Specifications and Models for Cross-Media Extraction, Metadata Publishing, Querying and Recommendations: Final Version

WP2: Christian Weigel, Patrick Aichroth, Marcel Sieland, Luca Cuccovillo, Johanna Björklund
WP3: Emanuel Berndl, Kai Schlegel, Andreas Eisenkolb
WP4: Thomas Kurz
WP5: Thomas Köllmer, Patrick Aichroth, Alex Bowyer

christian.weigel@idmt.fraunhofer.de, patrick.aichroth@idmt.fraunhofer.de,
marcel.sieland@idmt.fraunhofer.de, luca.cuccovillo@idmt.fraunhofer.de, johanna@cs.umu.se,
berndl@dimis.fim.uni-passau.de, schlegel@dimis.fim.uni-passau.de,
eisenkolb@dimis.fim.uni-passau.de, thomas.kurz@salzburgresearch.at,
thomas.koellmer@idmt.fraunhofer.de, alex.bowyer@zooniverse.org

Abstract
This deliverable summarizes the finale specifications in cross-media analysis, metadata publishing, querying and recommendations. It is a joint outcome of work packages WP2, WP3, WP4 and WP5, and describes the final plans MICO technologies as well as their combined operation.

Keyword List
Specification, Cross-Media Analysis, Metadata Publishing, Querying, Recommendations

Contents

1 Executive Summary **2**

2 Specifications and Models for Cross-Media Extraction & Orchestration Components **3**
2.1 Extractor Overview . . . 3
2.2 Extractor Updates and New Extractor Specifications . . . 4
2.2.1 Object and Animal Detection – OAD (TE-202) – UPDATE . . . 4
2.2.2 Temporal Video Segmentation – TVS (TE-206) – UPDATE . . . 5
2.2.3 Audiovisual Quality – AVQ (TE-205) – UPDATE . . . 5
2.2.4 Face detection – FDR (TE-204) – UPDATE . . . 5
2.2.5 Speech-Music Discrimination – SMD (TE-207) – UPDATE . . . 6
2.2.6 Speech-to-Text (TE-214) – UPDATE . . . 6
2.2.7 Redlink Text Analysis Extractor (TE-213, TE-220) – NEW . . . 6
2.2.8 Diarization (TE-214) – NEW . . . 9
2.2.9 Audio Demux – DMX (TE-214) – NEW . . . 10
2.2.10 Nudity Detection – NDE (TE-226) – Year 3 . . . 11
2.2.11 Generic Feature Extraction – GFE (TE-201) – Year 3 . . . 12
2.2.12 Video Segment Matching – VSM (TE-211) – Year 3 . . . 13
2.2.13 Stanford NLP Extractor (TE-213, TE-220) - Year 3 . . . 14
2.2.14 OpenNLP Named Entity Recognition Extractor (TE-220) - Year 3 . . . 16
2.2.15 OpenNLP Sentiment Analysis (TE-213) – Year 3 . . . 18
2.3 Extractor Pipelines . . . 19
2.4 Extractor Pipelines Planned for Year 3 . . . 20
2.5 Multilingual support . . . 21
2.5.1 Speech-to-Text Multilingual support . . . 21
2.5.2 Named Entity Recognition Multilingual support . . . 22
2.5.3 Sentiment Analysis Multilingual support . . . 22
2.6 Broker Overview . . . 22
2.7 Broker v2 wishlist . . . 23
2.8 Broker updates after the platform release . . . 25
2.8.1 Pipeline configuration . . . 25
2.8.2 MICO broker v2 . . . 27
2.9 Broker v3 design . . . 27
2.9.1 General broker principles and assumptions . . . 28
2.9.2 Relevant User Stories . . . 29
2.9.3 Relevant Components . . . 30
2.9.4 High-level data model . . . 31
2.9.5 Use Cases . . . 36
2.9.6 Registration Service . . . 46
2.9.7 Workflow planning and creation . . . 52
2.9.8 Workflow execution . . . 52

3 Specifications and Models for Cross-media Publishing **54**
3.1 Introduction . . . 54
3.2 Recapitulation . . . 54
3.3 Major Changes . . . 55

3.4 Specification . . . 57
3.5 Read and write the MICO Metadata Model . . . 58
3.5.1 Anno4j Querying . . . 59
3.5.2 Anno4j Custom Extensions . . . 60

4 Specifications and Models for Cross-media Querying **61**
4.1 Retrospect . . . 62
4.2 Extended Specification . . . 63
4.2.1 Extension of Media Fragment URIs . . . 63
4.2.2 GeoSPARQL . . . 70
4.2.3 Extension of Spatio-Temporal Functions . . . 80
4.2.4 Cover Functions . . . 85
4.3 Outlook . . . 87

5 Specifications and Models for Cross-media Recommendations **88**
5.1 Introduction . . . 88
5.2 Key user stories . . . 89
5.2.1 Greenpeace Magazine . . . 89
5.2.2 Shoof . . . 90
5.2.3 Snapshot Serengeti . . . 91
5.3 Available datasets . . . 93
5.3.1 Greenpeace Magazine and Shoof . . . 93
5.3.2 Snapshot Serengeti . . . 94
5.4 General architecture and common specifications . . . 96
5.5 Showcase specific specifications . . . 97
5.5.1 InsideOut10: Greenpeace Magazine & Shoof . . . 98
5.5.2 Zooniverse: Snapshot Serengeti . . . 100
5.6 Recommendation API reference . . . 102
5.7 Deviations from D5.2.1: First specifications . . . 106
5.7.1 TE-501. User Activity and Context Monitor . . . 106
5.7.2 TE-502. User Similarity Calculator . . . 106
5.7.3 TE-503. Item Similarity Calculator . . . 107
5.7.4 TE-504. Cross-Modal Content Recommender . . . 107
5.8 Work planned for year 3 . . . 108
Appendix 5.A Appendix: Subject recommendation for Snapshot Serengeti . . . 109
5.A.1 Test that species preferences match user behaviour (TP-506-01) . . . 109
5.A.2 Test that the subjects recommended match the preferred species (TP-506-02)] . 111
Appendix 5.B MICO WP5 Platform Integration . . . 112
5.B.1 Install on platform . . . 112
5.B.2 Running prediction.io inside a docker container . . . 112
5.B.3 Testing . . . 112

List of Figures

1 Animal detection pipeline (animal-detection) . . . 19
2 Face detection pipeline (face-detection). . . . 19
3 IO Showcase pipelines . . . 20
4 Broker v1 Pipeline Configuration . . . 26
5 Data model of the MICO broker: Overview . . . 32
6 Data model of the MICO broker: Content description and extractor configuration domains 33
7 Data model of the MICO broker: I/O data description . . . 34
8 Data model of the MICO broker: Platform management . . . 36
9 Overview of the MICO Use Cases . . . 37
10 Broker registration XSD: Basic types . . . 47
11 Broker registration XSD: Extractor Specification overview . . . 48
12 Broker registration XSD: Input data definition . . . 49
13 Broker registration XSD: Output data definition . . . 50
14 Broker registration XSD: User-configurable parameter definition . . . 51
15 Data model of the MICO broker: Platform management . . . 53
16 Basic annotation in the MICO data model . . . 54
17 Face Recognition example . . . 55
18 MICO Metedata Model Version 2.0 . . . 56
19 Exemplary Extractor Provenance . . . 58
20 Example: Rectangle . . . 64
21 Example: Circle . . . 65
22 Example: Ellipse . . . 66
23 Example: Polygon . . . 66
24 Example: Rotate ellipse . . . 68
25 Example: Translate . . . 70
26 Sending user event data and asynchronous triggering of Prediction.io . . . 99
27 Recommendation using both collaborative filtering and media analysis results . . . 99
28 Submitting user behaviour data to the recommendation API . . . 103
29 Simple item recommendation . . . 104
30 Greenpeace editor support . . . 105
31 Shoof Cross Media Recommendation . . . 105
32 Pseudocode: Determine subject content . . . 109
33 Pseudocode: Collect subject interest data . . . 110
34 Pseudocode: Determine species interest . . . 110
35 Pseudocode: Generate user profiles . . . 111
36 Pseudocode: Generate subject recommendation set . . . 111
37 Pseudocode: Recommend subject sets per user . . . 111

List of Tables

1 Overview of all MICO extractors. . . . 4
2 Text Analysis Extractor based on the Redlink Analysis Service (TE-213 and TE-220) . 7
3 TE-214 implementation: Speaker Diarization with LIUM . . . 9
4 TE-214 AudioDemux: Example implementation based on ffmpeg . . . 10
5 TE-226 implementation: Nudity classification . . . 11

6 TE-201 implementation: Generic Feature extraction using Fraunhofer XPX 12
7 TE-2111 implementation: Fraunhofer VSM 13
8 The Stanford NLP extractor (TE-213 - sentiment, TE-220 - neamed entity recognition) 15
9 The OpenNLP Named Entity Recognition Extractor implementation (TE-220) 16
10 The OpenNLP Sentiment Extractor implementation (TE-213) 18
11 Broker: functional requirements 24
12 Broker: Non-Functional Requirements 24

1 Executive Summary

This document provides the final specification draft for WP2-5 in MICO. Based on the initial specification from Dx.2.1, and considering the first software testing and evaluation results from our showcase parters InsideOut10 and Zooniverse, it covers all updates and newly introduced extractors and pipelines, and specifies the next version of the MICO broker, which will be implemented in year 3. Moreover, it describes version 2 of the annotation model, and finally provides an update to the querying and recommendation capabilities for the MICO platform.

Section 2 introduces the textual, visual and audio extractor enhancements and updates, and describes current pipelines and pipelines planned for year 3. Moreover, it outlines the technical requirements for extractor orchestration, relevant broker updates until now, and provides the broker v3 design to be implemented in year 3.

Section 3 describes the updated ontology for multimedia metadata (MICO MetadataModel[1]). This includes a summary of the model basics, lessons learned from the first two projects years, and how they lead to version 2 of the metadata model, which includes OWL specifications and descriptive documentation for improved usability of the model. In addition, a dedicated ontology has been designed to incorporate MICO showcase-specific needs. Finally, a description of Anno4j[2] is provided, which is a software designed to simplify RDF and SPARQL usage via MICO annotations.

Section 4 delivers an extended specification of SPARQL-MM, a multimedia query language that has been introduced in D4.2.1. It includes additional fragment accessor functions, an extension of the Media Fragment URI standard, a description of GeoSPARQL features and a introduction to SPARQL Inferencing Notations. Furthermore, it includes an outlook to features to be implemented in year 3.

Section 5 describes the recommendation framework for MICO, which aims at using selected results of cross-media analysis with user data, to provide cross-media recommendation. After providing the updated recommendation user stories and requirements, it describes the framework architecture, with a focus on the different needs of the relevant showcases Shoof, Greenpeace Magazine and Snapshot Serengeti. Moreover, the section includes the specification of interfaces and component interactions and a comparison with the previous D5.2.1, and a description of available data sources.

[1] `http://mico-project.bitbucket.org/vocabs/mmm/2.0/documentation/`

[2] `https://github.com/anno4j/anno4j`

2 Specifications and Models for Cross-Media Extraction & Orchestration Components

The following provides all relevant updates to the specifications and models of WP2 extractors and broker:

- An overview over WP2 *extractors* is provided in section 2.1. Extractor specifications that have been created or updated since D2.2.1 and D2.3.1 are provided in section 2.2, and current and year 3 pipelines are described in sections 2.3 and 2.4.
- An overview over *orchestration and the MICO broker* is provided in section 2.6. This includes the long-term 'wishlist' for extractor orchestration in section 2.7, relevant broker updates since broker v1 in 2.8, and a description of the final broker v3 design to be implemented in year 3 in Section 2.9.

2.1 Extractor Overview

D2.2.1 (provided in Oct 2014) specified **Technology Enablers (TE)**, which represent abstract descriptions of functionalities required by showcase partners. It also provided a specification of respective input and output data, often using TE-specific formats, which was then used to provide the first specification of the semantic data model provided with D3.2.1.

These specifications were followed by an intense implementation and system integration phase that lasted from Oct 2014 until April 2015. D2.3.1, which was provided in April 2015, therefore consequently described the concrete software implementations of the TE, which in this context are called **Extractors**. For some implementations, Open Source Software (OSS) was used, while others were adapted based on proprietary background from partners. Apart from significant efforts necessary for the implementation or adaptation, and integration of extractors, some extractors required R&D efforts for of parameter tuning, model training for machine learning algorithms and algorithm modifications.

The following table provides a compact overview over all extractors that have been developed for MICO and which are planned for the final phase of the project in Year 3. Apart from name, version, programming language and relevant TE, the table shows whether the extractor is Debian-packable (Deb.) – please note that some of the extractors represent work in progress and hence, while being present in the repository, they are not yet deployed to the MICO package server. The table also describes whether extractors are producing MICO metadata model RDF annotations, or a native format that need to be transformed to RDF afterwards.

The extractors serve different purposes: While most extractors provide *annotations*, others are necessary for intermediate, preparatory *processing steps*, and yet others are *annotation helpers* which convert native annotations from C++ extractors to RDF.

Details about extractor updates and new extractors provided since the previous deliverable are provided in Section 2.2.

Table 1 Overview of all MICO extractors.

Name	Ver.	Lang.	Deb.	RDF	TEs	Purpose
Object & Animal Detection	1.0.2	C++	yes	yes[1]	TE-202	annotation
Audio Demux	1.0.1	C++	yes	no	TE-214	processing step
Face Detection	1.0.3	C++	yes	yes[1]	TE-204	annotation
Diarization	1.0.1	Java	yes	no	TE-214	processing step
Kaldi2rdf	1.0.1	Java	yes	yes	TE-214	annotation helper
Kaldi2txt	1.0.0	Java	yes	no	TE-214	processing step
Redlink Text Analysis	1.0.0	Java	yes	yes	TE-213/220	annotation
ObjectDetection2RDF	1.0.3	Java	yes	yes	TE-202	annotation helper
Speech-to-Text	1.0.0	C++	yes	yes[1]	TE-214	annotation
Temporal Video Segmentation	1.1.3	C++	yes	yes	TE-206	annotation
Media Quality	1.0.1	C++	no	no	TE-205	annotation
Audio Editing Detection	1.0.0	C++	no	no	TE-224	annotation
Media Info	1.0.0	C++	no	yes[1]	TE-227	annotation
MediaTags2rdf	0.0.1	Java	no	yes	TE-227	annotation helper
Speech-Music-Discrimination	1.0.0	C++	no	no	TE-207	annotation
Nudity Detection	planned	C++	no	no	TE-226	annotation
Generic Feature Extraction	planned	C++	no	no	TE-211/207	processing step
Video Segment Matching	planned	C++	no	no	TE-211	annotation
Stanford NLP	planned	Java	no	no	TE-213/220	annotation
OpenNLP NER	planned	Java	no	no	TE-220	annotation
OpenNLP Sentiment	planned	Java	no	no	TE-213	annotation

[1] via annotation helper

2.2 Extractor Updates and New Extractor Specifications

This section contains updated and new extractor descriptions. In the cases where there has been no change since D2.2.1 and D2.3.1, the extractors are not included in the list. All information can then be gathered from either D2.2.1 or D2.3.1.

2.2.1 Object and Animal Detection – OAD (TE-202) – UPDATE

For the initial version of the blank image (emptiness) and animal detection extractor we integrated a HoG (Histogram of oriented Gradients) detector [DT05]. In order to train the detector for the Zooniverse data set, we annotated a large subset (region annotations) for all 43 animal classes of the Snapshot Serengeti data set. Though this extractor was released as first version of the MICO animal extractor we did not expect the best results due to the difficulty image content and the limitedness of the HoG approach for detection deformable objects such as animals. Therefore we rather considered the approach as baseline for the final implementation.

In Year 3, we will improve the blank image and animal detection extractor as well as the animal classification using two approaches:

1. We will use a pre-processing step that exploits the fact that about 80% of the Snapshot Serengeti subjects are 3-image sequences shot within a short time frame. We may use this information for pre-filtering interesting regions. Based on simple image differencing or advanced background modelling, this may help to (i) to identify blank images without any animal classification (ii) to improve detection performances of the animal detector.

2. We will apply a detection approach based on Deformable Part Models [Fel+10] which showed significantly better results in short pre-tests. Independent of Step 1 this will help to increase the rather bad results of the HoG-based detection approach.

2.2.2 Temporal Video Segmentation – TVS (TE-206) – UPDATE

Since the previous deliverable, this extractor has undergone some improvements in terms of RDF annotation and decoding. We extended the extractor such that it produces RDF annotations directly without an extra extractor (annotation-helper) in the pipeline. This has two advantages. The extractor is the first one, demonstrating that the RDF annotation via the C++ extractor API is viable although not as comfortable as in the Java world. In addition it saves some processing time and configuration overhead. In the video decoding parts we mainly improved the meta data extraction and the correct frame handling using the ffmpeg libraries.

The extractor it the most mature one in the MICO system. There is no need for extension in Year 3.

2.2.3 Audiovisual Quality – AVQ (TE-205) – UPDATE

In the previous deliverable (April 2015) we outlined the following plans:

1. We will combine the single technical measures into one traffic light kind of measure we call media quality. The media quality represents a accumulated, more understandable quality measure. It is the weighted combination of selected quality features and events. In order to find the right selection and weighting we will conduct further experiments within MICO.

2. We need to reduce the amount of data when storing video annotations to the RDF model.

These strategies are still valid. For the specification on how to do this, we now opted for an configurable event based annotation to reduce the amount of annotation data. This means that a quality annotation is only produced, when an configurable media quality threshold is exceeded. The extractor will still be able to produce fixed time step annotations. The next evaluation phase will show if these annotations can be used meaningfully and with acceptable query performance.

2.2.4 Face detection – FDR (TE-204) – UPDATE

The face detection extractor was deployed in June 2015 and subsequent releases. It uses the open source software libccv which implements the SURF-Cascade Detection [LZ13]. The evaluation report will describe how the training model will perform on the showcase evaluation. In order to decrease the amount of annotation data as well as the processing time in the video analysis case, we chose two pipeline-based approaches:

1. Face detection is run on frames at fixed time steps.

2. Face detection is run on frames extracted as shot boundaries or key frames from the Temporal Video Segmentation Extractor.

The pipeline section (Section 2.3) gives more information about this. However, the first approach will be more usable once the new broker version supports job-specific parametrization of the extractor from a user perspective. For Year 3, there are no major updates planned for this extractor. If required an additional face model could be trained if there is a usable annotated data set available for this purpose.

2.2.5 Speech-Music Discrimination – SMD (TE-207) – UPDATE

First experiments with the Speech-to-Text extractor (see Section 2.2.6) revealed, that when an audio track contains music, the text produces by the extractor becomes meaningless or even false. Especially, when named entity recognition is applied to such wrong results, wrong entities might be detected. We therefore considered to to include the Speech-Music Discrimination extractor in such a pipeline which detects and annotates time ranges in an audio signal, where silence, speech, speech and music and music is detected.

The first version of this extractor was left in development stage since it was not able to achieve an appropriate temporal resolution of the results (was $3s$). In year three we will extend the extractor in order to decrease that resolution to $0.5s$ and also to produce RDF annotations.

2.2.6 Speech-to-Text (TE-214) – UPDATE

The initial version of the ASR (automatic speech recognition) extractor based on KALDI had several drawbacks (memory consumption, processing time) making it not usable in a meaningful manner. Since the June 2015 release we therefore started to optimize both, the extractor and the the extractor pipeline to make the extractor usable for evaluation. The major change was that we split the data in small pieces of processing ranges using the new Diarization extractor (2.2.8) and extended the extractor to split the analysis and finally join the results. In addition two annotation helpers had been updated (kaldi2rdf for annotation for each recognized word) and newly integrated (kaldi2text plain text output for subsequent NER processing). We also added an extractor splitting the audio stream from a video and samples down the audio signal as pre-processing step for the Speech-to-Text extractor (section 2.2.9).

Yet there is still room for improvement. The training model are rather huge and the start-up of a pipeline with this extractor takes a significant amount of time since the model needs to be loaded into memory. Also processing time is an issue. The extractor currently processes a an audio track at approx. 3x video playing time.

Year 3 will see improvements in the support of languages other than Englishe (see Section 2.5, and if time allows, improvements to the extractor's computational performance.

2.2.7 Redlink Text Analysis Extractor (TE-213, TE-220) – NEW

Extractor that uses the Redlink Analysis Service part of the Redlink Semantic Platform [3] to extract Named Entities; link to Entities defined in custom vocabularies or Wikipedia; extract keywords; classify texts along classification schemes and or extract keywords.
The Extractor itself is open source. The usage of the Redlink service requires an account for Redlink [4].

2.2.7.1 Specific comments

Performance: This extractor does only put minor load on Mico as the processing is done by the Redlink Analysis Service. The extractor itself sends the input content (text/plain) to the service and processes the received analysis results. Every request to the extractor will generate a request to the Redlink service. Those requests will count to the limit set for the account.

Output: The extractor outputs RDF according to the MICO metadata model (Open Annotation)

[3] `http://redlink.co/semantic-platform/`
[4] `https://my.redlink.io/`

Table 2 Text Analysis Extractor based on the Redlink Analysis Service (TE-213 and TE-220)

Name	mico-extractor-named-entity-recognizer
Original license	Apache Software License 2.0
MICO integration license	Apache Software License 2.0
External dependencies	Redlink Analysis Service(`https://my.redlink.io/`)
Input data	Text - while the Redlink platform supports plain as well as several rich text formats the extractor is currently limited to the media type 'text/plain'
Output data	The extractor outputs RDF only. No binary content part is added to the processed content item
RDF persistence	The extractor writes Open Annotation (OA) annotations as defined by the MICO metadata model. Used Annotation Bodies for Named Entities, Linked Entities, Topic Classification, Sentiment Annotations and Keyword Extraction are taken from the Fusepool Annotation Model (`https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md`)
External Parameters	None
Internal Parameters	The Extractor need the 'Redlink Analysis Name' (-a) the 'Redlink Key' (-k) to be configured as parameters for the daemon. In addition the extractor allows to specify a 'Queue Name' (-q) parameter. This has to be used if multiple instance of this extractor configured for different 'Redlink Analysis' configurations can be addressed by the MICO broker.
Additional requirements	This extractor requires an account for the Redlink Analysis Service

and uses Annotations Bodies as defined by the Fusepool Annotation Model [5] to annotate features extracted form the analyzed textual content.

The Extractor supports the following types of annotations:

- Content Language (`fam:LanguageAnnotation` [6]): annotates the language of the processed text.
- Named Entities (`fam:EntityAnnotation` [7]): annotates named entities (person, organization, location and others) found in the text. The annotation also provides a selector with the exact position of the entity.
- Linked Entity(`fam:LinkedEntity` [8]): annotates the mention of an Entity as define by a controlled vocabulary in the processed text. This is similar to a Named Entity, but also provide the reference (URI) for the detected entity.
- Topic Classification(`fam:TopicClassification` and `fam:TopicAnnotation` [9]): annotation that classifies the processed text along topics defined by some classification scheme. Each `fam:TopicClassification` consists of one or more `fam:TopicAnnotations`. The confidence of a topic annotation defines how well the processed text fits to the topic. If topics are defined by a controlled vocabulary the annotations will also provide a reference (URI) to the topic.
- Sentiment Annotation (`fam:SentimentAnnotation` [10]): annotation that defines the sentiment of the processed text in the range of [-1..+1] where -1 sands for negative and +1 for a positive sentiment.
- Keyword Annotation(`fam:KeywordAnnotation` [11]): keywords are words and phrases with a special improtance for the processed text. Keyword annotations provide a metric [0..1] that defines the importance as well as the count how often the keyword is mentioned in the processed text.

Note that the set of annotations extracted form the processed text will depend on the configuration of the configuration of the Redlink analyser application.

[5] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md

[6] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#language-annotation

[7] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#entity-mention-annotation

[8] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#linked-entity-annotation

[9] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#topic-classification

[10] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#sentiment-annotation

[11] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#keyword-annotation

2.2.8 Diarization (TE-214) – NEW

Diarization consists in partitioning an audio stream into segments corresponding to different speakers. This is a useful preprocessing step for speech-to-text, as it divides potentially long audio files into manageable pieces. The diarization extractor is also able to identify different speakers, though this functionality is currently not exploited in MICO.

Table 3 TE-214 implementation: Speaker Diarization with LIUM

Name	mico-extractor-diarization
Original license	GNU General Public License, version 3
MICO integration license	Apache license, version 2.0
External dependencies	LIUM Speaker Diarization tool (`http://www-lium.univ-lemans.fr/diarization/doku.php/welcome`)
Input data	Audio wav file
Output data	Time-stamped segments containing speech, annotated with diarization information (XML).
RDF persistence	Not supported, as this is a pre-computiation of the Kaldi extractor
Additional requirements	None.

2.2.8.1 Specific comments

Performance: The LIUM extractor is running well, is fast and produces useful pre-processing results. The only downside from a software point of view is, that it is running another Java app out of Java by starting it as a separate process. However, as a draft solution it is satisfactory.

Output: Worth mentioning is that the segmentation information provided by the diarization tool could be one large segment if there are continuous speech by a single speaker. For long audio files it is necessary to have this in mind for other extractors using this information.

2.2.9 Audio Demux – DMX (TE-214) – NEW

This extractors serves as a pre-processing step to the Speech-to-Text extractor. It extracts the audio data stream from a video and changes its sampling rate to a rate best suited for automatic speech recognition.

Table 4 TE-214 AudioDemux: Example implementation based on ffmpeg

Name	mico-extractor-audiodemux
Original license	GPL or LGPL respectively
MICO integration license	Apache License 2.0
External dependencies	ffmpeg library (avformat, avcodec, avutil, avfilter, avresample)
Input data	video file
Output data	audio file with configurable sampling rate
RDF persistence	not required
External Parameters	input mime type, target sampling rate
Internal Parameters	none
Additional requirements	none

2.2.9.1 Specific comments

Performance: Pretty good processing performance due to optimized decoding and fast re-sampling via ffmpeg.

2.2.10 Nudity Detection – NDE (TE-226) – Year 3

The nudity extraction extractor detects inappropriate or adult-only content in images and videos. It required to pre-filter user generated content in the Inside Out showcases. Since the priority of this extractor is medium, we consider integrating an existing classification approaches and software for this task. We did a first review of the available approaches but did not decide yet which to go for. If it turns out that existing services or library solutions are not usable or are too bad in their results we will consider to drop that extractor within the MICO project duration.

Table 5 TE-226 implementation: Nudity classification

Name	Nudity Detection
Original license	not yet clear
MICO integration license	Apache License 2.0
External dependencies	most likely OpenCV
Input data	images videos
Output data	amount of skin occurrence /nudity
RDF persistence	native annotation
External Parameters	not yet clear
Internal Parameters	not yet clear
Additional requirements	none

2.2.10.1 Specific comments

Since efforts are limited we are definitely opt for an out-of-the-shelf component solution for this extractor. Training of new models is not planned.

2.2.11 Generic Feature Extraction – GFE (TE-201) – Year 3

This extractor will support the extractors Speech-Music-Discrimination (Section 2.2.5) and Video Matching (Section 2.2.12) by providing the functionality of Fraunhofer feature extraction framework XPX as a MICO extractor.

Table 6 TE-201 implementation: Generic Feature extraction using Fraunhofer XPX

Name	Generic Feature-Extraction
Original license	Proprietary (FhG)
MICO integration license	Apache Software License 2.0/Proprietrary (FhG)
External dependencies	xpx-api (FhG)
Input data	image, video, audio
Output data	binary feature container (afp)
RDF persistence	none / (if used for SMD, native annotation implementation)
External Parameters	feature depended
Internal Parameters	feature depended
Additional requirements	feature extraction configuration file (xml)

2.2.11.1 Specific comments

Performance: The Fraunhofer XPX framework is designed for high speed, multi-processor, feature extraction and classification.

Output: Since this extractor uses a very generic approach literally any output is possible. In the specific context of the MICO project the extractor will produce the features in a native binary format.

2.2.12 Video Segment Matching – VSM (TE-211) – Year 3

The video matching extractor finds video sequences originating from the same source that have been re-used in other videos. This can be used for de-duplication within a video collection, content tracking and provenance or copyright infringement detection.

Table 7 TE-2111 implementation: Fraunhofer VSM

Name	Video Segment Matching
Original license	Proprietary (FhG)
MICO integration license	Apache Software License 2.0/Proprietrary (FhG)
External dependencies	Fraunhofer Video Segment Matcher (vsmapi)
Input data	query content part URI, reference content part URIs
Output data	native json format with segment matches
RDF persistence	to be decided base on what needs to be modelled and at what level - native implementation
External Parameters	matching mode / i. e. matching accuracy vs. matching speed)
Internal Parameters	-
Additional requirements	requires proprietary features descriptor extracted by the Generic Feature Extractor GFE (section 2.2.11)

2.2.12.1 Specific comments

Performance: Depending on the amount of data, the matching will take several time. Due to the MICO architecture, video feature will not be stored in an in-memory data base but rather loaded from the MICO content storage whenever a matching process is taking place.

2.2.13 Stanford NLP Extractor (TE-213, TE-220) - Year 3

Stanford NLP [12] is a software framework for NLP processing provided by the Stanford NLP Group under GPLv3 [13] license.

This extractor uses Stanford NLP to support Named Entity Recognition (TE-220) and Sentiment Analysis (TE-213). For multilingual aspects, see Section 2.5.

2.2.13.1 Specific comments

Performance: Stanford NLP holds NLP models in memory. It does support multi-threading. Running this extractor will require sufficient memory to hold all used models. Scaling will mostly depend on the number and speed of the available CPU cores.

Output: As all MICO NLP extractors the Stanford NLP extractor will use the MICO metadata model v2 and create annotation bodies for Named Entities and Sentiment Annotations as defined by the Fusepool Annotation Model [14].
In detail the Stanford NLP extractor generates the following annotations:

- Language Detection: This is a pre-requirement for most NLP processing tools. The result will be encoded using a `fam:LanguageAnnotation` [15].
- Named Entity Recognition (NER): Named Entity Recognition allows to detect mentions of trained entity types (typically Persons, Organization and Locations) in texts. The Stanford NLP frameworks includes models for English, German and Chinese. Additional models are available via the Europeaner-Newspaper project [16] that publishes models for Dutch, German and French. Named Entity results will be represented by `fam:EntityAnnotation` [17].
- Sentiment Annotation: Stanford NLP provides Sentiment annotation support for English [Soc+]. The speciality of this solution is that it works on the parse tree and not on document level. Because of that it is possible to identify phrases of the content that carry most of the Sentiment. This feature allows to create a `fam:SentimentAnnotation` [18] with an assigned `oa:Selector` for the exact phrase in the text. In addition the extractor will also aggregate a sentiment value for the document as a whole.

[12] http://nlp.stanford.edu/software/index.shtml
[13] http://www.gnu.org/licenses/gpl-3.0.html
[14] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md
[15] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#language-annotation
[16] http://lab.kbresearch.nl/static/html/eunews.html
[17] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#entity-mention-annotation
[18] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#sentiment-annotation

Table 8 The Stanford NLP extractor (TE-213 - sentiment, TE-220 - neamed entity recognition)

Name	Stanford NLP Extractor
Original license	GPL v3.0
MICO integration license	Dual License: Apache Software License 2.0 and GPLv3 - with the intention to allow people the freedom of the ASL 2.0 to copy, branch, modify the extractors code. Running the extractor will require to confirm to the GPLv3 as the extractor links the Stanford NLP library.
External dependencies	Stanford NLP CoreNLP (`http://nlp.stanford.edu/software/corenlp.shtml`) and Sentiment(`http://nlp.stanford.edu/sentiment/`)
Input data	This extractor will support plain text input. Plain text originating from speech-to-text extractors will require a different configuration for optimal results.
Output data	RDF
RDF persistence	Web annotation (`http://www.w3.org/annotation/`) based annotations as defined by the MICO metadata model v2.0. Annotation Bodies for Named Entities and Sentiment Annotations as defined by the Fusepool Annotation Model (`https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md`)
External Parameters	none
Internal Parameters	The Extractor will require a configuration for NLP models based on the language and possibly the type (news articles, papers/articles, blogs, forum posts, ...) of the text. Especially text originating from speech-to-text extractors might need a different set of models to produce good results. It will allow multiple instances configured with a different set of models to be used in parallel.

2.2.14 OpenNLP Named Entity Recognition Extractor (TE-220) - Year 3

Extractor for Named Entity Recognition based on Apache OpenNLP [19] and the IXA Pipes [20] [RAR14] extensions. For multilingual aspects, please see Section 2.5.

Table 9 The OpenNLP Named Entity Recognition Extractor implementation (TE-220)

Name	(tbd) MICO OpenNLP Named Entity Recognition Extractor
Original license	Apache Software License 2.0
MICO integration license	Apache Software License 2.0
External dependencies	This Extractor will be based upon the OpenNLP frameworks and the IXA Pipes extensions as well as language models (`http://ixa2.si.ehu.es/ixa-pipes/`)
Input data	Plain text originating from written text or from speech to text transcoding
Output data	RDF
RDF persistence	Web annotation based annotations as defined by the MICO metadata model v2.0 with Named Entity annotations as defined by the Fusepool Annotation Model
External Parameters	none
Internal Parameters	The Extractor will require NLP processing models to be configured. It will allow multiple instances configured with a different set of models to be used in parallel.

2.2.14.1 Specific comments

Performance: OpenNLP holds NLP models in memory. Using this Extractor will require sufficient memory for loading all configured models in memory. For the IXA Nerc models this is about 5 GByte. OpenNLP can process concurrent requests on different CPU cores. Therefore scaling depends on the number and speed of available CPU cores.

Output: As all MICO NLP extractors the OpenNLP NER extractor will use the MICO metadata model v2 and create annotation bodies for Named Entities as defined by the Fusepool Annotation

[19] `http://opennlp.apache.org/`
[20] `http://ixa2.si.ehu.es/ixa-pipes/`

Model [21].

Named Entity Recognition (NER) allows to detect mentions of trained entity types (typically Persons, Organization and Locations) in texts. While OpenNLP comes with support for NER the quality of the default models is not sufficient for use in most application. The models distributed by IXA Pipes NERC [22] provide much better quality. IXA Pipe NERC provides NER models for Basque, English, Spanish, Dutch, German and Italian. Named Entity Results will be represented by `fam:EntityAnnotation` [23] annotations.

[21] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md

[22] https://github.com/ixa-ehu/ixa-pipe-nerc

[23] https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#entity-mention-annotation

2.2.15 OpenNLP Sentiment Analysis (TE-213) – Year 3

The extractor for Sentiment Analysis is based on Apache OpenNLP [24] and uses the Document Categorizer [25] functionality based on maximum entropy algorithm[26]. For multilingual aspects, please see Section 2.5.

Table 10 The OpenNLP Sentiment Extractor implementation (TE-213)

Name	MICO OpenNLP Sentiment Extractor
Original license	Apache Software License 2.0
MICO integration license	Apache Software License 2.0
External dependencies	This Extractor will be based upon the OpenNLP Document Categorizer functionality.
Input data	Plain text originating from written text or from speech to text transcoding
Output data	RDF
RDF persistence	Web annotation based annotations as defined by the MICO metadata model v2.0 with Named Entity annotations as defined by the Fusepool Annotation Model
External Parameters	none
Internal Parameters	The Extractor will require Sentiment Classification models to be configured. It will allow multiple instances configured with a different set of models to be used in parallel.

2.2.15.1 Specific comments

Performance: OpenNLP holds NLP models in memory. Using this Extractor will require sufficient memory for loading all configured models in memory. Document Classification models are expected to need memory in the range of 100 MByte of RAM each. OpenNLP supports multi-threading what means that multiple concurrent requests can be processed on different CPU cores. Scaling depends therefore on the number and speed of available CPU cores.

Output: As all MICO NLP extractors, the OpenNLP NER extractor will use the MICO metadata model v2 and create annotation bodies for Named Entities as defined by the Fusepool Annotation

[24] `http://opennlp.apache.org/`
[25] `https://opennlp.apache.org/documentation/1.6.0/manual/opennlp.html#tools.doccat`
[26] `https://opennlp.apache.org/documentation/1.6.0/manual/opennlp.html#opennlp.ml.maxent`

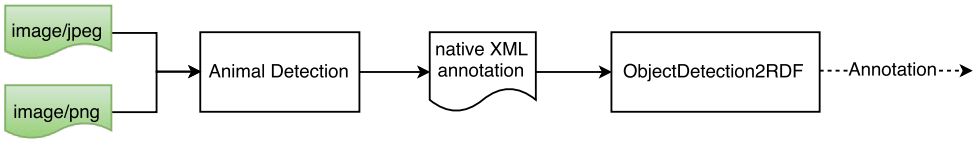
Model [27].
The Document Classifier will only provide a sentiment classification on document level. This means that the Extractor will provide a single `fam:SentimentAnnotation` [28] for the document. Technically it would be possible to split the content in multiple parts and perform separate classifications for those. Depending on requirements of the use cases this option might get explored.

2.3 Extractor Pipelines

For the first MICO platform release (June 2015) a number of extractor pipelines have been created. Details about how these pipeline are technically implemented gives Section 2.8.1. This section show the combination of the developed MICO extractors in different pipelines and shortly explains their purpose. It shows, which unstructured data a pipeline produces and which structured data in the MICO data model it annotates.

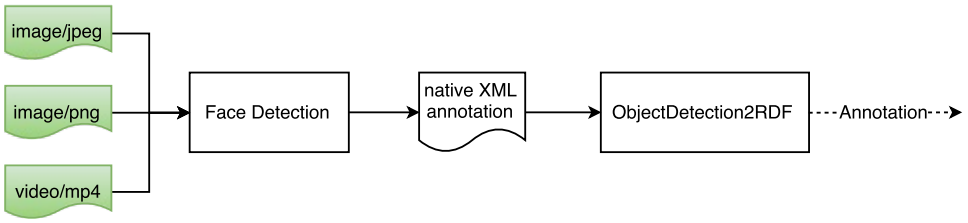
Animal detection pipeline (Figure 1): This pipeline is used in the Zooniverse showcase for blank image and animal detection. Due to the need of fixed parameters in the broker Version 1 implementations two pipelines exist which provide different detection models (blank, animal classification).

Figure 1 Animal detection pipeline (animal-detection)



Face detection pipeline (Figure 2): This pipeline is designed in two variants (image, video). It annotates detected face regions per image/frame. In the video case, we set a fixed time step (currently 3s), in order to reduce amount of annotations (compare Section 2.2.4). The pipeline resembles a sub set of the more complex Inside-Out pipeline and is used for evaluation of the face detection performance.

Figure 2 Face detection pipeline (face-detection).



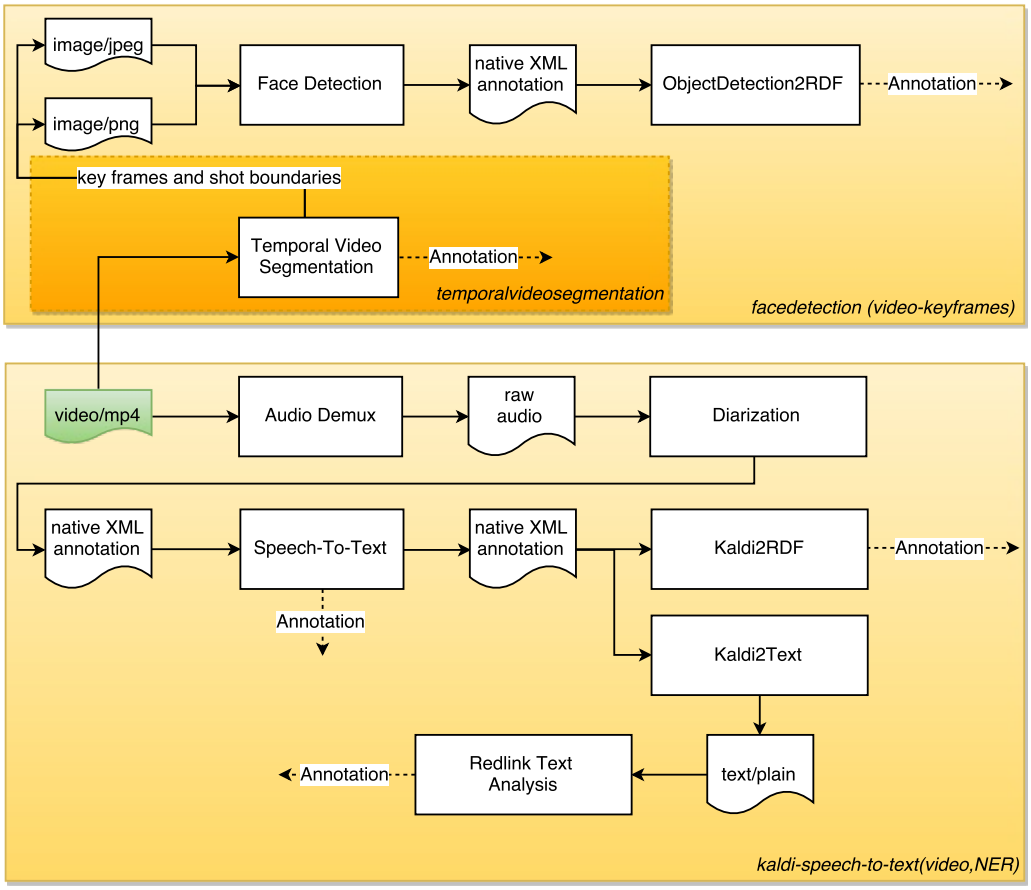
IO Showcase pipeline (Figure 3): This huge cross-media pipeline actually combines three pipelines (yellow/orange rectangles) into one. The pipelines are also available as separate pipelines in the system in order to reduces processing time if some annotation are not required. The pipelines contain most of the extractors, developed in the project until now. The input data is a video. The first pipeline annotates

[27] `https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md`

[28] `https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md#sentiment-annotation`

shots (i.e. edits in a video) in the visual stream and produces shot boundary frames (first frame of a shot) and key frames (important frames within a shot). The second pipeline is a modified face detection pipeline which operates on these shot boundary or key frames. The third pipeline uses the audio stream of the video converts that into text which is then processed by a named entity recognition extractor. This pipeline allows for queries such as "Give me all shots (temporal video segmentation) in a video where a person (face detection) says something (text-to-speech) about topic XYZ (named entity recognition)".

Figure 3 IO Showcase pipeline (io-demoshowcase-all, temporalvideosegmentation, kaldi-speech-to-text, face-detection)



2.4 Extractor Pipelines Planned for Year 3

For Year 3 we mainly plan to improve and extend the existing pipelines in order to (1) create more cross media annotations, (2) make the results more robust and thus the annotations more useful, and (3) make the extractors faster and less resource consuming. We will also add a limited number of new extractors such as the Video Segment Matching extractor or the new NLP extractors. As experienced after the first release, also completely new pipeline ideas will created once new extractors are ready and we play around with them. Due to the increasing robustness of the system, the pipeline creation process will

become much more user friendly and facilitate this process. Along with the improvements of the broker (see Section 2.9) this will make the MICO system ready-to-use for our early system users InsideOut, Zooniverse or Zaizi.

2.5 Multilingual support

During the final year of MICO, an important goal is to increase the number of supported languages. Both speech-to-text pipeline (TE-214), named entity recognition (TE-220) and sentiment analysis (TE-213) where primarily developed for the English language. This was due to the fact that anguage resources for English are easy to come by, and because all project members are fluent in English and English was also an importnat language across all the different use cases. Now that the pipeline is in place, a natural next step is to include support for more languages. This chapter will provide more information about ongoing and planed work related to multi lingual support.

2.5.1 Speech-to-Text Multilingual support

Towards this end, we have begun to convert existing language models and train new ones for the Kaldi system that is at the heart of the speech-to-text pipeline. The first languages to be considered are those prioritised by the MICO use-case partners, namely Italian and Arabic.

Initial versions of named-entity recognition (TE-220) and sentiment analysis (TE-213) are also in place, again primarily built for English. In the final implementation of these, we shall base our work on software libraries with extensive language cover such as Stanford NLP and OpenNLP.

A dialog has also been opened with Synthema, one of the leading partners of the EU project "Sharing audio-visual language resources for automatic subtitling" (SAVAS) [Poz13]. This project served to

- collect spoken and textual resources in six European languages (Basque, Spanish, Portuguese, Italian, French and German) from the broadcasters and subtitling companies acting as data providers within the consortium;
- transcribe and annotate the collected corpora into a form suitable to train acoustic and language models of Large Vocabulary Continuous Speech Recognition (LVCSR) systems using a combination of automatic and collaborative approaches;
- build a local META-SHARE[29] repository containing the collected and annotated SAVAS language resources to allow their reuse; and
- adapt and train dictation and transcription LVCSR systems with the SAVAS language resources.

We are presently doing an inventory of the SAVAS META-SHARE repository to identify resources that can help improve MICO's multilingual support in the speech-to-text pipeline. META-SHARE is a sustainable network of repositories of language data, tools and related web services documented with high-quality metadata, aggregated in central inventories allowing for uniform search and access to resources. Seeing that Italian is also a prioritized language in MICO, there are likely to be synergies between the two projects.

[29] Co-funded by the 7th Framework Programme of the European Commission through the grant agreement no. 249119.

2.5.2 Named Entity Recognition Multilingual support

With the addition of two extractors for Named Entity Reposition based Stanford NLP 2.2.13 and OpenNLP 2.2.14 users of the MICO platform will gain access to high quality language models of the following languages: English (OpenNLP and Stanford NLP), German (OpenNLP and Stanford NLP), Spanish (OpenNLP), French (Stanford NLP), Italien (OpenNLP), Dutch (OpenNLP and Stanford NLP), Basque (OpenNLP) and Chinese (Stanford NLP).

The UMU group are also training a model for Swedish, and will investigate the feasibility of adding Arabic. The Swedish model is based on data från the public resource Språkbanken, but it is still an open question what data set to use for Arabic.

2.5.3 Sentiment Analysis Multilingual support

In the final version of the MICO plattform there will be two Extractors supporting (different types) of Sentiment Analysis. First the Stanford NLP extractor 2.2.13 supports a phrase level sentiment detection. This can e.g. be used extract those phrases in document that carry the most sentiment. The OpenNLP sentiment classification 2.2.15 is based on document classification that is trained to classify for their sentiment. This implementation can only provide sentiment values for document as a whole (or subsection if one calculates multiple classification for different section of a document).

Stanford NLP provides a model for the English language. While the tool also provides a training tool the requirements for training sets are rather high. In addition the sentiment classifier also requires a parse tree for the text. Stanford NLP Shift-Reduce Constituency Parser [30] has support for English, German, French, Arabic and Chinese. So for training of Sentiment analysis for other of those languages one would also need to train a parser model first.

Sentiment classification for OpenNLP is easier to train as a training set only requires to assign documents with their sentiment category. However OpenNLP does not provide any document classifier model that can be used for sentiment classification.

During the last year of the project, the UMU plans to provide two classification models; one for rating the confidence of a given text, and one for assessing the competence w.r.t. to a particular task of its author. Both analysers are motivated by the Serengeti use case, where we want to discover competent but non-confident users and encourage them not to leave the effort.

2.6 Broker Overview

The initial framework for orchestration of extractors via pipelines, aka **MICO broker** v1, was provided and described with D6.2.1 (Oct. 2014). While it was clear that there was room for improvement regarding the broker, and a first qualitative overview of respective requirements was provided in D2.3.1, it was also clear that no major updates of the broker could be introduced during the preparation of the first platform release (Oct.2014 - April 2015) - they would have further complicated the already very challenging activities for extractor and model adaptation, implementation and integration. Moreover, it seemed reasonable to consolidate the broker requirements after the platform release, in order to validate the rather 'theoretical' requirements and to complement them with the practical experiences of platform release preparations.

Hence, the broker activities consisted of several threads and respective results:

1. The elicitation of requirements before and during the platform release preparations, which lead to a broker 'wishlist' described in section 2.7

[30] `http://nlp.stanford.edu/software/srparser.shtml`

2. The introduction of 'unavoidable' updates related to pipeline configuration as described in section 2.8.1, and updates to the event API right after the platform release in order to support error handling and progress communication, as described in section 2.8.2.

3. The design of the final broker v3, which will be implemented in year 3, and is described in section 2.9.

2.7 Broker v2 wishlist

The discussions following broker v1, and the practical experiences and lessons learned during and after platform release and evaluation phase, lead to an extended 'wishlist' for future MICO broker versions, considering input and prioritization of requirements from all WPs. Hence, *all requirements mentioned in the following go beyond broker v1 capabilities*. They are separated into *functional* (11) and *non-functional* (12) requirements.

ID	NAME	Description / Motivation	Prio
FR-1	EIP	The broker should directly or indirectly support Enterprise Integration Patterns (EIP) to orchestrate the processing, including, router, aggregator (FR-3), splitter (FR-2)	H
FR-2	Support for multiple output items	The broker should be able to deal with extractors that provide output of various types. The types are specified by the common data model resulting from T2.2. Each extractor should produce a fixed number of outputs of fixed types.	H
FR-3	Support for multiple input items	The broker should be able to deal with extractors that require input of various types, and hence waiting for several processes ('aggregator' from FR-1). Each extractor should however consume a *fixed* number of inputs of *fixed* types.	H
FR-4	No output	Some transitions do not produce new content parts, but are required by broker v1 to provide at least an empty one to trigger execution of other services; would be necessary to add some other sort of notification so the broker can send to the next service in sequence	M
FR-5	Dependencies	An extractor should declare dependencies that must be satisfied *before* being called: Needs existing data / output from other extractors, expressing semantic and syntactical interdependencies.	H
FR-6	Status tracking	The broker should maintain a list of active jobs, and track at which extractor each job is being processed (related with FR-13).	L
FR-7	Execution planning	V1 does not foresee execution planning as such: The format generated by an extractor is provided to and processed by all extractors that are able to handle it. Therefore, it currently executes all possible paths, regardless of whether needed / feasible or not. What is needed is a consideration of type constraints, expression of dependencies, optimal path, computation only of what is required, storage of preliminary results for expensive extractors, workflow balancing	M
FR-7a	Manual execution planning	The user specifies the entire execution plan through a graph-based GUI, i.e., that which is already available for Camel	L

ID	NAME	Description / Motivation	Prio
FR-7b	(Semi-)Automatic execution planning	The user provides extractor descriptions, input types and goal types, and the broker supports respective workflow creation according to that.	M
FR-8	Dynamic execution	Dynamic execution of the processing workflow based on dependency information (e.g. depending on the language identified, different path needs to be followed), see also EIP router pattern. This may, however, influence FR-2 and FR-3 regarding fixed number of inputs/outputs.	M
FR-9	Types support	The extractor needs to support the common data model developed in T2.2	H
FR-10	Loop avoidance	The execution plan should avoid unterminated loops.	M
FR-11	Intertwined annotation	It would be nice to have support for human interaction within workflows.	L
FR-12	Pull content	Apart from the current support of 'pushing' content to the platform, 'pull' support would be useful whenever an extractor needs to get input from the knowledge base: previous extractor results, manual annotations, etc., see EIP file transfer.	L
FR-13	Logging	Logging / monitoring of workflow execution (related to FR-6).	H
FR-14	Content-based queries	It would be nice to support the execution of content-based queries, which requires consideration of instantiation constraints because respective MICO components require matching through one central DB	L
FR-15	Extractor versioning	Versioning of components should be supported	H

Table 11: Broker: functional requirements

ID	NAME	Description / Motivation	Prio
NFR-1	Failure tolerance	What happens when an extractor fails? In v1, a failing service pushes messages back into the queue, trying to execute over and over again (infinite loop) - a distinction between permanent errors and recoverable errors is needed.	H
NFR-2	Compatibility	Keep as many broker v1 components as possible (messaging queue, etc.), as modifications can cause significant effort for other components developers.	H
NFR-3	Extensibility	Extensibility, here, is related to EIP supported (related to FR-1).	M
NFR-4	Testability	V1 provided only very limited support for testing pipelines and extractors, due the existing interdependencies.	M
NFR-5	Stability	Avoid the use of unstable components, especially regarding third-party developments.	M

Table 12: Broker: Non-Functional Requirements

While it will not be feasible to implement all requirements mentioned in the following (hence the

term 'wishlist'), a considerable amount of them have been either implemented, or have been considered into the broker design, and will be implemented in y3.

2.8 Broker updates after the platform release

The following describes the 'unavoidable' updates related to pipeline configuration that were necessary for the platform release, and updates of the event API that were absolutely necessary after the platform release, and will be used until broker v3 is fully implemented.

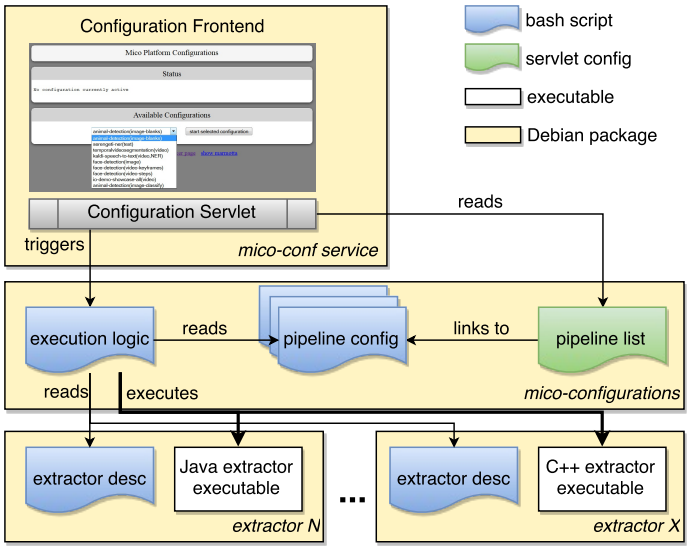
2.8.1 Pipeline configuration

We described the first version of the broker (broker v1) in an early stage of the project in D6.2.1 (Oct. 2014). The approach used simple mime type-based connections (i.e. string comparison) of extractor as pipelines. That meant, that as soon as a running extractor daemon registers itself with the broker, all possible connections were established, including unintended connections or even connections producing loops. Therefore we needed to extend the system in order to support the following features:

1. Standardized way of parameter specification passed to the extractor during start-up
2. Means of pipeline configuration defining the extractors involved and their parameters
3. End-User controlled start-up and shut-down of extractors establishing a pipeline for a specific purposes

These features needed to be compatible with the rather limited connection and data transmission capabilities in broker v1. We opted for an approach using a mixture of bash scripts and servlet configurations which is shown in Fig. 4. Every extractor deployed for the MICO system is obliged to support standard start-up and shut-down command line parameters. It also needs to be packaged along with a short description bash script specifying the name, description and system (native, Java) it is running for. For easy changes and updates, the pipelines are configured in a separate Debian package. They specify the extractors to be loaded and the parameters to be passed in addition to the run/stop arguments. Therefore the pipeline designer for the broker v1 needs a good knowledge about the extractors and their CLI parameters. The mico-configuration package also provides a servlet configuration which links pipeline names to pipeline configuration scripts and the actual extractor execution logic script. The servlet and front-end are provided as a third Debian package. The service reads the current configuration and triggers the execution logic for a selected pipeline which then start or stops the corresponding extractors.

Figure 4 Broker v1 Pipeline Configuration



2.8.2 MICO broker v2

During evaluation of the platform release and the first real usage of the system, it turned out that the communication channel between extractors to the broker had to be urgently extended: In broker v1, information about new content parts is the only data that the extractor could send to the broker. After getting a notification about new content, the broker then had to assume that the extractor finished processing successfully. However, if an extractor produced more than one new content part, all extractor errors apart from the first one would go unnoticed. Moreover, in this approach, extractors had to send a messages about new content parts, even when they did not provide any. Most importantly, the broker did not support the reporting and hence proper handling of errors during processing.

Considering the urgency of such improvements, we decided to introduce an intermediate broker v2 with extended capabilities before implementation v3 in year 3. V2 improves the event API, which deals with broker-extractor communication: The broker uses it to send information to extractors when new content is added to the platform, to trigger the required extraction processes. Extractors use the API to store extracted metadata in the triple store, and to inform the broker about new extracted binary content after pushing it to the data store.

A new message type was added to the event API to distinguish between following events during extraction processes:

- *new content part*: As in v1, this is used by the broker to trigger next extractors for new parts.
- *extraction finished*: Upon reception of this message, the broker can be sure that the extraction was successfully finished and the extractor is ready to process more content.
- *extraction aborted*: This indicates that an extractor was not able to process content - this message can contain more detailed information about the reason why the process was canceled which can be used to detect and avoid problems, e.g. related to the content itself, problems with the environment such as insufficient disk space or memory, etc.
- *extraction progressed*: This is an optional event, which is useful to indicate the progress in long-running processes.

V2 was implemented for the Java event API, and is currently implemented for the C++ event API as well - it will be used by extractors until broker v3 is fully implemented.

2.9 Broker v3 design

The following provides several sections on broker v3 design. The implementation phase has started already, and will be completed in year 3:

- principles and assumptions, in section 2.9.1
- relevant user stories, in section 2.9.2
- relevant components, in section 2.9.3
- the high-level broker data model, in section 2.9.4
- relevant use cases, in section 2.9.5
- the specification of the registration service, a key component for the v3 approach, in section 2.9.6

- information about workflow planning, in section 2.9.7
- information about workflow execution, in section 2.9.8

2.9.1 General broker principles and assumptions

The broker design is based on several high-level principles, insights and assumptions, which are described in the following:

To begin with, there are several principles and insights related to **extractor registration and model**:

- a key assumption is that some parts of the extractor information can and should be provided upon packaging by the developer (extractor properties, input and output), while other parts of the extractor information may be provided after packaging, by other developers or by showcase administrators (semantic mapping of extractors, and information / feedback about pipeline performance); hence, registration information is provided at different times
- extractor information should be separated into *syntactical* information, which is more or less fixed and can typically be provided upon extractor packaging by the developer, and *semantic* information describing what annotations are about, which is at least partially subjective, depending on the usage scenario, and will at least partially be revised continuously. Such information will often not be provided by the developer of an extractor, but by other developers providing or consuming relevant information, or by other actors, e.g. showcase admins. It will often be provided after extractor packaging, and does not require component adaptation - hence, it should not be communicated through and when packaging extractors.
- for that purpose, a new service for extractor registration and discovery is introduced, which will provide functionalities to store and retrieve extractor information, supporting both a REST API for providing extractor registration information, and a front-end for respective user interaction, which is more suitable to complement information that is not or cannot be known to an extractor developer at packaging time. It will use Marmotta for the extractor model storage, and workflow planning and execution can reuse this information for their purposes.
- information about the extractor output format (which may be RDF or a native format that requires conversion) should be communicated, and used by the broker to validate / control conversion; in order to locate specific data, LDPath could be used.
- the broker model should reuse existing information e.g. regarding syntactical types using existing ontologies as far as possible, however it should cache it in order to improve performance for related queries; wherever applicable, extractors and extractor versions, types etc. should be unique identified via URN

Related to **workflow planning and execution**, we came to the following conclusions:

- That Apache Camel would be a good choice for workflow execution via Camel routes, supporting many EIP, and that it should be complemented by components (to be developed by us) which simplify the task of retrieving information from the knowledge base to put it into Camel messages, in order to control the processing process based on rules, e.g. in order to invoke different paths within a route depending on languages detected. This feature allows for 'on-demand routing' within processing graphs and helps to avoid the need to specific many pipelines for the same task in such cases.

- Even if it was decided that the broker does not deal with managing scalability directly (but instead rely on the extractor nodes dealing with such aspects themselves, e.g. via parallelization and load balancing), information about whether an extractor is a singleton, and its resource dependencies, should be captured.
- Manual pipeline creation is a difficult process, due to the many constraints and interdependencies: It does not only depend on a 'syntactic match' between extractors, it also depends on semantic aspects (e.g. a face recognition cannot make sense of any image region, but only of image regions that represent faces, i.e. have a certain semantic meaning). Moreover, it heavily depends on the type of content, and the goal / use cases an annotation is used for. Considering this, it is extremely desirable to simplify the task of pipeline creation. While automatic workflow creation is not realistic considering these constraints, it is possible to provide a semi-automatic process that supports workflow creation, considering the various constraints.
- the broker needs to be able to define and track processing of content sets, as not all pipelines are to be invoked upon all content items.
- In order to support semi-automatic workflow creation, several data sources should be applied: Apart from syntactical types, and semantical types, the broker should also support storage and use 'feedback' from showcase admins on which extractors and pipelines worked well for which content set and use cases. All of these data sources will then provide useful information for workflow creation, but also about e.g. which extractor would only require support for an additional mime type in order to work in a pipeline etc.

2.9.2 Relevant User Stories

For the design of the MICO broker v3, we selected relevant user stories from [Lin+15], extending them considering the platform release and the evaluation phase:

- US-01: as a showcase admin, I want the MICO system to support the specific queries defined for my needs, so that our projects can be extended and improved
- US-02: as a showcase admin, I want the MICO system to support as many interesting queries as possibly for the given content, so that business and end-users can explore and enrich the content
- US-03: as a workflow creator/maintainer, I want the MICO system, to support me by route creation
- US-04: as a workflow creator/maintainer, I want the MICO system, to show me the current processing state
- US-05: as a extractor developer, I want to connect my local extractor to the MICO system, to extend the platform functionality.
- US-06: as a extractor developer, I want the MICO system, to call my extractor and store the result of extraction process
- US-07: as a extractor developer, I want to stop my locally running extractor
- US-08: as a MICO user, I want the MICO system, to analyze my a/v content(-set) and store/provide the results

- US-09: as a MICO user, I want the MICO system, to re-analyze my content(-set) with a new workflow (other extractors or extractor settings)
- US-10: as a MICO user, I want to cancel ongoing jobs if results are not satisfying to adapt extractor configurations
- US-11: as a MICO user, I want validate and rate the extractor results for my content set
- US-12: as a MICO user, I want the MICO system to analyze my content with specific extractors based on metadata provided by the platform

2.9.3 Relevant Components

The MICO broker design involves the following components in order to address the aforementioned user stories:

Extractor
The primary goal of an extractor is to produce/convert annotations of the input content in RDF format, to provide intermediate processing steps for other extractors, or to convert native output to RDF, as described in Section 2.1.

Extractor registration service
The main goal of the registrations service is support storage and retrieval / discovery of extractor information, focusing especially on the input and output - required data / formats, and provided data / annotations. In addition this service is also responsible for providing the extractors with correct connection parameters (e.g. the storage URI). More information about the registration service is provided in section 2.9.6.

Workflow planner
The workflow planner, which is described in more detail in section 2.9.7, is the responsible for the creation of a workflow, i.e. the composition of a complex processing chain of registered extractors that aims at a specific user need or use case.

Data store
The data store is responsible for persisting the input data provided by the users of the MICO system and to persist and retrieve annotations and the binary data produced by the extractors. A more detailed description is provided in section 2.9.4

Item injector The item injector is responsible for injecting items, e.g. binary data such as a video, an audio file or an image, as well as respective item sets into the system. As reported in section 2.9.8, the injector is both triggering the execution of the required workflow, and storing the information about the input data.

Workflow executor The workflow executor is responsible for conducting workflow processing, as requested by the user. It is responsible for triggering the appropriate extractors, providing them with the data they need, following the rules imposed by the workflow planner. More information about this is provided in section 2.9.8

Auxiliary component The auxiliary component is necessary to support dynamic routing within a workflow - i.e., it supports routing based on annotations / results within the workflow by getting data to the Data store and providing it within a route in order to make decisions about the further processing. Its specifications are provided in section 2.9.8

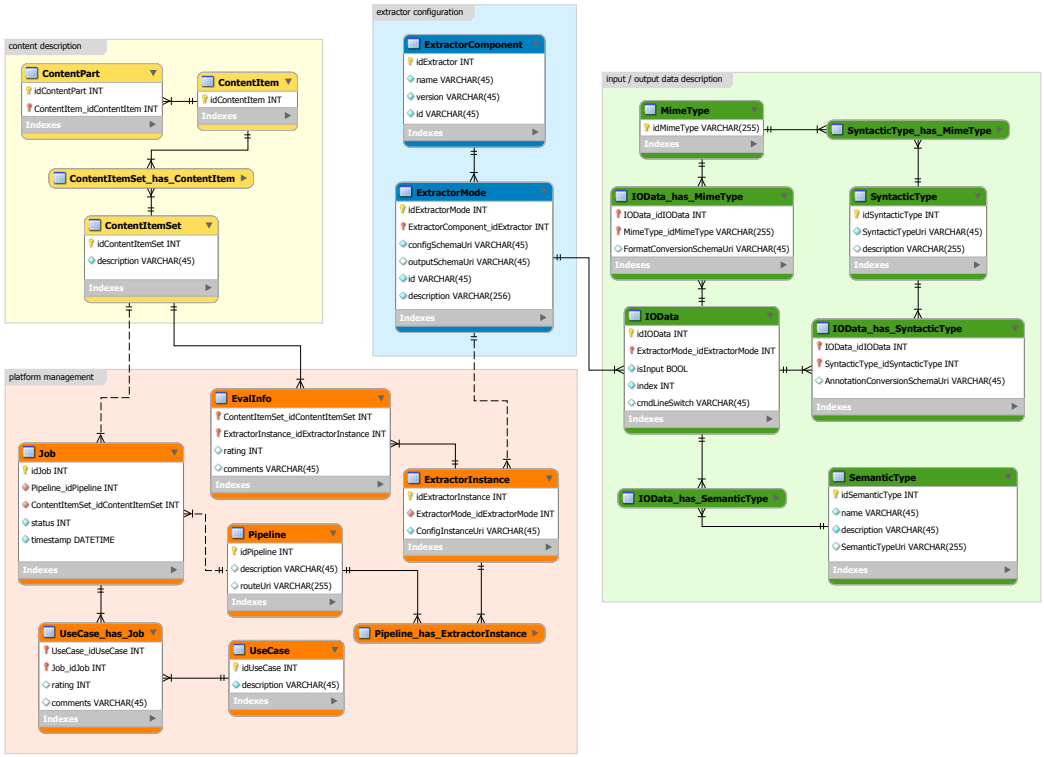
2.9.4 High-level data model

The data model of the MICO broker was designed in order to capture the key information needed to address the general principles outlined in section 2.9.1, and considering the key requirements from section 2.7 and use cases in section 2.9.5. The broker data model is using URIs as elementary stored data: As a result, the broker system can fully exploit the RDF annotations presented in section 3, and can support extractor registration, manual and semi-automatic workflow creation, and capturing of feedback related to previous annotation jobs (i.e. processing workflows applied to a defined content set). The model can be split into four interconnected domains:

1. **Content description**: This domain captures information about content that is aggregated and produced by the system
2. **Extractor configuration**: This domain captures information about possible configurations of the extractors which are registered
3. **I/O data description**: This domain captures information about the data required or produced by extractors, which provides the basis for pipeline creation
4. **Platform management**: This domain captures information about previous jobs e.g. from showcase administrators, thereby providing important feedback that can be reused for new or adapted workflows

An overview of the data model is provided in figure 5, with each layer being represented by a different color. The following paragraphs will describe the domains in more detail.

Figure 5 Data model of the MICO broker: Overview



2.9.4.1 Content description domain

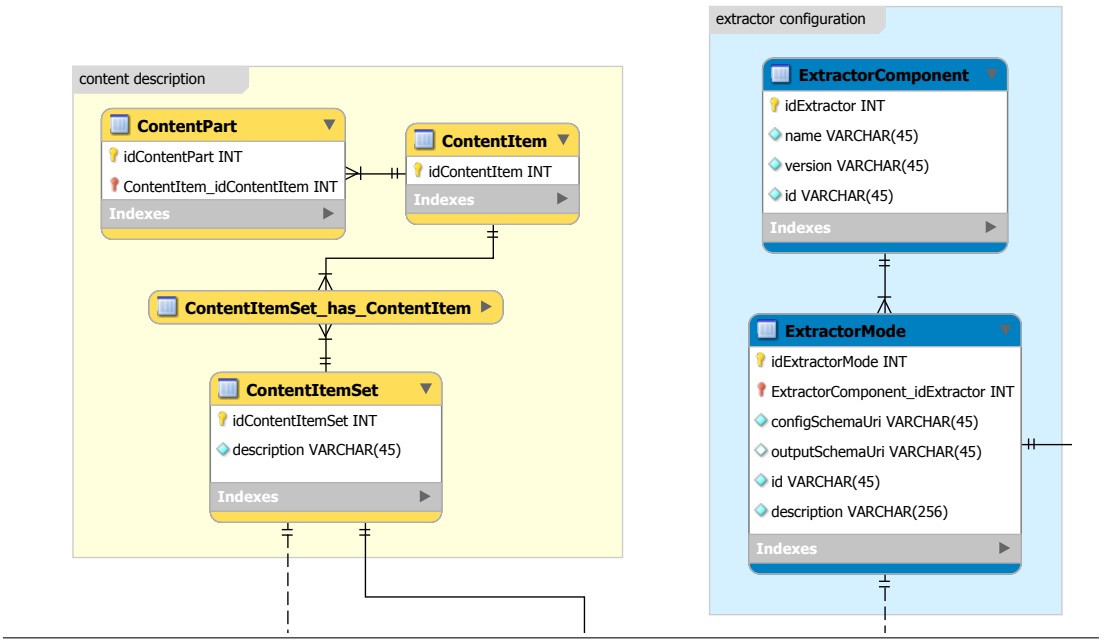
The content description domain, depicted in figure 6, consists of three main tables:

ContentItem
: captures information about the location (URI) of Content Items that have been stored within the system. As described in section 3, Content Items combine media resources and their respective analysis results.

ContentPart
: captures information about the location (URI) of Content Parts – i.e., media resources associated to a Content Item, as well as the output annotation produced by the extractors.

ContentItemSet
: allows grouping of several Content Items into one set. A pre-existing Content Set can be used, for instance, to run different pipelines on the same set, or to repeat the analysis with an updated extractor pipeline configuration.

Figure 6 Data model of the MICO broker: Content description and extractor configuration domains



2.9.4.2 Extractor configuration domain

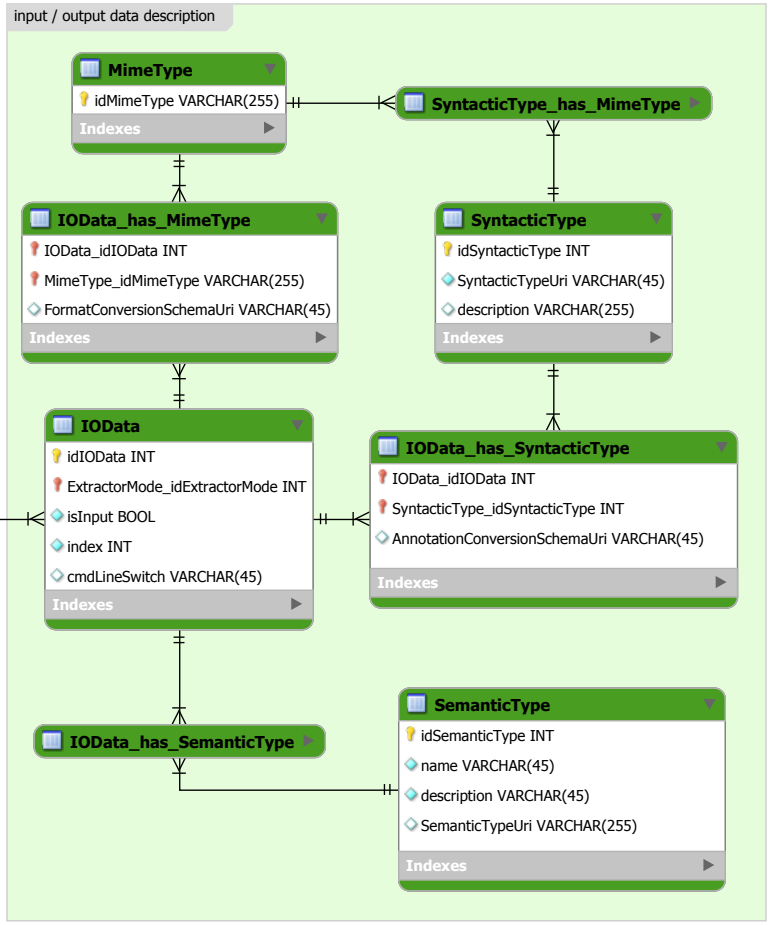
The extractor configuration domain, as depicted in figure 6, consists of two tables:

ExtractorComponent
: captures information about an Extractor Component registered within the MICO system.

ExtractorMode
: captures information about a concrete functionality (there can be 1..n functionalities per extractor), provided by an Extractor Component. In particular, it includes the URI of a configuration schema provided by the developer[31] and, for extractors creating annotations in a format different than RDF, the URI of the output schema.

[31] cft. extractor registration schema in section 2.9.6

Figure 7 Data model of the MICO broker: I/O data description



2.9.4.3 Input/Output data description domain

The input/output description domain is depicted in figure 7. This domain stores the core information necessary to validate, create and execute extractor pipelines and workflows, according to user selections/requirements.

IOData

represents the core data entity for the respective input or output to a given ExtractorMode. The optional field *cmdLineSwitch* can be used to control in which format a binary output, e.g. an image, is provided. For extractors requiring multiple inputs or providing multiple outputs, the relative index is stored as an attribute.

MimeType

captures the MIME type [32] of the I/O data. RDF data produced by extractors will be labeled as type "rdf/mico".

IOData_has_MimeType
is the table connecting I/O data to MimeType. The optional entry *FormatConversionSchemaURI* can be used to signal that an extractor is a 'helper' with the purpose of converting binary data from one format to another one (e.g. PNG images to JPEG), whenever such functionality is not directly supported by the extractor itself.

SyntacticType
is the syntactic type of the I/O data. For MICO extractors that produce RDF annotations, the stored URI should correspond to an RDF type, preferably to one of the types defined by the MICO Metadata model described in section 3. For binary data, this URI corresponds to a Dublin Core format [33]

IOData_has_SyntacticType
connects I/O data to SemanticType, and signals via the optional entry *AnnotationConversion-SchemaUri* if an extractor produce native XML annotations, which need to be converted to an RDF type.

SemanticType
captures high-level information about the semantic type associated with the I/OD data. It can be used e.g. by showcase administrators to quickly discover new or existing extractors that may be useful to them, even if the syntactical type is not (yet) compatible - this information can then be exploited to request an adaptation or conversion.

2.9.4.4 Platform management domain

ExtractorInstance
is the elementary unit storing the URI of a specific instance of an Extractor Mode, i.e. a configured extraction functionality available to the platform. The information stored in the URI includes e.g. the parameter and i/o data selection, according to the schema in section 2.9.6 and the information stored during the registration by the extractor itself.

EvalInfo
is information about the analysis performance of an Extractor Instance on a specific Content Item Set. This can be added by the developer to signal an appropriate input set to a showcase administrator, or by a showcase administrator to signal data sets for which his detector is working better or worse than expected.

Pipeline
captures the URI of the corresponding workflow configuration, i.e. the composition of Extractor Instances and respective parameter configuration.

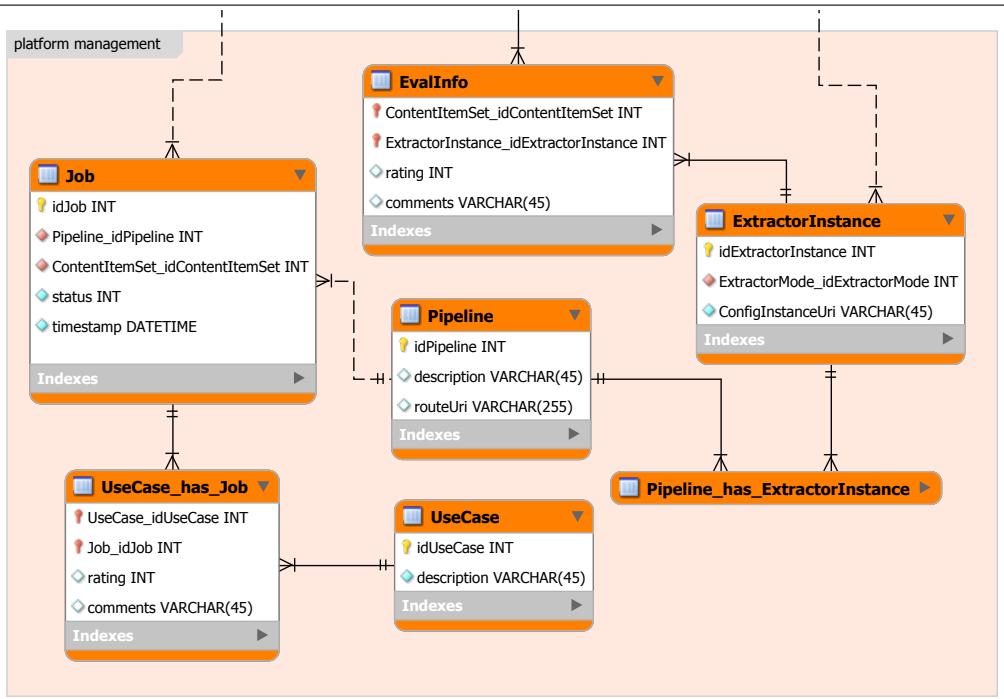
UseCase
is a high-level description of the goal that a user, e.g. showcase administrator, wants to achieve.

[32] `http://www.iana.org/assignments/media-types/media-types.xhtml`
[33] `http://dublincore.org/documents/dces/`

Job
: is a unique and easy-to-use entity that links a specific Pipeline to a specific Content Item Set. This can e.g. be used to verify the analysis status.

UseCase_has_Job
: is a table connecting a Use Case to a specific Job, which can be used to provide feedback, e.g. to rate how well a specific Pipeline has performed on a specific Content Item Set.

Figure 8 Data model of the MICO broker: Platform management



2.9.5 Use Cases

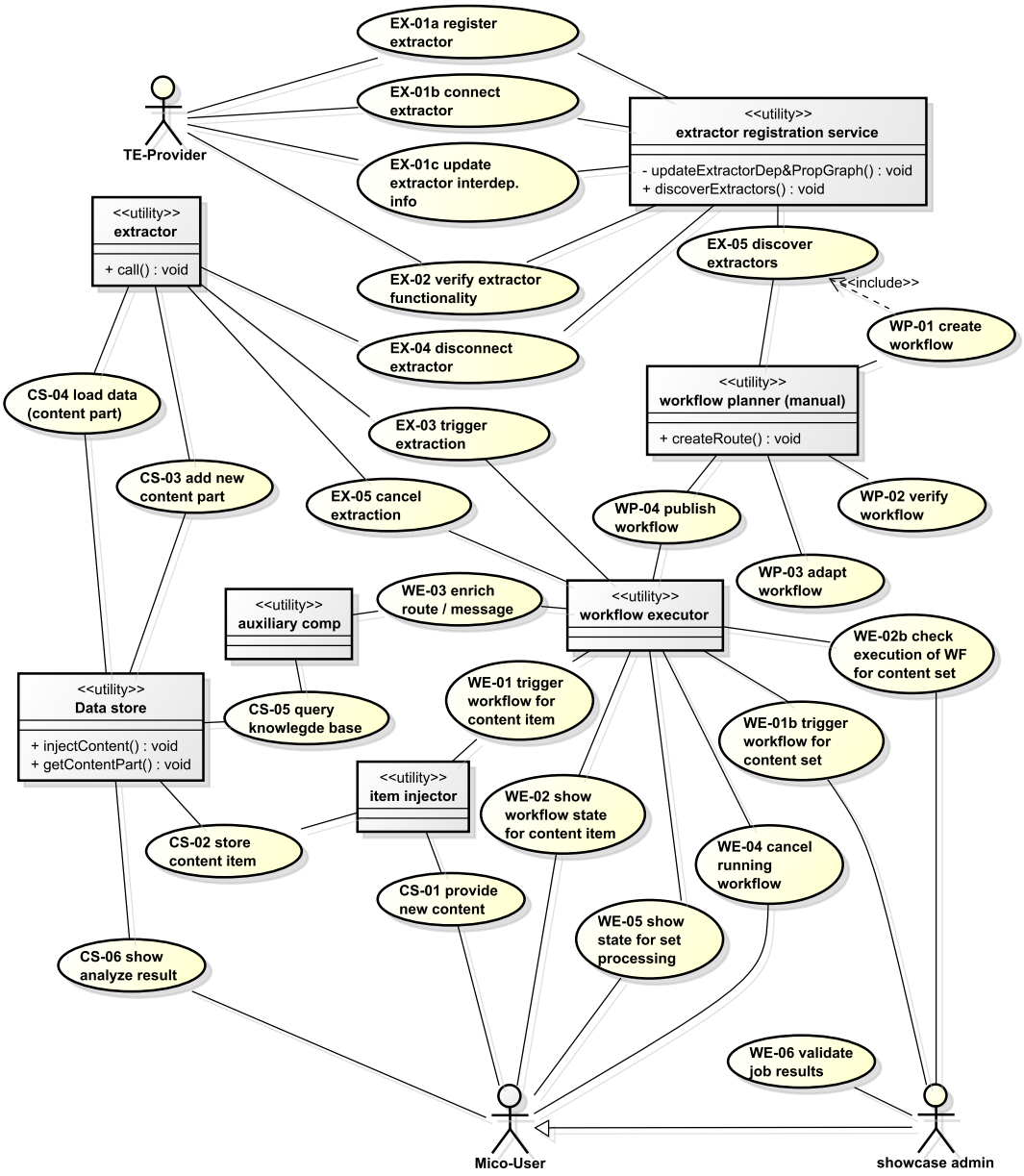
The following sections address the outcome of a refined analysis of the system requirements of the MICO broker v3:

- section 2.9.3 reports the set of components involved, with a brief overview over their main purposes and functionalities.
- section 2.9.5 presents the identified Use Cases in a tabular format.

figure 9 presents an overview of both use cases and system components.

The identified Use Cases (UC) are focusing on functional requirements, i.e. requirements that the MICO system needs to fulfill in order to adhere both to the general assumptions described in Section 2.9.1 and to the User Stories (US) reported in Section 2.9.2. Their description follows the following template:

Figure 9 Overview of the MICO Use Cases



ID:	*UC-XX (xx: unique increment number)*
Name:	*representative use case name (reflects involved component and user actions)*
Related US:	*parent user stories IDs*
Actors:	*relevant actors of the use case*
Description:	*brief description of the use case*
Pre-conditions:	*activities that must take place or conditions that must be given, before the use case can be initiated*
Post-conditions:	*state of the system after use case completion*
Normal Flow:	*detailed description of the interaction between actors (user, system, components), that will take place during execution of the use case under normal, expected conditions*
Alternative Flows:	*alternative interaction that can take place within this use case*

The rest of the section reports the details of every UC displayed in figure 9.

ID:	**UC- 01**
Name:	EX-01a register extractor
Related US:	US-05
Actors:	user, extractor, extractor registration service, triple store (Marmotta)
Description:	extractor registers his capabilities (consume, provide, settings, ...) at registration service
Pre-conditions:	registration service is running and accessible
Post-conditions:	registration service knows all possible configuration of extractor
Normal Flow:	1) extractor connects with registration service 2) extractor sends its configuration schema (consume, provides, settings ...) to service 3) service checks configuration data and stores it in triple store 4) service confirms registration process 5) extractor and service close connection

ID:	**UC- 02**
Name:	EX-01b connect extractor
Related US:	US-05
Actors:	user, extractor, extractor registration service
Description:	an extractor connects to the platform during initialization
Pre-conditions:	platform is running and accessible, necessary extractors are registered
Post-conditions:	the extractor is running and waits for extraction call from platform, platform knows how to call extractor
Normal Flow:	1) user starts extractor with specific configuration 2) extractor connects to platform and send current configuration 3) platform checks if configuration is available in registration store 4) platform confirms connection 5) extractor is ready and waits for call

ID:	**UC- 03**
Name:	EX-01c update extractor interdependency
Related US:	US-05
Actors:	platform user, extractor registration service
Description:	a platform user gathered information, that two extractors can be combined (one extractor can handle the output of the other one) and he wants to add that information to the platform and make it accessible for future route creations
Pre-conditions:	registration service is running and accessible, the extractors are registered
Post-conditions:	the updated dependency information are stored and can be used for future route creation
Normal Flow:	1) user connects to registration service 2) user sends updated dependency information 3) registration service updates information about extractors 4) registration service confirms update 5) user disconnects from registration service

ID:	**UC- 04**
Name:	EX-02 verify general extractor functionality
Related US:	US-05
Actors:	user, extractor, platform (broker)
Description:	after connecting a new extractor, it should be possible to test the general functionality by sending a sample extraction call and check/verify the generated output
Pre-conditions:	platform is running and accessible, necessary extractors are registered
Post-conditions:	the extractor has processed the sample call and the user has checked/verified the generated output
Normal Flow:	1) user selects sample call (if extractor supports several modes) and content 2) platform/broker triggers extractor (UC-EX-03 trigger extraction) 3) user verifies manually or automatically that the generated output looks like expected

ID:	**UC- 05**
Name:	EX-03 trigger extraction
Related US:	US-02, US-06, US-08
Actors:	extractor, platform (broker, storage)
Description:	during workflow process the broker has to call several extractors, this use case describes the call of one extractor
Pre-conditions:	platform is running and accessible, extractor is connected to platform, broker processes a workflow
Post-conditions:	the extractor has analyzed the content and uploaded the results
Normal Flow:	1) platform/broker triggers extraction process 2) extractor loads content part from storage 3) extractor analyzes content 4) extractor uploads results (content part and/or metadata) to storage 5) extractor informs broker that processing is finished

ID:	**UC- 06**
Name:	EX-04 disconnect running extractor
Related US:	US-07
Actors:	user, extractor, platform (broker)
Description:	User stops local running extractor, which itself disconnects from platform
Pre-conditions:	platform is running and accessible, necessary extractor is connected
Post-conditions:	platform is running, extractor is stopped
Normal Flow:	1) User signals stop to running extractor (e.q. press 'q' on cmd line) 2) extractor recognizes stop signal 3) extractor disconnects from platform 4) platform acknowledges disconnect 5) extractor process stops

ID:	**UC- 07**
Name:	EX-05 discover extractors
Related US:	US-03
Actors:	showcase admin, workflow planner, extractor registration service
Description:	sometimes it is necessary to get information about available extractors, e.g. for workflow creation to find and connect them to a pipeline
Pre-conditions:	platform registration service is running and available
Post-conditions:	registration service provided information about suitable extractors
Normal Flow:	1) During workflow creation the planer requests information about extractors from registration service 2) registration service analyzes request 3) registration service generates response 4) registration service sends response to requester

ID:	**UC- 08**
Name:	WP-01 create new workflow / camel route
Related US:	US-01, US-02, US-03
Actors:	workflow planer, extractor registration service
Description:	a workflow planer creates a new workflow based on the available extractors
Pre-conditions:	platform is running and accessible, necessary extractors are registered
Post-conditions:	the workflow planer has a new route definition, which is ready for publishing to workflow executor
Normal Flow:	1) workflow planer requests information about available extractors (UC-EX-05) 2) user arranges extractors and creates a complete workflow 3) (opt.) user triggers route with sample content (UC-WP-02) to verify route functionality

ID:	**UC- 09**
Name:	WP-02 verify workflow
Related US:	US-01, US-03
Actors:	workflow planer, broker
Description:	a workflow planer should verify a newly generated workflow by processing sample content
Pre-conditions:	platform is running and accessible, necessary extractors are registered
Post-conditions:	the platform ran a test workflow on the sample content
Normal Flow:	1) workflow planer selects workflow and sample content 2) broker triggers workflow execution

ID:	**UC- 10**
Name:	WP-03 adapt an existing workflow
Related US:	US-03
Actors:	user, workflow planer
Description:	sometimes an existing workflow needs to be adapted cause of newly available extractors etc.
Pre-conditions:	a workflow was created, platform is running and accessible
Post-conditions:	an adapted workflow is stored
Normal Flow:	1) user selects workflow to adapt from existing workflows 2) user adapts existing workflow within workflow planer 3) user publishes adapted/changed workflow (UC-WP-04)

ID:	**UC- 11**
Name:	WP-04 publish workflow
Related US:	US-03
Actors:	user, workflow planer, workflow executor
Description:	after creation of a new workflow with the planer it needs to be published to workflow executor, to be available for execution
Pre-conditions:	a workflow was created/changed, platform is running and accessible
Post-conditions:	an workflow is stored in workflow executor
Normal Flow:	1) user selects workflow to publish and the target workflow executor 2) workflow planer sends workflow to workflow executor 3) workflow executor confirms receiving

ID:	**UC- 12**
Name:	WE-01a trigger job for content item
Related US:	US-08, US-09
Actors:	mico user, workflow executor
Description:	a mico user wants to analyze content with a defined workflow and therefore starts a new job
Pre-conditions:	platform and storage service is running and accessible, necessary extractors are connected
Post-conditions:	a workflow was triggered for a content item
Normal Flow:	1) user selects content item to analyze 2) user selects appropriate workflow for his needs 3) workflow executor provides job reference to the user 4) workflow executor starts workflow (triggers first extractor ...) 5) workflow executor signals process complete

ID:	**UC- 13**
Name:	WE-01b trigger job for content set
Related US:	US-08, US-09
Actors:	mico user, workflow executor
Description:	a mico user wants to analyze content set with a defined workflow and therefore starts a new job
Pre-conditions:	platform and storage service is running and accessible, necessary extractors are connected
Post-conditions:	a workflow was triggered for a content set
Normal Flow:	1) user selects content set to analyze 2) user selects appropriate work flow for his needs 3) workflow executor provides job reference to the user 4) workflow executor starts work flow (triggers first extractor ...) 5) workflow executor triggers flow for all items in the set

ID:	**UC- 14**
Name:	WE-02a show job state for content item
Related US:	US-04, US-09
Actors:	mico user, workflow executor
Description:	mico user wants to know if processing is still ongoing or has finished (with success or error)
Pre-conditions:	platform and storage service is running and accessible
Post-conditions:	the user got information about a job
Normal Flow:	1) user selects content item to analyze 2) user selects appropriate workflow for his needs 3) workflow executor starts workflow (triggers first extractor ...) 4) workflow executor signals process complete

ID:	**UC- 15**
Name:	WE-02b show job state for content set
Related US:	US-04, US-09
Actors:	mico user, workflow executor
Description:	mico user wants to know if processing is still ongoing or has finished (with success or error)
Pre-conditions:	platform and storage service is running and accessible
Post-conditions:	the user got information about a job
Normal Flow:	1) user selects content item to analyze 2) user selects appropriate workflow for his needs 3) workflow executor starts workflow (triggers first extractor ...) 4) workflow executor signals process complete

ID:	**UC- 16**
Name:	WE-03 enrich route / message
Related US:	US-12
Actors:	workflow executor, auxiliary component
Description:	sometimes the auxiliary component needs to load extractor results which are necessary for decisions during dynamic routing
Pre-conditions:	a workflow with dynamic parts and the auxiliary component is in progress
Post-conditions:	the auxiliary component has added the information to the route message, so that a dynamic router can use it to make decisions
Normal Flow:	1) auxiliary component loads data from triple store 2) auxiliary component transforms data into a format that the dynamic router can consume 3) auxiliary component puts information in route message

ID:	**UC- 17**
Name:	WE-04 cancel running workflow
Related US:	US-10
Actors:	showcase admin, workflow executor
Description:	sometimes it is necessary to abort a job, e.g. when showcase admin notices a wrong extractor configuration
Pre-conditions:	a workflow is in progress
Post-conditions:	a workflow is canceled, and no new extractors are triggered
Normal Flow:	1) showcase admin connects to workflow executor 2) workflow executor lists active workflows 3) showcase admin selects workflow to stop 4) workflow executor stops selected workflow

ID:	**UC- 18**
Name:	WE-05 show state for content set processing
Related US:	US-09
Actors:	mico user, workflow executor
Description:	a mico user wants to know the how many files of a set are analyzed/processed by a workflow
Pre-conditions:	platform and storage service is running and accessible, necessary extractors are connected
Post-conditions:	broker displays progress information to the user
Normal Flow:	1) user connects to monitoring utility of workflow executor 2) user opens progress monitor 3) user selects content set and pipeline 4) system shows progress state

ID:	**UC- 19**
Name:	WE-06 validate extractor results
Related US:	US-11
Actors:	mico user, registration service
Description:	after finishing a job the called user should be able to rate the results of an extractor for the processed content set
Pre-conditions:	at least one content item of a set was processed by an extractor of a workflow produced some results
Post-conditions:	extractor metadata is updated by user feedback and can be used for upcoming workflow planning
Normal Flow:	1) user connects to registration service 2) user selects job and/or extractor to evaluate 3) user provides feedback 4) registration service updates extractor metadata

ID:	**UC- 20**
Name:	CS-01a provide new content (via cli)
Related US:	US-08
Actors:	mico-user, item-injector, data-storage
Description:	a mico user wants to upload / store new content in mico platform
Pre-conditions:	platform and storage service is running and accessible
Post-conditions:	a new content item with at least one content part is created
Normal Flow:	1) mico user provides storage access information to injector 2) user provides new data to injector 3) injector connects to storage 4) injector requests location (id) for new item from storage 5) data-storage provides item id 6) item injector transfers data to storage

ID:	**UC- 21**
Name:	CS-01b provide new content (via broker)
Related US:	US-08
Actors:	item injector, data store, mico user
Description:	add new content to the platform
Pre-conditions:	platform storage is up and running
Post-conditions:	the new content is stored in platform storage
Normal Flow:	1) mico user connects to broker 2) mico user opens upload view 3) mico user selects file(s) to upload 4) item injector stores file(s) in data store

ID:	**UC- 22**
Name:	CS-02 store content item
Related US:	US-08
Actors:	item injector, data store
Description:	add new content to the platform
Pre-conditions:	platform storage is up and running, CS-01 was invoked
Post-conditions:	the new content is stored in platform storage
Normal Flow:	1) inject tool creates new content item in storage 2) inject-tool upload binary data to platform storage 3) inject tool updates meta data in triple store

ID:	**UC- 23**
Name:	CS-03 add new content part
Related US:	US-08
Actors:	extractor, data store, triple store
Description:	an extractor add a new content part to an item during extraction process
Pre-conditions:	an extractor is processing a content
Post-conditions:	a new content part and its meta data is stored
Normal Flow:	1) during analyze process the extractor produces or extracts new binary data 2) the extractor uploads binary data to platform storage 3) extractor send meta data to triple store

ID:	**UC- 24**
Name:	CS-04 load data
Related US:	US-08
Actors:	extractor, data store
Description:	an extractor loads data (binary content) from storage to analyze it
Pre-conditions:	the content exists in data store
Post-conditions:	the extractor has copied the binary content to a local storage
Normal Flow:	1) extractor gets triggered 2) extractor opens connection to storage 3) extractor request data from storage 4) extractor stores data on local storage

ID:	**UC- 25**
Name:	CS-05 query knowledge base
Related US:	US-08
Actors:	triple store, auxiliary component
Description:	auxiliary camel component loads meta data from triple store by LD-Path
Pre-conditions:	meta data exists in triple store
Post-conditions:	auxiliary component retrieved data to process it
Normal Flow:	1) auxiliary component gets triggered with LD-Path 2) auxiliary component sends LD-Path request to triple store 3) triple store responses data selected by LD-Path 4) auxiliary component can further process received data

ID:	**UC- 26**
Name:	CS-06 show analyze result
Related US:	US-09
Actors:	broker, triple store, mico user
Description:	a mico user inspects analyze results
Pre-conditions:	content was uploaded to the platform
Post-conditions:	platform delivered meta data to user
Normal Flow:	1) user connects to platform 2) user selects content item to inspect 3) broker requests meta data from triple store 4) broker shows data to the user

2.9.6 Registration Service

As outlined in 2.9.1, one key assumption for the broker is that some extractor information is provided at packaging time by the developer (extractor properties, input and output), while other extractor information will typically be provided after packaging time, by other developers, showcase administrators etc. (semantic mapping of extractors, and information / feedback about pipeline performance). This also implies a separation into *syntactical* information, which is more or less fixed, and *semantic* information, which is at least partially subjective and not fixed.

The registration service represents a new component, meant to provide one central point to register and query extractor information, considering the aforementioned needs. Details about the respective extractor / broker data model and registration process have been described in section 2.9.4. As outlined in previous sections, Marmotta will be used by the broker for the broker / extraction model storage. Workflow planning and execution can then reuse this information, and sections 2.9.7 and 2.9.8 will refer to that.

The registration service will support both a REST API for providing extractor registration information, and a front-end for respective user interaction, to complement information that is not or cannot be provided by an extractor developer at packaging time, including feedback on how well certain pipelines or extractors performed for content sets.

For the purpose of this document and implementation work in y3, it is especially important to agree on the registration information that will be provided by developers, as this is includes a lot of information, and respective changes will create significant effort for developers. Hence, the following focuses on the schema to be used for extractor registration via the REST API, now provided as an XML Schema Definition (XSD). Alternatively, the same information can also be provided as JSON data to the service.

The first entities that are specified in the XSD are the possible types of input and output data, and of extractor parameters, as shown in figure 10

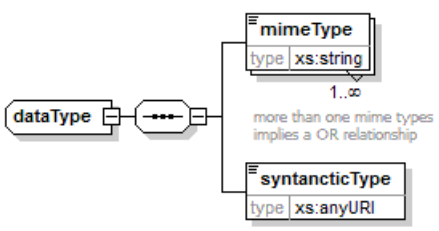
The types *dataType* and *semanticType* reflect the data model design from section 2.9.4. *primitiveType*, on the other hand, defines a closed set of possible primitive types, which are then used / combined by extractors to communicate how exactly to interpret the information.

On a high level perspective, as depicted in figure 11, every extractor is providing basic information about the component, including *name*, *version* number and *id*.

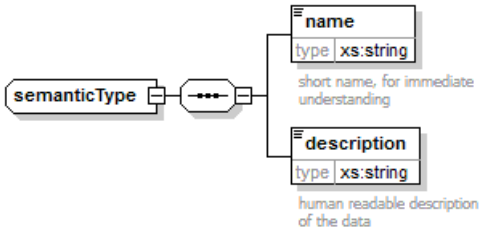
Moreover, one or more *modes* need to be specified, capturing a specific behavior and functionalities of a component (there can be several modes to one component), which may include a pre-defined configuration provided by the developer. Every mode needs to provide the broker with:

1. a unique *id*

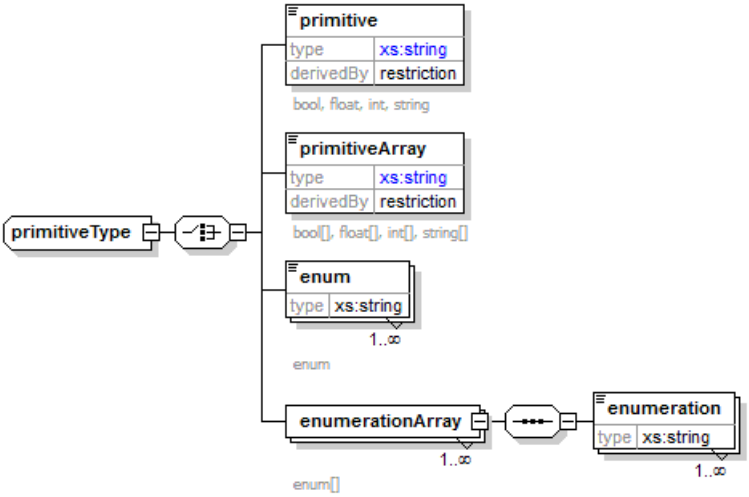
Figure 10 Broker registration XSD: Basic types



(a) dataType



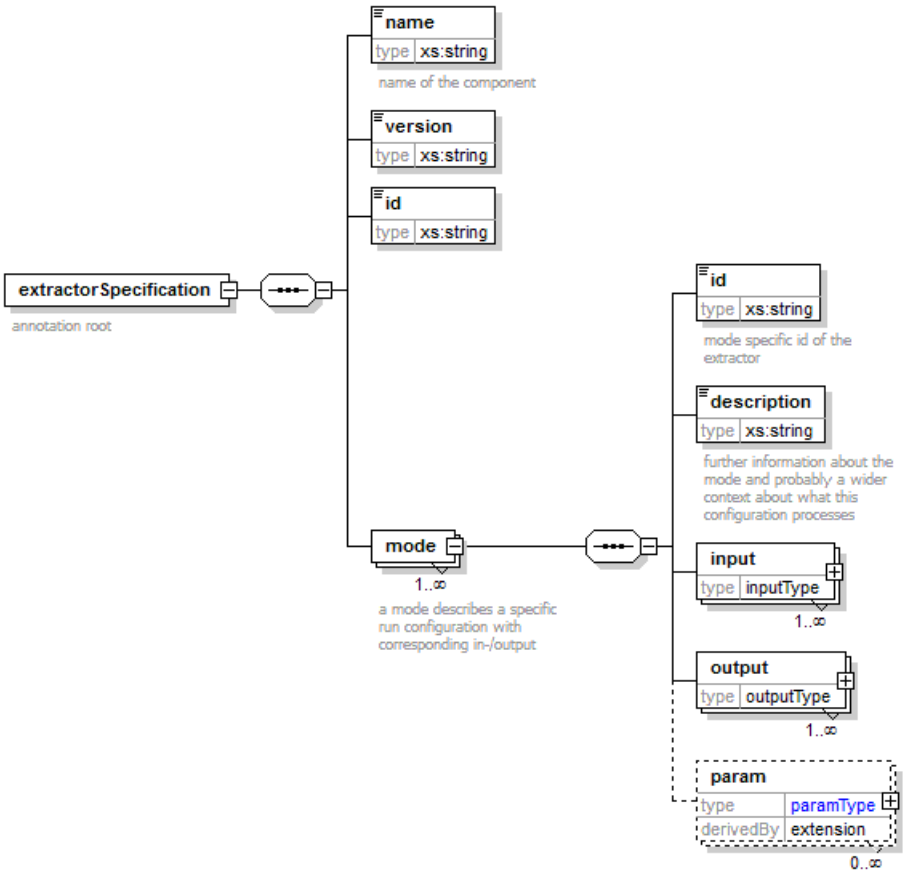
(b) semanticType



(c) primitiveType

2. a *description*
3. a non-empty list of required *input* - all inputs need to be provided at once to the extractor.
4. a non-empty list of required *output* - all outputs are generated by the extractor.
5. (opt.) a non-empty list of *parameters*, that the user or showcase administrator can select at startup, when creating a pipeline.

Figure 11 Broker registration XSD: Extractor Specification overview



As depicted in figure 12, every input data must define a specific *semanticType* and a specific *dataType*. By default, if many MIME types are listed inside the *dataType* entry [34], the input can be represented in any of these format. The *cmdLineSwitch* attribute, if present, allows the broker to force the extractor to work only with the specified one.

For extractor modes, it is possible to pre-filter the input depending on its annotated properties, using one or more *constraints*. Each constraint is defined by a *name*, a non-empty list of *allowedValues*, an *allowedType* that is specifying how to parse the allowed values and the constraints, and the *dataLocation* – i.e., an LDpath – where the Broker can retrieve the necessary information for the pre-filtering.

[34] cfr. Figure 10(a)

Figure 12 Broker registration XSD: Input data definition

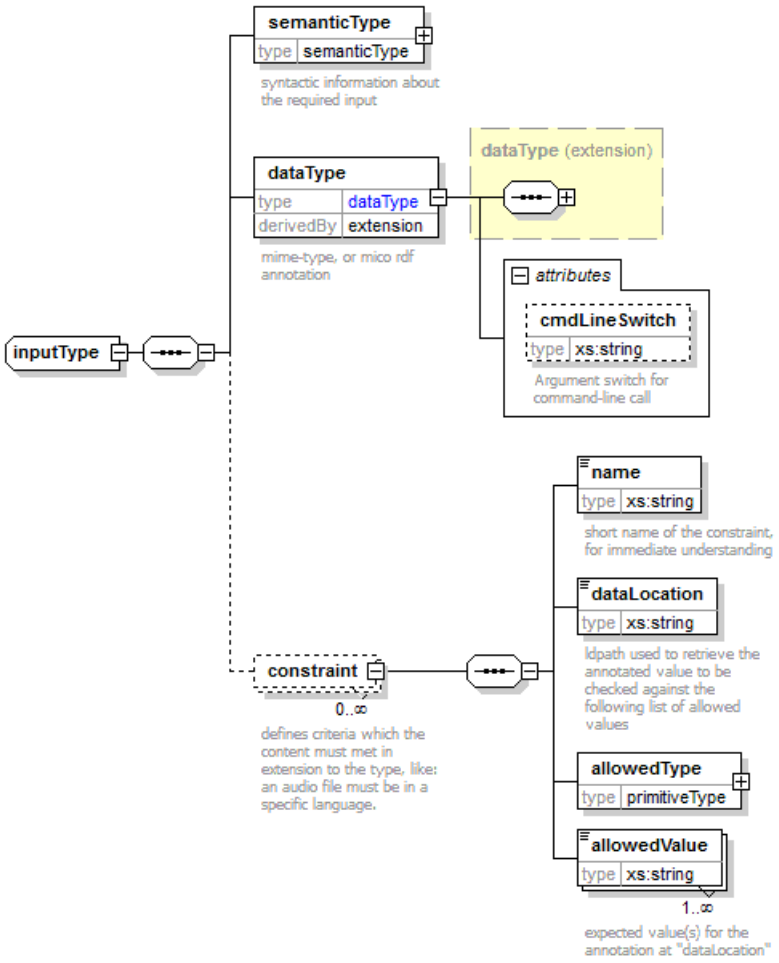
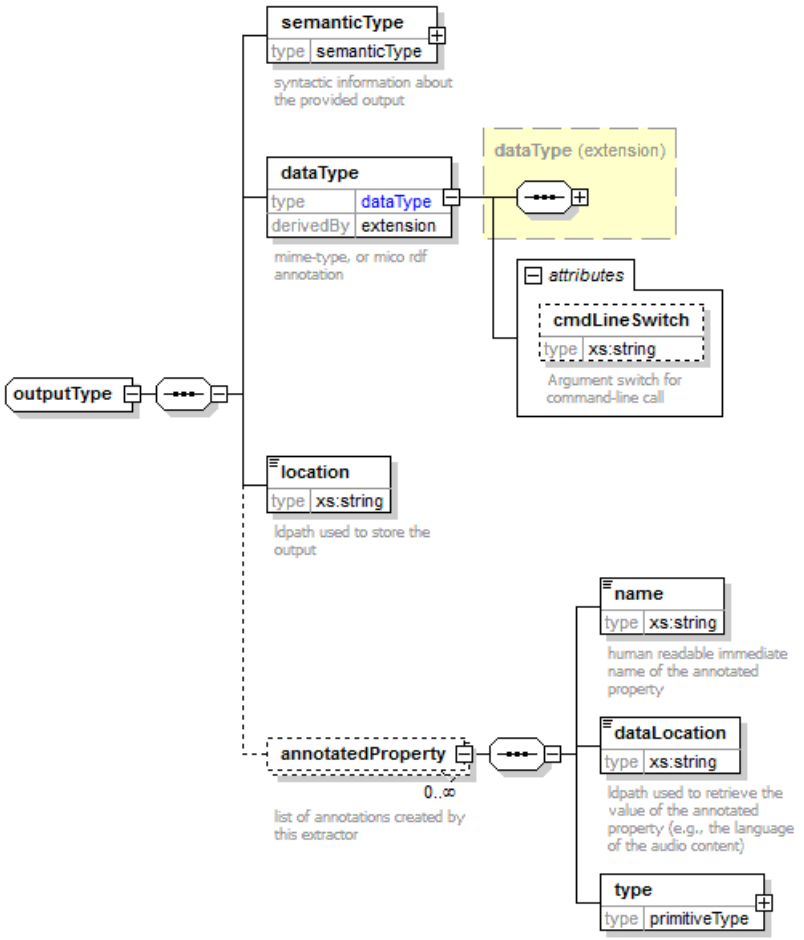


Figure 13 provides the definition of output data per extractor mode. Every output data has to define a specific *semanticType*, a specific *dataType* and a *location* – expressed as LD-path – reporting where exactly data is stored. By default, if many MIME types are listed inside the *dataType* entry, the output will be replicated for every entry in the list. The *cmdLineSwitch* attribute, if present, allows the broker to force the extractor to produce an output *only* for the selected format(s).

If the extractor mode is also annotating some specific properties, every *annotatedProperty* has to be specified in terms of its *name*, *type* and *dataLocation*, which once again is expressed as LDpath.

Figure 13 Broker registration XSD: Output data definition



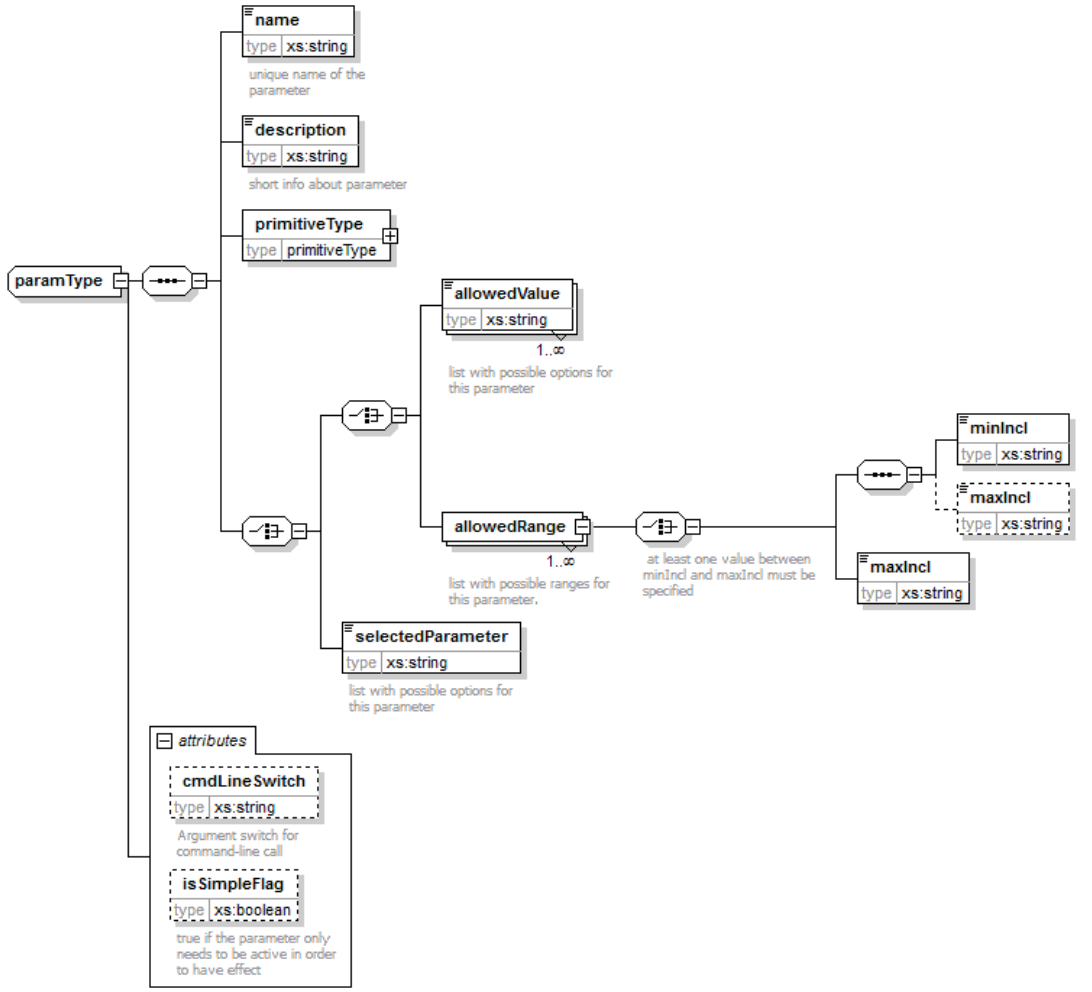
User-configurable parameters of the extractors are modeled as described in figure 14. During the registration, a parameter *must* include

1. a *name*
2. a *description*
3. a *primitiveType*, so that the broker may filter out illegal parameters
4. one between
   (a) a non-empty list of *allowed values*
   (b) a non-empty list of *allowed ranges*. an AllowedRange is specified by setting at least one between *maxIncl* and *minIncl*
5. a defined *cmdLineSwitch* attribute, that allows the broker to start the extractor correctly

6. a defined *isSimpleFlag* attribute, that informs the broker about the parameter being a simple switch or not

If the same XSD is used as a model to persist the information about an ExtractorInstance as defined in section 2.9.4, then instead of choosing a list of allowed values or ranges, is it possible to omit the attributes and to store the value of the parameter in the element *selectedParameter*.

Figure 14 Broker registration XSD: User-configurable parameter definition



2.9.7 Workflow planning and creation

As outlined in previous sections, one of the key goals of broker v3 is to support the process of creating new pipelines, using the aforementioned information from the broker model and a respective front-end. It is not realistic that this will work in a fully automatic manner: Pipeline and extractor performance always depend the context / use case and content, hence it can only be a semi-automatic approach. However, even a semi-automatic workflow can provide huge usability benefits, which may very well be one of the key elements for the cross-media vision of the project.

In order to support workflow creation, the service will exploit several 'information pillars', including *mimeType*, *syntacticType*, *semanticType*, and feedback on how well existing pipelines and extractors performed on content sets, as outlined in the previous sections 2.9.1, 2.9.4 and 2.9.6. In essence, it will use these 'pillars' to check compatibility of extractors on several levels and find possible matches, but it is important to note that the 'pillars' do not represent a simple hierarchy. Instead, they can be used to support the creation of pipelines in a variety of ways. For instance, an indication of extractors that have matching *mimeType* and *syntacticType*, but not a matching *semanticType* can be used to signal to the service which extractors could match and hence should be 'linked' via a new *semanticType*. Vice versa, if it turns out that e.g. to extractors seem to provide similar output, as signaled by *syntacticType* and *semanticType*, but the *mimeType* does not fit, this can be treated as indicated that a simple extension of the extractor to support a new *mimeType*, e.g. via format conversion, could do the trick.

For instance, broker v3 could be used for semi-automatic workflow creation for video shot detection: Using the broker v3 front-end, the user could search for extractors that provide locations (timestamps) for interesting points inside a video. One possible entry in the result list the system may show the TVS-extractor (section 2.2.2) with a configuration for shot detection. After selection of the TVS extractor, the workflow planner would then check the input dependencies of that component, query possible extractors that provide output data in the required format, and show them to the user. The user can then browse the list and select one or several suitable extractors to combine them as part of a new workflow. This step can be repeated several times, thereby completing the workflow creation. Once finished, the new pipeline can be stored, to be used for execution jobs or possible later reuse and extension by the same or other system users.

2.9.8 Workflow execution

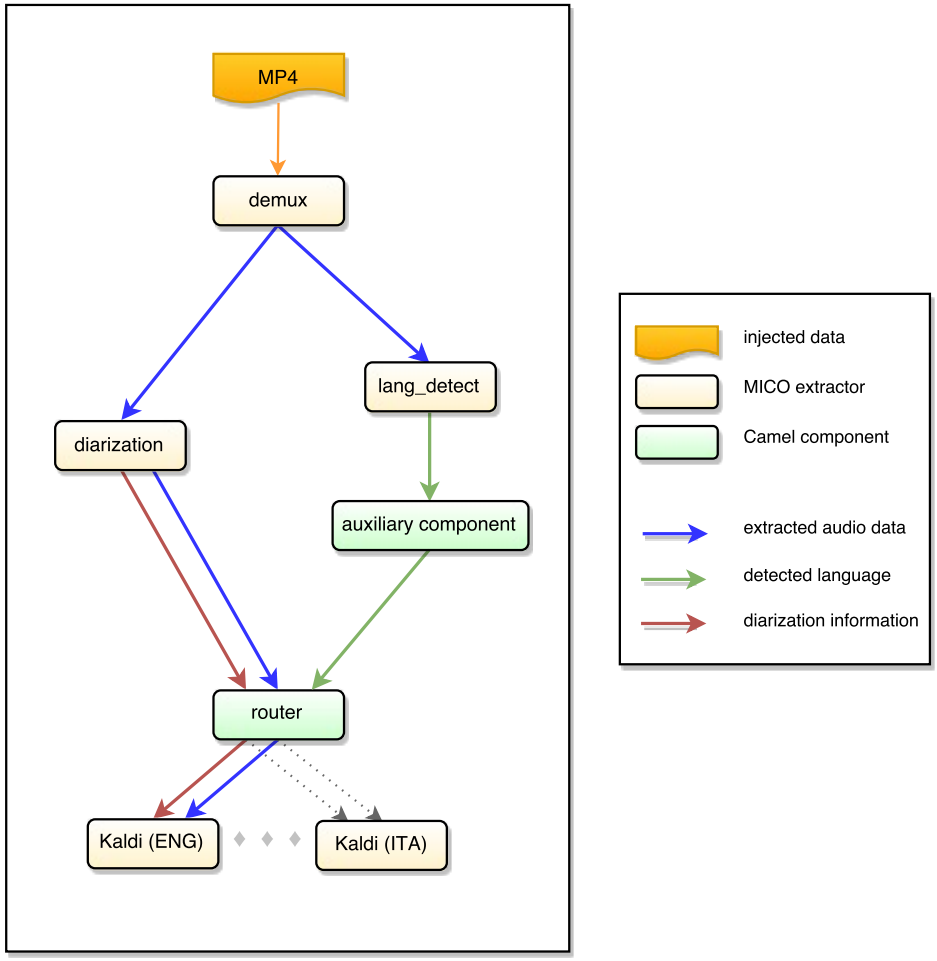
Workflow execution involves four main components:

- The *workflow executor* as core component, which orchestrates and uses the other components and is based on the Apache Camel framework [Fou04].
- The *RabbitMQ* message broker, which is used as communication layer to loosely couple extractors and the MICO platform [PS04].
- The *micoRabbitComp*, which is a Camel endpoint component that connects the Camel framework with the MICO platform, and triggers the extractors via RabbitMQ.
- The *auxiliary component*m which is also a MICO-specific extension to Camel that allows extraction of information from Marmotta to support dynamic routing based on information provided by extractors of the MICO platform.

Workflow / route execution and related component interaction is illustrated in the following, using a sample pipeline as example, which is about extracting spoken words from a video. The workflow

involves several extractors, the auxiliary component, and a dynamic router to complete the task. The connections between components and their execution order is depicted in figure 15.

Figure 15 Data model of the MICO broker: Platform management



The first extractor *audio demux* extracts the audio stream from an injected video file and stores it within platform storage for further processing. The next two extractors are *diarization* and *language detection*, which are analyzing the audio content in parallel. Finally, the *diarization* extractor annotates information about the speaker and tries to detect start and end points of sentences, and language detection is used to identify the spoken language, storing it in the knowledge base. Afterwards, a Camel component loads that data from storage, and puts it to the Camel message, before a Camel component for dynamic routing checks the language information and triggers a *Kaldi* extractor that is configured with the language model according to the detected language.

3 Specifications and Models for Cross-media Publishing

3.1 Introduction

The MICO platform is an environment that allows to break up the hidden semantics of media in context by orchestrating sets of different components that jointly analyse content, each adding its bit of additional information to the final result. Instead of publishing the results in proprietary or varying formats, the platform relies on the Resource Description Framework (RDF) as a solid foundation for a unified persistence layer, which allows semantic interlinking on the level of single resources and comprehensive querying with SPARQL.

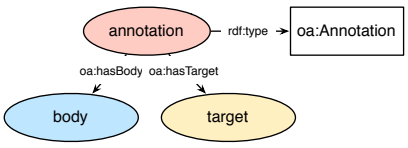
All results that are produced by the orchestrated extractors (intermediate as well as final results) are made available following the MICO metadata model vocabulary (introduced in Dx.2.1). This section covers the refinement of the Model in year 2 of the project. Version 2 of the MICO Metadata model is based on lessons learned of existing extractors and performed pipelines of the MICO platform. As recapitulation, subsection 3.2 presents a short summary of important terms in the model context. Subsection 3.3 will highlight all important changes of the new MICO Metadata Model in contrast to version 1.0. Subsection 3.4 contains the detailed specification and documentation of the refined vocabulary. In order to support developers with a lack of expertise in Semantic Web technologies, subsection 3.5 presents Anno4j[35], a Java library to read and write the MICO Metadata Model.

3.2 Recapitulation

The main workflow implemented by the framework is the analysis of media information units, in this context called **(Content) Items**. **(Content) Items** are representations of media resources together with their subsequent analysis results, called **(Content Parts)**. **(Content) Parts** are results of analysis components with different media types that are directly related to the same **(Content) Item**. In other words, a **(Content) Item** is a semantic grouping of information objects (**Content Parts**) considering a specific multimedia asset. As extraction and analysis processes in general produce a manifold of different results and output formats, unified ways of storing and querying the content and its context are required. To enable comprehensive cross-media querying, the underlying data model must be capable to reflect the performed workflow chain including the interaction and mutual dependencies between all analysis steps. Metadata as well as provenance can be persisted in close relationship to the multimedia content. Our existing model basically builds upon the Web Annotation Data Model (WADM, former Open Annotation Data Model, see D3.1.1) and several existing ontologies like Dublin Core, Media Fragments PROV-O, and FOAF.

The concept of (web) annotations is adapted and extended to the cross-media environment. The baseline consists of three concepts, namely **annotation**, **body**, and **target** (see figure 16).

Figure 16 Basic annotation in the MICO data model

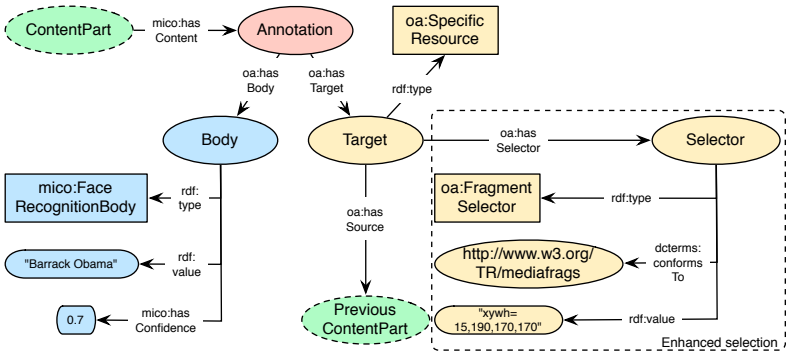


[35] https://github.com/anno4j/anno4j

The **body** contains ways of describing and classifying the results or outcomes of the extractors. In general, every extractor has its own output type (e.g. `mico:FaceRecognitionBody` or `mico:NERBody`). The **target** describes what the body is about and usually refers to the input multimedia asset or previous **(Content) Part** and allows the creation of a traceable workflow chain. The **target** can be extended with more precise selections to only refer to a temporal or spatial fragment of a multimedia asset (e.g. a frame of a video). The actual annotation object connects the body and the target and also can be enriched with context and provenance information about the extraction workflow (e.g. timestamps, creator and configuration).

As an overall example, figure 17 shows the results of an extractor that runs a face recognition algorithm. Therefore, its body is typed as an instance of the class mico:FaceRecognitionBody. The content of the extraction result contains a name (corresponding to the person that is recognised) and a confidence value (which reflects how sure the extractor is about its findings). The target of the annotation refers to the input of the extraction process. In this case, the target links to its preceding content part, which itself contains information about the input picture. As the face recognition algorithm also supports the coordinates of the located face, a link to the sole media item is not enough. For this purpose, a selection refines the target to specify the selected spatial fragment of the picture.

Figure 17 Face Recognition example



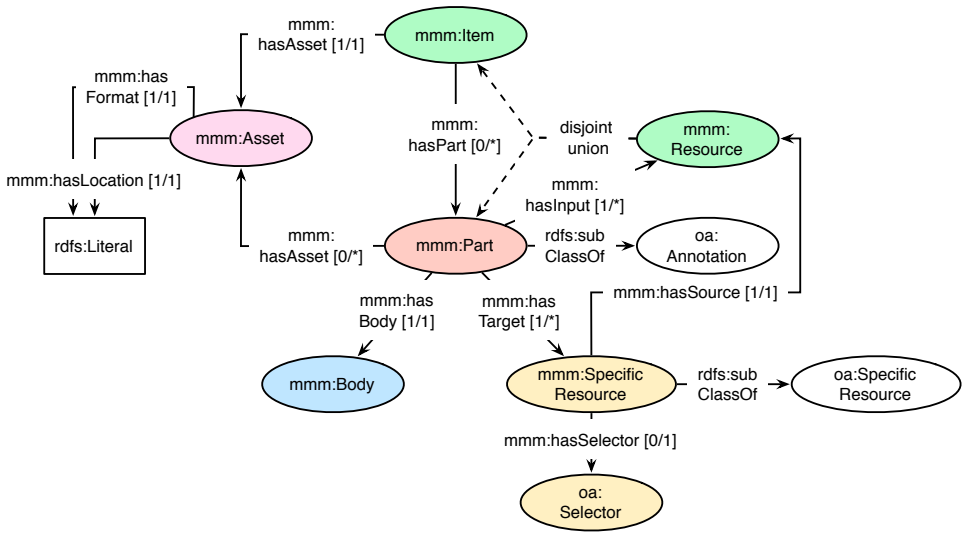
3.3 Major Changes

The second iteration of the MICO Metadata model is based on lessons learned and includes basic restructuring, renaming, and refinement of the semantics of different components. Furthermore stricter multiplicities of the relationships between components were introduced to allow a well-defined querying. The following list describes a summarised overview over the major changes and explanations of the refined model, which are also illustrated in figure 18.

Namespace changes Due to upcoming new MICO vocabulary (e.g. broker model vocabulary), the namespace and abbreviation for the MICO Metadata Model `mico` - `http://www.mico-project.eu/ns/platform/1.0/schema#` was changed to `mmm` - `http://www.mico-project.eu/ns/mmm/2.0/schema#`.

Namespace for Extractor Bodies To split and distinguish between schema vocabulary and Use-Case specific vocabulary a second vocabulary, called MICO Metadata Model Terms `mmmterms` (`http:`

Figure 18 MICO Metedata Model Version 2.0



`//www.mico-project.eu/ns/mmmterms/2.0/schema#`) was created. It includes application-related body and selector classes which are used in MICO Use-Cases.

Renaming of ContentItem and ContentPart `mico:ContentItem` and `mico:ContentPart` are renamed to `mmm:Item` and `mmm:Part` to avoid semantic misunderstandings that they always are connected to binary multimedia content. `mmm:Parts` can solely consist of metadata annotations without a binary representation (e.g. multimedia file).

MMM versions of OA vocabularies The `mmm` introduces extended versions of selected Open Annotation classes and predicates to modify multiplicities, domains or ranges (e.g. `mmm:SpecificResource`, `mmm:Body`, `mmm:hasSource`).

Merging `ContentPart` with `Annotation` The first version of the model defined that a `mico:ContentPart` has exactly one `oa:Annotation` because of deprecated requirements. The refined model merges both concepts which results in the `mmm:Part` class which extends the `oa:Annotation`. Hence, `mmm:Part` is a direct counterpart of an Annotation in the MICO context.

Initial multimedia asset is moved into the `Item` In the previous model, the initial multimedia asset (usually the multimedia upload) was defined as a special ContentPart which didn't denote the actual relevance of the content in relation to the overall Content Item. Content Items represent media resources together with their subsequent analysis results. Content Parts are therefore only results of analysis components. As a result the refined model integrates the initial multimedia asset directly on the level on the Item and not as a standalone Part. All subsequent Parts then semantically refer to the same overall Item with directly connected multimedia asset.

Multimedia assets on Part level The location information of a binary file output, formerly presented in the SpecificResource/Target of an Annotation is now included at the level of Parts. Semantically,

these multimedia assets denote a representation of the whole annotation. (e.g a cropped image for a face detection annotation or a xml file for a low-level feature).

Modelling of a Multimedia asset Multiple multimedia assets on the level of Item (e.g. initial upload) or Part (binary representations of extractor results) are modelled with multiple `mmm:hasAsset` relationships instead of the former `mico:hasLocation` relationship. A `mmm:Asset` can be defined with a format (`dc:format`) and the location of the binary asset (`mmm:hasLocation`).

Introduction of mmm:Resource Because of the modification that initial multimedia assets are moved to the Item, the selection of a annotation has to be able to either refer to a Item or to a Part. Therefore mmm:Resource denotes the distinct union of Item and Part instances and is used to limit the range of the mmm:hasSource predicate.

Input and Target Provenance The former model had inconsistence in specifying the difference of input and target of an annotation. The target side was misused to either use it to denote the input Content Part of an analysis (e.g. input XML file) or to denote the real target of an annotation (e.g. input image where a face was detected). The new model allows a clear separation between both. While multiple inputs now reside on the Part level with `mmm:hasInput` (also a preparation for an extensive broker model), the target of an annotation clearly denotes what the annotation is about.

Extractor Provenance As preparation for the broker model, which will also lead to an RDF specification in the future, some features are already included in the new model. A `mmm:Part` will link to the instance that is responsible for its creation via the relationship `oa:annotatedBy`. In this case, the "instance" is a mode of a given extractor. An extractor has different modes, while every mode is defined by its inputs, outputs, as well as different dynamic parameters. An example for this could be a Named Entity Recognition extractor, that takes different languages for every defined mode. An exemplary excerpt can be seen in figure 19. The given "extractor" has three different modes (namely "mode1" to "mode3"). The Part in this case is extracted by "mode3" under its specified configuration.

Include Fusepool Annotation Model The Fusepool Annotation Model (FAM) defines several annotations for textual content. It supports user level annotations like named entities, topics, sentiment, as well as machine level annotations like tokens, POS tags, lemmas, or stems. For a detailed description of the model and its vocabulary elements, see the FAM specification[36]. The following annotations are added to the MICO ontology: language annotations, entity mention annotations, linked entity annotations, topic classifications, and sentiment annotations. All of them are documented and described in the documentation of mmmterms[37]. MICO adopts the user level annotation of the FAM for extractors processing textual content.

3.4 Specification

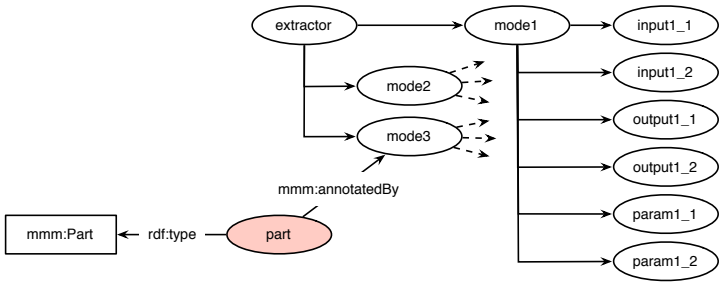
As part of this deliverable, the appendix contains four documents about the specification of the MICO model:

`mmm` Specification Contains the machine-readable OWL specification of the `mmm` vocabulary. Also published online at `http://www.mico-project.eu/ns/mmm/2.0/schema#`.

[36] `https://github.com/fusepoolP3/overall-architecture/blob/master/wp3/fp-anno-model/fp-anno-model.md`

[37] `http://mico-project.bitbucket.org/vocabs/mmmterms/2.0/documentation/`

Figure 19 Exemplary Extractor Provenance



mmm Documentation Contains the human readable specification and documentation of the `mmm` vocabulary. Also published online at `http://mico-project.bitbucket.org/vocabs/mmm/2.0/documentation/`.

mmmterms Specification Contains the machine-readable OWL specification of the `mmmterms` vocabulary. Also published online at `http://www.mico-project.eu/ns/mmmterms/2.0/schema#`.

mmmterms Documentation Contains the human readable specification and documentation of the `mmmterms` vocabulary. Also published online at `http://mico-project.bitbucket.org/vocabs/mmmterms/2.0/documentation/`.

3.5 Read and write the MICO Metadata Model

Most of the people interacting with the Semantic Web, no matter if it is a researcher, programmer, or a data-scientist, do not make (full) use of the Semantic Web technologies, as it requires much effort in order to use your data in a Semantic Web conform way. You have to face challenges like familiarising yourself with the de-facto standard RDF (Resource Description Framework [MM04]) as well as its querying language SPARQL (SPARQL Protocol and RDF Query Language [GSP]). Best practices, ontologies and different specifications that allow to express the data and information in extensible and meaningful ways even make it more difficult.

The MICO project uses Semantic Web technologies in an extensive way to model a uniform way for heterogenous cross-media analysis results and the corresponding workflow chains. At the same time, the project focus on lowering the barrier to Semantic Web technologies for non Semantic Web developers to support a wider range of cross-multimedia analysis adoption. Therefore we implemented a software approach during the project, that tries to facilitate the programmatic access to RDF with the well known programming language Java. The result is the open-source library Anno4j[38], which was already introduced in Dx3.1. The key features were an extensible creation of MICO (see section 3.4) or generic W3C Web Annotations with plain Java objects, predefined implementations for Bodies and Targets conform to the MICO Metadata Model or W3C Web Annotation Data Model, and persisting of the generated RDF to a local or remote SPARQL 1.1 endpoint.

Consistent with a second iteration of the MICO Metadata Model, Anno4j has also been reworked. Besides necessary refactoring of the code base to allow for example multiple inheritance of entities, concurrent usage of Anno4j instances, Anno4j was adapted to the new version of the MICO Metadata

[38] `https://github.com/anno4j/anno4j`

Model. Furthermore, Anno4j was enhanced with convenient methods to export created RDF information in various formats like JSON-LD, Turtle, or XML. The two most important changes are the introduction of querying and custom extensibility of Anno4j will be presented in the following.

3.5.1 Anno4j Querying

Anno4j now provides an expressive way of querying persisted annotations or parts of them, using a path-based criteria, which reduces the effort to learn non-experts. Anno4j uses the LDPath[39] syntax to define multiple criteria of the resulting annotation. LDPath is a path-based query language similar to XPath [DR14] or SPARQL Property Paths [Sea10]. It showed up that, particularly for non-experts, path-based declaration of annotation criteria with LDPATH is more convenient in contrast to a pattern-based query languages like SPARQL.

Listing 1 shows an example where multiple criteria are used to define an Annotation where the Body should be a `mmmterms:AnimalDetectionBody` and an Elephant was detected. A collection of individual criteria define the desired characteristic of the resulting annotations. Hence, a developer is not obliged to create a complex and overloaded equivalent SPARQL query. There is no need for special LDPath capable endpoint, because LDPath criteria are directly translated to an equivalent SPARQL 1.1 query, which allows developers to reuse generic SPARQL 1.1 endpoints for their use-cases.

Listing 1: Anno4j Query Example

```
1  List<Annotation> annotations = queryService
2          .addCriteria("oa:hasBody[is−a mmmterms:AnimalDetectionBody]")
3          .addCriteria("oa:hasBody/rdf:value", "Elephant")
4          .execute();
```

Besides basic path criteria, Anno4j also supports a wide range of different condition types:

- Forward and reverse path conditions
- Recursive Path like OneOrMore(+) or ZeroOrMore(*)
- Comparison methods like equal, greater, or lower
- Union of multiple paths
- Type or Datatype conditions
- Logical combination of conditions
- Custom functions

For further and detailed description of the LDPath criteria, please refer to Anno4j[40] and LDPath specification[41].

[39] `http://marmotta.apache.org/ldpath/`
[40] `https://github.com/anno4j/anno4j`
[41] `http://marmotta.apache.org/ldpath/`

3.5.2 Anno4j Custom Extensions

The path criteria accepted by Anno4j will cover essential requirements of the end-users. To be more flexible to the users needs, Anno4j provides a powerful way to extend the basic querying mechanism by allowing the user to register custom LDPath syntax, such as function predicates, test functions and filters. Developers can introduce new query elements by defining LDPath syntax and registering a partial query evaluator which transforms the new query element to valid SPARQL 1.1 to ensure full compatibility with generic SPARQL endpoints. This extension mechanism was induced by the requirement to integrate query extensions like SPARQL MM (see section 4) into Anno4j. A SPARQL-MM integration can, for example, query for temporal or spatial relationships of different annotations. Considering the previous example in listing 1, we could extend the query to search for annotations where an elephant stands left besides a giraffe by using a custom function in the LDPath query (see listing 2). The parameter of the `fn:leftBesides` function contains an LDPATH query to select the giraffe annotation. The custom function `fn:leftBesides` doesn't represent an actual serialised relationship between two annotations, but rather uses annotation information like the target and selection to evaluate the corresponding function.

Listing 2: Anno4j Custom Function Example

```
1  List<Annotation> annotations = queryService
2          .addCriteria("oa:hasBody[is-a mmmterms:AnimalDetectionBody]")
3          .addCriteria("oa:hasBody/rdf:value", "Elephant")
4          .addCriteria("fn:leftBesides(oa:hasBody/rdf:value, Giraffe)")
5          .execute();
```

c

4 Specifications and Models for Cross-media Querying

With SPARQL-MM we introduced an Multimedia-specific extension to the existing de-facto standard query language in the Semantic Web SPARQL. It introduces spatio-temporal filter and aggregation functions to handle media resources and fragments that follow the W3C standard for Media Fragment URIs. Basic requirements for this extension as well as a basic specification are already defined in D4.1.1. and D4.2.1. Additionally we set a proper basis for all technology enablers by aligning them to real world use cases in D7.1.1 and D8.1.1. In D4.2.1 the specification fulfilled the requirements TE-401 (Support query by spatio-temporal relationship), TE-402 (Support full-text search on structured datasets), TE-403 (Support storage and retrieval for structured data), TE-405 (Provide a SPARQL query interface), and TE-406 (SPARQL 1.1 Support). With the second specification we extend this functionality to:

TE-412: Support complex spatio-temporal fragments By extending the rudimentary Media Fragment URIs to more complex shapes and transformations we support more fine grained annotation and retrieval for spatio-temporal fragments.

TE-413: Support geographical queries We achieve this by integrating the GeoSPARQL standard from the Open Geospatial Consortium [Ope11] to Apache Marmotta.

TE-404: Support SPARQL-MM query building We achieve this aim by adding SPARQL-MM functionality to Anno4J (as described in the example in 3.5). The integration currently contains all spatial and temporal relation functions that are specified for SPARQL-MM and is continuously adapted to new features. This integration makes the access of annotations and media fragments seamless.

TE-409: Support media analysis functions and properties We achieve this by cover complex SPARQL queries (that use and combine existing analysis features) within filter functions and ”magic properties” using SPARQL Inferencing Notation [42].

TE-410: Support image and video fragment presentation We achieve this by providing a web service that supports media fragments as query parameter and thus serves cropped images and videos. As this service is not query but platform specific it is going to be described in the final version of the MICO platform D6.2.3.

We do not specify TE-408 (Fuzzy SPARQL), TE-411 (Pivot Vocabularies) and TE-407 (RDF based similarity search) here yet due to the project internal task prioritization. This are considered for the next iteration of SPARQL-MM.

In this chapter we describe how we build on top of the existing work to extend the functionality, support missing features and improve the efficiency of SPARQL-MM. We divided the chapter in 3 subsections:

Retrospect gives an overview of existing features of SPARQL-MM.

Extended Specification outlines how we extend SPARQL-MM feature-wise in the this iteration cycle and informs about the underlying technologies.

[42] `http://spinrdf.org/`

Outlook shows how we are going to address the missing features in the next iteration. Additionally we give a brief overview on optimization strategies that will allow a more efficient evaluation of SPARQL-MM. As a third point we show how we are going to automatically generate test-data of arbitrary cardinality for high-scalability tests.

The sample data and queries are related to MICO use cases, which gives the reader a proper understanding of how to use SPARQL-MM in real world scenarios.

4.1 Retrospect

SPARQL-MM as query language for Linked Media is an important pillar for the bridge between Multimedia and the Semantic Web. In D4.1.1 we described many features that has to be fulfilled to make this pillar stable and robust. Like described in [KSK15], SPARQL-MM focused in the first iteration mainly on functions for spatial and temporal object retrieval. As example (adapted from [Aic+15]) we show, how SPARQL-MM can be used to issue queries like:

> Find scenes where Barack Obama is left beside the Greenpeace MD during the UN climate change summit ordered by length.

The associated SPARQL-MM query could look like this:

```
PREFIX mm:      <http://linkedmultimedia.org/sparql-mm/ns/1.0.0/function#>
PREFIX oa:      <http://www.w3.org/ns/oa#>
PREFIX ex:      <http:mico-project.eu/examples/vocabulary/>
PREFIX dbpedia: <http://dbpedia.org/resource/>

SELECT ?scene WHERE {
  ?a3     oa:hasBody ?event;
          oa:hasTarget ?s3.                              # there are resources about an event

  ?event  schema:Event;
          schema:summary ?description.                   # which has a description

  FILTER mm:fulltext-search(str (?description),
                            "UN climate conference","en")  # about 'UN climate conference'

  ?a2     oa:hasBody dbpedia:Barack_Obama;
          oa:hasTarget ?s2.                              # and there are resources about Obama

  ?a1     oa:hasBody ?p1;
          oa:hasTarget ?s1.                              # and there are resources
  ?p1 ex:ceo_of dbpedia:Greenpeace.                      # about the MD of Greenpeace.

  FILTER mm:leftBeside(?s2,?s1)                          # Obama has to be at the left of the MD and

  FILTER mm:intersects(?s3,?s2)                          # has to be appear at the same time like the event.

  BIND (mm:boundingBox(?s1,?s2) AS ?scene)               # Wrap the results to scenes

  ORDER BY DESC(mm:duration(?scene))                     # and show the longest first.
```

Like described in the first specification, SPARQL-MM supports by now:

- Media Fragments and Media Fragment URIs, like described in [Tro+12].
- 10 spatial topological relations based on the Dimensional Extended nine-Intersection Model (DE-9IM) [CFO93]
- 8 spatial directional relations based in simple center point comparison model
- 13 temporal relations based on Allen's interval algebra [All83]

Furthermore SPARQL-MM integrates:

- full-text query and full-text search
- all features defined by SPARQL 1.1 query language [HS13]

In the next section we describe, how SPARQL-MM is extended in several ways.

4.2 Extended Specification

In this section we specify several extensions for Multimedia querying in MICO. It includes an extension for Media Fragment URIs, a description of the GeoSPARQL implementation, an extension of SPARQL-MM regarding accessor-functions and an introduction of the SPARQL Inferencing Notation.

4.2.1 Extension of Media Fragment URIs

The existing standard for Media Fragment URIs [Tro+12] specifies the syntax for constructing media fragment URIs and explains how to handle them over the HTTP protocol. It supports addressing media along four dimensions, which are spatial, temporal, track (e.g. audio) and id (e.g. a chapter). Unfortunately the spatial dimension is weak with respect to representation power because it just supports rectangular regions. This may be the reason why, in comparison to the temporal fragments that are supported by all major web browsers, spatial fragments lead a niche existence. Therefor we decided to extend the standard with a) a wider range of geometric shapes b) basic translations of shapes and c) animations of shapes in a temporal range. As described in D4.2.1 the SPARQL-MM model already considers such extensions and therefore has to be just slightly adapted.

The specification is an outcome of many discussions with the research community in the area of Linked Media. The extensions are mainly inspired by discussions at the World Wide Web conference 2015[43] and the work from Tom Steiner[44] and Olivier Aubert[45].

In the following we interpret coordinates regarding the standard: ”Pixel coordinates are interpreted after taking into account the resource’s dimensions, aspect ratio, clean aperture, resolution, and so forth, as defined for the format used by the resource. If an anamorphic format does not define how to apply the aspect ratio to the video data’s dimensions to obtain the ”correct” dimensions, then the user agent must apply the ratio by increasing one dimension and leaving the other unchanged.”[Tro+12].

Extending Shapes
Currently the Media Fragment URI standard supports only rectangular selections. The rectangle can be specified as pixel coordinates or percentages. According the standard ”Rectangle selection is denoted by the name `xywh`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 4 comma-separated integers. The integers denote x, y, width and height, respectively, with x=0, y=0 being the top left corner of the image. If percent is used, x and width are interpreted as a percentage of the width of the original media, and y and height are interpreted as a percentage of the original height.” Inspired by SVG Basic Shape specification in [Fer01] we recommend four shapes in addition (or substitution) to `xywh`:

[43] `https://lists.w3.org/Archives/Public/public-media-fragment/2015May/0003.html`
[44] `https://github.com/tomayac/dynamic-media-fragments`
[45] ~urlhttp://olivieraubert.net/dynamic-media-fragments/

Rectangle: `rect=x,y,w,h[,rx,ry]`
This is an extension of the existing rectangular basic shape. Rectangle selection is denoted by the name `rect`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 4 or 6 comma-separated integers. The integers denote x, y, width, height and (optionally) the x and y radius (rx and ry) of the ellipse used to round off the corners of the rectangle respectively. x=0, y=0 being the top left corner of the image. If percent is used, x, width and rx are interpreted as a percentage of the width of the original media, and y, height and ry are interpreted as a percentage of the original height. An example is outlined in Figure 20. The formal syntax specification is outlined in listing 3.

Listing 3: BNF for Media Fragment Extension: rect

```
rectprefix    = %x72.65.63.74                                   ; "rect"
rectparam     = [ unit ":" ] 1*DIGIT "," 1*DIGIT "," 1*DIGIT "," 1*DIGIT
                                 [ "," 1*DIGIT "," 1*DIGIT ]
unit          = %x70.69.78.65.6C                                ; "pixel"
                / %x70.65.72.63.65.6E.74                        ; "percent"
```

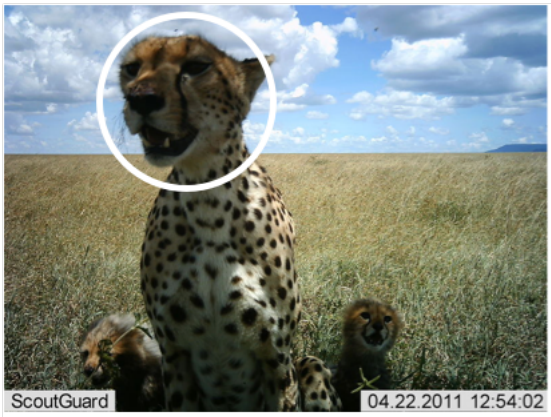
Figure 20 Example: `rect=75,11,161,283,20,20`

Circle: `circle=x,y,r`
Circle selection is denoted by the name *circle*. The value is an optional format *pixel:* or *percent:* (defaulting to pixel) and 3 comma-separated integers. The integers denote x and y as the center of the circle and r as the radius. x=0, y=0 being the top left corner of the image. If percent is used, x and r are interpreted as a percentage of the width of the original media, and y is interpreted as a percentage of the original height. An example is outlined in Figure 21. The formal syntax specification is outlined in listing 4.

Listing 4: BNF for Media Fragment Extension: circle

```
circleprefix    = %x63.69.72.63.6c.65                    ; "circle"
circleparam     = [ unit ":" ] 1*DIGIT "," 1*DIGIT "," 1*DIGIT
```

Figure 21 Example: `circle=136,72,65`



Ellipse `ellipse=cx,cy,rx,ry`
Ellipse selection is denoted by the name `ellipse`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 4 comma-separated integers. The integers denote cx, cy (the center of the ellipse) and rx, ry (the radius of the ellipse). x=0, y=0 being the top left corner of the image. If percent is used, cx and rx are interpreted as a percentage of the width of the original media, and cy and ry are interpreted as a percentage of the original height. An example is outlined in Figure 22. The formal syntax specification is outlined in listing 5.

Listing 5: BNF for Media Fragment Extension: ellipse

```
ellipseprefix    = %x65.6c.6c.69.70.73.65            ; "ellipse"
ellipseparam     = [ unit ":" ] 1*DIGIT "," 1*DIGIT "," 1*DIGIT ","
  1*DIGIT
```

Polygon `polygon=x1,y2*(,xn,yn)`
Polygon selection is denoted by the name `polygon`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 2*n comma-separated integers (with $n \in \mathbb{N}$). The integers denote x, y as starting point and xn, yn as points on the polyline that borders the polygon; the polygon is closed. x=0, y=0 being the top left corner of the image. If percent is used, x and xn are interpreted as a percentage of the width of the original media, and y and yn are interpreted as a percentage of the original height. An example is outlined in Figure 23. The formal syntax specification is outlined in listing 6.

Listing 6: BNF for Media Fragment Extension: polygon

```
polygonprefix    = %x70.6f.6c.79.67.6f.6e            ; "polygon"
polygonparam     = [ unit ":" ] 1*DIGIT "," 1*DIGIT *("," 1*DIGIT ","
  1*DIGIT)
```

Figure 22 Example: `ellipse=271,252,30,50`

Figure 23 Example: `polygon=213,265,263,198,333,265`

As the shapes identify regional fragments there is no necessity for ”open shapes” so we skipped line and polyline. The existing shape *xywh=x,y,w,h* can be substituted to *rect=x,y,w,h*.

Introducing Transformation

Currently the Media Fragment URI standard does not support shape transformation. Even with the shape extensions we introduced in the former section, the identification of spatial fragments is limited. Additionally, with regard to further extensions e.g. animations, a proper representation of shape transformation and translation is lacking. We aim to overcome this limitations by introducing shape transformations. Inspired by SVG Transformation specification in [Fer01] we recommend:

Translate: `translate=x[,y]`

Translate transformation is denoted by the name `translate`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 1 or 2 comma-separated integers. The integers denote x horizontal and y (optionally) for vertical translation. x=0, y=0 being the top left corner of the image. If percent is used, x is interpreted as a percentage of the width of the original media, and y is interpreted as a percentage of the original height. The formal syntax specification is outlined in listing 7.

Listing 7: BNF for Media Fragment Extension: translate

```
translateprefix    = %x74.72.61.6e.73.6c.61.74.65    ; "translate"
translateparam     = [ unit ":" ] 1*DIGIT [ "," 1*DIGIT ]
```

Scale: `scale=x[,y]`
Scale transformation is denoted by the name `scale`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 1 or 2 comma-separated integers. The integers denote x horizontal and y (optionally) for vertical scale. x=0, y=0 being the top left corner of the image. If percent is used, x is interpreted as a percentage of the width of the original media, and y is interpreted as a percentage of the original height. The formal syntax specification is outlined in listing 8.

Listing 8: BNF for Media Fragment Extension: scale

```
scaleprefix    = %x73.63.61.6c.65                  ; "scale"
scaleparam     = [ unit ":" ] 1*DIGIT [ "," 1*DIGIT ]
```

Rotate `rotate=a[,x,y]`
Rotate transformation is denoted by the name `rotate`. The values are and 1 or 3 comma-separated integers. The x and y value (optional) is an optional format `pixel:` or `percent:` (defaulting to pixel). The integers denote a as rotation angle and x,y as center of rotation. x=0, y=0 being the top left corner of the image. If percent is used, x is interpreted as a percentage of the width of the original media, and y is interpreted as a percentage of the original height. The formal syntax specification is outlined in listing 9.

Listing 9: BNF for Media Fragment Extension: rotate

```
rotateprefix    = %x72.6f.74.61.74.65                ; "rotate"
rotateparam     = [ unit ":" ] 1*DIGIT [ "," 1*DIGIT "," 1*DIGIT ]
```

Skew `rotate=a[,x,y]`
Skew transformation is denoted by the name `skew`. The values are and 1 or 2 comma-separated integers. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) and 1 or 2 comma-separated integers. The integers denote x horizontal and y (optionally) for vertical skew. x=0, y=0 being the top left corner of the image. If percent is used, x is interpreted as a percentage of the width of the original media, and y is interpreted as a percentage of the original height. The formal syntax specification is outlined in listing 10.

Figure 24 Example: `ellipse=157,150,66,145&rotate=345`

Listing 10: BNF for Media Fragment Extension: skew

```
skewprefix    = %x73.6b.65.77                          ; "skew"
skewparam     = [ unit ":" ] 1*DIGIT [ "," 1*DIGIT ]
```

Transformations in Media Fragment URIs are only considered if one and only one shape is defined. Transformations can be stacked. If a transformation type occurs more than once, only the first value is considered.

Animating Transformation

The static transformations we introduced are a useful tool for a more special definition of Media Fragment and may be sufficient for still images. But spatial fragments often needs to transform over time (for videos, interactive charts etc.). Therefore we introduce animated transformations. In order to keep urls short we just use the prefix a + the name of the transformation that has be animated:

Animated Translate: `aTranslate=d1,x1[,y1]*[;dn,xn[,yn]]`
Animated translate transformation is denoted by the name `aTranslate`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) plus a semicolon-separated list of 2 or 3 comma-separated numbers. The first number of each number set (d) is defined as duration and may be defined in percent (for videos) or milliseconds (for images). The other numbers represents the translation as specified. The formal syntax specification is outlined in listing 11.

Listing 11: BNF for Media Fragment Extension: aTranslate

```
atranslate    = %x61.54.72.61.6e.73.6c.61.74.65      ; "aTranslate"
atranslate    = [ unit ":" ] 1*NUMBER, 1*DIGIT [ "," 1*DIGIT ]
  *[1*NUMBER, 1*DIGIT [ "," 1*DIGIT ]]
```

Animated Scale: `aScale=d1,x1[,y1]*[;dn,xn[,yn]]`

Animated scale transformation is denoted by the name `aScale`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) plus a semicolon-separated list of 2 or 3 comma-separated numbers. The first number of each number set (d) is defined as duration and may be defined in percent (for videos) or milliseconds (for images). The other numbers represents the scaling as specified. The formal syntax specification is outlined in listing 12.

Listing 12: BNF for Media Fragment Extension: aScale

```
1  ascaleprefix    = %x61.53.63.61.6c.65                ; "aScale"
2  ascaleparam     = [ unit ":" ] 1*NUMBER, 1*DIGIT [ "," 1*DIGIT ]
     *[1*NUMBER, 1*DIGIT [ "," 1*DIGIT ]]
```

Animated Rotate: `aRotate=d1,r1[,x1,y1]*[;dn,rn[,xn,yn]]`
Animated rotate transformation is denoted by the name `aRotate`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) plus a semicolon-separated list of 2 or 4 comma-separated numbers. The first number of each number set (d) is defined as duration and may be defined in percent (for videos) or milliseconds (for images). The other numbers represents the rotaion as specified. The formal syntax specification is outlined in listing 13.

Listing 13: BNF for Media Fragment Extension: aRotate

```
1  arotateprefix    = %x61.52.6f.74.61.74.65            ; "aRotate"
2  arotateparam     = [ unit ":" ] 1*NUMBER, 1*DIGIT [ "," 1*DIGIT ","
     1*DIGIT] *[1*NUMBER, 1*DIGIT [ "," 1*DIGIT "," 1*DIGIT]]
3
```

Animated Skew: `aSkew=d1,r1[,x1,y1]*[;dn,rn[,xn,yn]]`
Animated skew transformation is denoted by the name `aSkew`. The value is an optional format `pixel:` or `percent:` (defaulting to pixel) plus a semicolon-separated list of 2 or 4 comma-separated numbers. The first number of each number set (d) is defined as duration and may be defined in percent (for videos) or milliseconds (for images). The other numbers represents the skew as specified. The formal syntax specification is outlined in listing 14.

Listing 14: BNF for Media Fragment Extension: aSkew

```
1  askewprefix    = %x61.53.6b.65.77                   ; "aSkew"
2  askewparam     = [ unit ":" ] 1*NUMBER, 1*DIGIT [ "," 1*DIGIT ]
     *[1*NUMBER, 1*DIGIT [ "," 1*DIGIT ]]
3
```

Animated Transformations in Media Fragment URIs are only considered if one and only one shape is defined. Animated transformations can be stacked. If an animated transformation type occurs more than once, only the first value is considered. Figure 25 shows how a spatial fragment is animated over time in both scale and translation. In this case there is no translation until 70% of the temporal fragment (8 seconds overall), in the remaining time the shape translates linearly. The scaling is linear over the whole fragment playtime.

Figure 25 Example: `ellipse=90,84,7,12&t=0,8&aTranslate=0.7,0,0;0.3,-13,-33 &aScale=percent:1,3,1`

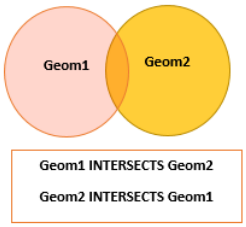
On the web[46] we provide a draft implementation of our media fragment extension proposal. This work is on a very early stage and represents only a snapshot of current and ongoing work in an upcomming W3C working group. To keep on track we recommend to subscribe to the newsletter of the Media Fragments Working Group[47].

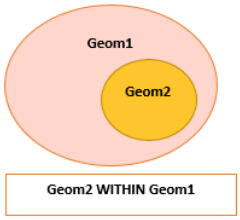
4.2.2 GeoSPARQL

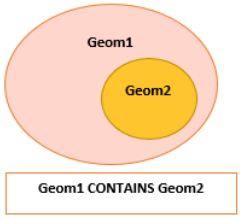
"The OGC GeoSPARQL standard supports representing and querying geospatial data on the Semantic Web. GeoSPARQL defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data [...]. In addition, GeoSPARQL is designed to accommodate systems based on qualitative spatial reasoning and systems based on quantitative spatial computations."[Ope11]. In MICO we planed to integrate this standard in the Open Source framework Apache Marmotta which builds the basis for Metadata Storage Layer of the MICO platform. To achieve this we used the instruments of the Open Source community and mentored a development process under the organizational umbrella of the Google Summer of Code 2015 (GSoC[48]) project. The whole development is described in the official Apache Issue tracking system under *MARMOTTA-584*[49]. In this section we give a short overview of the implemented parts of the standard. The function vocabulary uses the prefix `geof:` `http://www.opengis.net/def/function/geosparql/`.

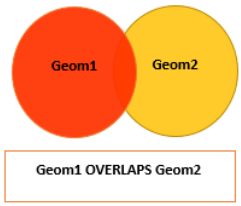
Simple Features Topological Relations

[46] `http://tkurz.github.io/media-fragment-uris-ideas/`
[47] `http://www.w3.org/2008/WebVideo/Fragments/`
[48] `https://developers.google.com/open-source/gsoc/`
[49] `https://issues.apache.org/jira/browse/MARMOTTA-584`

Function	intersects
Syntax	geof:sfIntersects(?geom1, ?geom2)
Description	returns true if p1.shape contains p2.shape.
Example	

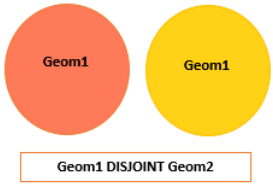
Function	within
Syntax	geof:sfWithin(?geom1, ?geom2)
Description	returns TRUE if geom1 is completely inside geom2.
Example	

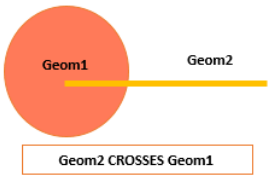
Function	contains
Syntax	geof:sfContains(?geom1, ?geom2)
Description	returns TRUE if and only if no points of geom2 lie in the exterior of geom1, and at least one point of the interior of geom2 lies in the interior of geom1.
Example	

Function	overlaps
Syntax	geof:sfOverlaps((?geom1, ?geom2)
Description	returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other.
Example	

Function	touches
Syntax	geof:sfTouches(?geom1, ?geom2)
Description	returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.
Example	

Function	equals
Syntax	geof:sfEquals(?geom1, ?geom2)
Description	returns TRUE if the given geometries represent the same geometry.
Example	

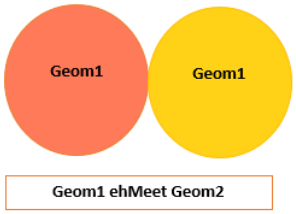
Function	disjoint
Syntax	geof:sfDisjoint(?geom1, ?geom2)
Description	returns TRUE if the Geometries do not ”spatially intersect” - if they do not share any space together.
Example	

Function	crosses
Syntax	geof:sfCrosses(?geom1, ?geom2)
Description	returns TRUE if the supplied geometries have some, but not all, interior points in common.
Example	

Eigenhofer Topological Relations

Function	equals
Syntax	geof:ehEquals(geom1, geom2)
Description	returns TRUE if the given geometries represent the same geometry.

Function	disjoint
Syntax	geof:ehDisjoint(?geom1, ?geom2)
Description	returns TRUE if the Geometries do not ”spatially intersect” - if they do not share any space together.

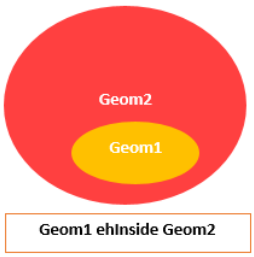
Function	meet
Syntax	geof:ehMeet(?geom1, ?geom2)
Description	returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.
Example	

Function	overlap
Syntax	geof:ehOverlap(?geom1, ?geom2)
Description	returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other.

Function	crosses
Syntax	geof:sfCrosses(?geom1, ?geom2)
Description	returns TRUE if the supplied geometries have some, but not all, interior points in common.

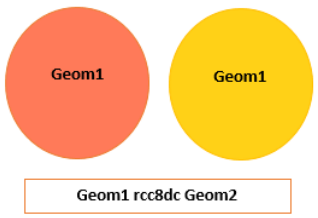
Function	covers
Syntax	geof:ehCovers(?geom1, ?geom2)
Description	returns TRUE if the subject spatialObject spatially covers the object spatialObject. DE-9IM: T*TFT*FF*.
Example	

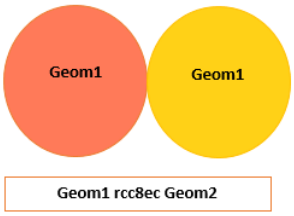
Function	coveredBy
Syntax	geof:ehCoveredBy(?geom1, ?geom2)
Description	returns TRUE if the subject spatialObject is spatially covered by the object spatialObject. DE-9IM: TFF*TFT** (inverse of covers).

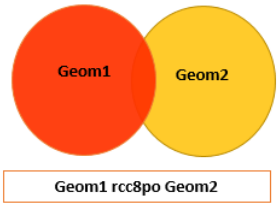
Function	inside
Syntax	geof:ehInside(?geom1, ?geom2)
Description	returns TRUE if the subject spatialObject is spatially inside the object spatialObject. DE-9IM: TFF*FFT**.
Example	

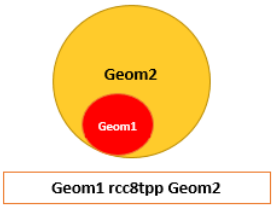
RCC8 Topological Relations

Function	equals
Syntax	geof:rcc8eq(geom1, geom2)
Description	returns TRUE if the given geometries represent the same geometry.

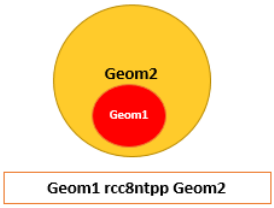
Function	disconnected
Syntax	geof:rcc8dc(geom1, geom2)
Description	returns TRUE if the Geometries do not ”spatially intersect” - if they do not share any space together.
Example	

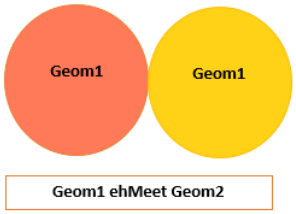
Function	externally connected
Syntax	geof:rcc8ec(geom1, geom2)
Description	returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.
Example	

Function	partially overlapping
Syntax	geof:rcc8po(geom1, geom2)
Description	returns true if geom1 is within geom2 and their borders touches in almost one point.
Example	

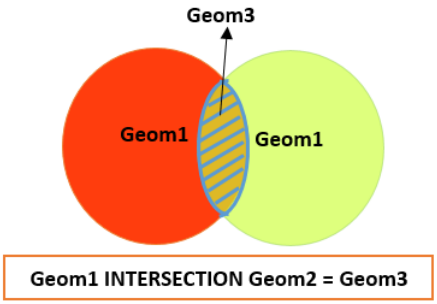
Function	tangential proper part
Syntax	geof:rcc8tpp(geom1, geom2)
Description	returns true if geom1 is within geom2 and their borders touches in almost one point.
Example	

Function	tangential proper part inverse
Syntax	geof:rcc8tppi(geom1, geom2)
Description	returns true if geom1 contains geom2 and their borders touches in almost one point.

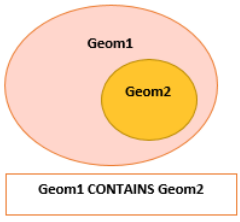
Function	non-tangential proper part
Syntax	geof:rcc8ntpp(geom1, geom2)
Description	returns true if geom 1 is within geom 2 and their borders doesn't touches in almost one point.
Example	

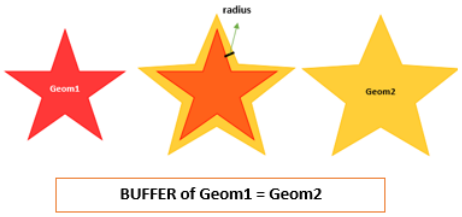
Function	non-tangential proper part inverse
Syntax	geof:rcc8ntppi(geom1, geom2)
Description	returns true if geom1 contains geom2 and their borders doesn't touch in any point.
Example	

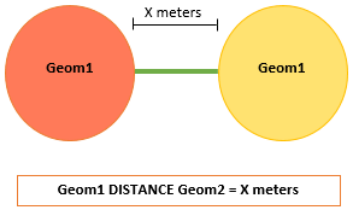
Non-Topological Functions

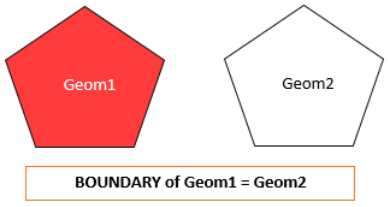
Function	intersection
Syntax	geof:intersection(?geom1, ?geom2)
Description	returns a geometry that represents the shared portion of geom1 and geom2.
Example	

Function	convex hull
Syntax	geof:convexHull(?geom1)
Description	the convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set.
Example	

Function	contains
Syntax	geof:ehContains((?geom1, ?geom2)
Description	returns TRUE if the subject spatialObject spatially contains the object spatialObject.
Example	

Function	buffer
Syntax	geof:buffer(?geom1, radius, units:anyUnit)
Description	this function returns a geometric object that represents all Points whose distance from geom1 is less than or equal to the radius measured in units.
Example	

Function	distance
Syntax	geof:distance(?geom1, ?geom2,units:anyUnit)
Description	returns the shortest distance in units between any two Points in the two geometric objects.
Example	

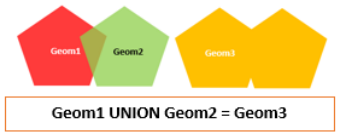
Function	boundary
Syntax	geof:boundary(?geom1)
Description	this function returns the closure of the boundary of geom1.
Example	

Function	difference
Syntax	geof:difference (geom1, geom2)
Description	this function returns a geometric object that represents all Points in the set difference of geom1 with geom2.
Example	

Function	envelope
Syntax	geof:envelope (geom1)
Description	this function returns the minimum bounding box of geom1.
Example	

Function	relate
Syntax	geof:relate(geom1, geom2, pattern-matrix)
Description	returns TRUE if the spatial relationship between geom1 and geom2 corresponds to one with acceptable values for the specified pattern-matrix. Otherwise, this function returns false. Pattern-matrix represents a DE-9IM intersection pattern consisting of T (true) and F (false) values.

Function	symmetric difference
Syntax	geof:symDifference(geom1, geom2)
Description	this function returns a geometric object that represents all Points in the set symmetric difference of geom1 with geom2.

Function	union
Syntax	geof:union(geom1, geom2)
Description	this function returns a geometric object that represents all Points in the union of geom1 with geom2.
Example	

Function	get spatial reference ID
Syntax	geof:getSRID (geom)
Description	returns the spatial reference system URI for geom.

For more information we recommend the *user and developer documentation* at the Apache Marmotta Wiki [50] which builds the basis for this section.

4.2.3 Extension of Spatio-Temporal Functions

As described in D3.1.1 we represent spatial-temporal media sections using the Media Fragments URI specification from the Word Wide Web consortium [Tro+12] - which entails many advantages including information coherence (the identifier of media and fragment is in one place). But this also leads to some issues within the Semantic Web where all the information is expressed using RDF, a graph based approach that tries to formalize all information spread on nodes and edges. SPARQL-MM tries to integrate both approaches within one access mechanism. In D4.2.1 (the basic specification of SPARQL-MM) we introduced many functions that allows to compare fragments with each other. But first tests in real world scenarios showed that an important feature is missing: accessor functions which allows to extract information from media fragment urls and use it in other query parts like projection, filtering and ordering. In this section we define three different kinds of accessor functions; the function identifier follow the schema defined in D4.2.1., the input and output properties are aligned to the SPARQL-MM object model, which uses the prefix `lmo: http://linkedmultimedia.org/sparql-mm/ns/1.0.0/ontology#`.

Spatial Accessor Features (SF):

In order to support pixels and percent we introduce `lmo:unitNumber` which is defined as:

Listing 15: Defintion: lmo:unitNumber

```
lmo:unitNumber = [ unit ":" ] 1*number
number         = INTEGER | DECIMAL
unit           = %x70.69.78.65.6C         ; "pixel"
/ %x70.65.72.63.65.6E.74  ; "percent"
```

Function F402-SF-a: getArea

```
lmo:unitNumber mm:getArea (
   lmo:SpatialEntity entity
)
```

This function returns the area of a spatial entity as unit number. If the property is not a spatial entity, null is returned.

[50] https://wiki.apache.org/marmotta/GSoC/2015/MARMOTTA-584

Function F402-SF-b: getBoundingBox

```
lmo:Rectangle mm:getBoundingBox (
   lmo:SpatialEntity entity
)
```

This function returns the rectangular bounding box for a spatial entity. If the property is not a spatial entity, null is returned.

Function F402-SF-c: getXY

```
lmo:Point mm:getXY (
   lmo:SpatialEntity entity
)
```

This function returns the left upper point of the bounding rectangle of a spatial entity. If the property is not a spatial entity, null is returned.

Function F402-SF-d: getHight

```
lmo:unitNumber mm:getHight (
   lmo:SpatialEntity entity
)
```

This function returns the height of the bounding box for a spatial entity as unit number. If the property is not a spatial entity, null is returned.

Function F402-SF-e: getWidth

```
lmo:unitNumber mm:getWidth (
   lmo:SpatialEntity entity
)
```

This function returns the width of the bounding box for a spatial entity as unit number. If the property is not a spatial entity, null is returned.

Function F402-SF-f: getCenter

```
lmo:Point mm:getCenter (
   lmo:SpatialEntity entity
)
```

This function returns the center of the spatial entity as point. If the property is not a spatial entity, null is returned.

Temporal Accessor Features (TF):

Currently we only support Normal Play Time (NPT) as specified in the Media Fragment URI standard.

Function F402-TF-a: getDuration

```
xsd:decimal mm:getDuration (
   lmo:TemporalEntity entity
)
```

This functions returns the duration of a temporal entity as decimal. If the property is not a spatial entity, null is returned.

Function F402-TF-b: getStart

```
xsd:decimal mm:getStart (
   lmo:TemporalEntity entity
)
```

This functions returns the start time of a temporal entity as decimal. If the property is not a spatial entity, null is returned.

Function F402-TF-c: getEnd

```
xsd:decimal mm:getEnd (
   lmo:TemporalEntity entity
)
```

This functions returns the end time of a temporal entity as decimal. If the property is not a spatial entity, null is returned.

Generic Accessor Features (GF):

These features allow the retrieval of general information regarding Media Fragments and lmo:unitNumbers.

Function F402-GF-a: isMediaFragment

```
xsd:boolean mm:isMediaFragment (
   xsd:string entity
)
```

This functions returns true if the string can be parsed to a MediaFragment, false otherwise.

Function F402-GF-b: isMediaFragmentURI

```
xsd:boolean mm:isMediaFragmentURI (
   xsd:string entity
)
```

This functions returns true if the string can be parsed to a MediaFragmentURI, false otherwise.

Function F402-GF-c: hasTemporalFragment

```
xsd:boolean mm:hasTemporalFragment (
   xsd:string entity
)
```

This functions returns true if the string can be parsed to a MediaFragment(URI) and has a Temporal Fragment, false otherwise.

Function F402-GF-c: hasSpatialFragment

```
xsd:boolean mm:hasSpatialFragment (
   xsd:string entity
)
```

This functions returns true if the string can be parsed to a MediaFragment(URI) and has a Spatial Fragment, false otherwise.

Function F402-GF-d: toPixel

```
xsd:decimal mm:toPixel (
   lmo:unitNumber number,
   xsd:decimal conversionNumber (OPTIONAL)
)
```

This functions returns the unitNumber converted to pixels based in conversion number. If no conversion number is given, only values for pixel units are returned, null otherwise.

Function F402-GF-e: toPixel

```
xsd:decimal mm:toPixel (
   lmo:SpatialFragment shape,
   lmo:Rectangle conversionShape (OPTIONAL)
)
```

This functions returns the Spatial Fragment converted to pixels based in conversion shape. If no conversion shape is given, only values for pixel fragments are returned, null otherwise.

Function F402-GF-f: toPercent

```
xsd:decimal mm:toPercent (
   lmo:unitNumber number,
   xsd:decimal conversionNumber (OPTIONAL)
)
```

This functions returns the unitNumber converted to percent based in conversion number. If no conversion number is given, only values for percent units are returned, null otherwise.

Function F402-GF-g: toPercent

```
xsd:decimal mm:toPercent (
   lmo:SpatialFragment shape,
   lmo:Rectangle conversionShape (OPTIONAL)
)
```

This functions returns the Spatial Fragment converted to percent based in conversion shape. If no conversion shape is given, only values for percent fragments are returned, null otherwise.

4.2.4 Cover Functions

The MICO platform is able to extract information of various kind from various content formats like text, image, video, etc. All the information is represented in a common annotation schema, which supports low-level features like color histograms as well as hight level features like concept detection. This wide range of results makes information retrieval a data expert task. To prevent this and to make results accessible even for non-experts we are going to provide high-level filter functions, which are internally translated into complex SPARQL queries and cover the complexity of metadata representation from users. The SPARQL Inferencing Notation SPIN[51] (a W3C member submission from 2011) gives as a proper basis for that. In this section we give an overview over the SPIN Modeling Vocabulary, "a light-weight collection of RDF properties and classes to support the use of SPARQL to specify rules and logical constraints."[52]. Especially we will focus on SPIN Functions and Magic Properties that support our use case of covering complex functions. The section is inspired by the SPIN primer [53].

SPIN Constraints and Rules

SPIN rules and constraints are used to execute inferences on classes. Whereby rules are represented as SPARQL CONSTRUCT queries and create new inferences (triples) based on the existing state, constraints can only be used to verify that certain invariants are not valid via creating new instances of type `spin:ConstraintViolation`. Listing 16 shows a constraint that assures that an image of type `ex:ImageWithPerson` contains at least one `ex:Person`. The variable `?this` is always bound to the tested resource.

Listing 16: SPIN Constraint example

```
ex:ImageWithPerson
  spin:constraint [
    rdf:type sp:Construct ;
    sp:text """
      # must have this relation
      CONSTRUCT {
        _:cv a spin:ConstraintViolation ;
        spin:violationRoot ?this ;
        rdfs:label "Image must contain a person" .
      } WHERE {
        ?this ex:includes/rdf:type ex:person .
      }
    """ ;
] .
```

SPIN Functions and Magic Properties

SPIN Functions are "boxed" queries that declare new SPARQL functions. This functions hide the complexity of the underlying SPARQL query and lower the barrier to SPARQL for non-experts. The arguments of the function are described in the `spin:constraint` part as an ordered list of `spl:Argument`(s), which specify the variables that are used within the attendant SPARQL query. Constraints for further

[51] `http://spinrdf.org/`
[52] `http://www.w3.org/Submission/2011/SUBM-spin-modeling-20110222/`
[53] `http://spinrdf.org/spinsquare.html`

argument checks can also be added here. The `spin:body` defines the SPARQL SELECT query the `spint:returnType` specifies the type of the query. The function in listing 17 is taken from a real world use case and returns all subjects that were annotated by a specific annotator since a certain timestamp.

Listing 17: SPIN Function example

```
ex:filteredAnnotationsSince
  rdf:type spin:Function ;
  rdfs:comment "Returns subjects that were annotated with a specfic
                annotation body type since a certain timestamp." ;
  spin:constraint ([
  rdf:type spl:Argument ;
    spl:predicate sp:arg1 ;
    spl:valueType rdf:Resource ;
    rdfs:comment "The type of the annotation body." ;
  ],[
    rdf:type spl:Argument ;
    spl:predicate sp:arg2 ;
    spl:valueType xsd:dateTime ;
    rdfs:comment "The datetime." ;
  ]);
  spin:body [
    rdf:type sp:Select ;
    sp:text """
      SELECT DISTINCT ?subject WHERE {
        ?subject mico:hasContentPart ?cp .
        ?cp mico:hasContent ?annot .
        ?annot oa:hasBody ?body .
        ?body rdf:type ?arg1 .
        ?annot  oa:annotatedAt ?date .
        FILTER {xsd:dateTime(?date) > ?arg2}
     }
    """ ;
  ] ;
  spin:returnType oa:Annotation .
```

This function can than used to issue a simplified SPARQL query; in this case the query returns all subjects that are annotated by the `mico:FaceRecognitionBody` since October the 26th:

Listing 18: SPARQL SPIN Function example

```
SELECT *
WHERE {
  BIND (ex:filteredAnnotationsSince(
    mico:FaceRecognitionBody,
    "2015-10-26T00:00:00+00:00"^^xsd:dateTime
  ) AS ?annotation) .
}
```

In order to make access to complex operations even more natural, SPIN supports the definition of so called Magic properties. Typically, a magic property is backed by a query time calculation function that determines bindings of the variables on the left or right side of the predicate. In comparison to SPIN functions they provide more flexibility because they are able to return multiple values and work

on unbound input or output variables.

4.3 Outlook

As mentioned there are some specified features lacking within this specification. From use case prioritization we decided that similarity search is the next feature we are going to integrate. In this section we give a short outlook on the next steps. Furthermore we give an overview how we are going to make the current implementation more efficient to work on a larger scale. As third point we are going to describe, how we create large scale SPARQL-MM test sets to validate the performance improvements.

Similarity Search

In RDF there are several ways to define similarity. This can include subtype systems, predicate co-occurrence, various types of literal similarity metrics, and graph based approaches like graph distance, weighted graph distance, max-flow etc. In MICO we will define proper distance metrics for Multimedia similarity search based on the MICO Metadata Model, which will be a combination of the above mentioned metrics. An integration in the SPARQL environment will enable users to seamlessly combine similarity search with other SPARQL constructs.

Performance improvements

The current implementation of SPARQL-MM is sufficient for smaller multimedia databases. But if the amount if images and annotations increases the performance of query evaluation should be improved. Therefor we wan't to optimize SPARQL-MM queries in a 3 step optimization process:

1. Re-factor SPARQL operation tree to move SPARQL-MM functions to an optimal position
2. Use database histograms to improve the selection and joining performance
3. Use optimized indexes for media fragment to improve SPARQL-MM geometric functions

Scalability Testing

To measure performance improvements for high scalability we need appropriate test data, namely annotated media fragments of arbitrary cardinality plus corresponding SPARQL-MM queries varying in type and complexity. We are working on an automatic testset creator that builds upon statistics from the real world test set Microsoft COCO[54] that includes more than 300,000 images with about 5 annotations each (from a controlled vocabulary with about 80 categories). We will run the tests with different SPARQL-MM versions which allows a proper comparison of optimization strategies.

[54] http://mscoco.org/

5 Specifications and Models for Cross-media Recommendations

5.1 Introduction

During its initial phase and until year 2, WP5 focused mostly on PredictionIO and a respective monitoring API for data collection. However, as it turned out in discussions with showcase partners during year 2, showcases required support for content-based recommendation, for collaborative filtering (based on user-item interactions), and recommendation using rule-based approaches - it became clear that requirements has to be adjusted, that the PredictionIO-focused framework had to be significantly extended, and that the WP5 framework had to be tightly integrated into the overall MICO platform. All of the aforementioned points then also lead to a readjustment of WP5 partner responsibilities.

In year 3, WP5 therefore aims at a fully integrated cross-media recommendation framework, i.e., a framework that can exploit different information sources, using different recommendation approaches: Depending on the specific use cases, it will allow combination of extractor annotations, content metadata, and usage information, using different media types as input, in order to provide recommendations for other media types as output. At the same time, of course, this means that different approaches and information sources have to be connected: For instance, collaborative filtering will typically work fine on an item level, but not on the level of media fragments, e.g., to recommend specific news segments (related to one specific topic) within news show items, which will require the use of automatic or manual annotation on a fragment level. The same goes for the linking of different media types, which requires additional metadata, like describing that various documents, images and videos are actually about the same species.

The following provides an update and description of the delta with respect to previous deliverables, summarizing requirement adaptations, and providing a design for the WP5 framework as an integral part of the MICO platform. Implementation of this WP5 framework design has started, and will be completed in year 3. More specifically, this section will include:

- an overview over the prioritized and new or adapted WP5 user stories in section 5.2, which are mainly connected to Snapshot Serengeti for Zooniverse (Work Package 7: Crowd Sourcing Platform), and to the Greenpeace Magazine showcase for InsideOut10 (Video Sharing Platform). Both showcases require cross-media extraction, and recommendation that offers previously unknown, but interesting content to end-users.
- relevant datasets, which are key to be able to develop and evaluate WP5 components, are briefly described in section 5.3.
- a more detailed discussion on the recommendation requirements and needs of showcase partners for Shoof, Greenpeace Magazine and Snapshot Serengeti in section 5.5.
- a WP5 framework architecture overview in section 5.4.
- a specification of selected WP5 component interfaces / APIs in section 5.6.

5.2 Key user stories

Many goals of the MICO project were initially described with the Use Case Compendium that was released in April 2014, being part of the requirement analysis documents D7.1.1 and D8.1.1. Since then, new showcases and showcase aspects emerged, which has strongly influenced the needs and priorities of showcases, especially regarding WP5: Some user stories (US) and technology enablers (TE) lost importance, or had to be adapted, and others became more important. The following provides a new set of user stories that were defined in discussions with the showcase partners, which supersede the user stories from the compendium document, and which represent the goals that the following specification and implementation work for year 3 will aim at.

5.2.1 Greenpeace Magazine

The following three user stories describe the scope of WP5 for the Greenpeace Magazine. The result is a REST interface that can be accessed from the Wordlift/Helixware software preparing the Greenpeace Magazine website.

ID	Name	Description
US-60	*Non-Subscriber Recommendation*	As a Greenpeace User (Non Subscriber) I want to receive meaningful links to related articles so that the number of page views per session and CTR is increased (so that he/she will need eventually to subscribe - increasing the opportunity of bringing the user to the subscribe page). Recommendation Input: • Output of Text analysis (entity extraction) components on GP magazine articles • Data on user behaviour for CF
US-61	*Subscriber Recommendation*	As a Greenpeace User (Subscriber) I want to receive meaningful links to related articles so that the number of page views per session and CTR is increased. Recommendation Input: • Output of Text analysis (entity extraction) components on GP magazine articles • Data on user behaviour for CF

ID	Name	Description
US-62	*Editor Support*	As a Greenpeace Editor I want videos and video key frames (images) that are related to the textual content of the current article so that I spend less time in crafting and finalizing new content. Recommendation Input: • Output of text analysis (entity extraction) components • Output of ASR / entity extraction from the audio stream of the video • Data on user behaviour for CF (optional, using editor feedback)

5.2.2 Shoof

Very similar, to the Greenpeace Magazine, the Shoof project profits immensely on item recommendations, that derive from content analysis.

ID	Name	Description
US-63	*Cross-Media Recommendation*	As a Shoof Admin I want to find content interesting for a user by using results from the WP2 extractors applied on Shoof content so that the quality of the information on the application main stream can improve. Recommendation Input: • Output of WP2 extractors (ASR, entity extraction, container MD (e.g. geodata), TVS, content similarity) for Shoof material • Optional: admin feedback
US-64	*User Data Recommendation*	As a Shoof Admin I want content interesting for a user by using a analysis of their click behavior so that the quality of information on the application main stream can improve. Recommendation Input: • Data on user behaviour (Shoof User Data)

5.2.3 Snapshot Serengeti

Making the best out of a volunteer's time and motivation is the driving force behind the user stories relevant for the Zooniverse project. While preprocessing subjects gives valuable data, the processes of using this information to actually modify how subjects are presented to the volunteer is a special form of a recommendation problem.

ID	Name	Description
US-65	*Subject complexity mix*	As a Zooniverse Admin, I want to know the complexity of different images so that I can ensure users get an appropriate level of complexity in the images they are presented with. Recommendation Input: • Results of TE-202
US-66	*Volunteer posting analysis*	As a Zooniverse Admin I want to gain insight from textual analysis of posts so that I can understand user interest, image complexity and learn other things about subjects and how users classify them. Recommendation Input: • Results of TE-212 and TE-215
US-67	*Volunteer clustering*	As a Zooniverse Admin I want to identify clusters of volunteers so that we can target users with a different user experience according to their "group", expertise or knowledge. Recommendation Input: • Geordi Data (user behaviour data) • User profile data (from TE-506) • Talk comments and discussion threads

ID	Name	Description
US-68	*Volunteer education*	As a Zooniverse Admin I want to know which piece of education I should give to a volunteer so that I can get the best out of each volunteer, by improving the quality and quantity of annotations they provide. Recommendation Input: • Geordi Data (user behaviour data) • User profile data (from TE-506) • Talk comments and discussion threads
US-69	*Volunteer-Subject recommendation*	As a Zooniverse Admin I want to know which set of subjects to present to a specific volunteer using what we know about user interest, likely image content, user cluster/expertise, so that I can get the best out of each volunteer, by improving the quantity of annotations they provide. Recommendation Input: • User profile data (from TE-506) • Image analysis data (blankness/animals present) (from TE-202) • Outputs from US-65 (complexity), US-66 (user interest), US-67 (clusters)

5.3 Available datasets

For each of the user stories listed in the previous section, different datasets are needed for implementation and evaluation purposes. The following tables summarize these datasets for the InsideOut10 and Zooniverse showcases.

5.3.1 Greenpeace Magazine and Shoof

Dataset	Description
Extractor data	Data from the extractors is stored inside Marmotta and is queried using the SPARQL endpoint, using Anno4j, as described in section 3.5. To remove complexity and interdependencies between recommendation module and extractors, it is assumed that all content items to be considered for a recommendation task, have previously been annotated within the platform, using the appropriate pipeline. As described in the WP2 broker chapter of this deliverable, broker v3 will provide improved functionalities to define such pipelines, and to perform respective *jobs*, i.e. to execute a specific pipeline using a specific content set. Furthermore, it is assumed that content items referred to by user activities that are fed to the API are mappable to content items analyzed by the platform, as described in the WP2 high-level broker data model. Using this approach, the recommendation provides a *best effort* service: If annotations are available, the recommendation will use it. If only a subset of annotations exists, recommendation will still work, but its quality might decrease. For the Greenpeace Magazine and Shoof showcase, any pipeline configuration that produces named entities which can be processed within the recommendation module can be used. With reference to the extractor overview table in section 2.1, the following extractors are required by this specific recommendation: **audiodemux** to extract audio from video files for further processing, which is especially useful if the audio stream contains speech that can be further processed to deduce information about the content of the video. For details, see section 2.2.9. **kaldi2txt** to extract audio transcriptions for further processing. The task of speech-to-text extraction for different language is implemented using the Kaldi speech recognition toolkit. For details, see section 2.2.6. **redlink-analysis** provides textual analysis for Greenpeace articles and videos, and for Shoof videos. The NER capabilities of this extractor will be used for subsequent clustering and therefore support recommendation of similar items. For details, see section 2.2.7.

User behavior data	User behavior is tracked as pseudonymous log data, e.g., from Google Analytics. The minimal set of required fields is as follows • date: date of the logging event • eventType: type of the logged event, see below • pagePath: URL to the respective content item • userId: numeric value for each user For the Greenpeace Magazine, these events are mostly page impressions from registered and unregistered users. For Shoof, there are more event types, like interactions with the app: • new content uploaded • video watched • video shared • content rated

5.3.2 Snapshot Serengeti

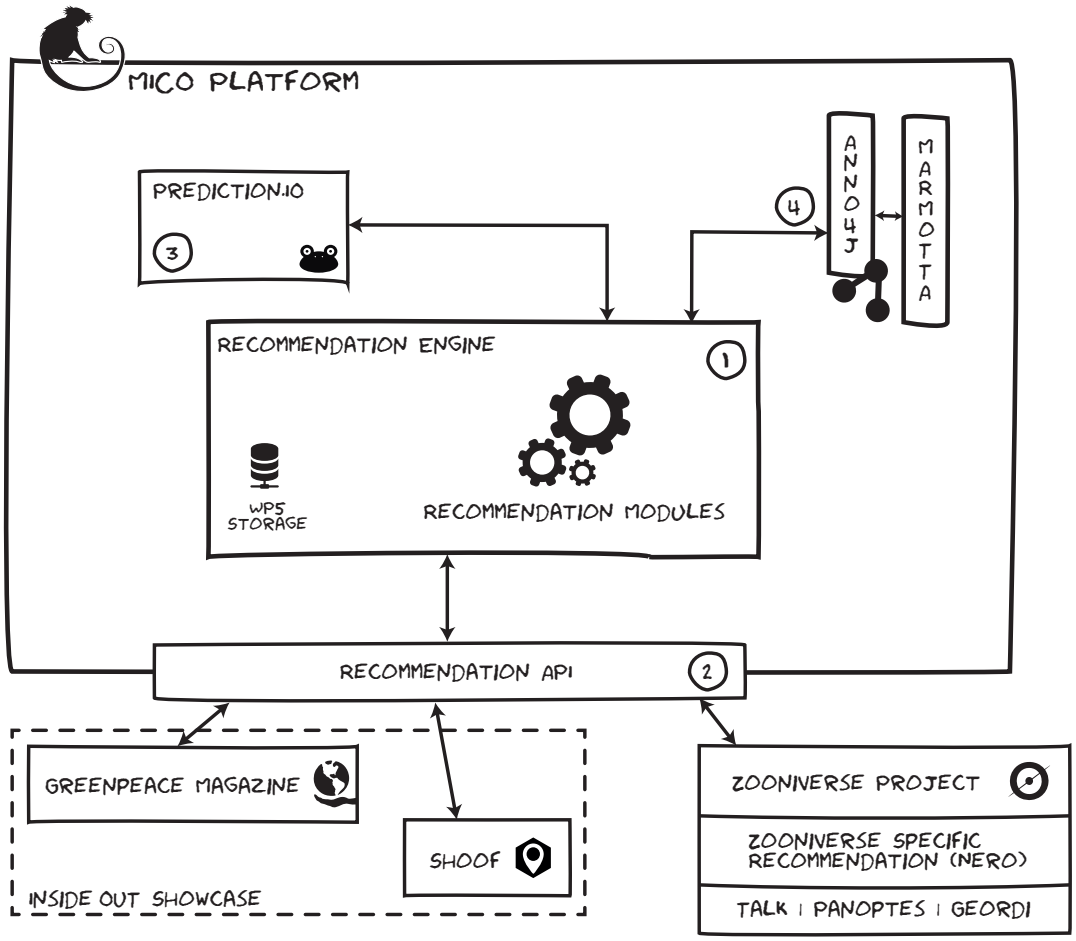
Dataset	Description
Geordi data	Geordi (`https://github.com/zooniverse/geordi/`) is Zooniverse's REST-based user event storage system. Since February 2015 Zooniverse have collected detailed analytics and behavior data for every visitor to Snapshot Serengeti, including which buttons the users clicked and how they classified their images. This is a MySQL database.
User profile data	As part of TE-506 (see Validation doc) we were able to derive information about which species each user is most interested in. Initial experiments suggest this data may not be especially useful, but for now we keep it under consideration as a possible data source. This is CSV data.
Classification and subject data	Snapshot Serengeti's backend database, Ouroboros, contains detailed information about classifications so far for each subject, and about the history of which user classified which subject, and when. This is a MongoDB database.
Talk comments and discussion threads	Every volunteer on Snapshot Serengeti is able to comment or ask questions about an image they are classifying, using Zooniverse's "Talk" discussion board system. For each subject, the discussion history is available, as are general topic based threads. Each comment is attributed to a user and contains a timestamp. This is a MongoDB database.

Subject content data	Using the aggregate answers from Snapshot Serengeti volunteers, we produced a CSV containing the known content of each image (blank, or the species present). This data is in a CSV (see Validation deliverable).
TE-202 animal/blankness detection output	As part of validation of the MICO platform (see Validation deliverables), we produced a database containing MICO's recommendations about the likely animals (or lack of animals) present in each image. This is stored in a PostgresSQL database. We have not yet generated this for all images, but this data can be regenerated by re-running tests against the MICO platform.

5.4 General architecture and common specifications

WP5 is located 'between' WP2 extraction and the needs of a showcase partner in the sense that it uses extraction like a user would: On one hand, it uses the annotations provided by extraction, and the querying infrastructure in a similar way as showcase partners. On the other hand, recommendation can be considered a high-level 'extractor' itself, which provides information to the showcase partners. However, the difference is that in contrast to WP2, in which extractors provide new information annotation per analyzed item, WP5 typically exploits information about groups of items and/or their annotations.

The following provides a bird's eye view on the overall architecture of the WP5 framework:



It is important to note that there is a significant conceptual difference between the two showcases: For InsideOut10, there are two applications, both relying directly on the MICO Platform for recommendations. Both the Greenpeace Magazine and Shoof showcases are similar in terms of the in- and output of the recommendation module, and they are therefore discussed jointly in the following, wherever applicable.

For Zooniverse, input from the platform is *combined* with results from internal processing, e.g. for frequently changing data streams that are hard to synchronize with the MICO platform, or for data that

required deep understanding of the business logic.

Please also note that in contrast to previous deliverables, the strong functional decomposition of the recommendation module into several technology enablers was now considered infeasible (due to the modified functional requirements), and hence dropped in favor of *one* technology enabler for cross-media recommendations.

1 Recommendation engine including custom recommendation modules

Basic services that might be reusable for several recommendation use case are implemented here.

2 Recommendation API

Code that is specific to a single user story / need will be put in separate modules that connect to the Recommendation Modules. The logic for triggering recommendation and I/O with systems outside of the platform is implemented here. While media analysis data is available via Anno4j, the recommendation API also offers capabilities to collect user data from the showcases, and forwards it e.g. to the Prediction.io event server for re-training the recommendation model.

In the case of Zooniverse, the recommendation API will be used largely to obtain insights from the MICO platform about the analysed images and text content, and Zooniverse will also be developing a Zooniverse-specific recommendation engine, Nero, to further harness these insights in a real-time application environment. This is necessary because decisions to take forward the analyses from MICO and turn them into full recommendations of how to modify the user's workflow can only be done within the context of the Zooniverse web application, and with reference to other Zooniverse data sources including Panoptes (the new Zooniverse back-end system, Talk (user comments), and Geordi (user behaviour)). Unlike InsideOut, where a lot of MICO's recommendations can be delivered as standalone pieces of metadata that can be offered directly to a user (related videos for example), Zooniverse can only really harness MICO recommendations in an algorithmic, real-time context, where changes made to user workflows will be tracked and will influence subsequent recommendations.

3 Prediction.IO

As described in the previous deliverables, Prediction.io will be responsible for machine-learning based collaborative filtering tasks. While Prediction.io offers a rich feature set, it comes with many dependencies. To simplify using it inside the platform, *docker* was used. For WP5, a custom dockerfile was written that allows to easily deploy Prediction.io with a single command and encapsulate its dependencies. A quick introduction on how to use it, i.e., integrate prediction.io to the MICO platform is desccibed inside the appendix 5.B.

4 Marmotta & Anno4j

Data is stored inside the platform using Marmotta. WP5 will query this data using the Marmotta SPARQL endpoint, using the querying capabilities of Anno4j when applicable. Anno4j is discussed inside this deliverable in chapter 3.

5.5 Showcase specific specifications

While the general architecture is designed for being capable to cope with any showcase that might fit into the MICO ecosystem, the main focus of this is to provide recommendations for the specific showcases by InsideOut10 and Zooniverse. Shoof, the Greenpeace Magazine and Snapshot Serengeti are briefly discussed in the following sections 5.5.1 and 5.5.2.

5.5.1 InsideOut10: Greenpeace Magazine & Shoof

The InsideOut showcase includes two recommendation cases. As discussed before, the relevant User Stories for Greenpeace Magazine are:

- **US-60**: Non-Subscriber recommendation
- **US-61**: Subscriber recommendation
- **US-62**: Editor support

The relevant user stories for Shoof are:

- **US-63**: Cross-Media-Recommendation
- **US-64**: User Data Recommendation

The main similarity of both scenarios is the need for a recommendation based on user data. The main difference, on the other hand, is the type of content involved. While the Greenpeace Magazine naturally involves a lot of text (which is accompanied by audio, video and images) the focus of Shoof is on videos.

Collaborative filtering tasks will be done by Prediction.io. Its event server is exposed by the recommendation API, training is performed asynchronously. The process of sending user event data to the platform and the model retraining is shown as a sequence diagram in figure 26.

For the cross-media recommendation, collaborative filtering will be combined with item similarity results from WP2 extractors. An example to this is the combination of video analysis with speech-to-text analysis and NER. This allows to cluster the videos regarding their topic - without metadata provided by a human. Recommendations are accessible by simple GET requests. Figure 27 shows an example for the Greenpeace Magazine showcase.

Figure 26 Sending user event data and asynchronous triggering of Prediction.io

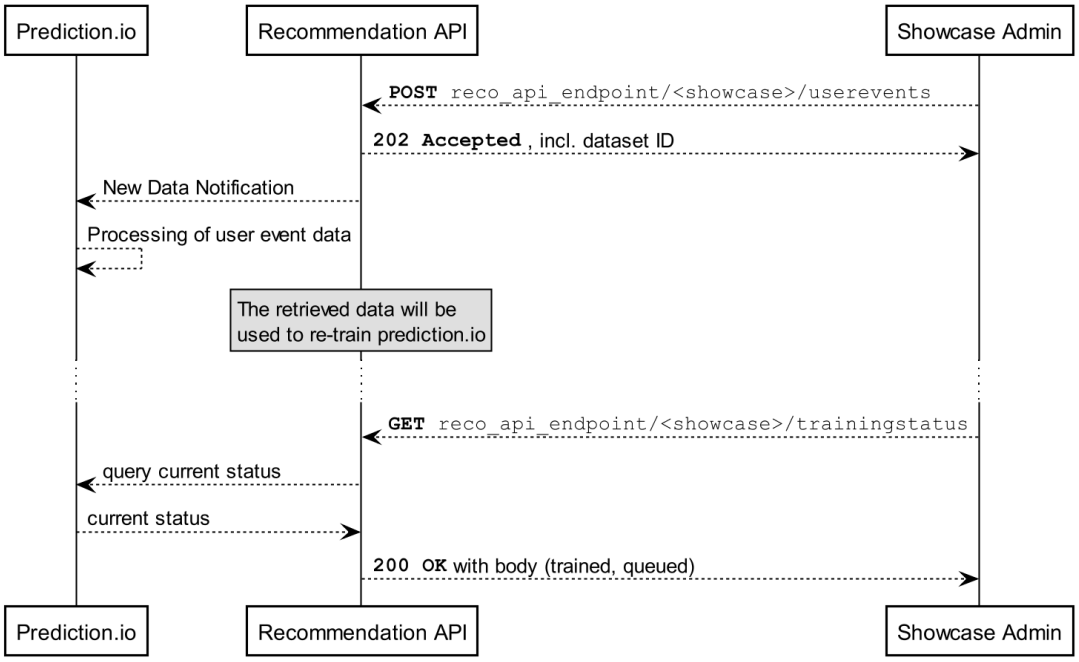
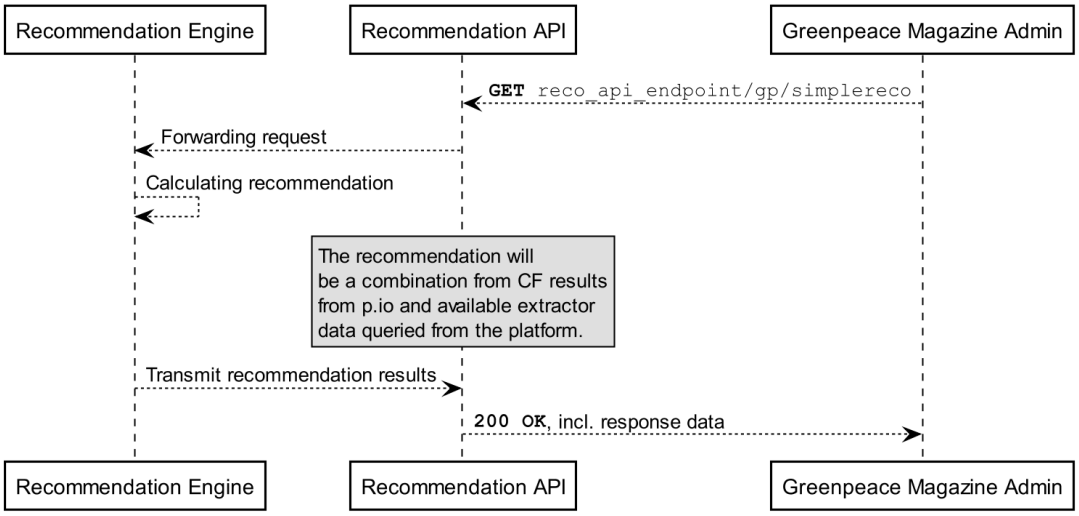


Figure 27 Recommendation using both collaborative filtering and media analysis results (**US60 - US62**)



5.5.2 Zooniverse: Snapshot Serengeti

In contrast to the InsideOut10 showcases, Zooniverse already includes infrastructure for user behavior analysis and recommendation. Moreover, due to delays regarding framework development in WP5, Zooniverse has implemented a simple subject recommendation system based on user interest, using a combination of MySQL queries and Python scripts. However, for MICO, the focus on user interest as a key input for recommendation is not considered relevant anymore - details of the implementation are listed in appendix 5.A for reference, but they will not play a role for year 3. A description of the test data can be found in the TE-506 section of D.7.3.2 *Use Cases: Validation Report (First Evaluation 2015)*.

The second main difference is the large amount of projects inside Zooniverse – 52 at the time of this writing – which makes it impossible to cover all of them in the scope of this project. Therefore, the focus is on Snapshot Serengeti, and all the implementation will focus on this Zooniverse showcases. However, the developed architecture and software components will be reusable for other projects within Zooniverse.

US-65 to US-68 are applicable to any project, and US-69 refers to "blanks" and other concepts which are Snapshot Serengeti-specific, but its general goal of recommending subjects could be adapted to any other Zooniverse project.

5.5.2.1 Workflows

As described above, Zooniverse will be developing a new component, Nero, for Zooniverse-specific recommendation. This will be built on Panoptes, the new Zooniverse back-end, rather than Ouroboros, the legacy back-end. As such, we expect Snapshot Serengeti to be ported to the new infrastructure in order to allow to continue the respective studies.

Panoptes posts classifications made by users into an Apache Kafka message bus as JSON objects. Nero will pick up and respond to classifications from this message bus. In response to classifications, Nero sends the current metadata about the project, subject and user belonging to that classification to an algorithm, which will take appropriate action as configured. This algorithm might be built into Nero or could be a separate service, e.g., an HTTP web service that takes a POST request with JSON data, and returns an updated version of that JSON data, along with actions to be taken (like enqueue this subject for users [x,y,z]; or stop showing this subject to users). It is required to have this process be stateless to ease scalability and deployment (e.g., to AWS Lambda).

Recommendation in the Zooniverse showcase will be a two-part process, with content-based recommendations from MICO being further enhanced and harness in a real-time application situation using the Zooniverse-specific recommendation engine, Nero. Nero will handle the business of modifying the user experience in real time, generating new recommendations and acting upon them dynamically. This cannot be done at the MICO platform level without building a lot of Zooniverse business logic into the platform and sacrificing code agility and execution efficiency.

5.5.2.2 Interfaces

All communiation between Zooniverse components and the MICO recommendation API will happen via a HTTP REST API. We will investigate the possibility of building awareness of the Zooniverse concepts of subjects (collections of images and talk comments) and users (posters of comments, actors of classifications) into the MICO API, so that Zooniverse might be able to ask a question of the API such as "please give me the current insights and metadata for this Zooniverse subject, or for this Zooniverse user". The inputs that Zooniverse would expect to provide in a request are:

- user ID or subject ID
- specific query parameters/filters (equivalent to e.g. “I only want to know about animal species” or “I want to know about user skill”) The outputs that would be returned could include any/all of:
- a grouping or expertise level for a user
- a list of interests, topics or vocabulary used by that user
- a list of subjects matched to that user
- a list of likely species contained in the images of a subject
- a list of interesting highlights from discussions about a subject (e.g. user disagreement, sentiments such as doubt, frustration, strong positive feelings, etc)
- other related ideas as the use cases develop

In order to answer such questions, it is expected that the MICO recommendation engine needs to pre-process many of the data sources detailed under “data available” above, especially the talk and subject content data, and outputs from sentiment and image analysis extractors. Hence, it may be reasonable to create an API interface so that Zooniverse can upload these, or alternatively, the recommendation engine could provide a mechanism for ingesting these data sources.

This also implies that the platform needs the ability to cope with “new data”, and thus all answers given back to Zooniverse must indicate which version of the data sources were used to derive their answer.

5.6 Recommendation API reference

This section defines the recommendation module interface that is exposed for the showcases, and the sequence diagrams and REST interfaces in the following represent the functionality required to address the aforementioned user stories.

The interfaces presented here are simplified in that they do not include user management and access control. To protect the system from non-authorized access, every request will be secured with an access token. OAuth v1 ([HL10]) and an approach for granting access of resources to different clients will be implemented for this purpose.

Apart from specialized cross-media recommendation, the recommendation engine also provides basic collaborative filtering capabilities that can be used by all showcases that have user event data to analyse (e.g., views, clicks, likes, etc.). Figures 28 and 29 specify the generic components. Recommendation that includes data from the WP2 extractors is implemented separately. Figure 30 shows the basic REST interface for getting recommended items for a currently edited article. Figure 31 shows the request for getting videos that might be interested for a certain Shoof user as described in US-63: *Shoof cross media recommendation*. The recommendation interfaces for the Snapshot Serengeti showcase will be developed along the guidelines given in section 5.5.2.2.

The following table provides a summary of the specified API and the covered user stories:

Interface	**Related User Story**
Figure 28: Sending user event data	US60 - US64
Figure 29: Simple item Recommendation	US60, US61, US64
Figure 30: Greenpeace editor support	US62
Figure 31: Shoof Cross Media	US63

Figure 28 Submitting user behaviour data to the recommendation API (**US60 - US64**)

This call submits a list of user events to the recommendation module. The recommendation engine will start retraining upon new data being sent to the system, as long as there is no running training job. The showcase site can access the current status of the engine to determine whether the transmitted data is already included in the recommendation.

```
POST reco_api_endpoint/<showcase>/userevent
```

```
events: {
    [
        {
            "event_type": <Event Type>
            "user_id": <User Id>
            "item_id": <Item Id>
            "time_stamp": <Time Stamp>
            "properties": {<Custom Properties>}
        },
        {
            "event_type": ...
            "user_id": ...
            "item_id": ...
            "time_stamp": ...
            "properties": ...
        },
        ...
    ]

}
```

Figure 29 Simple item recommendation. (**US-60**, **US-61**, **US-64**)

This call retrieves a number of recommendations based on user behavior data transmitted to the platform. It provided, the recommendation is customized for a user (US-61: *Subscriber Recommendation*) or generic based on the behavior of all known users (US60: *Non-Subscriber recommendation*.)

```
GET reco_api_endpoint/<showcase>/simplereco?entityId=<entity id>&userId=<user id>&limit=<limit>
```

```
response: {
    meta: <technical meta data>
    recommendations: [          // array of objects sorted by confidence/score
        {
            entity: <entity id>      // the entity driving the recomendation
            item: <recommended item id>    // the recommended item
            confidence: 0.8765          // the confidence/score
        },
        {
            entity: ...
            item: ...
            confidence: ...
        },
        ...
    ]
}
```

Request parameters:

entity id The id of the item to get recommendations for (e.g., news article or video)

user id Optional: The id of the user to get recommendations for.

limit The maximum number of items returned.

Response parameters:

entity id The id of the item the recommendation is for

entity id The id of the recommended item

confidence Confidence value of the recommendation, Range: (0, 1]

meta Additional showcase dependent data

Figure 30 Greenpeace editor support (**US-62**)

This call retrieves videos and images related to the current article.

```
GET reco_api_endpoint/gp/editorsupport?articleId=<article_id>
```

```
response: {
    meta: <technical meta data>
    recommendations: {
      related_articles:
        [
          {
            <references to related articles from greenpeace magazine>
          }, ...
        ],
        related_media:
        [
          {
            <references to related video or audio items>
          }, ...
        ]
    }
}
```

Figure 31 Shoof Cross Media Recommendation (**US-63**)

This call retrieves content relevant for the specified Shoof user.

```
GET reco_api_endpoint/shoof/mediareco?userId=<user_id>
```

```
response: {
    meta: <technical meta data>
    recommendations: [          // array of objects sorted by
    confidence/score
        {
            entity: <entity id>      // the entity driving the recomendation
            item: <recommended item id>    // the recommended item
            confidence: 0.2333456          // the confidence/score
            source: "video-asr-ner"
        },
        {
            entity: ...
            item: ...
            confidence: ...
            source: ...
        },
        ...
    ]
}
```

5.7 Deviations from D5.2.1: First specifications

The following outlines some clarifications of points that were mentioned in previous WP5 specification documents, by commenting on design choices in D5.2.1 for relevant technology enablers. The main source for respective changes were updated requirements, and the focus on user stories instead on showcase-independent technology enablers.

To signal which text passages are taken from D.5.2.1, they are *framed*.

5.7.1 TE-501. User Activity and Context Monitor

> *The main outcome will be a Activity Monitoring API implemented as REST services integrated in the MICO platform. The API will be very simple and will allow integrators to send users' preferences data in a concrete fixed format. The data structure used for representing users' preferences must fit with the proposed recommendation models for WP5 but also must be consistent with the representations foundations in MICO. For those reasons, a reasonable approach is to use an ontology for representing users preferences.*

Comment:
There is no dedicated Activity Monitoring API: For the Zooniverse showcase, this task was taken over by *Geordi*, which has direct access to the Zooniverse platform. For the needs of the InsideOut showcases, the activity monitoring was integrated into the *Recommendation API*, using the storage back-end provided by the Prediction.io event server.

5.7.2 TE-502. User Similarity Calculator

> *Correlation or vector-based implementations are nowadays provided by many open-source Recommender systems like Apache Mahout, Prediction.io or MyMediaLite. All these implementations use Ratings data for computing the similarity between users, using boolean or numeric values as rates. Also, most of these frameworks allow to define custom or complementary domain-dependent similarity functions. Independently of the selected framework for the final implementation, it is important to expose an API for easing the inclusion of custom data for calculating the similarity between users. [...]*

Comment:
No changes. Prediction.io was chosen as a collaborative filtering based recommender for WP5. Its functionality is exposed via an API as described in section 5.6.

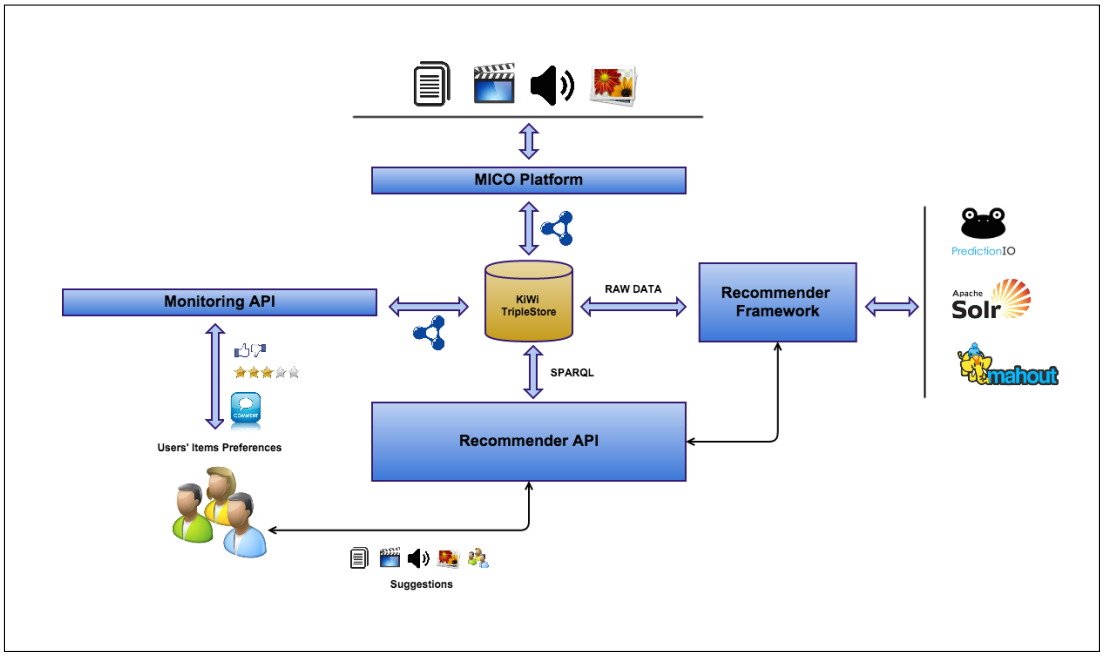
5.7.3 TE-503. Item Similarity Calculator

> *As stated in section 4 (Preliminaries and Basics for the Metadata Model), all the extracted features from the content analyzed in MICO will be stored as triples following the specified aforementioned Metadata Model. These features will be part of the items' vectors that will feed the similarity functions. [...]*

Comment:
No changes. Extracted metadata is stored as triples inside the platform's Marmotta which is queried using Anno4j.

5.7.4 TE-504. Cross-Modal Content Recommender

D5.2.1 proposed the following architecture for a cross-modal content recommender:



Comment:
The architecture diagram from section 5.4 differs mainly with respect to the presentation and inclusion of showcases. However, the main components and their connectivity remained the same:

Mico Platform No changes. The cross media recommendation relies heavily on the work done by the extractors.

KiWi Triple Store No changes The KiWi triplestore is used within Marmotta. It is preferably queried using Anno4j, SPARQL queries remain possible.

Recommender Framework Prediction.io is used as a collaborative filtering framework. There was no need to integrate Apache Solr and Mahout in addition to that.

Recommender API No changes.

Monitoring API Integrated to the Recommender API, see description on TE-501

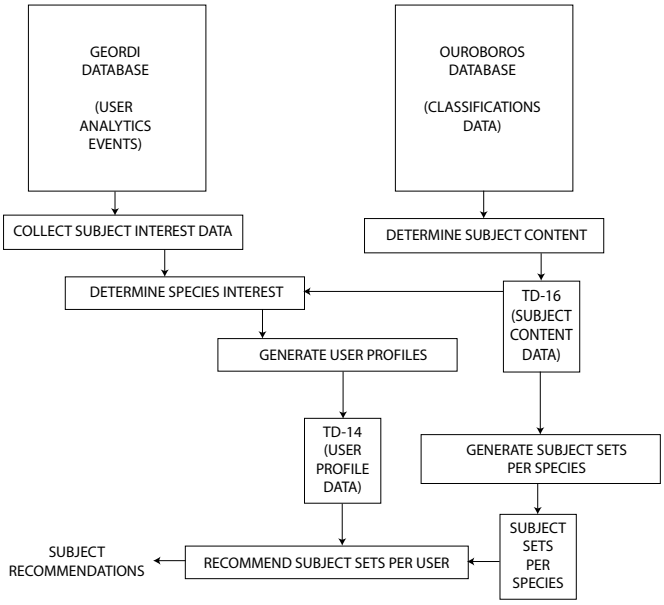
5.8 Work planned for year 3

To summarize, the main goals for WP5 in year 3 are

- to complete the general recommendation framework, by implementing the remaining components, as outlined in the previous chapters
- to fully integrate the framework into the MICO platform, including stable interfaces between WP5 and the MICO extractor and broker infrastructure
- to complement the above with an implementation of the showcase-specific components needed for cross-media evaluation for the prioritized user stories

5.A Appendix: Subject recommendation for Snapshot Serengeti

The following diagram illustrates how recommended subjects are derived for each user based on just two things, user analytics data from our *Geordi* database, and classifications data from the *Ouroboros* database:



Each of the narrow rectangles represents a different stage of the scripting/querying process. These will be referred to below. Each of the larger rectangles represents an intermediate MySQL data table or CSV.

5.A.1 Test that species preferences match user behaviour (TP-506-01)

This test focussed on validating that a recommendation system trained upon user analytics data and species content data could correctly identify a user's species preferences.

First, subject species content (TD-16) was calculated for all past Snapshot Serengeti users using a combination of MySQL queries and Python scripts ["DETERMINE SUBJECT CONTENT"]

Figure 32 DETERMINE SUBJECT CONTENT - Pseudocode

```
for each subject
  combine all user classifications for this subject
  find the mode species (ie. that which the most users specified)
  store this as the dominant species for this subject
next subject
```

This was done by aggregating all user classifications for the subjects and taking the most commonly occurring answer, or "blank", as the correct indication of subject content - as shown in the pseudocode above. Where more than one species was present (only 6% of images) we take the more numerous species.

Now that we have this data, we are then able to determine users' preferred species by looking at which subjects they prefer. The general approach for generating TD-14 was as follows (again this was done using a combination of MySQL queries and Python scripts).

The procedure was

- Identify and tally indications of positive interest in a subject from the analytics data ["COLLECT SUBJECT INTEREST DATA" in diagram above]. We included the following user events:
  - User favourites the subject
  - User shares the subject via Facebook, pinterest or Twitter
  - User views the discussion page for a subject
  - User views the map for a subject

Figure 33 COLLECT SUBJECT INTEREST DATA - Pseudocode

```
for each user
  for each "positive interest" user analytics event
    increment interest counter for that subject for that user
  next analytics event
  store total interest by subject for each user
next user
```

- Convert each tallied indication of interest in a subject, to an indication of interest in the dominant species (or "blank") of that subject ["DETERMINE SPECIES INTEREST" in diagram above]. Each indication of interest is assigned a simple numerical score of 1.

Figure 34 DETERMINE SPECIES INTEREST - Pseudocode

```
for each user
  for each subject with positive interest
    determine which species is present (from TD-16)
    increment interest counter for that species for that user
  next subject
  store total "expressions of interest" per species for this user
next user
```

- Assign a score per user, per species, based on the sum of total expressions of interest in that species by that user ["GENERATE USER PROFILES" in diagram above]. So for example, if a user favourites an image of a zebra and shares two images containing lions, they would score 2.0 for lions and 1.0 for zebra. A higher score indicates a stronger interest in that species, and the two highest scoring species for a user can be deemed as that user's "favorite" species.

Figure 35 GENERATE USER PROFILES - Pseudocode

```
for each user
  for each species with positive interest
    add an entry in that user's profile that they like that species
    count number of expressions of interest for that species
    assign that number as a score to the user profile entry
  next species
next user
```

5.A.2 Test that the subjects recommended match the preferred species (TP-506-02)]

This test focussed on ensuring that the correct subjects would be recommended (in line with the calculated species preferences for each user) for a user once that user's species preferences were known. Subject recommendation was performed as follows, using a combination of MySQL queries and Python scripts:

- A random set of 100 subjects was chosen for each species and "blank", using the TD-16 data. We ensured that these images did not contain other secondary species.

Figure 36 GENERATE SUBJECT RECOMMENDATION SET - Pseudocode

```
for each species
  initialize subject set for this species
  while subject set contains < 100 items
    current subject = random subject
    while current subject does not contain this species
      pick another random subject
    end while
    add this subject to the subject set
  end while
next species
```

Figure 37 RECOMMEND SUBJECT SETS PER USER - Pseudocode

```
for each user
  create a subject recommendation set
  pick the two species with highest score from TD-14
  for each favourite species
    pick 20 random subjects from the subject set for that species
    add these subjects to the recommendation set for that user
  next favourite species
next user
```

5.B MICO WP5 Platform Integration

To make testing and deployment more easy, Prediction.io with a customized MICO recommendation template is provided encapsulated as a docker image, derived from the official prediction.io dockerfile on GitHub (`https://github.com/sphereio/docker-predictionio`).

5.B.1 Install on platform

```
$ sudo apt-get install curl
$ curl -sSL https://get.docker.com/ | sh
$ sudo usermod -aG docker user
```

Logout, Login again

5.B.2 Running prediction.io inside a docker container

Docker accepts git repositories instead of local files as references to Dockerfile & Co. This allows us to store the Dockerfile inside the project's Bitbucket. In a public release this will be published on a public repository, making this adjustment step only required for development.

```
$ docker build --tag="wp5"
  https://user@bitbucket.org/mico-project/recommendation.git#master:platform/docker
$ docker run -p 8000:8000 -p 7070:7070 -p 9000:9000 wp5
```

The Engine is deployed to port 7070, an overview page of the Prediction.io status is available on port 8000. Scripts for training the engine with Greenpeace Magazine data are accessible from the Dockerfile.

5.B.3 Testing

On build, a minimal set of entries (*user buys item*, *user rates item*) is given to the engine as training data. Alternatively, the REST endpoint can be used directly using `curl`. To Retrieve *num=2* recommendations for user *user=1*:

```
$ curl -H "Content-Type: application/json" \
       -d '{ "user": "1", "num": 2 }' http://localhost:8000/queries.json
```

Resulting in something similar to:

```
{
   "itemScores":[
      {
         "item":"2",
         "score":3.996582770292206
      },
      {
         "item":"0",
         "score":1.3457
      }
   ]
}
```

References

[Aic+15] Patrick Aichroth et al. "MICO – Media in Context." In: *Proceedings of the 2015 IEEE International Conference on Multimedia and Expo (ICME)*. Torino, Italy, June 2015.

[All83] J.F. Allen. "Maintaining Knowledge About Temporal Intervals." In: *Communications of the ACM* 26.11 (1983), pp. 832–843.

[CFO93] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. "A Small Set of Formal Topological Relationships Suitable for End-User Interaction." In: *SSD*. Ed. by David J. Abel and Beng Chin Ooi. Vol. 692. Lecture Notes in Computer Science. Springer, 1993, pp. 277–295. URL: http://dblp.uni-trier.de/db/conf/ssd/ssd93.html#ClementiniFO93.

[DR14] Michael Dyck and Jonathan Robie. *XML Path Language (XPath) 3.1*. W3C Working Draft. http://www.w3.org/TR/2014/WD-xpath-31-20140424/. W3C, Apr. 2014.

[DT05] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection." In: *Computer Vision and Pattern Recognition*. 2005.

[Fel+10] P.F. Felzenszwalb et al. "Object Detection with Discriminatively Trained Part Based Models." In: *IEEE Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645.

[Fer01] Jon Ferraiolo. *Scalable Vector Graphics (SVG) 1.0 Specification*. W3C Recommendation. http://www.w3.org/TR/2001/REC-SVG-20010904. W3C, Sept. 2001.

[Fou04] The Apache Software Foundation. *Apache Camel*. Oct. 2004-2015. URL: http://camel.apache.org/ (visited on 10/30/2015).

[GSP] Steve H. Garlik, Andy Seaborne, and Eric Prud'hommeaux. *SPARQL 1.1 Query Language*. http://www.w3.org/TR/sparql11-query/. URL: http://www.w3.org/TR/sparql11-query/.

[HL10] E. Hammer-Lahav. *The OAuth 1.0 Protocol*. RFC 5849. http://www.rfc-editor.org/rfc/rfc5849.txt. RFC Editor, Apr. 2010. URL: http://www.rfc-editor.org/rfc/rfc5849.txt.

[HS13] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. 2013. URL: http://www.w3.org/TR/sparql11-query/.

[KSK15] Thomas Kurz, Kai Schlegel, and Harald Kosch. "Enabling access to Linked Media with SPARQL-MM." In: *Proceedings of the 24nd international conference on World Wide Web (WWW2015) companion (LIME15)*. 2015. DOI: 10.1145/2740908.2742914.

[Lin+15] Chris Lintott et al. *D7.2.1 & D8.2.1 Combined Use Cases: First Prototype*. Deliverable. MICO, 2015. URL: http://www.mico-project.eu/wp-content/uploads/2015/08/Combined-Deliverable7.2.1-8.2.1UseCasesFirstPrototype1.pdf.

[LZ13] Jianguo Li and Yimin Zhang. "Learning SURF Cascade for Fast and Accurate Object Detection." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 3468–3475. ISBN: 1063-6919. DOI: 10.1109/CVPR.2013.445.

[MM04] Frank Manola and Eric Miller. *RDF RDF Primer*. W3C Community Draft. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. W3C, Feb. 2004.

[Poz13] *Annual Public Report 2013*. Tech. rep. Sharing AudioVisual language resources for Automatic Subtitling (SAVAS), 2013.

[PS04] Inc. Pivotal Software. *RabbitMQ - Messaging that just works*. Oct. 2004-2015. URL: https://www.rabbitmq.com/ (visited on 10/30/2015).

[RAR14] Josu Bermudez Rodrigo Agerri and German Rigau. "IXA pipeline: Efficient and Ready to Use Multilingual NLP tools." In: *Proceedings of the 9th Language Resources and Evaluation Conference (LREC2014) Reykjavik, Iceland* (2014).

[Sea10] Andy Seaborne. *SPARQL 1.1 Property Paths*. W3C Working Draft. http://www.w3.org/TR/2010/WD-sparql11-property-paths-20100126/. W3C, Jan. 2010.

[Soc+] Richard Socher et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)* ().

[Tro+12] Raphaël Troncy et al. *Media Fragments URI 1.0 (basic)*. W3C Recommendation. W3C, Sept. 2012.

[Ope11] Open Geospatial Consortium. *OGC GeoSPARQL - A Geographic Query Language for RDF Data*. Tech. rep. Open Geospatial Consortium, 2011.