The MICO Broker: An Orchestration Framework for Linked Data Extractors

Patrick Aichroth, Marcel Sieland, Luca Cuccovillo, and Thomas Köllmer

Fraunhofer Institute for Digital Media Technology IDMT,
Ehrenbergstraße 31, 98693 Ilmenau, Germany
patrick.aichroth|marcel.sieland|luca.cuccovillo|thomas.koellmer@idmt.fraunhofer.de

Abstract. This paper describes the *MICO broker*, a management and orchestration framework for Linked Data extractors. It outlines the initial version of the broker, illustrates the key challenges and requirements for extractor orchestration in the MICO project, and provides an improved MICO broker design and implementation that addresses these key challenges. The paper describes the interaction with the Linked Data approach applied in MICO for this purpose, especially regarding the broker data model, semi-automatic workflow creation and workflow execution.

Keywords: extractor orchestration, cross-media analysis, metadata extraction, linked data, workflow creation, workflow execution

1 Introduction

MICO[1] is a EU research project that provides a platform for distributed analysis of textual, image, video and audio content, including cross-media metadata extractors, a common metadata model, advanced metadata querying, cross-media recommendation, and persistence functionalities based on Linked Data.

One of the core challenges of the project is the development of the MICO *broker*, which is depicted as "service orchestration" within the overall architecture in Figure 1. The MICO broker includes all necessary technologies to orchestrate heterogeneous media extractors, thereby supporting complex analysis workflows which allow combination and reuse of otherwise disconnected results from standalone extractors, to achieve improved analysis performance.

An example MICO workflow (a so called *pipeline*) is described in Figure 2): It uses mp4 video containers as input, and consists of three partial workflows (see yellow/orange rectangles), which can also be invoked individually:

1. shot detection, and shot boundary and key frame extraction
2. face detection, which operates on extracted boundary or key frames
3. audio demux, speech2text and named entity recognition

The resulting annotations can be used for queries such as "Give me all shots in a video where a person says something about topic X". It is important to note that the described workflow could be further extended and improved, e.g., using speaker identification, face recognition, or extraction of metadata from the respective MP4 container or API where the content may have been crawled, all of which demonstrates: There is a huge potential in combining extractors which are typically used in isolation.

The following will describe the challenges, design and implementation of the MICO broker to support such workflows, thereby exploiting the Linked Data based MICO infrastructure: Section 2

[1] MICO project website: http://www.mico-project.eu/

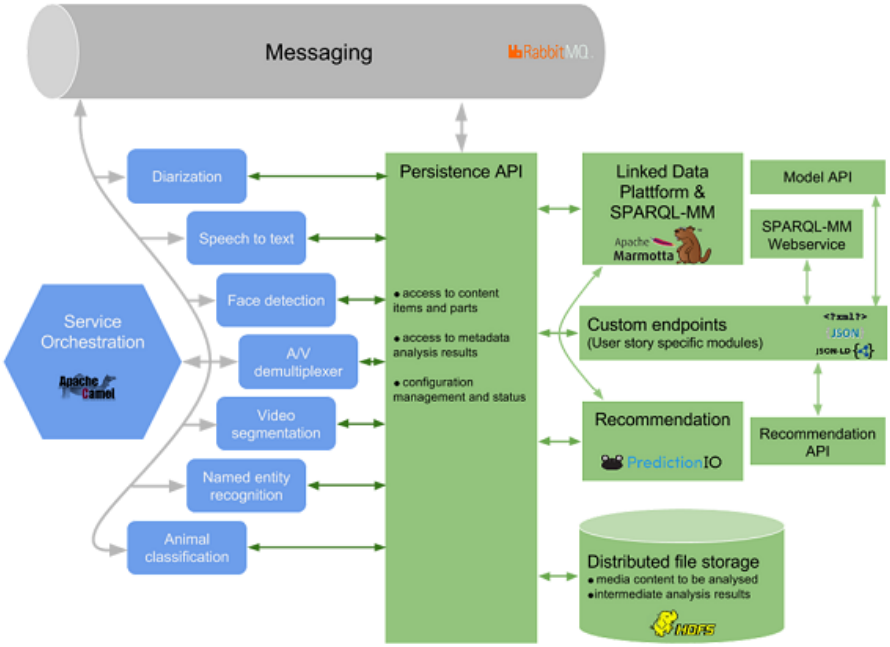

Fig. 1. overall MICO platform architecture

will describe the initial v1 MICO broker functionalities and limitations, and section 3 will outline the relevant requirements to be addressed. Section 4 will then provide an overview of the broker approach and its components. The related broker model that extends the MICO metadata model is outlined in section 5, and the approaches to workflow creation and execution are described in sections 6 and 7. Section 8 will provide conclusions and an outlook.

2 MICO broker v1: Initial work and limitations

The initial v1 of the MICO broker was implemented on top of RabbitMQ [3], following the principles of AMQP[2]. Extractor orchestration was implemented using two different message queues: A *content item input queue* for receiving new content, and a temporary *content item reply-to queue* created per content item, with each extractor providing results. For service registration, broker v1 provided a *registry queue* for analysis service registration, and a temporary *reply-to queue* that was created by each service discovery event, to take care of service registration and analysis scheduling. The implementation completely decoupled extractors from extractor orchestration, in order to support free choice of extractor programming language and potential distribution of extraction tasks. For convenience purposes, an *Event API* exposes a basic set of instructions for extractors to interact with the broker, available in both Java and C++ (other languages supporting AMQP could be used as well).

The described v1 proved to be quite stable and robust, especially regarding the RabbitMQ messaging infrastructure. However, extractor orchestration was based on simple, mime-type-based comparison: Upon registration of a new extractor process, *all* connections possible for that mime type were established, including unintended ones and loops. Hence, it was clear that further improvements were necessary.

[2] Advanced Message Queuing Protocol 1.0, `https://www.amqp.org/`

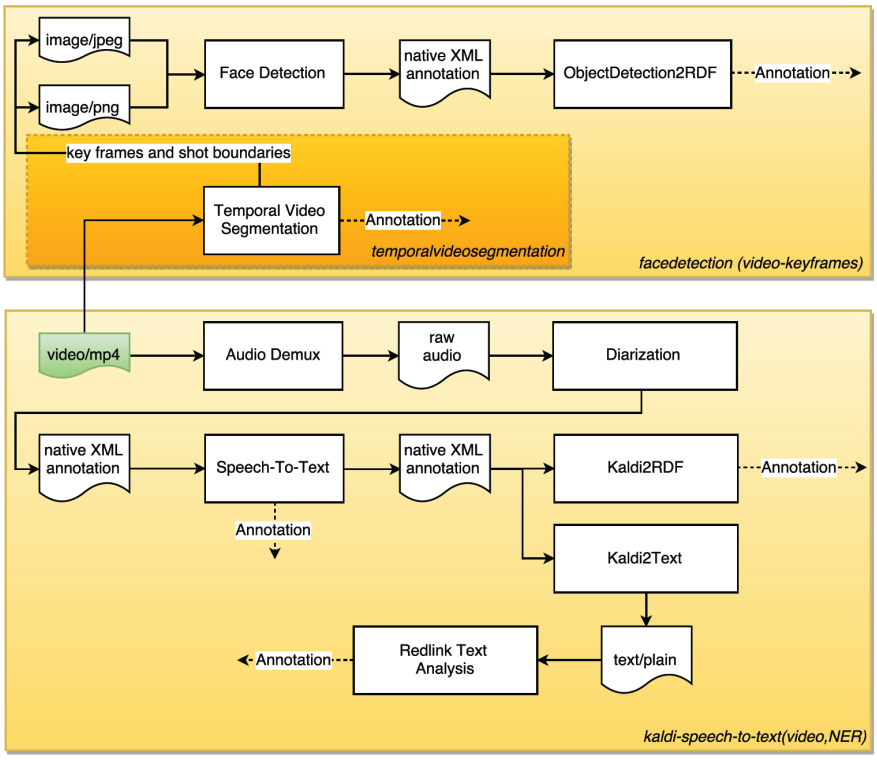

Fig. 2. example MP4 video analysis workflow in MICO

To address some of the most pressing questions, while still keeping compliance with the Event API, some instant improvements were provided shortly after broker v1. The so-called *pipeline configuration tools*, consisting of a mixture of bash scripts and servlet-based Web UI configurations support

- Standard extractor parameter specification
- User-controlled pipeline configuration
- User-controlled service startup/shutdown.

Beyond this, however, the project soon required a more advanced approach for workflow orchestration and execution, including a distinction between syntactical versus semantic input and output of extractors (substituting the simple mime type approach), support for multiple inputs and outputs, extractor parametrization, a more usable approach to defining workflows, and support for dynamic routing in workflows and general support for EIP (Enterprise Integration Patterns) for workflow execution.

3 MICO Broker requirements

Based on broker v1 experiences, an extensive list of requirements for orchestration was compiled, and then prioritized, resulting in the following key points:

1. *General requirements* including backward-compatibility regarding the existing infrastructure (in order to reduce efforts for extractor adaptation, especially regarding RabbitMQ, but also

the Event API wherever possible); reuse of established existing standards / solutions where applicable; and support for several workflows per MICO system, which was not possible with v1.
2. Requirements regarding *extractor properties and dependencies*: support for extractor configuration, and support for different extractor "modes", i.e., different functionalities with different input, output or parameter sets, encapsulated within the same extractors; support for more than one extractor input and output; support for different extractor versions; and distinction between different I/O types: mime type, syntactic type (e.g., image region), semantic concept (e.g., human face).
3. Requirements regarding *workflow creation*: avoiding loops and unintended processing; extractor dependency checking during planning and before execution; simplifying the workflow creation process (which was very complicated).
4. Requirements regarding *workflow execution*: error handling, workflow status and progress tracking and logging; support for automatic process management; support for routing, aggregation, splitting within extraction workflows (EIP support); and support for *dynamic routing*, e.g, for context-aware processing, using results from language detection to determine different subroutes (with different extractors and extractor configurations) for textual analysis or speech2-to-text optimized for the detected language.

4 Overview: MICO broker v2 and v3

MICO broker v2 and v3 address the requirements outlined in section 3. V2 focuses on changes and extensions to the Event API, especially related to error management, progress communication, provision of multiple extractor outputs, and registration. The goal of v2 was to provide an earlier API update, which gave extractor developers an opportunity to adapt to it, also considering overall data model changes which are also influenced by the new broker model described in section 5. V3, in contrast, was meant to focus on the addition of new broker components for registration, workflow creation and execution.

The following describes the principles and assumptions for design and implementation, and the components used to provide the respective functionalities.

4.1 Principles and Assumptions

Regarding **extractor registration and model**, assumptions and principles include:

- Some parts of extractor information can be provided during packaging by the developer (extractor I/O and properties), while other parts can only be provided after packaging, by other developers or showcase administrators (semantic mapping, and feedback about pipeline performance): Registration information is provided at different times.
- Extractor input and output should be separated into several meta types (a) *mime-types* e.g., 'image/png', (b) *syntactic types* e.g., 'image region', and (c) *semantic tags* e.g., "face region". Mime-types and syntactical types are pre-existing information that a extractor developer / packager can refer to using the MICO data model or external sources, while semantic tags are subjective, depending on the usage scenario, will be revised frequently, and are often provided by other developers or showcase administrators. Often, they cannot be provided at extractor packaging time, nor do they need to be, as they do not require component adaptation. As a consequence, different ways of communicating the various input and output types are needed.
- A dedicated service for extractor registration and discovery can address many of the mentioned requirements, providing functionalities to store and retrieve extractor information,

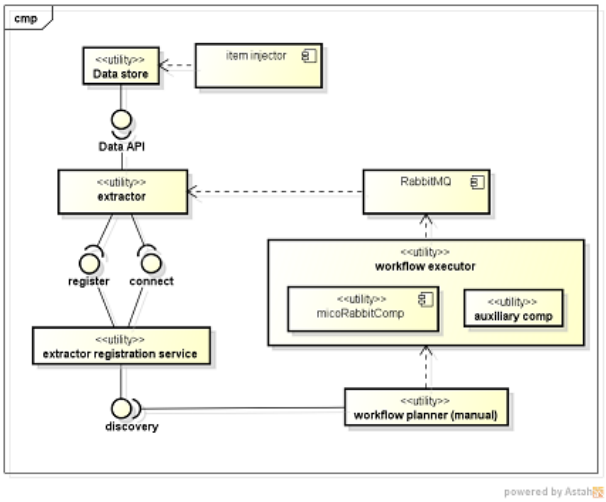

Fig. 3. MICO broker components

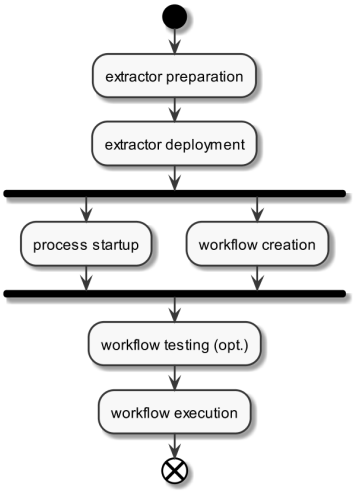

Fig. 4. MICO extractor lifecylcle

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 using Linked Data. Workflow planning and execution can reuse this information for their purposes.
– Existing Linked Data sources and the MICO metadata model (MMM) should be reused as far as possible, e.g., for syntactic types, but that related information should be cached by broker components for performance reasons; wherever applicable, extractors and extractor versions, types etc. should be uniquely identified via URIs

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

– Apache Camel is a good choice for workflow execution, supporting many EIP and all core requirements for the project. It should be complemented by MICO-specific components for retrieving data from Marmotta to put them into Camel messages, in order to support dynamic routing.
– The broker should not deal with managing scalability directly, but allow later scalability improvements by keeping information about extractor resource requirements, and allowing remote extractor process startup and shutdown.
– Manual pipeline creation is a difficult process, due to the many constraints and interdependencies, depending on the aforementioned types, but also content, goal, and context of an application. Considering this, we found that it would be extremely desirable to simplify the task of pipeline creation using a semi-automatic workflow creation approach that considers the various constraints. Additionally it is supposed to store and use feedback from showcase admins on which extractors and pipelines worked well for which content set and use cases.
– We need to support content sets and the mapping of such sets to their specific workflows.

4.2 Broker Components

The resulting broker includes and interacts with the components depicted in Figure 3. As mentioned above, the **registration service** provides a REST API to register **extractors** (which

produce or convert annotations and store them as Linked Data / RDF), thereby collecting all relevant information about extractors. It also provides global system configuration parameters (e.g., the storage URI) to the extractors, and retrieval and discovery functionalities for extractor information that are used by the **workflow planner**. The workflow planner is responsible for the semi-automatic creation and storage of workflows, i.e., the composition of a complex processing chain of registered extractors that aims at a specific user need or use case. Once workflows have been defined, they can be assigned to content sets. Finally, the **item injector** is responsible of injecting content sets and items into the system, thereby storing the input data, and triggering the execution of the respective workflows (alternatively, the execution can also be triggered directly by a user). Workflow execution is then handled by the **workflow executor**, which uses Camel, and a MICO-specific **auxiliary component** to retrieve and provide data from the data store to be used for dynamic routing within workflows. Finally, all aforementioned Linked Data and binary data is stored using the **data store**.

4.3 Extractor Lifecyle

From an extractor perspective, the high-level process can be summarized as depicted in Figure 4: **Extractor preparation** includes the preparation and packaging of the extractor implementation, including registration information that is used to automatically register the extractor component upon **extractor deployment**, and possible test data and information for the extractor. As soon as extractor registration information is available, it can be used for **workflow creation**, which may include **extractor / workflow testing** if the required test information was provided earlier. For planning, or the latest for execution, the broker will then perform an **extractor process startup**, and **workflow execution** will then be performed upon content injections or user request, as outlined above.

The following sections will provide more details on the broker model, workflow planning and execution, thereby clarifying how Linked Data is exploited within these domains.

5 MICO broker model and Linked Data

The data model of the MICO broker was designed to capture the key information needed to to support extractor registration, workflow creation and execution, and collecting feedback from annotation jobs (i.e., processing workflows applied to a defined content set), thereby addressing the general principles outlined in section 4, and considering the key requirements from section 3. It uses URIs as elementary data and extends the MICO MetadataModel (MMM)[3] presented in [1, ch. 3]. The broker data model is composed of four interconnected domains, represented by different colors in figure 5, which are described in the following:

- The **content description** domain (yellow) with three main entities: *ContentItem* captures information of items that have been stored within the system. As described in [1, ch. 3.2], MICO items combine media resources and their respective analysis results in *ContentPart*. *ContentItemSet* are used to group several *ContentItem*s into one *ContentItemSet*. Such a Set can be used, to run different pipelines on the same set, or to repeat the analysis with an updated extractor pipeline configuration.
- The **extractor configuration** domain (blue) with two main entities: *ExtractorComponent*, which captures general information about registered extractors, e.g., name and version and *ExtractorMode*, which contains information about a concrete functionality (there must be at least one functionality per extractor), which includes information provided by the developer

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

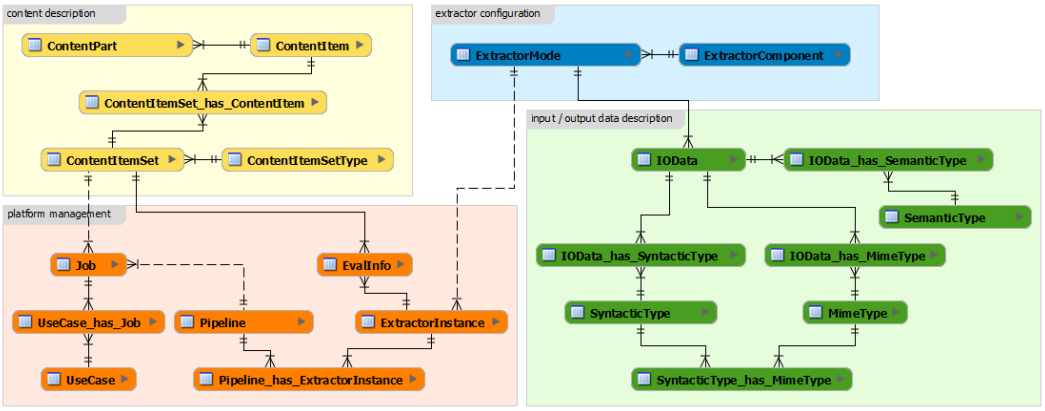

Fig. 5. MICO broker model overview

at registration time, e.g., a human-readable description and configuration schema URI. For extractors which create annotations in a format different from RDF, it includes a output schema URI.

- The **input/output data description** domain (green) stores the core information necessary to validate, create and execute extractor pipelines and workflows: *IOData* represents the core entity for the respective input or output to a given *ExtractorMode*.
  MimeType is the first of three pillars for workflow planning, as outlined in Section 4.1. RDF data produced by extractors will be labeled as type *rdf/mico*. textitIOData_has_MimeType connects I/O data to MimeType. It has an optional attribute *FormatConversionSchemaURI* which signals 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).
  The *SyntacticType* of data required or provided by extractors is the second pillar for workflow planning. For MICO extractors which produce RDF annotations, the stored URI should correspond to an RDF type, preferably to one of the types defined by the MICO Metadata model([1, ch. 3.4]). For binary data, this URI corresponds to a Dublin Core format [2].
  SemanticType is the third pillar for route creation and captures high-level information about the semantic meaning associated with the I/O data. It can be used 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.
- The **platform management** domain (orange) combines several instances related to platform management: *ExtractorInstance* is the elementary entity 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 the parameter and I/O data selection and further information stored during the registration by the extractor itself.
  EvalInfo holds information about the analysis performance of an *ExtractorInstance* on a specific *ContentItemSet*. This can be added by a showcase administrator to signal data sets for which an extractor is working better or worse than expected.
  Pipeline captures the URI of the corresponding workflow configuration, i.e., the composition of *ExtractorInstance*s and respective parameter configuration.
  UseCase is a high-level description of the goal that a user, e.g., showcase administrator, wants to achieve.
  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 relation that connects a *UseCase* 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 *ContentItemSet*

As outlined in 4.1, a key broker assumption 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 or showcase administrators. The registration service is one central point to register and query that extractor information, which provides the information needed for workflow planning (see section 6) and execution (see section 7). The registration service provides 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.

6 Semi-automatic Workflow creation

During the MICO project, the manual creation of workflows turned out to be more complicated and difficult than expected, as it depends not only on extractor interdependencies and constraints on multiple levels, but also on the content at hand, and the context and goal of an application. In order to address this problem, the idea of a semi-automatic workflow creation process emerged. It was implemented using the idea of finding possible combinations of matching extractors using the Linked Data *information pillars* outlined in sections 4 and 5. *MimeType* and *syntacticType* signal syntactical interoperability, and *semanticTags* signal a semantic match. Beyond that, if available, feedback on how well workflows performed on content sets can be used as well.

It is important to note that these pillars do not represent a simple hierarchy: For instance, the indication of two extractors providing and consuming a matching *mimeType* and *syntacticType*, but lacking the same *semanticType* can be used to signal to the service which extractors *could* match and hence should be linked via a new *semanticType*, requiring human feedback to create this link. Vice versa, if it turns out that two extractors seem to provide similar output, as signalled by *syntacticType* and *semanticType*, but the *mimeType* does not fit, this can be exploited as a signal that a simple extension of the extractor to support a new *mimeType*, e.g., via format conversion, could do the trick to create interoperability.

Figures 6 and 7 provide screenshots of the current workflow creation tool for MICO. In this example, the creation process started from multimedia content with mp4 video and text, where the user could incrementally add suitable extractors proposed by the GUI using the aforementioned *pillars*. However, Figure 6 shows a workflows that is validated, while Figure 7 shows the same but invalid workflow, which is due to a slightly different audio demux extractor configuration: The latter does not provide the output of type audio/wav, which leads to an incompatibility that is signaled within the GUI. After completion, the user can store the resulting workflow as a Camel route, which can then be used to execute the workflow.

7 Workflow execution and dynamic routing

Once workflows have been created and stored as Camel routes, using the semi-automatic approach (see section 6), they can be assigned to content items and sets, and execution can be triggered via the item injector or the user / showcase admin (see section 4).

The actual workflow execution is performed using four main components, two of which were already mentioned in section 4): The *workflow executor* as master component, which uses the other components and is based on Apache Camel, and the *auxiliary component*, a MICO-specific

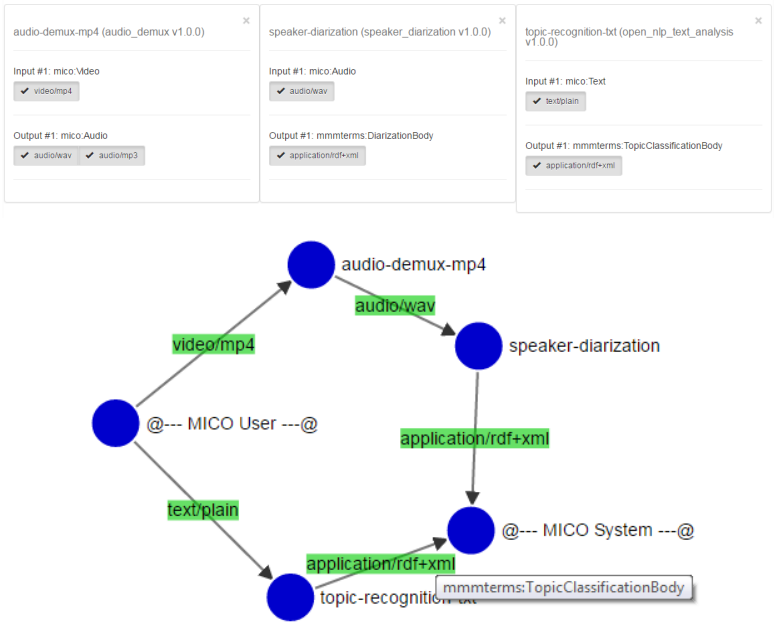

Fig. 6. A complete extractor workflow

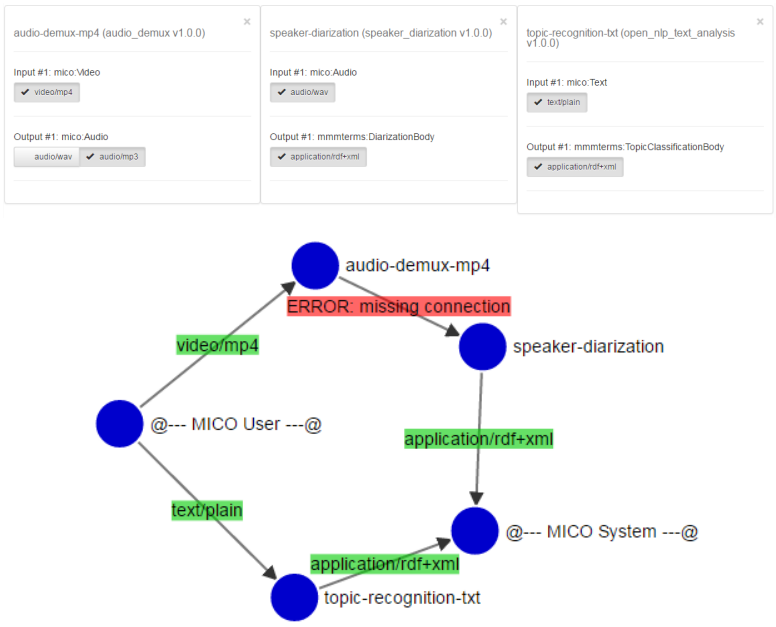

Fig. 7. An extractor workflow with one missing connection

extension to Camel that allows Linked Data retrieval to support dynamic routing. In addition, the *RabbitMQ* message broker serves as communication layer to loosely couple extractors and the MICO platform, and a MICO-specific Camel endpoint component that connects Camel with the MICO platform, and triggers extractors via RabbitMQ.

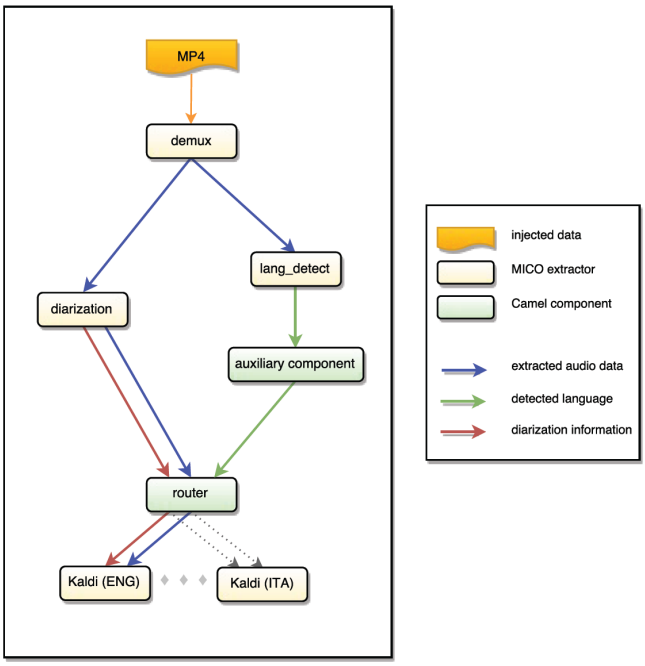

Fig. 8. MICO workflow execution with dynamic routing

Dynamic routing based on Linked Data works as depicted in the short example workflow for extracting spoken words from an mp4 video (figure 8). After audio demuxing (*demux*), the audio stream from the mp4 video is stored and provided to *diarization*[4] and language detection (*lang_detect*. Both analyze the audio content in parallel, and store their annotations in Marmotta.

At this point, dynamic routing is applied to optimize performance: The *auxiliary component* loads the detected language from Marmotta and puts it into the Camel message header – it knows where to locate the detected language, as *lang_detect* described by the storage location with its registration data via LDPath [4]. Afterwards, the language information within the Camel message header is evaluated by a *router* component, which triggers the *Kaldi* extractor optimized for the detected language. Beyond this example, there are many use cases where such dynamic routing capabilities can be applied.

8 Conclusion and outlook

This paper has described the challenges and requirements of cross-media extraction orchestration based on Linked Data within the MICO project, and how they were addressed with a mix of existing frameworks and MICO-specific extensions.

[4] The segmentation of audio content based on speakers and sentences.

While all major requirements could be met, there is still a lot of potential for future improvements: For semi-automatic workflow planning and creation, usability could be further enhanced, e.g., by allowing definition and combination of sub-graphs. Moreover, project experiences within the final project phase are likely to result in further demands with respect to process monitoring and management, and regarding scalability improvements.

Acknowledgements

This work has been partially funded by the European Commission 7th Framework Program, under grant agreement no. 610480.

References

1. Aichroth, P., Bjoerklund, J., Schlegel, K., Kurz, T., Köllmer, T.: Dx.2.2 Specifications and Models for Cross-Media Extraction, Metadata Publishing, Querying and Recommendations: Final Version. Deliverable, MICO (October 2015), http://www.mico-project.eu/wp-content/uploads/2016/01/Dx.2.2-SPEC_final_READY_FOR_SUBMISSION.pdf
2. Board, D.U.: DCMI Metadata Terms. Tech. rep., Dublin Core Metadata Initiative (jun 2012), http://dublincore.org/documents/dcmi-terms/
3. Pivotal Software, Inc.: Rabbitmq - messaging that just works (oct 2004-2015), https://www.rabbitmq.com/
4. Schaffert, S., Bauer, C., Kurz, T., Dorschel, F., Glachs, D., Fernandez, M.: The Linked Media Framework: Integrating and Interlinking Enterprise Media Content and Data. Proceedings of the 8th International Conference on Semantic Systems - I-SEMANTICS '12 (2012), http://dl.acm.org/citation.cfm?id=2362504