Platform
Final Version

Deliverable Nr **Title:**	Deliverable D6.2.3 Platform: Final Version
Delivery Date:	October 2016
Author(s):	Horst Stadler (Salzburg Research) Thomas Kurz (Salzburg Research) Patrick Aichroth (Fraunhofer IDMT)
Publication Level:	Public

Table of Contents

Documentation Information

Item	Value
Identifier	D6.2.3
Author(s)	Horst Stadler (Salzburg Research) Patrick Aichroth (Fraunhofer IDMT)
Document Title	Platform: Final 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)

Executive Summary

This deliverable contains the final version of the MICO platform and integration of the enabling technology components from work packages 2-5. It provides an implementation of the initial architecture proposed in D6.1.1[1] and extended in D6.2.2[2].

The implementation documentation is available online to make access easier for developers:

- C++/JAVA API documentation:http://mico-project.bitbucket.org/api/3.1/
- 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.

In order to demonstrate the MICO platform functionality within a web environment we created a generic MICO platform demonstrator, which can be found on http://demo3.mico-project.eu/.

It is purely Platform based, which means it only uses the web services that have been described above. Additionally it combines several frontend software projects that have been developed and/or adapted within the project, which are Squebi (for SPARQL result presentation), the workflow-manager (for the creation and the selection of workflows), and the balloon synopsis (for extraction result presentation). Thus the demo supports:

- Workflow creation
- Workflow selection
- Data ingest (text, image, video, audio)
- Type-specific result presentation
- Generic result presentation
- SPARQL access

[1] MICO project deliverable D6.1.1 System Architecture and Development Guidelines, Sebastian Schaffert and Sergio Fernández, 2014
[2] MICO project deliverable D6.2.2 Platform: Refined Version, Horst Stadler and Patrick Aichroth, 2015

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[3] and some rework is outlined in D6.2.2[4]. This section will provide a summary of the system architecture that will be the basis for the final evaluation. The architecture evolved based on the initial architecture and it's shortcomings that showed up in the first evaluation phase. The adoptions and new requirements are mainly caused by the extended broker design and are specified in the combined deliverable D2.3.2[5].

Components

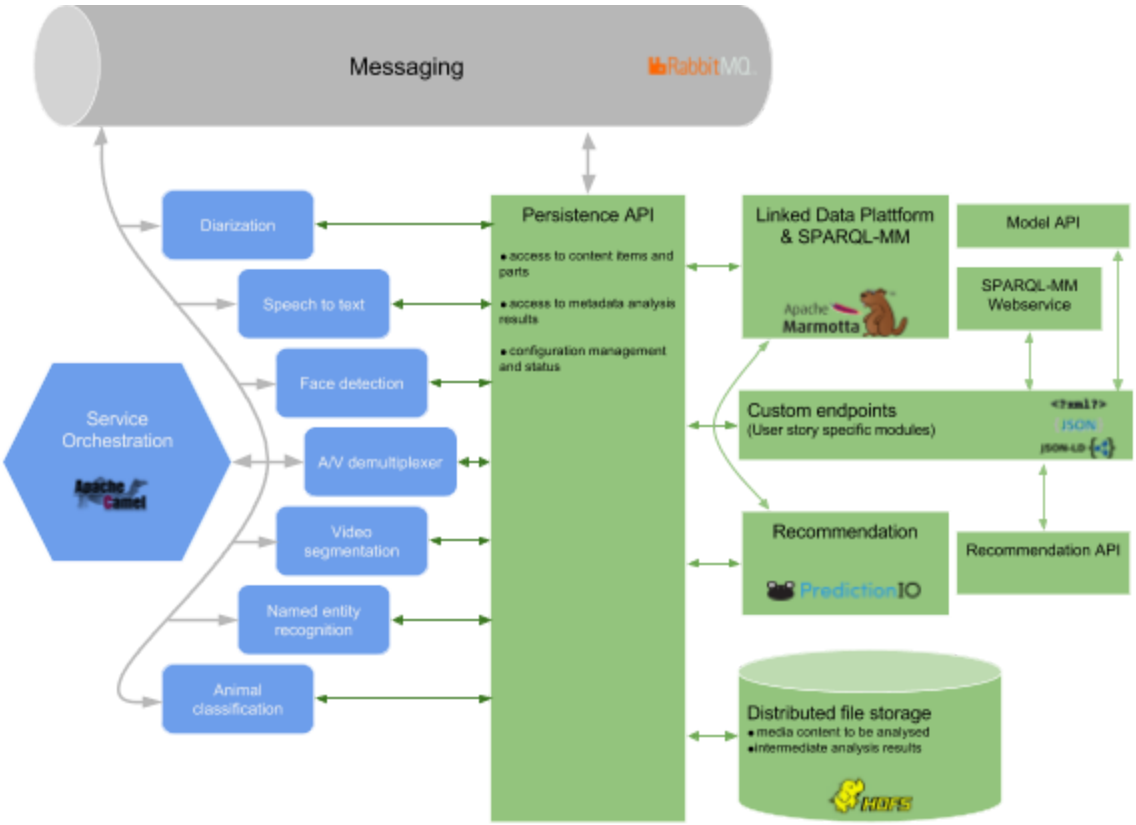


Figure 1:MICO architecture

[3] MICO project deliverable D6.2.1 Platform: Initial Version, Sebastian Schaffert and Sergio Fernández, 2014
[4] MICO project deliverable D6.2.2 Platform: Refined Version, Horst Stadler and Patrick Aichroth, 2015
[5] MICO project combined deliverable D2.3.2/D3.3.2/D4.3.2/D5.3.2 Enabling Technology Modules: Final Version – Interim Report, Patrick Aichroth, Johanna Björklund, Kai Schlegel, Thomas Kurz and Thomas Köllmer, 2016

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 is modeled as a workflow. 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 final version of the broker to overcome this and other drawbacks are specified in the combined deliverable D2.3.2.

- *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.
  The extractors and their abilities get registered on installation, as this information is needed to define new workflows.

- *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.

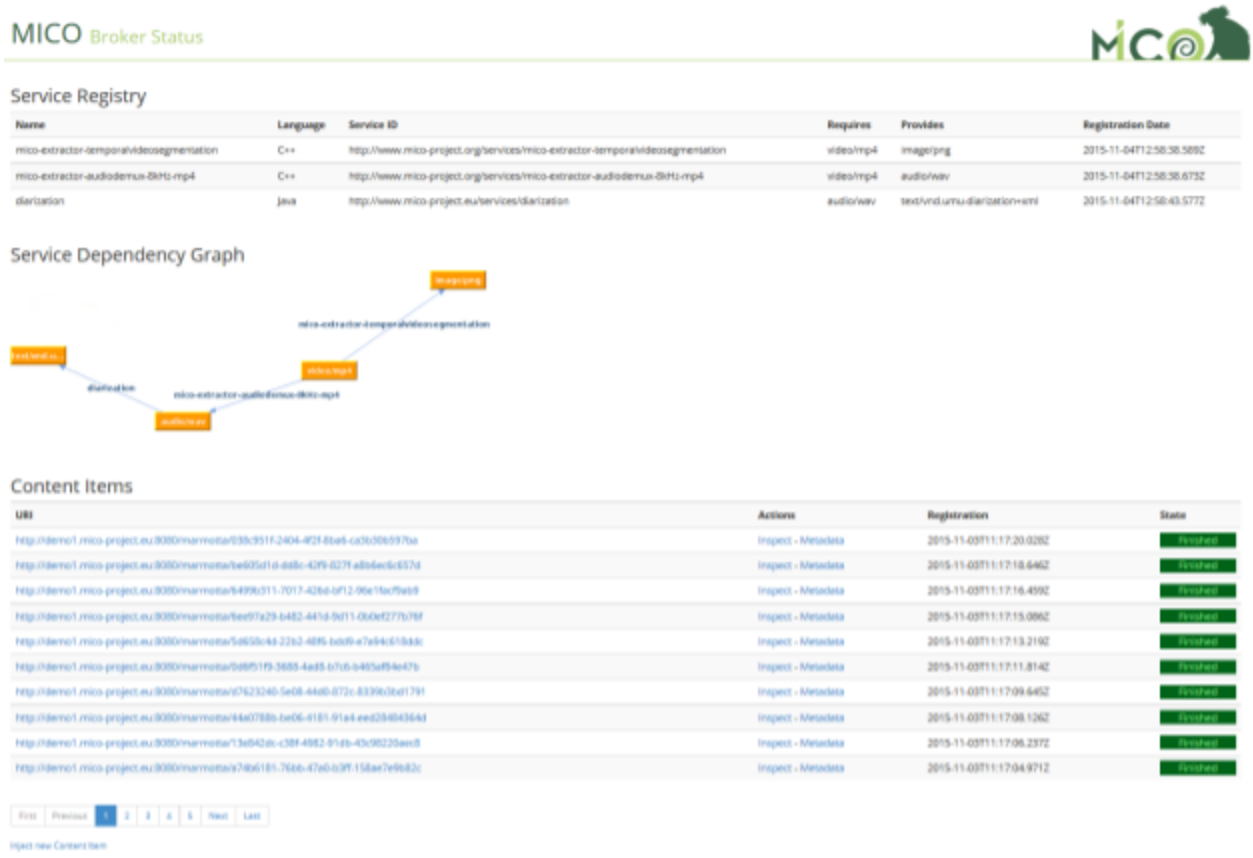


Figure 2: MICO broker user interface

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 start up the required extractors for their specific workflow, we set up a simple user interface:

Figure 3: Execution plan configuration

The first evaluation phase brought up some limitations that were addressed in the final platform release:

- Especially for analyses it is necessary to get feedback about the progress and current state. Therefore we enhanced the API so extractors can report their progress.
- The API got extended to enable reporting of problems or errors to feeder systems or the user.
- To inform feeder systems, like custom endpoints, if extractors are not running and therefore workflows can't be processed checks with proper results have been implemented.

Messaging

The messaging follows an event based approach. Therefore the *Advanced Message Queuing Protocol*[6] (AMQP) is used. As a platform and language independent middleware that implements AMQP, we rely on *RabbitMQ*[7] 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 red 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*[8] 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 or errors.
- *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

[6] http://www.amqp.org
[7] https://www.rabbitmq.com
[8] 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[9] is used as metadata storage. It allows to access to the metadata using LDPath[10] web service or SPARQL[11] . Moreover the SPARQL implementation is extended by SPARQL-MM[12] 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[13]. Marmotta also supports access (with some restrictions[14]) to store and fetch RDF data according the Linked Data Platform specification[15].
As backend for Apache Marmotta the KiWi Triplestore[16] is used, that in turn utilizes PostrgreSQL[17] as its relational database storage.

However, the platform is not restricted to this. Marmotta itself supports several backends like Sesame Native (which is now called RDF4J[18]), Big Data and Titan DB[19]. Moreover, by using a standard communication protocol (namely SPARQL), the possible replacement of the whole metadata storage layer by any other RDF backend is straightforward. This allows the usage of a broad range of backends, like Fuseki[20], OpenLink Virtuoso[21], etc. This list can also extended to graph databases like e.g. NEO4J[22] or Allegro Graph[23] by using an RDF implementation like provided by Apache Tinkerpop[24].

[9] http://marmotta.apache.org
[10] https://marmotta.apache.org/ldpath/language.html
[11] http://www.w3.org/TR/sparql11-overview/
[12] https://github.com/tkurz/sparql-mm
[13] http://www.w3.org/TR/media-frags/
[14] https://marmotta.apache.org/platform/ldp-module.html
[15] http://www.w3.org/TR/ldp/
[16] https://marmotta.apache.org/kiwi/triplestore.html
[17] http://www.postgresql.org/
[18] http://rdf4j.org/
[19] http://titan.thinkaurelius.com/
[20] https://jena.apache.org/documentation/serving_data/
[21] https://virtuoso.openlinksw.com/
[22] https://neo4j.com/
[23] http://franz.com/agraph/allegrograph/
[24] https://tinkerpop.apache.org/

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[25] (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[26] (FTP), as it was chosen as the first storage type to go with. Additionally we support simple local filesystems for development purposes.
Platform components, like the broker and the extractors, that have to deal with binary data are compiled to support all three types of file storage backend. The selection of a specific backend used by a MICO platform instance is made in one single place, in the configuration file of broker webapp.

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[27] is used.
For cross media recommendation, i.e., combining annotations from different extractors to get similar items, a custom api endpoint was created[28].

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.

[25] https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Overview
[26] https://tools.ietf.org/html/rfc959
[27] https://prediction.io
[28] http://mico-project.bitbucket.org/api/rest/

- *(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 implemented an endpoint for our use case partners that can be used as a template for custom endpoints. This is important especially regarding the format of the results and the data format an endpoint provides.

The 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:

Action:	Create and upload new content part	
Path:	/broker/inject/create	
Request method:	POST	
Parameters:	`type`	Syntactical type
	`name`	Name of the content type
Payload:	Data to store as the content part	
Responses	200	*application/json*

		`{`
		`"itemUri":` URI of the created content item as String
		`"Created"`: Current date and time
		`"syntacticalType"` : Type from the input parameter
		`"assetLocation"`:
		`{`
		`"namespace"`: Namespace of the asset
		`"localName"`: Unique identifier
		`}`
		`}`

	500	Error

Second step:

Action:	Trigger the analysis process	
Path:	/broker/inject/submit	
Request method:	POST	
Parameters:	`item`	URI of item
	`route`	Workflow ID (if not set all extractors matching the syntactical type are run; behaviour of platform version < 3)
Responses	200	OK
	500	Error
	503	Error (Workflow not available)

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:

Action:	Analyze JPEG image by providing the image	
Path:	/showcase-webapp/zooniverse/animaldetection/	
Request method:	POST	
Parameters	`mode`	Animal detector mode (`dpm` or `yolo`)
Payload:	The image to analyze.	
Responses	200	*application/json*

		`{`
		`"id":` ID of the created content part as String
		`"status": "submitted"`
		`"mode":` Animal detection mode

		`}`
	500	Error

or providing the URL of an image:

Action:	Analyze JPEG image by providing the URL of the image	
Path:	/showcase-webapp/zooniverse/animaldetection/	
Request method:	PUT	
Parameters:	`url`	HTTP URL to fetch the image from
	`mode`	Animal detector mode (`dpm` or `yolo`)
Responses	200	*application/json*

		`{`
		`"id":` ID of the created item as String
		`"status": "submitted"`
		`"mode":` Animal detection mode
		`}`

	500	Error

The result can be fetched by:

Action:	Fetch result of the animal detection	
Path:	/showcase-webapp/zooniverse/animaldetection/<item ID>	
Request method:	GET	
Responses	200	*application/json*

		`{`
		`"status": "inProgress"` or `"finished"`
		`"processingBegin":` Start of processing as DateTime
		
		The following fields are only available if `status` is `finished`:
		`"objectsFound":` Total number of identified animals as Integer
		`"objects": [` Array of identified animals.
		`{`
		`"algorithmVersion":` Algorithm identifier as String
		`"confidence":` Identification confidence level as Float
		`"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 }, …] }
	404	A content item 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

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

The result can be fetched with

Action:	Fetch result of the comment analysis	
Path:	/showcase-webapp/zooniverse/extanalysis/<content item ID>	
Request method:	GET	
Responses	200	*application/json*
		{ "id": 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 {

		"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
		}, …]
		}

	404	A content item 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.	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, 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[29]	no[30]	partly [31]	partly [32]	partly [33]	yes	no[34]	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

[29] The enterprise version of RiakCS can be used as a back-end, allowing multi-datacenter replication. See RiakCS for restrictions.
[30] 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.
[31] only for replicas
[32] It does not have an ability to run its database or trackers in multiple locations.
[33] 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.
[34] 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 [35]	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[36]	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[37]	Apache License 2.0	Apache License 2.0	New BSD
NFR-3	RHEL/ Fedora, SLES/ OpenSuse, Debian/Ubuntu	RHEL/ Fedora/CentOS, Debian/Ubuntu	Java	RHEL/ CentOS, Ubuntu	RHEL/ Fedora/CentOS/Pidora, SLES/ OpenSuse, 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, SmartOS	RHEL/ Fedora/CentOS, SLES/ OpenSuse, 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	libxtreemfs (Java, C++), FUSE

The installation, configuration and maintenance is very time-intense, as any distributed storage system is very complex and therefore it is necessary to deal more intense with it to get and keep

[35] see: http://lists.ceph.com/pipermail/ceph-users-ceph.com/2014-January/007540.html
[36] The last update was some time ago. (Even it might be used in practice, it seems to be poorly supported.)
[37] JAVA and C++ drivers (clients) are released under Apache license 2.0

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 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. In the final platform version the broker added support for workflows that enable a user to model the sequence of extractors data should be analysed. The workflow is stored at the platform and a workflow ID is provided with each data that should be analysed. To be backwards compatible a workflow gets created on demand using the syntactical type of the input data and extractors if no workflow ID is given.
One drawback is that extractors required by an workflow won't get started automatically if they are not running. For the built-in workflows a web interface is available to start the required extractors. For new workflows the required extractors need to be started manually or a startup configuration, like the ones for the built-in workflows, needs to be created.

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/pages/documentation/.
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[38]. 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[39]: Needed for communicating with the RabbitMQ master.
- libhdfs3[40]: 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[41]: 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):

[38] https://bitbucket.org/mico-project/platform/#markdown-header-prerequisites
[39] https://github.com/CopernicaMarketingSoftware/AMQP-CPP
[40] https://github.com/PivotalRD/libhdfs3
[41] http://kaldi-asr.org

Package name	Description
libmico-api1	This package contains the compiled shared libraries of the MICO C++ API platform version 1. They are needed by all binary implementations of MICO C++ services.
libmico-api2	This package contains the compiled shared libraries of the MICO C++ API platform version 2. They are needed by all binary implementations of MICO C++ services.
libmico-api3	This package contains the compiled shared libraries of the MICO C++ API platform version 3. 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-default-camel-routes	Built-in workflows.
mico-registration-service	The web service that enables the registration of extractors on installation is provided with this package.
mico-conf	This package contains the distribution of the MICO Platform 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-extractor-registration	Tools to register an extractor at the MICO platform.
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-audiodemux-registration	Registration data for mico-extractor-audiodemux
mico-extractor-ccv-facedetection	MICO service for detection of faces in videos.
mico-extractor-ccv-facedetection-registration	Registration data for mico-extractor-ccv-facedetection
mico-extractor-diarization	This package contains the Speaker Diarization extractor daemon, responsible of providing support for TE-214 in MICO.
mico-extractor-diarization-registration	Registration data for mico-extractor-diarization
mico-extractor-speech-to-text	MICO service for automatic transcription, based on Kaldi.
mico-extractor-speech-to-text-registration	Registration data for mico-extractor-speech-to-text
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-kaldi2rdf-registration	Registration data for mico-extractor-kaldi2rdf
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-kaldi2txt-registration	Registration data for mico-extractor-kaldi2txt

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-named-entity-recognizer-registration	Registration data for mico-extractor-named-entity-recognizer
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-object-detection-rdf-registration	Registration data for mico-extractor-object-detection-rdf
mico-extractor-temporalvideosegmentation	MICO service for detection of shot boundaries and keyframes in videos.
mico-extractor-temporalvideosegmentation-registration	Registration data for mico-extractor-temporalvideosegmentation
mico-extractor-animaldetection-dpm	DPM animal detection.
mico-extractor-animaldetection-dpm-registration	Registration data for mico-extractor-animaldetection-dpm
mico-extractor-animaldetection-dpm	Yolo animal detection.
mico-extractor-animaldetection-yolo-registration	Registration data for mico-extractor-animaldetection-yolo
mico-extractor-opennlp-ner	OpenNLP NER extractor
mico-extractor-opennlp-ner-registration	Registration data for mico-extractor-opennlp-ner
mico-extractor-opennlp-ner-models-model-de	German model for mico-extractor-opennlp-ner
mico-extractor-opennlp-ner-models-model-en	English model for mico-extractor-opennlp-ner
mico-extractor-opennlp-ner-models-model-es	Spanish model for mico-extractor-opennlp-ner

mico-extractor-opennlp-ner-models-model-it	Italian model for mico-extractor-opennlp-ner
mico-extractor-opennlp-text-classifier	OpenNLP text classifier extractor.
mico-extractor-opennlp-text-classifier-registration	Registration data for mico-extractor-opennlp-text-classifier
mico-extractor-competence-classification-model-en	English competence classification model for mico-extractor-opennlp-text-classifier
mico-extractor-text-lang-detect	Language detection extractor.
mico-extractor-text-lang-detect-registration	Registration data for mico-extractor-text-lang-detect
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.
mico-kaldi-lib	Kaldi is a toolkit for speech recognition written in C++. It is intended for use by speech recognition researchers. MICO build of OSS Kaldi speech to text library.
mico-kaldi-lib-dev	MICO headers of OSS Kaldi speech to text library.
mico-kaldi-data	Contains packaged versions of the Kaldi data files needed by MICO.

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[42].

[42] https://www.virtualbox.org

Docker

The core components of the MICO platform are available as Docker[43] image too (see https://bitbucket.org/mico-project/platform/src//docker/?at=master for the Dockerfiles). It turned out that the way the components are bundled as Debian packages are not suitable for building proper Docker images. Therefore it would be necessary to setup and maintain a parallel distribution and deployment infrastructure beside the existing packages. As it is already we decided to This would lead to a considerable additional amount of time, so we decided to not pursue this further.

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/pages/documentation/

Continuous Integration

Bamboo[44] 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). Due to the architecture of the system we decided to not set up automatic full integration tests, but instead enable automatic tests on the component level.
The dashboard is available at http://ci.mico-project.eu.

[43] https://www.docker.com
[44] https://www.atlassian.com/software/bamboo

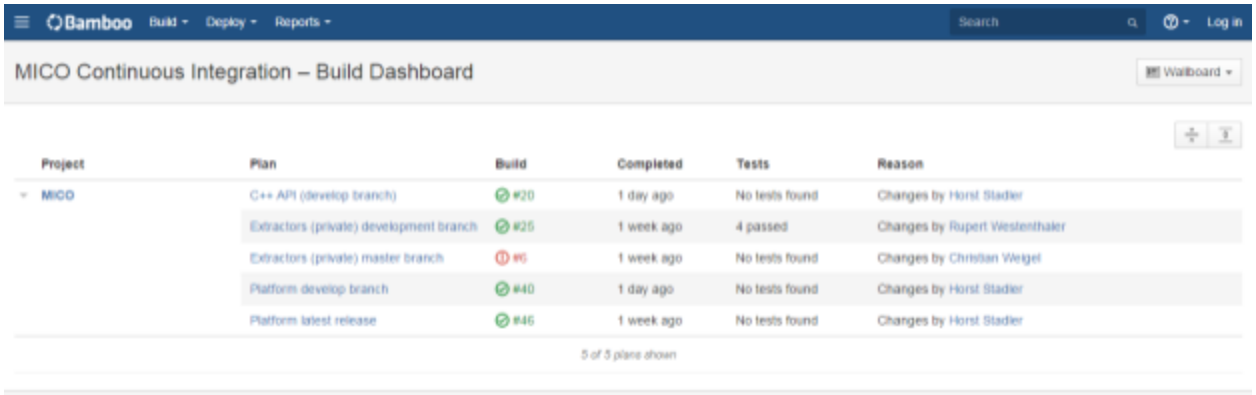


Figure 4: Continous Integration Dashboard

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.

We took a look at the very powerful centralized logging stack based on Logstash[45], Elasticsearch[46] and Kibana[47]. This allows 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.
The downside is that it is bloated and time consuming to set up and maintain. Therefore it does not make sense to deliver it with the MICO platform by default. We also had a look at the very lightwight log.io[48], which is quite easy to setup but it does not help to investigate on problems afterwards, as it only shows current logs and.

[45] https://www.elastic.co/products/logstash
[46] https://www.elastic.co/products/elasticsearch
[47] https://www.elastic.co/products/kibana
[48] http://logio.org

Platform dependencies

The following table lists the first level java dependencies of the MICO platform with their license.

Nr	Group	artifact	type	version	scope	LICENSE
1	cglib	cglib	jar	3.1	compile	Apache 2
2	ch.qos.logback	logback-classic	jar	1.1.2	compile	MIT license (and EPL v1.0)
3	ch.qos.logback	logback-classic	jar	1.1.3	compile	MIT license (and EPL v1.0)
4	ch.qos.logback	logback-core	jar	1.1.2	compile	MIT license (and EPL v1.0)
5	ch.qos.logback	logback-core	jar	1.1.3	compile	MIT license (and EPL v1.0)
6	com.beust	jcommander	jar	1.48	compile	Apache 2
7	com.github.anno4j	anno4j-core	jar	2.2.0.MICO	compile	Apache 2
8	com.github.anno4j	anno4j-ontologies-mmm	jar	2.0.1	compile	Apache 2
9	com.github.tkurz	sparql-mm	jar	1.0	compile	Apache 2
10	com.github.zafarkhaja	java-semver	jar	0.9.0	compile	MIT License
11	com.google.guava	guava	jar	18.0	compile	Apache 2
12	com.google.protobuf	protobuf-java	jar	2.6.0	compile	Copyright 2014, Google Inc
13	com.h2database	h2	jar	1.3.174	compile	EPL 1.0
14	com.jayway.restassured	rest-assured	jar	2.5.0	test	Apache 2
15	com.rabbitmq	amqp-client	jar	3.3.4	compile	Apache 2
16	com.sun.jersey	jersey-client	jar	1.9	compile	GPL2 (/w Classpath Exception)

17	com.thetransactionco mpany	cors-filter	jar	1.9	compile	Apache 2
18	commons-collections	commons-collections	jar	3.2.1	runtime	Apache 2
19	commons-configurati on	commons-configurat ion	jar	1.10	runtime	Apache 2
20	commons-io	commons-io	jar	2.4	compile	Apache 2
21	commons-logging	commons-logging	jar	1.1.3	compile	Apache 2
22	commons-net	commons-net	jar	3.2	compile	Apache 2
23	commons-validator	commons-validator	jar	1.4.1	compile	Apache 2
24	eu.mico.platform.cam el	mico-rabbitmq-comp	jar	3.1.0	compile	Apache 2
25	eu.mico-project.platfo rm	event	jar	3.1.0	compile	Apache 2
26	eu.mico-project.platfo rm	fam-anno4j	jar	2.0.2	compile	Apache 2
27	eu.mico-project.platfo rm	marmotta-mico	jar	3.1.0	compile	Apache 2
28	eu.mico-project.platfo rm	mmm-anno4j	jar	3.1.0	compile	Apache 2
29	eu.mico-project.platfo rm	persistence	jar	3.1.0	compile	Apache 2
30	eu.mico-project.platfo rm	storage-core	jar	3.1.0	compile	Apache 2
31	io.prediction	client	jar	0.9.5	compile	Apache 2
32	javax.el	javax.el-api	jar	2.2.4	test	GPLv2 with classpath exception
33	javax.json	javax.json-api	jar	1.0	compile	CDDL 1.1 or GPL v2
34	javax.servlet	javax.servlet-api	jar	3.0.1	provided	CDDL 1.1 or GPL v3
35	javax.servlet	jstl	jar	1.2	compile	

36	jgraph	jgraph	jar	5.13.0.0	compile	LGPL) version 2.1 and JGraph License version 1.1
37	kaldi	kaldi	jar	2.2.0	compile	Apache 2
38	langdetect	langdetect	jar	1.2.0	compile	Apache 2
39	lium	lium	jar	1.3.0	compile	GPL v3
40	org.apache.camel	apt	jar	2.17.2	compile	Apache 2
41	org.apache.camel	camel-core	jar	2.17.2	compile	Apache 2
42	org.apache.camel	camel-servlet	jar	2.17.2	compile	Apache 2
43	org.apache.camel	camel-test	jar	2.17.2	test	Apache 2
44	org.apache.commons	commons-email	jar	1.4	compile	Apache 2
45	org.apache.commons	commons-lang3	jar	3.1	compile	Apache 2
46	org.apache.commons	commons-vfs2	jar	2.0	compile	Apache 2
47	org.apache.httpcomponents	httpclient	jar	4.5	compile	Apache 2
48	org.apache.marmotta	kiwi-caching-hazelcast	jar	3.4.0-SNAPSHOT	compile	Apache 2
49	org.apache.marmotta	ldpath-core	jar	3.4.0-SNAPSHOT	compile	Apache 2
50	org.apache.marmotta	marmotta-backend-kiwi	jar	3.4.0-SNAPSHOT	compile	Apache 2
51	org.apache.marmotta	marmotta-core	tests	3.4.0-SNAPSHOT	test	Apache 2
52	org.apache.marmotta	marmotta-core	jar	3.4.0-SNAPSHOT	compile	Apache 2
53	org.apache.marmotta	marmotta-ldcache-common	jar	3.4.0-SNAPSHOT	compile	Apache 2
54	org.apache.marmotta	marmotta-ldcache-file	jar	3.4.0-SNAPSHOT	compile	Apache 2
55	org.apache.marmotta	marmotta-ldpath	jar	3.4.0-SNAPS	compile	Apache 2

				HOT		
56	org.apache.marmotta	marmotta-sparql	jar	3.4.0-SNAPSHOT	test	Apache 2
57	org.apache.marmotta	marmotta-sparql	jar	3.4.0-SNAPSHOT	compile	Apache 2
58	org.apache.marmotta	marmotta-versioning-common	jar	3.4.0-SNAPSHOT	compile	Apache 2
59	org.apache.maven	maven-core	jar	3.0.4	compile	Apache 2
60	org.apache.maven	maven-plugin-api	jar	3.0.4	compile	Apache 2
61	org.apache.maven.plugin-tools	maven-plugin-annotations	jar	3.4	compile	Apache 2
62	org.apache.opennlp	NER	jar	1.2.0	compile	Apache 2
63	org.apache.opennlp	Competence	jar	1.2.0	compile	Apache 2
64	org.apache.opennlp	Sentiment	jar	1.2.0	compile	Apache 2
65	org.apache.tika	tika-core	jar	1.4	compile	Apache 2
66	org.codehaus.jackson	jackson-core-asl	jar	1.9.13	compile	Apache 2
67	org.codehaus.jackson	jackson-mapper-asl	jar	1.9.13	compile	Apache 2
68	org.glassfish	javax.json	jar	1.0	compile	GPLv2 with classpath exception
69	org.hamcrest	hamcrest-core	jar	1.3	test	BSD License
70	org.hamcrest	hamcrest-core	jar	1.3	test	BSD License
71	org.hamcrest	hamcrest-core	jar	1.3	compile	BSD License
72	org.hamcrest	hamcrest-library	jar	1.3	test	BSD License
73	org.jboss.resteasy	resteasy-client	jar	3.0.8.Final	compile	Apache 2
74	org.jboss.resteasy	resteasy-jackson-provider	jar	3.0.8.Final	compile	Apache 2
75	org.jboss.resteasy	resteasy-jaxrs	jar	3.0.8.Final	compile	Apache 2

76	org.jboss.resteasy	resteasy-multipart-provider	jar	3.0.8.Final	compile	Apache 2
77	org.jboss.resteasy	resteasy-servlet-initializer	jar	3.0.8.Final	compile	Apache 2
78	org.jboss.weld.se	weld-se-core	jar	2.1.2.Final	test	Apache 2
79	org.jboss.weld.servlet	weld-servlet-core	jar	2.1.2.Final	runtime	Apache 2
80	org.jgrapht	jgrapht-core	jar	0.9.0	compile	EPL 1.0 LGPL 2.1
81	org.jgrapht	jgrapht-ext	jar	0.9.0	compile	
82	org.jsondoc	jsondoc-core	jar	1.2.16	compile	MIT
83	org.jsondoc	jsondoc-ui	jar	1.2.16	compile	MIT
84	org.mockito	mockito-all	jar	1.10.19	test	MIT
85	org.openrdf.sesame	sesame-model	jar	2.7.16	compile	Aduna BSD license
86	org.openrdf.sesame	sesame-query	jar	2.7.16	compile	Aduna BSD license
87	org.openrdf.sesame	sesame-repository-api	jar	2.7.16	compile	Aduna BSD license
88	org.openrdf.sesame	sesame-repository-contextaware	jar	2.7.16	compile	Aduna BSD license
89	org.openrdf.sesame	sesame-repository-sail	jar	2.7.16	test	Aduna BSD license
90	org.openrdf.sesame	sesame-repository-sparql	jar	2.7.16	compile	Aduna BSD license
91	org.openrdf.sesame	sesame-rio-api	jar	2.7.16	compile	Aduna BSD license
92	org.openrdf.sesame	sesame-rio-rdfxml	jar	2.7.16	test	Aduna BSD license
93	org.openrdf.sesame	sesame-rio-turtle	jar	2.7.16	test	Aduna BSD license
94	org.openrdf.sesame	sesame-sail-memory	jar	2.7.16	test	Aduna BSD license

95	org.slf4j	log4j-over-slf4j	jar	1.7.7	compile	MIT License
96	org.slf4j	slf4j-api	jar	1.7.7	compile	MIT License
97	org.slf4j	slf4j-ext	jar	1.7.7	compile	MIT License
98	org.slf4j	slf4j-log4j12	jar	1.7.7	compile	MIT License
99	org.springframework	spring-aop	jar	4.1.1.RELEASE	compile	Apache 2
100	org.springframework	spring-context	jar	4.1.1.RELEASE	compile	Apache 2
101	org.springframework	spring-web	jar	4.1.1.RELEASE	compile	Apache 2
102	org.springframework	spring-webmvc	jar	4.1.1.RELEASE	compile	Apache 2
103	org.webjars	angular-file-upload	jar	1.6.5	compile	MIT
104	org.webjars	angularjs	jar	1.2.15	compile	MIT
105	org.webjars	angular-ui-bootstrap	jar	0.12.1	compile	MIT
106	org.webjars	bootstrap	jar	3.1.1	compile	MIT

MICO Platform Online Showcase

In order to demonstrate the MICO platform functionality within a web environment we created a generic MICO platform demonstrator, which can be found on http://demo3.mico-project.eu/.

It is purely Platform based, which means it only uses the web services that have been described above. Additionally it combines several frontend software projects that have been developed and/or adapted within the project, which are Squebi (for SPARQL result presentation), the workflow-manager (for the creation and the selection of workflows), and the balloon synopsis (for extraction result presentation). Thus the demo supports:

- Workflow creation
- Workflow selection
- Data ingest (text, image, video, audio)
- Type-specific result presentation
- Generic result presentation
- SPARQL access

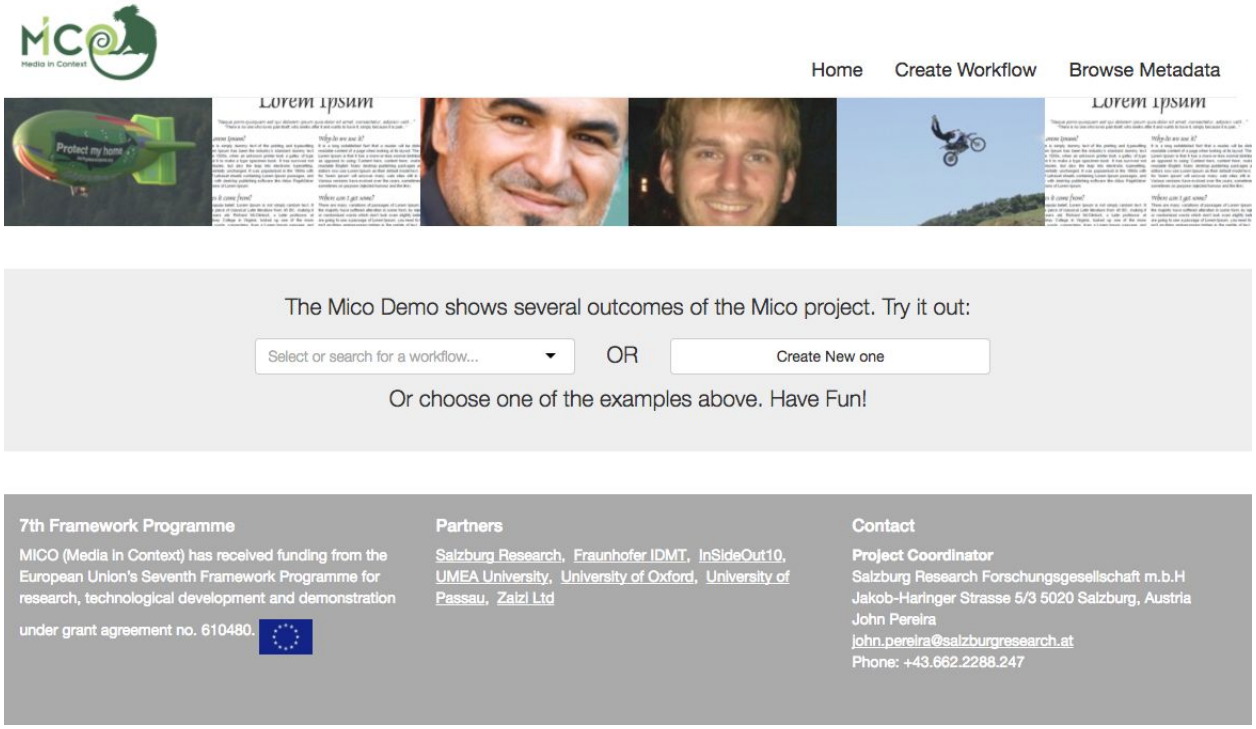
The demo backbone is based in the yeoman angular generator[49] which determines the basic project structure. The demo uses angularjs, an open source javascript application framework that allows to build well structured web applications by following the MVVM (Model–view–viewmodel) design pattern. Additionally we use SASS[50] for structured CSS generation. In order to support a fast development and a standardized deployment, we use grunt[51] (a Javscript task runner) with several plugins. The whole project is public abvailable at https://bitbucket.org/mico-project/mico-public-demo, where the reader can find more information about building and installation.

In this demo you can create/select and test analysis pipelines You can select sample media assets or upload custom media files including image, video, audio and text. When analysis is done, you can watch the results on type specific result pages or get deeper into the generated metadata via using generic annotation views and/or issue SPARQL queries. The demo is purely frontend based and uses the standard MICO platform services. It is designed to demonstrate MICO technology in web-friendly way. Hint: The demo is a (as the name says) a demonstrator and thus not exhaustively tested with the big set of all existing browsers. It works best with Chrome browser.

[49] https://github.com/yeoman/generator-angular
[50] http://sass-lang.com/
[51] http://gruntjs.com/

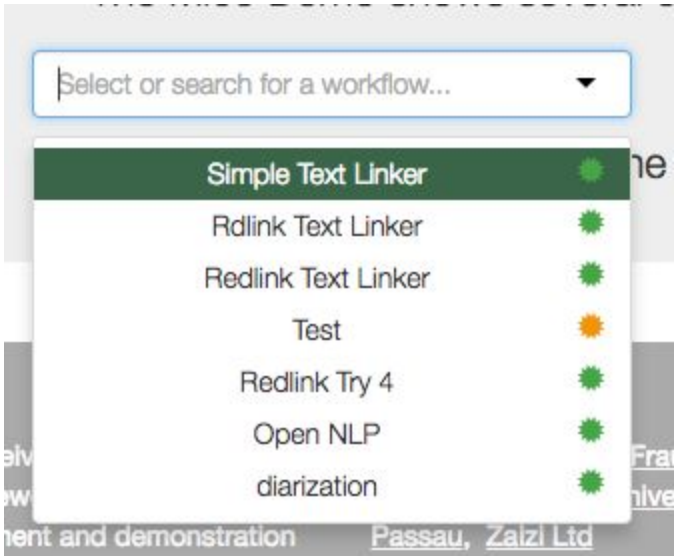
Main Page

The central page of the demo includes links to main demo parts on the top. A list of various prepared demo assets (including images, video, text and audio samples) give users an easy entrypoint without the hurdle of a private asset upload. A click on the demo thumbnails redirects to the regarding result presentation immediately. In the center of the page the users can select from various predefined workflows. Additionally they can create a new one (which leads them to the workflow management section).



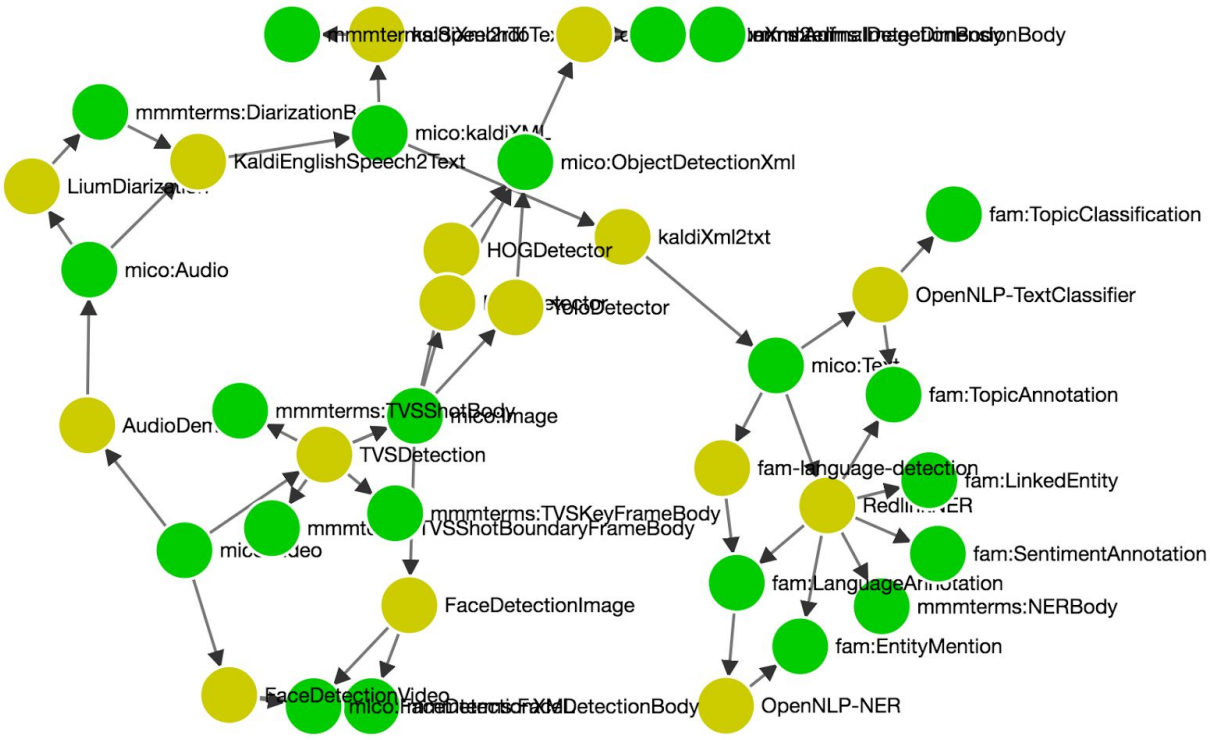
Workflow Selection

The selection of a workflow defines, which data can be ingested by the user. In the public demo, several workflows including image, video, text and audio processing are predefined. The current status of workflow is indicated by a color schema, whereby green denotes *okay*, orange denotes *temporarily unavailable*, and red means *broken*. The workflow status is depended on status of the applied extractor components.



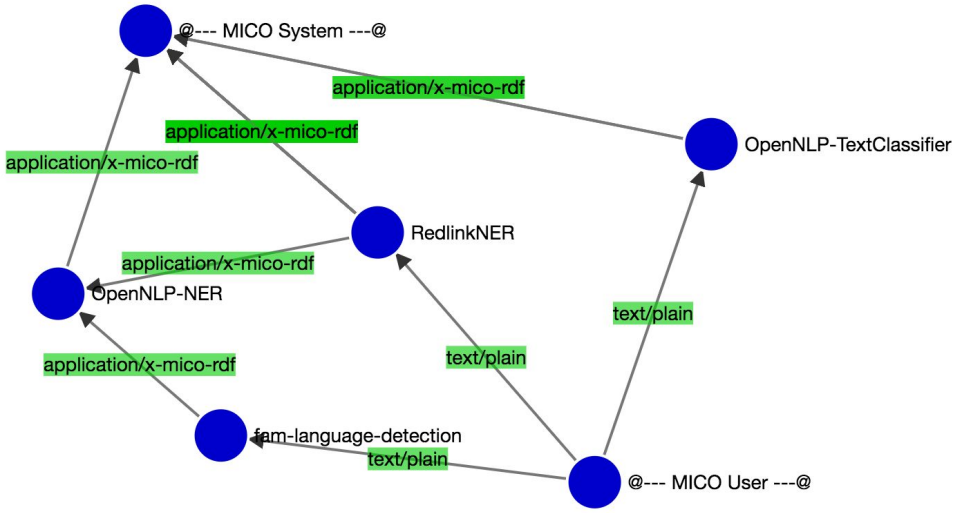
Workflow Creation

The demo allows the maintenance of existing and the creation of new workflows. Thereby the demo integrates the Workflow Management App, which is build on top of a Spring Application that provides Restful a set of task specific web services. The UI gives users an overview of all extractors and its interplay by a dependency graph visualization.



The workflow creation is enabled by a straightforward extractor selection process (by selecting and unselecting extractor components). Analogous to the extractor overview, the created

workflows are visualized as a graph, which allows a) a what you see is what you get (WYSIWYG) experience during the creation process and b) a immediate overview on input and output values and assets.

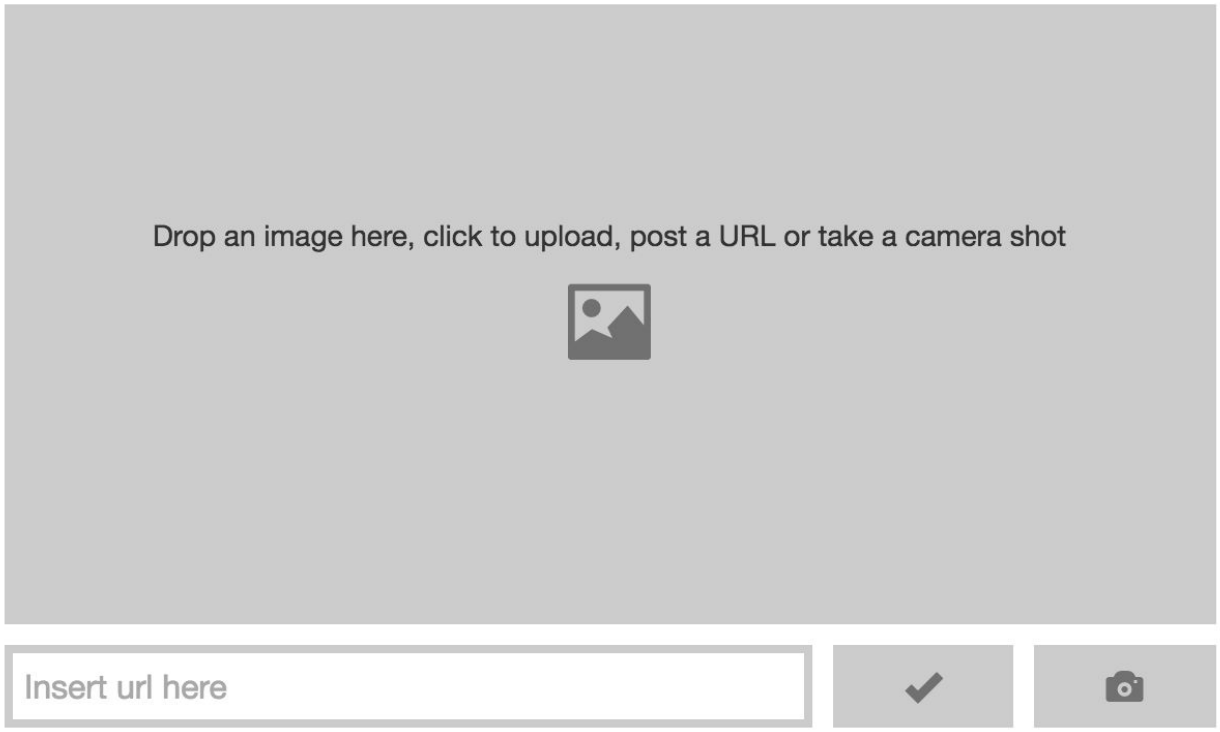


Data Ingest

The ingestion UI is aligned with the selected workflow. Thus, there currently exist 5 different ingest UIs:

- The Image Ingest allows to push images to the system by providing an URL, by drag and drop desktop images, by common file system selection and by using an onboard webcam to take a picture.
- The Text Ingest allows drag and drop, file select and input by using a textarea.
- The Video/Audio Ingest provides drag and drop and file select. To overcome the issue that audio and video processing may take a while (depending on asset size) the UI also enables process notification by email.
- As MICO workflows support multi asset as well as multi content type processing a Mixed Ingest UI allows the upload of multiple files within one step

In order to get an impression of the ingest UI we added a screenshot of the of the Image Ingest Interface.



Result Display

MICO produces a bunch of metadata and assets during workflow execution. To enable users, which are not yet familiar with RDF and MICO specific ontologies to get an overview on the results that have been created, the demo supports the display of several result types in a well known manner.

Such displays are for example:

- Image Media Fragments are displayed as rectangular shapes
- Video Fragments are displayed via keyframes
- Recognized Entities in text are displayed by as colorization and popups
- Video subtitles are displayed under the regarding video and are aligned to the current video timestamp
- Recognized Entities are displayed together with metadata within type specific information boxes

The list of type specific displays is not fixed and may be adapted and extended with new extractor types.

SPARQL Endpoint

SPARQL is a powerful query language with a syntax that is aligned to common query languages like SQL. Nevertheless, SPARQL query building can be a complex task because the language is not yet widely used and deals a lot with URLs, which can be confusing to new users. In order to lower the barrier for SPARQL query writing we developed Squebi[52], a clean SPARQL UI which supports:

- customization of SPARQL result visualization
- support for SPARQL 1.1 (update and select)
- bookmarkable uris that define queries and the visualization type
- support for SPARQL query editing
    - Auto-creation of URIs prefixes
    - Autocompletion for well known ontologies

The demo integrated Squebi and provides the user additionally with many generic as well as workflow specific examples.

[52] https://github.com/tkurz/squebi

Get examples!

```
SELECT * WHERE {
  ?subject ?property ?object
} LIMIT 20
```

Run

Table JSON XML

Rows 1 to 10 of overall 20

subject	property	object
http://demo3.mico-project.eu:8080/marmotta	http://www.w3.org/ns/ldp#contains	http://demo3.mico-project.eu:8080/marmotta/3dd2da82-7e84-4e86-8f9b-c29cfa0167d0

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[53]. The resulting components allow arbitrary signature and verification of HTML documents via XPATH. The can also be used to sign and verify copyright and provenance information embedded within the HTML, and provide the following *MicroformatSignature* class API:

MicroformatSignature class	
Constructor: Builds a new object for creating and verifying microformat signatures for HTML/XML documents.	
`public MicroformatSignature(`	
`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
`)`	
`buildFor:` Creates a new microformat signature node for the given HTML/XML node.	
`public Node buildFor(`	
`Node node`	HTML/XML node to sign (can also be a Document object)
`)`	
`@return Node`	Signature node created

[53] http://microformats.org

`checkFor:` Verifies the embedded microformat signature for a given HTML/XML node.	

`public CheckResult checkFor(`
 	
`Node node`
 	
`)`
	HTML/XML node to check (can also be a Document object)

`@return CheckResult`	Enum: ● `signatureValid` ● `signatureInvalid` ● `signatureNotFound`
`checkAllDocumentSignatures` : Verifies all microformat signatures found in the given document.	

`public Map<String, CheckResult>`
`checkAllDocumentSignatures(`
 	
`Document doc`
 	
`)`
	Document to examine

`@return Map<String, CheckResult>`	Map containing for each signed node: ● XPath expression and ● CheckResult value Enum: ○ `signatureValid` ○ `signatureInvalid` ○ `signatureNotFound`

These components have been developed and integrated into the platform in year 3, and can 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).

The following code shows an example for a HTML page containing a signed license part (bold). The *Microformat Extractor* component extracts the license information and validates the signature in order to verify the authenticity of the information.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head profile='http://microformats.org/profile/rel-license'>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Signature Example</title></head>
<body><div class="hnews hentry item"><div class="entry-content"><p>
```

Ilmenau, Germany
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam

<!-- Rel-License is one of several microformats. By adding rel="license" to a hyperlink,
a page indicates that the destination of that hyperlink is a license for the current page.
E.g. with the following hyperlink the author indicates that the page is licensed
under a Creative Commons 2.0 Attribution Required license.-->

 License information: cc by 2.0 <abbr class="signaturemethod" title="RSA"></abbr><abbr class="digestmethod" title="SHA256"></abbr><abbr class="manifest" title="example"></abbr><abbr class="digestvalue" title="LnG72Gz9SoxLQCOGOO+gWAs3KuA="></abbr><abbr class="signaturevalue" title="bFP72p9Be [...] HfAKw=="></abbr><abbr class="keyinfo" title="X.509"></abbr><abbr class="keyvalue" title="MIICtz [...] xQrb"></abbr> </body></html>