



# Platform Refined Version

<b>Deliverable Nr Title:</b>	Deliverable D6.2.2 Platform: Refined Version
<b>Delivery Date:</b>	November 2015
<b>Author(s):</b>	Horst Stadler (Salzburg Research) Patrick Aichroth (Fraunhofer IDMT)
<b>Publication Level:</b>	Public

# Table of Contents

[Table of Contents](#)

[Documentation Information](#)

[Executive Summary](#)

[Introduction](#)

[System Architecture](#)

[Components](#)

[Service Orchestration](#)

[Extractors](#)

[Messaging](#)

[Persistence API](#)

[Linked Data Platform](#)

[File Storage](#)

[Recommendation](#)

[Endpoints](#)

[Implementation and Integration](#)

[Endpoints for data ingestion and result retrieval](#)

[Animal detection endpoint](#)

[Text analysis endpoint](#)

[Storage layer](#)

[Requirements](#)

[Comparison of available remote storage systems](#)

[Execution plan configuration](#)

[Development Infrastructure](#)

[Distribution](#)

[Debian package repository](#)

[Maven repository](#)

[Virtual Machine](#)

[Docker](#)

[Demo servers for use-case partners](#)

[Source code](#)

[Continuous Integration](#)

[Logging](#)

[Content provenance and Trust](#)

## Documentation Information

Item	Value
Identifier	D6.2.2
Author(s)	Horst Stadler (Salzburg Research) Patrick Aichroth (Fraunhofer IDMT)
Document Title	Platform: Refined Version
Actual Distribution Level	Public

## Document Context Information

Project (Title/Number)	MICO - "Media in Context" (610480)
Work Package / Task	Work Package 6: Framework Implementation and Integration
Responsible person and project partner	Horst Stadler (Salzburg Research)

## Quality Assurance / Review

Name / Partner / QA Control /	Thomas Köllmer (Fraunhofer IDMT) Thomas Kurz (Salzburg Research)
-------------------------------	---

## **Copyright**

This document contains material, which is the copyright of certain MICO consortium parties, and may not be reproduced or copied without permission. The commercial use of any information contained in this document may require a license from the proprietor of that information. Neither the MICO consortium as a whole, nor a certain party of the MICO consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.

Neither the European Commission, nor any person acting on behalf of the Commission, is responsible for any use which might be made of the information in this document.

The views expressed in this document are those of the authors and do not necessarily reflect the policies of the European Commission.

## Executive Summary

This deliverable contains the refined version of the MICO platform and integration of the current enabling technology components from work packages 2-5. It provides an implementation of the initial architecture proposed in D6.1.1<sup>1</sup>, as well as the early evolution of some aspects within the project.

## Introduction

This deliverable contains an overview of the current MICO platform architecture and outlines the major extensions and adaptations that are planned in the final year of the project regarding work package 6.

A lot of the implementation documentation is available online to make access easier for developers:

- C++/JAVA API documentation: <http://mico-project.bitbucket.org/api/1.2/>
- Source code: <https://bitbucket.org/mico-project/platform>

The target audience of this document are mainly technical people interested in working with the MICO platform. At this stage of the project developers in the consortium. The foundations of many aspects are omitted to make this concise and useful document for that target group.

---

<sup>1</sup> MICO project deliverable D6.1.1 System Architecture and Development Guidelines, Sebastian Schaffert and Sergio Fernández, 2014

# System Architecture

The initial approach of the MICO system architecture was defined in D6.1.1. The first adaptations to this approach are specified in D6.2.1<sup>2</sup> and an overview is given in the Linked Media workshop paper<sup>3</sup>. This section will provide a summary of the system architecture that will be the basis for the final evaluation. It is based on the initial architecture and addresses shortcomings that showed up in the first evaluation phase and will be addressed in the upcoming platform version 2.0. The adoptions and new requirements are mainly caused by the extended broker design and are specified in the combined deliverable D2.2.2<sup>4</sup>.

## Components

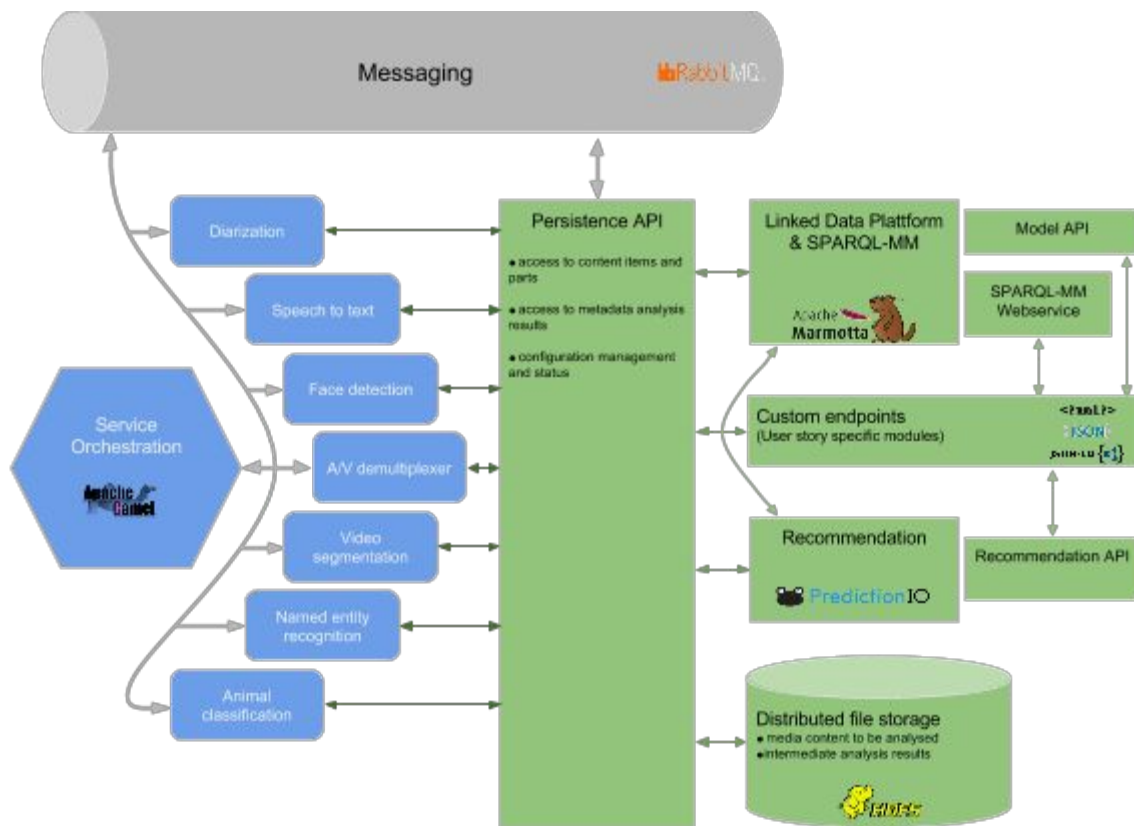


Figure 1: MICO architecture

<sup>2</sup> MICO project deliverable D6.2.1 Platform: Initial Version, Sebastian Schaffert and Sergio Fernández, 2014

<sup>3</sup> 3<sup>rd</sup> International Workshop for Linked Media LIME, MICO - Towards Contextual Media Analysis, Sebastian Schaffert, Sergio Fernández and Thomas Kurz, 2015

(<http://www.www2015.it/documents/proceedings/companion/p735.pdf>)

<sup>4</sup> MICO project combined deliverable D2.2.2/D3.2.2/D4.2.2/D5.2.2 Specifications and Models for Cross-Media Extraction, Metadata Publishing, Querying and Recommendations: Final Version, Patrick Aichroth, Johanna Björklund, Kai Schlegel, Thomas Kurz and Thomas Köllmer, 2015

## Service Orchestration

One of the main components of the MICO platform is the service orchestration. It provides a range of functions:

- *Data ingestion:* The starting point to trigger an analysis is to ingest the content that gets analyzed and to inform the broker, which is the name of the module that handles the service orchestration, that new content is available and needs to be processed. An user has several several ways to ingest content and trigger processing:
  - The broker provides an REST API to upload new content.
  - Custom endpoints provide a use case specific way to deal with it.
  - Using the command line tool *mico-inject*. This is mainly used for development.
- *Triggering extractors:* The orchestration service is responsible to distribute the processing tasks to the specific extractors in the correct order. The order is specified by an execution plan, which depends on the given input and requested output. To achieve this, the broker has to manage the execution plans as well as the running extractors. The intermediate and final results are stored on the linked data platform and file storage. The first version of the broker had several limitations specifying execution plans, as each extractor was only able to consume one type of input (defined as MIME type) and produce one output type. The extractors were chained together if an output type matches the input type of another extractor. This is quite inflexible if an extractor outputs different results or multiple inputs are required. The approach for the extended version of the broker to overcome this and other drawbacks are specified in the combined deliverable D2.2.2. This will also lead to some minor adaptations of the platform architecture.
- *Registration of extractors:* As the platform is designed as a distributed system, the broker provides a mechanism where extractors can register and unregister, so the broker can trigger their execution with the specific input and is able to take care of the next step according to the execution plan, regardless where the extractors run. For the next iteration, multiple instances of the same extractor can be run, to support load distribution.
- *Central configuration:* All components that need to fetch input or store output can do this using the persistence API or can access the linked data platform and file storage directly (e.g. extractors, clients like custom endpoints). Therefore they need to know how to connect to the linked data service and storage service. As we have a distributed architecture, it is not a good idea to have a central point as intermediate station. Each client needs to directly access the relevant service. Configuring each client (extractor, custom endpoint, ...) separately is very expensive and error prone. Therefore the broker as a central component also distributes the needed configuration to the components

during registration.

For now the configuration parameters include:

- Storage base URI
- Marmotta base URI

The only information a component needs to access the platform is the name (and port, if it differs from the default) of the RabbitMQ server. This allows to send configuration messages and receive configuration replies.

- **Status and debugging interface:** For debugging purposes the broker provides a simple user interface giving some information about the current configuration, status of the broker and requests. It also allows to inject new content and inspect existing content. One of the shortcomings that came up during the evaluation phase is, that the provided capabilities enabled by the registered extractors and configured execution plans can not be easily found out by feeder systems.

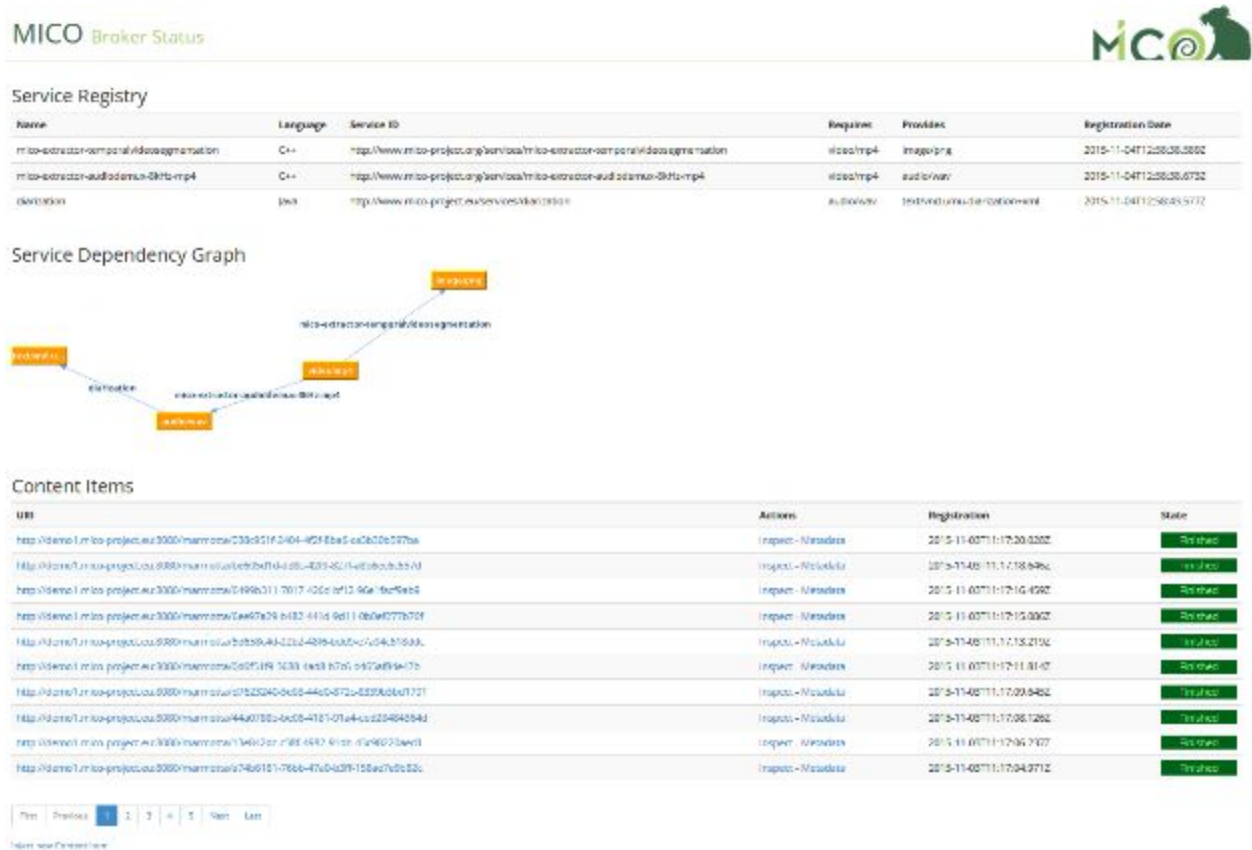


Figure 2: MICO broker user interface



## Extractors

The extractors are the components that do the actual data processing. Each extractor needs a defined set of input data and outputs a defined set of intermediate and final results. To enable the use case partners to configure the required execution plan, we set up a simple user interface to start the required execution plans:

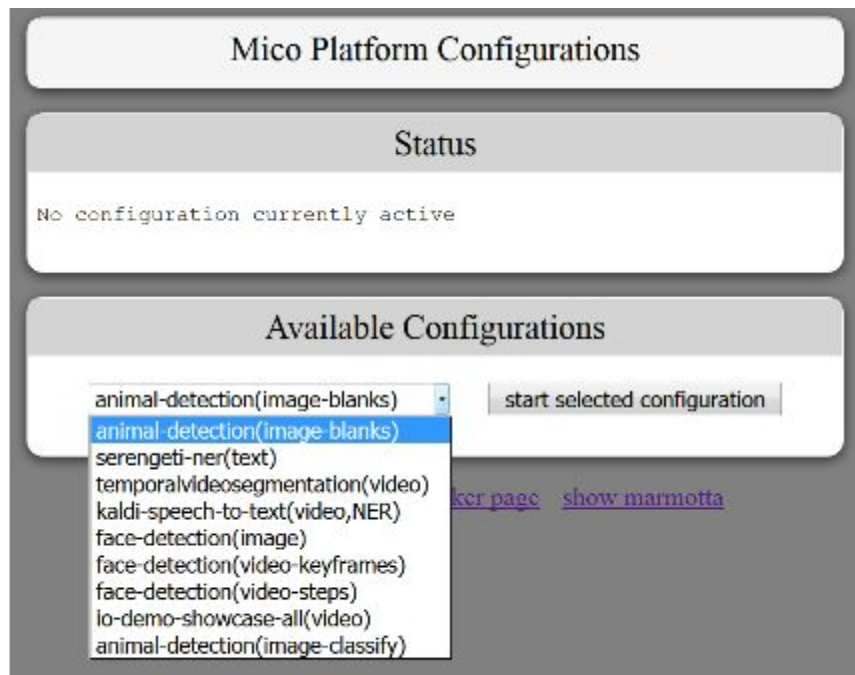


Figure 3: Execution plan configuration

The evaluation phase brought up some limitations that will be addressed in the upcoming platform:

- Especially for analyses it is necessary to get feedback about the progress and current state.
- At the moment there are no mechanisms to report problems or errors to feeder systems or the user.
- For feeder systems, like custom endpoints, it is not easy to find out which extractors and execution plans are available and if a requested output can be provided. Moreover the updated platform should set up required extractors just by defining the input type(s) and requested output type(s).

## Messaging

The messaging follows an event based approach. Therefore the *Advanced Message Queuing Protocol*<sup>5</sup> (AMQP) is used. As a platform and language independent middleware that implements AMQP, we rely on *RabbitMQ*<sup>6</sup> as message and event exchanges. In the MICO platform we use one-to-one messaging, where a publisher writes messages into a queue that is read by a consumer, as well as one-to-many messaging, where a single or a set of consumers can register to a queue for specific messages, and producers send the message to these consumers.

The entire message routing is done with *RabbitMQ*. As AMQP does not define a message interchange format we use *Protocol Buffers*<sup>7</sup> for that. It allows a language independent format definition of messages that gets compiled into a highly efficient binary representation.

The following message types are currently used:

- *Registration event*: Allows an extractor to register or unregister by providing its service identifier and input queue name.
- *Analysis event*: The broker sends this type of message to an extractor to trigger processing of a content item. The extractor replies using this message to inform the broker about new (intermediate) results.
- *Content event*: Informs the broker about a new content item, so it gets registered and prepared for the upcoming analysis.
- *Discovery event*: Allows a broker to request a registration of available extractors.
- *Configuration discover event*: Used by extractors to get the configuration for the linked data storage and binary storage. This is also used by brokers to check if there is another broker available and adopt its configuration values.
- *Configuration event*: Reply to the configuration discover event, sent by the broker.

This set of messages will be extended to enable extractors progress information, error reporting, etc.

## Persistence API

The persistence API provides feasible access to content parts, content items and metadata. *Content Items* are a collection of media resources together with their context in the MICO platform. A context covers all other resources that are directly related to the main object. For example, an HTML document might contain images and videos – the content item would in this case be the HTML document itself, as well as all images and videos it contains. Each resource within a content item is called *Content Part*. Both types also may have additional metadata like

---

<sup>5</sup> <http://www.amqp.org>

<sup>6</sup> <https://www.rabbitmq.com>

<sup>7</sup> <https://developers.google.com/protocol-buffers/>

provenance, creation, format, etc. Intermediate or final results of the analysis process are stored as content items and content parts, too.

The API handles all the communication with the broker via the message bus, saves and loads files from the file storage and stores and retrieves linked media data. It is available for for JAVA and C++.

## Linked Data Platform

Apache Marmotta<sup>8</sup> is used as metadata storage. It allows to access to the metadata using LDPATH<sup>9</sup> web service or SPARQL<sup>10</sup>. Moreover the SPARQL implementation is extended by SPARQL-MM<sup>11</sup> to enable multimedia querying functions by introducing spatio-temporal filter and aggregation functions to handle media resources and fragments that follow the W3C standard for Media Fragment URIs<sup>12</sup>. Marmotta also supports access (with some restrictions<sup>13</sup>) to store and fetch RDF data according the Linked Data Platform specification<sup>14</sup>.

As backend for Apache Marmotta the KiWi Triplestore<sup>15</sup> is used, that in turn utilizes PostgreSQL<sup>16</sup> as its relational database storage.

## File Storage

Depending on the respective application purpose, different types of file storage systems are suitable. One of main objectives of the MICO platform is to provide a distributed system. It is quite evident that the file storage has to fit to the needs of a distributed system too. So we decided to use the Hadoop File System<sup>17</sup> (HDFS) as our distributed storage (see section [Storage layer](#) to have a more detailed overview of requirements and available solutions). Nevertheless powerful systems, like HDFS, are very complex and the a proper setup can be very time-consuming, especially in environments where the user won't benefit from the additional functions. Therefore we also support storage systems that are simpler to set up, like the

File Transfer Protocol<sup>18</sup> (FTP), as it was chosen as the first storage type to go with. Additionally we will support simple local filesystems for development purposes.

<sup>8</sup> <http://marmotta.apache.org>

<sup>9</sup> <https://marmotta.apache.org/ldpath/language.html>

<sup>10</sup> <http://www.w3.org/TR/sparql11-overview/>

<sup>11</sup> <https://github.com/tkurz/sparql-mm>

<sup>12</sup> <http://www.w3.org/TR/media-frags/>

<sup>13</sup> <https://marmotta.apache.org/platform/ldp-module.html>

<sup>14</sup> <http://www.w3.org/TR/ldp/>

<sup>15</sup> <https://marmotta.apache.org/kiwi/triplestore.html>

<sup>16</sup> <http://www.postgresql.org/>

<sup>17</sup> <https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Overview>

<sup>18</sup> <https://tools.ietf.org/html/rfc959>

## Recommendation

The recommendation functionalities will be driven by an engine providing basic functionalities and custom recommendation modules addressing user story specific modules. For collaborative filtering tasks following a machine-learning approach, Prediction.IO<sup>19</sup> will be used. Data storage and querying will be provided by the file storage layer and the linked data layer.

## Endpoints

To provide additional benefit for a typical platform user, it is important to have easy-to-use interfaces, so a user does not need to worry about the details or internals of the MICO platform or one of its components (like extractors). From a user perspective a valuable interface has to provide the following functions:

- *Content ingestion*: The content that has to be analyzed needs to be provided to the platform. This might also be just a location (e.g. an URI) where the content lives, so the endpoint can fetch it for analysis. To reference the content the caller might get its platform specific IDs.
- *(Auto-)Configuration of extractors*: To perform the analysis the proper extractors have to be in place. So it has to be ensured that the request can be processed.
- *Start analysis*: In most cases the processing should start as soon the content is provided. Anyway there might be situations where this event has to be triggered explicitly. The client in turn needs to get notified, if the processing finishes or is interested in progress updates. This can be achieved by registering callbacks. Otherwise a regular polling for results is necessary.
- *Providing results*: One of the main reasons to have a custom endpoint is to provide the requested analysis results, without any intermediate or unneeded results, in a proper data format.

These functions may be combined in a single request or split up among multiple request, depending on the needs.

As an endpoint needs to be customizable to fit on the needs of a specific user story, we will provide a template for developing endpoints, that is capable of the major basic functionalities without extra effort, but can easily be extended to fit custom needs. This is important especially regarding the format of the results and the data format an endpoint provides.

We plan to go with the Apache Maven Archetypes<sup>20</sup> plugin, which is a project templating toolkit, for the JAVA part. For extractors written in C++ we have a template and a howto that will be extended and get released to the public as soon this is mature.

---

<sup>19</sup> <https://prediction.io>

<sup>20</sup> <https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>

As first exemplary custom endpoints a web service for the text analysis and animal (blank) detection got implemented. The interface is described in section [Endpoints for data ingestion and result retrieval](#). These will be used as a foundation to develop the templates.

## Implementation and Integration

This section contains details of selected implementation tasks.

### Endpoints for data ingestion and result retrieval

The platform itself provides all the necessary interfaces to ingest content, start the processing and fetch results.

For example the broker provides an REST API to upload new content and trigger its analysis:

First step:

<b>Action:</b>	Create a new content item
<b>Path:</b>	/broker/inject/create
<b>Request method:</b>	POST
<b>Responses</b>	200 <i>application/json</i>
	<pre>{   "uri": URI of the created content item as String }</pre>
	500 Error

Second step:

<b>Action:</b>	Upload new content part
<b>Path:</b>	/broker/inject/add
<b>Request method:</b>	POST
<b>Parameters:</b>	ci URI of content item
	type MIME type of the content type
	name Name of the content type

<b>Payload:</b>	Data to store as the content part
<b>Responses</b>	200 <i>application/json</i>
	<pre>{   "uri": URI of the created content item as String }</pre>
	500 Error

Third step:

<b>Action:</b>	Trigger the analysis process
<b>Path:</b>	/broker/inject/submit
<b>Request method:</b>	POST
<b>Parameters:</b>	ci URI of content item
<b>Responses</b>	200 OK
	500 Error

The results can be queried by using the SPARQL-MM service or, depending on the extractors used, fetch the (intermediate) results from the file storage.

As described in section [Endpoints](#), there are several reasons for the need of a custom endpoint. We implemented the following two for the evaluation phase.

### Animal detection endpoint

The analysis of a new JPEG image can be started by:

<b>Action:</b>	Analyze JPEG image by providing the image
<b>Path:</b>	/broker/zooniverse/animaldetection
<b>Request method:</b>	POST
<b>Payload:</b>	The image to analyze.
<b>Responses</b>	200 <i>application/json</i>

	<pre>{   "id": ID of the created content part as String   "status": "submitted" }</pre>
	500 Error

or providing the URL of an image:

<b>Action:</b>	Analyze JPEG image by providing the URL of the image
<b>Path:</b>	/broker/inject/add
<b>Request method:</b>	PUT
<b>Parameters:</b>	<code>url</code> HTTP URL to fetch the image from
<b>Responses</b>	200 <i>application/json</i>
	<pre>{   "id": ID of the created content part as String   "status": "submitted" }</pre>
	500 Error

The result can be fetched by:

<b>Action:</b>	Fetch result of the animal detection
<b>Path:</b>	/broker/zooniverse/animaldetection/<content part ID>
<b>Request method:</b>	GET
<b>Responses</b>	200 <i>application/json</i>
	<pre>{   "status": "InProgress" or "finished"   "processingBegin": Start of processing as DateTime    The following fields are only available if status is finished:   "processingEnd": End of processing as DateTime   "objectsFound": Total number of identified animals as Integer   "objects": [ Array of identified animals.   {     "algorithmVersion": Algorithm identifier as String     "confidence": Identification confidence level as Float</pre>

	<pre> "animal": Name of the species as String "x": x axis of rectangle marking the animal in pixels as Integer "y": y axis of rectangle marking the animal in pixels as Integer "w": width of rectangle marking the animal in pixels as Integer "h": height of rectangle marking the animal in pixels as Integer }, ... ] } </pre>
404	A content part with the given ID could not be found.
500	Error

Example of an animal detection result:

```

{
  "status": "finished",
  "processingBegin": "2015-11-03T11:39:37Z",
  "objectsFound": 20,
  "objects": [{
    "w": 190,
    "algorithmVersion": "01-vsBlank-HOG-CLASS",
    "confidence": "3.261539936065674",
    "h": 270,
    "y": 0,
    "x": 487,
    "animal": "wildebeest"
  },
  {
    "w": 87,
    "algorithmVersion": "01-vsBlank-HOG-CLASS",
    "confidence": "2.1107399463653564",
    "h": 124,
    "y": 37,
    "x": -49,
    "animal": "wildebeest"
  },
  ...
  ],
  "processingEnd": "2015-11-03T10:39:40Z"
}

```



## Text analysis endpoint

The Serengeti comments analysis services can is triggered by

<b>Action:</b>	Analyze comment
<b>Path:</b>	/broker/zooniverse/textanalysis
<b>Request method:</b>	POST
<b>Payload:</b>	<p><i>application/json</i></p> <pre>{   "comment": The comment to analyze as String. }</pre>
<b>Responses</b>	<p>200 <i>application/json</i></p> <pre>{   "id": ID of the created content item as String   "status": "submitted" }</pre> <p>503 Service is unavailable. This indicates, that the required extractors are not available.</p> <p>500 Error</p>

The result can be fetched with

<b>Action:</b>	Fetch result of the comment analysis
<b>Path:</b>	/broker/zooniverse/textanalysis/<content item ID>
<b>Request method:</b>	GET
<b>Responses</b>	<p>200 <i>application/json</i></p> <pre>{   "id": Content item ID as String   "status": "InProgress" or "finished"    The following fields are only available if status is finished:   "sentiment": The sentiment value as Float   "topics": [ Array of identified topics   {</pre>

	<pre> "label": Topic label as String "confidence": Identification confidence level as Float "uri": Link to DBPedia knowledge base about topic as String }, ... ] "entities": [ Array of identified entities { "label": Entity label as String "uri": Link to DBPedia knowledge base about entity as String }, ... ] } </pre>
404	A content part with the given ID could not be found.
500	Error

Example of a text analysis result:

```

{
  "id": "280197d7-4a27-43c2-8e84-735677971e27",
  "status": "finished",
  "sentiment": 0.3161099552911122,
  "topics": [
    { "label": "Sport", "confidence": 0.3983224928379059,
      "uri": "http://dbpedia.org/resource/Sport" },
    { "label": "Environment", "confidence": 0.24880832433700562,
      "uri": "http://dbpedia.org/resource/Environment" },
    { "label": "Religion", "confidence": 0.3528691828250885,
      "uri": "http://dbpedia.org/resource/Religion" }
  ],
  "entities": [
    { "label": "running",
      "uri": "http://dbpedia.org/resource/Running" },
    { "label": "lions",
      "uri": "http://dbpedia.org/resource/Lion" },
    { "label": "Serengeti",
      "uri": "http://dbpedia.org/resource/Serengeti" },
    { "label": "love",
      "uri": "http://dbpedia.org/resource/Romance_(love)" },
    { "label": "lions", "uri":
      "http://www.mico-project.eu/ns/cv/serengeti#lion" },
    { "label": "through",
      "uri": "http://dbpedia.org/resource/Ford_(crossing)" }
  ]
}

```

## Storage layer

In the first version of the MICO platform we decided to use the FTP as content storage protocol. In order to benefit from the advantages of the distributed approach, the platform has to support a distributed storage system as well. Therefore we defined requirements that are relevant in the context of the MICO platform and compared available distributed storage systems with respect to these requirements.

## Requirements

	ID	Name	Description / Motivation	Priority
Functional Requirements	FR-1	Data Distribution	Data is not required to be moved to a central storage (geographic distribution of the storage servers)	Medium
	FR-3	Pseudo-Streaming	Skipping data fragments or retrieval of specific fragments.	Medium
	FR-6	Native transport encryption	Protect data transferred over the network. Related for both FR-4 and FR-5.	Medium
	FR-7	Data integrity	Check the integrity of information to detect data corruption.	Low / Medium
	FR-8	Replication	Protect from component failure and minimize latency.	Low
Non-Functional Requirements	NFR-1	Maturity	The system should be mature and preferably easy to install.	High
	NFR-2	License	An open license for the allows to provide easy to set up platform installation packages.	High
	NFR-3	Supported platforms	Availability and portability across different software (and hardware) platforms.	High
	NFR-6	Scalability	Easy way to adapt to the needs of the resources currently required.	High
	NFR-7	Access API	Simple integration into the platform: The data storage system should fit into the	High

		API and existing parts of platform.	
--	--	-------------------------------------	--

Another aspect that might require future work in commercial applications, especially in commercial scenarios, is access control for storage: In some application scenarios, it will be necessary to limit access to content provided, e.g. based on certain roles and groups. This requires provenance tracking, as described above, and two more additions to the system: Storage of respective limitation in the model, and enforcement of access control constraints especially in the storage domain. While the former is relatively easy to add to the existing MICO model, and will be provided in year 3 in an interaction between T6.4 and work package 3, the latter requires support of access control within storage itself, which is more effort. As HDFS provides support for POSIX conform Access Control Lists, a respective storage layer could be included in the future.

### Comparison of available remote storage systems

Based on the requirements we had a look at available remote storage systems and did an evaluation of them regarding the prioritized requirements. Based on these results we have chosen to use HDFS as storage system, while still being able to switch to FTP, if necessary.

	BeeGFS	Ceph	CRATE	Eucalyptus	GlusterFS	HDFS	MogileFS	GridFS	Swift	RiakCS	XtreemFS
FR-1	no	yes	no	no <sup>21</sup>	no <sup>22</sup>	partly <sub>23</sub>	partly <sub>24</sub>	partly <sub>25</sub>	yes	no <sup>26</sup>	yes
FR-3	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes
FR-4	no	yes	no	yes	no	yes	no	yes	yes	yes	yes
FR-6	no	no	yes	yes	no	yes	no	yes	no	yes	yes

<sup>21</sup> The enterprise version of RiakCS can be used as a back-end, allowing multi-datacenter replication. See RiakCS for restrictions.

<sup>22</sup> GlusterFS supports striping and replication. It also provides a geo-replication feature, to mirror data across geographically distributed clusters. Therefore, this is useful in case of disaster recovery but not for distribution.

<sup>23</sup> only for replicas

<sup>24</sup> It does not have an ability to run its database or trackers in multiple locations.

<sup>25</sup> MongoDB allows replica sets to be deployed in multiple data centers. Clients can be statically configured to read data from a specific replica. Write access is possible only on the primary instance.

<sup>26</sup> Only the commercial extension Riak Enterprise offers a geographical distribution of data. However, this feature does not support access of the closest data.

FR-7	no	partly <sup>27</sup>	no	no	no	yes	no	no	yes	yes	yes
FR-8	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes
NFR-1	yes	yes	yes	yes	yes	yes	no <sup>28</sup>	yes	yes	yes	yes
NFR-2	proprietary (free of charge), Client: GPL	LGPL 2.1	Apache License 2.0	GPL v3	GPL v3	Apache License 2.0	GPL / Artistic License	AGPL v3.0 <sup>29</sup>	Apache License 2.0	Apache License 2.0	New BSD
NFR-3	RHEL/Fedora, SLES/Open Suse, Debian/Ubuntu	RHEL/Fedora/CentOS, Debian/Ubuntu	Java	RHEL/CentOS, Ubuntu	RHEL/Fedora/CentOS/Pidora, SLES/Open Suse, Debian/Ubuntu	Java (Linux, Windows)	Perl, Java, Ruby, PHP, Python	Fedora/CentOS, Debian/Ubuntu, FreeBSD, OS X, Solaris, Windows,	Python (Ubuntu, Fedora/CentOS)	Debian/Ubuntu, RHEL/Fedora, FreeBSD, OS X, Solaris, Smart OS	RHEL/Fedora/CentOS, SLES/Open Suse, Debian/Ubuntu
NFR-6	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
NFR-7	POSIX	librados (C, C++, Python, Ruby), S3, Swift, FUSE	JAVA, Python, Ruby, PHP, Scala, node.js, Erlang, mono/.NET, Go	HTTP	libglusterfs, FUSE, NFS, SMB, Swift, libgfapi	Java and C client, HTTP	Perl, HTTP	C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala	python-swiftclient	HTTP	libxtr emfs (Java, C++), FUSE

<sup>27</sup> see: <http://lists.ceph.com/pipermail/ceph-users-ceph.com/2014-January/007540.html>

<sup>28</sup> The last update was some time ago. (Even it might be used in practice, it seems to be poorly supported.)

<sup>29</sup> JAVA and C++ drivers (clients) are released under Apache license 2.0

The installation, configuration and maintenance is very time-intensive, as any distributed storage system is very complex and therefore it is necessary to deal more intensely with it to get and keep it running properly. Moreover the advantages of HDFS is negligible for small systems. Hence it might not fit for every use case. That is the reason to keep FTP as an option. In addition we will provide a local storage module to use the local file system as storage system.

## Execution plan configuration

In a typical cross-media analysis scenario, it takes several different extractors that do the processing. As the current broker implementation just handles static execution plans, a simple user interface was developed so end users (like our show case partners) are able to set up the required execution plan (see figure 3).

As the target platform is Debian Linux (Jessie AMD64 architecture) and the extractors run as daemons, the execution plan configurations are implemented as shell scripts, which bring up the required extractors with the execution plan specific configuration.

## Development Infrastructure

This section gives an overview of the development infrastructure that is in place and publicly available.

### Distribution

The MICO platform is available as virtual machine image and as Debian packages. Both sources provide the main components of the platform (service orchestration, persistence API including Apache Marmotta with contextual extensions, binary storage and messaging) as well as the available extractors. Some of them have to be installed separately, as they are not publicly available.

The only difference between the two is, that the virtual image uses HDFS as storage and the default Debian package setup uses FTP.

### Debian package repository

All developed modules are available as Debian packages and designed for Jessie with AMD64 architecture. The public repository is available at <http://apt.mico-project.eu/>. Instructions how to install the platform from scratch is available on the MICO project webpage <http://www.mico-project.eu/mico-release/#package-section>.

Some of the extractors are not publicly available. The necessary credentials for these extractors can be requested from the [project coordinator](#).

The MICO platform and the extractors rely on many libraries<sup>30</sup>. While all JAVA dependencies are resolved using Maven, not all dependencies of the C++ persistence API and some extractors are available as Debian packages. Therefore we created Debian packages for these and made them available via the MICO package repository:

- AMQP-CPP<sup>31</sup>: Needed for communicating with the RabbitMQ master.
- libhdfs3<sup>32</sup>: This is a native HDFS client implementation and does not make use of the Java Native Interface, keeping it lightweight with a small memory footprint.
- Kaldi<sup>33</sup>: Kaldi is a toolkit for speech recognition written in C++ and used by the speech-to-text extractors.

A list of Debian packages available from the MICO package repository (some of them are not public available):

Package name	Description
libmico-api1	This package contains the compiled shared libraries of the MICO C++ API. They are needed by all binary implementations of MICO C++ services.
libmico-api-dev	This package contains the MICO C++ API header files and documentation. These are needed for implementing custom MICO services in C++.
libmico-api-java	This package contains the compiled libraries of the MICO Java API. They are needed by all Java-based implementations of MICO services. Note that the libraries can also be installed via Maven (recommended).
mico-apt-key	This package contains the public GPG key used to sign packages in the MICO repository.
mico-apt-repository	This package contains the sources.list and public GPG key used to sign packages in the MICO repository (replaces mico-apt-key).
mico-base	This package contains some base configurations used by the MICO platform. Configuration options will be asked interactively when installing this package.
mico-broker	This package contains the distribution of the MICO Broker Web Application.
mico-conf	This package contains the distribution of the MICO Platform

<sup>30</sup> <https://bitbucket.org/mico-project/platform/#markdown-header-prerequisites>

<sup>31</sup> <https://github.com/CopernicaMarketingSoftware/AMQP-CPP>

<sup>32</sup> <https://github.com/PivotalRD/libhdfs3>

<sup>33</sup> <http://kaldi-asr.org>

	Configuration Web Application.
mico-marmotta	This package contains the distribution of Apache Marmotta used by the MICO Platform.
mico-persistence	This package is a configuration-only package setting up ProFTPD as persistence server for the MICO platform.
mico-platform	This is a metapackage installing a complete setup of the MICO platform, including the server and C++ client components and online documentation. It installs a lightweight http server with a simple entry page for single point access to the services.
mico-rabbitmq	This is a configuration package setting up RabbitMQ for the MICO platform. It configures the necessary RabbitMQ extensions and user permissions.
mico-extractor-configurations	MICO configuration scripts for extractor pipelines.
mico-service-ocr	This package contains a binary version of the sample OCR service implemented in C++ to use with the MICO platform.
mico-extractor-audiodemux	MICO service for audio demuxing and downsampling.
mico-extractor-ccv-facedetection	MICO service for detection of faces in videos.
mico-extractor-diarization	This package contains the Speaker Diarization extractor daemon, responsible of providing support for TE-214 in MICO.
mico-extractor-speech-to-text	MICO service for automatic transcription, based on Kaldi.
mico-extractor-kaldi2rdf	This package contains the extractor daemon, responsible of providing support for speech to text in MICO. It prepares the result of the mico-extractor-speech-to-text extractor and provides it as RDF.
mico-extractor-kaldi2txt	This package contains the extractor daemon, responsible of providing support for speech to text in MICO. It prepares the result of the mico-extractor-speech-to-text extractor and provides it as plain text.
mico-extractor-named-entity-recognizer	This package contains the Named-Entity Recognizer Daemon, responsible of providing support for TE-220 in MICO. Its implementation is currently based on a third-party service.
mico-extractor-object-detection-rdf	This package contains the object-detection to RDF extractor daemon, responsible of providing support for faces and animal detection in MICO.
mico-extractor-temporal-segments-rdf	This package contains the Temporal Segments to RDF extractor daemon, responsible of providing support for TE-206 in MICO.



mico-extractor-temporalvideo segmentation	MICO service for detection of shot boundaries and keyframes in videos.
mico-extractors-3rdparty	Additional MICO 3rd party libraries needed by extractors.
libamqpcpp2	This package contains the AMQP-CPP C++ library. It is used for communicating with a RabbitMQ message broker.
libamqpcpp-dev	AMQP-CPP development package.
libhdfs3-1	This library provides native C/C++ access to HDFS. Libhdfs3, designed as an alternative implementation of libhdfs, is implemented based on native Hadoop RPC protocol and HDFS data transfer protocol. It gets rid of the drawbacks of JNI, and it has a lightweight, small memory footprint code base.
libhdfs3-dev	libhdfs3 development package.
kaldi	Kaldi is a toolkit for speech recognition written in C++. It is intended for use by speech recognition researchers.
kaldi-dev	Kaldi (including OpenFst) development package.
kaldi-libs	Kaldi (and OpenFst) shared libraries.

## Maven repository

The released versions of the JAVA platform API are available from our Maven repository at <http://mvn.mico-project.eu/>.

## Virtual Machine

The virtual machine image provides a ready-to-use installation of the platform and can be downloaded as Open Virtualization Format (OVF) from <http://apt.mico-project.eu/download/MICO%20Platform%20current.ova>. This format is supported by most current virtualization software. We recommend the use of VirtualBox<sup>34</sup>.

---

<sup>34</sup> <https://www.virtualbox.org>

## Docker

For future releases we are thinking about providing Docker<sup>35</sup> images for the MICO platform, as it is designed to quickly build, ship, and run distributed applications at scale and would simplify deployment.

## Demo servers for use-case partners

In order to offer the best possible support to our use case partners during the evaluation phase, we set up a server for each use case partner running the MICO platform. By having the MICO platform running on a server accessible by every partner, we were able to clarify questions, as well as understand and fix problems in a quick and easy way on short-notice.

## Source code

The source code of the MICO project platform (and other components) is publicly available at: <https://bitbucket.org/mico-project/platform>

Installation instructions and first steps are described here:

<http://www.mico-project.eu/mico-release/>

## Continuous Integration

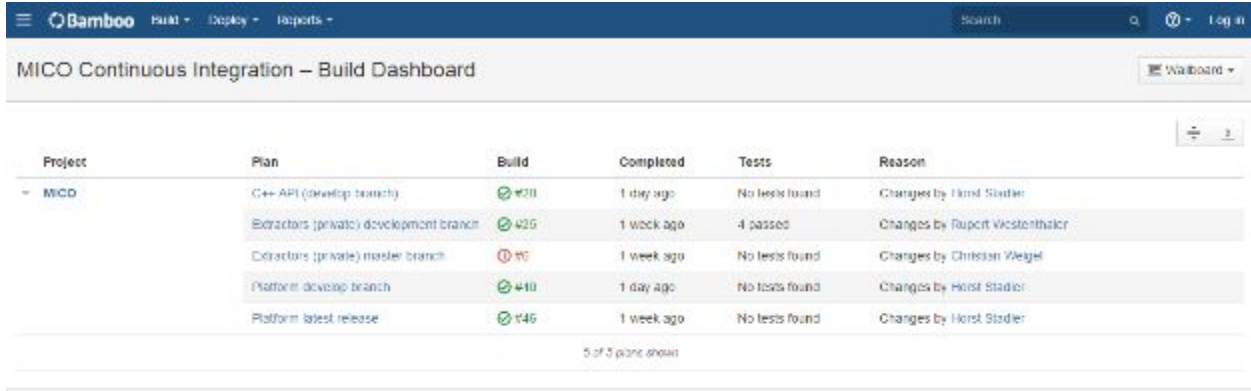
Bamboo<sup>36</sup> is used as continuous integration platform for the MICO components. We have created build plans for our main components (MICO platform including JAVA API, C++ API and extractors). To take full advantage, the next step is to set up a (semi-)automatic deployment as well as adopt and extend the build plans to these needs. This will also include nightly-builds to further facilitate the development process

The dashboard is available at <http://ci.mico-project.eu>.

---

<sup>35</sup> <https://www.docker.com>

<sup>36</sup> <https://www.atlassian.com/software/bamboo>



Project	Plan	Build	Completed	Tests	Reason
MICO	C++ API (develop branch)	420	1 day ago	No tests found	Changes by Horst Stodier
	Extractors (private) development branch	425	1 week ago	4 passed	Changes by Rupert Westenthaler
	Extractors (private) master branch	45	1 week ago	No tests found	Changes by Christian Weigel
	Platform develop branch	410	1 day ago	No tests found	Changes by Horst Stodier
	Platform latest release	445	1 week ago	No tests found	Changes by Horst Stodier

5 of 5 plans shown

Figure 4: Continuous Integration Dashboard

## Logging

One downside, that turned out during the evaluation phase and has not been addressed yet is the lack of a centralized logging system. The MICO platform consist of many different components that might run, due to its distributed architecture, on different systems. As all components produce logging data in different locations and files with diverse log formats, it can get very cumbersome and time-consuming to get through the logging data to locate and bring together the required information.

Therefore we propose to set up centralized logging system for the MICO platform based on Logstash<sup>37</sup>, Elasticsearch<sup>38</sup> and Kibana<sup>39</sup>. This allows us to search through all of the logs and identify issues in a single place. It is also useful to identify issues that span multiple servers by correlating their logs during a specific time frame.

To accomplish this, Logstash will be used to collect, parse, normalize and transport the logging data. This gets stored into Elasticsearch a RESTful NoSQL store and search engine that allows data analyzation in real-time. The user interface to explore, filter and visualize the logging data will be driven by Kibana.

<sup>37</sup> <https://www.elastic.co/products/logstash>

<sup>38</sup> <https://www.elastic.co/products/elasticsearch>

<sup>39</sup> <https://www.elastic.co/products/kibana>

## Content provenance and Trust

One important aspect of the system is to support provenance tracking, i.e. to be able to identify where information stems from or was created across the whole acquisition and annotation chain. On one hand, this is supported by the annotation and broker models, which keep track of annotators and on their involvement within workflows and jobs, as described for work package 2 and 3. On the other hand, this should also include the ability to authenticate metadata and annotations that are created outside of the MICO system, and are imported e.g. via crawling from websites / portals. After consideration of several alternatives, the choice was to enable this by modifying existing FHG background components to sign and authenticate microformats<sup>40</sup>. The resulting components allow arbitrary signature and verification of HTML documents via XPATH. They can also be used to sign and verify copyright and provenance information embedded within the HTML, and provide the following *MicroformatSignature* class API:

<b><i>MicroformatSignature</i> class</b>									
<b>Constructor:</b> Builds a new object for creating and verifying microformat signatures for HTML/XML documents.									
<pre>public MicroformatSignature (     InputStream keystoreInputStream     String keystorePassword     String keyAlias     String keyPassword )</pre>	<table border="1"> <tr> <td>InputStream keystoreInputStream</td> <td>Input stream for Java keystore that contains private key used for signing</td> </tr> <tr> <td>String keystorePassword</td> <td>Password for keystore</td> </tr> <tr> <td>String keyAlias</td> <td>Alias of the private key</td> </tr> <tr> <td>String keyPassword</td> <td>Password for private key</td> </tr> </table>	InputStream keystoreInputStream	Input stream for Java keystore that contains private key used for signing	String keystorePassword	Password for keystore	String keyAlias	Alias of the private key	String keyPassword	Password for private key
InputStream keystoreInputStream	Input stream for Java keystore that contains private key used for signing								
String keystorePassword	Password for keystore								
String keyAlias	Alias of the private key								
String keyPassword	Password for private key								
<b>buildFor:</b> Creates a new microformat signature node for the given HTML/XML node.									
<pre>public Node buildFor (     Node node )</pre>	<table border="1"> <tr> <td>Node node</td> <td>HTML/XML node to sign (can also be a Document object)</td> </tr> </table>	Node node	HTML/XML node to sign (can also be a Document object)						
Node node	HTML/XML node to sign (can also be a Document object)								

<sup>40</sup> <http://microformats.org>

@return Node	Signature node created
<u>checkFor</u> : Verifies the embedded microformat signature for a given HTML/XML node.	
<pre>public CheckResult checkFor(     Node node )</pre>	HTML/XML node to check (can also be a Document object)
@return CheckResult	Enum: <ul style="list-style-type: none"> <li>• signatureValid</li> <li>• signatureInvalid</li> <li>• signatureNotFound</li> </ul>
<u>checkAllDocumentSignatures</u> : Verifies all microformat signatures found in the given document.	
<pre>public Map&lt;String, CheckResult&gt; checkAllDocumentSignatures(     Document doc )</pre>	Document to examine
@return Map<String, CheckResult>	Map containing for each signed node: <ul style="list-style-type: none"> <li>• XPath expression and</li> <li>• CheckResult value</li> </ul> Enum: <ul style="list-style-type: none"> <li>○ signatureValid</li> <li>○ signatureInvalid</li> <li>○ signatureNotFound</li> </ul>

These components will be integrated into the platform in year 3, and can then be used to authenticate content and metadata including authorship and related copyright information (as extracted from content, the web, or derived from the retrieval context). They will be complemented by tools to extract authorship and copyright information from content and metadata based, in order to store them in the knowledge base and query them.