and browse actions
- Most popular streams per period
- 50 most frequently searched words in a period

## Glossary

Term	Description
Asset	a media clip (piece of content) consisting of media files, metadata and stills.
ASX file	XML file that refers to Windows Media files that need to be played.
Blob	binary large object; a database field which can contain any type of data, such as images or video.
Collection	a group of assets.
CQL	Contextual Query Language; a language for formulating queries in information retrieval systems.
D-Bus	Desktop Bus; protocol for communication between applications.
End-user application (EGA)	an application that gives users access to the facilities provided by MediaMosa.
FFmpeg	a programme for converting audio and video streams.
HTML snippet	a piece of HTML, Javascript or CSS code that can be placed on a web page.
Job	a task that MediaMosa does in the background, while the user does other things.
Media repository	a data storage space containing media files.
Media file	a file that can be stored and downloaded through MediaMosa, such as a Quicktime file.
OAI harvester	an application that retrieves metadata in an information retrieval system.
OAI-PMH	Open Archives Initiative Protocol for Metadata Harvesting; application-independent protocol for providing and requesting metadata.
POST request	an HTTP request to carry out an action on a server.
Provider application	an application that allows content providers to provide content to MediaMosa.
PUT request	an HTTP request used to upload a source.
REST	Representational State Transfer; a type of software architecture.

<b>Term</b>	<b>Description</b>
Servlet	a Java programme running in a servlet container on a server.
Servlet container	an interface between servlets and the server they are running on.
SOA	Service Oriented Architecture; architecture in which a process is divided into layers of service that exchange information.
Still	a non-moving image (single frame) captured from a video file.
Streaming	direct distribution of audio and video over the Internet.
Symbolic link	a temporary link to a media file located outside the web root.
Ticket	a code that identifies a user submitting an upload or play request to MediaMosa as an authorised user. A ticket is used only once and expires after a short time.
Transcoding	converting a media file from one file format to another, or changing the bit rate or resolution.
Transcoding profile	a profile for transcoding media files, which specifies the transcoding tool to be used, the bit rate and the frame size, among other parameters.
Transcoding tool	a programme for transcoding a media file, such as FFmpeg or Windows Media Encoder.
White label EGA	a template for developing an end-user application.