Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022), pages 2498–2506
Marseille, 20-25 June 2022

The CLAMS Platform at Work: Processing Audiovisual Data from the American Archive of Public Broadcasting

Marc Verhagen, Kelley Lynch, Kyeongmin Rim, James Pustejovsky
Department of Computer Science,
Brandeis University,
Waltham, Massachusetts,
{verhagen,kmlynch,krim,jamesp}@brandeis.edu

Abstract

The Computational Linguistics Applications for Multimedia Services (CLAMS) platform provides access to computational content analysis tools for multimedia material. The version we present here is a robust update of an initial prototype implementation from 2019. The platform now sports a variety of image, video, audio and text processing tools that interact via a common multi-modal representation language named MMIF (Multi-Media Interchange Format). We describe the overall architecture, the MMIF format, some of the tools included in the platform, the process to set up and run a workflow, visualizations included in CLAMS, and evaluate aspects of the platform on data from the American Archive of Public Broadcasting, showing how CLAMS can add metadata to mass-digitized multimedia collections, metadata that are typically only available implicitly in now largely unsearchable digitized media in archives and libraries.

Keywords: Multimedia platforms, Web services, Archives, Interoperability

1. Introduction

The CLAMS project provides access to tools that analyze multimedia content. The intended audiences for the platform are (1) researchers focused at processing multimedia content and (2) organizations like archives and libraries that seek to enrich their materials with metadata. It would be beneficial for those organizations to be able to process their sources and improve the search functionality that they offer their customers and the public at large. The processing tools that allow that need to be affordable and relatively easy to use, and ideally it should be possible to adapt the tools to the data in question.

In this paper we describe the first stable version of the CLAMS platform, following up on work in (Rim et al., 2019). We first give an overview of the system as it is now, pointing out the differences with its previous incantation. Then we describe some of the results of our work on the American Archive of Public Broadcasting (AAPB). We focus on what workflows were valuable for the AAPB data and which issues came up while processing the data. Finally we give an overview of related work.

2. CLAMS Architecture

The main ingredients of the CLAMS platform are the applications, the REST API, the MMIF specifications and its associated Software Software Development Kit (SDK), the workflow managers, and the visualization components. An overview of those components and some of their interactions is shown in Figure 1.

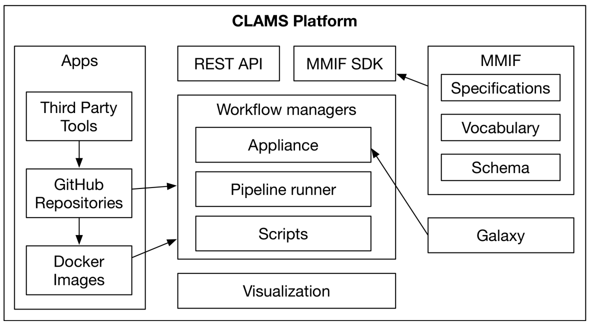


Figure 1: CLAMS Architecture

2.1. Applications

Applications are processing services that are typically, but not always, based on off-the-shelf publicly available processing tools with permissive licenses. We use the term *application* or *app* to refer to the software that takes a processing tool and embeds it as a CLAMS web service and we use *tool* to refer to the embedded processing tool.

CLAMS uses a microservices architecture where each application performs a specific processing task on video, audio, image or text data, or a combination thereof. Each application exposes a RESTful API on a Flask or Gunicorn server and has access to all CLAMS protocols, which include mechanisms for parameter passing, error handling, metadata definition and input data handling, as well as an interface to the MMIF interchange format. Typically, an application runs as a Docker container, but technically all the platform requires is access to a RESTful server listening to GET and POST requests and responding with either service metadata or an output file in the MMIF format.

For our current purposes, the most useful applications have been the following: (1) audio segmenters that distinguish between speech and non-speech, (2) Automatic Speech Recognition (ASR) and alignment of transcripts with video time frames, using Kaldi (Povey et al., 2011) and the Gentle forced aligner[1], (3) the post-ASR component that introduces capitalization and punctuation, which is not available in the Kaldi output, based on the Python fastpunct module[2], (4) components that recognize the bars-and-tone and slate sections in a video[3], (5) Optical Character Recognition (OCR) with EAST (Zhou et al., 2017) and Tesseract[4], and (6) Named Entity Recognition (NER) with spaCy.[5]

2.2. The Multi-Media Interchange Format

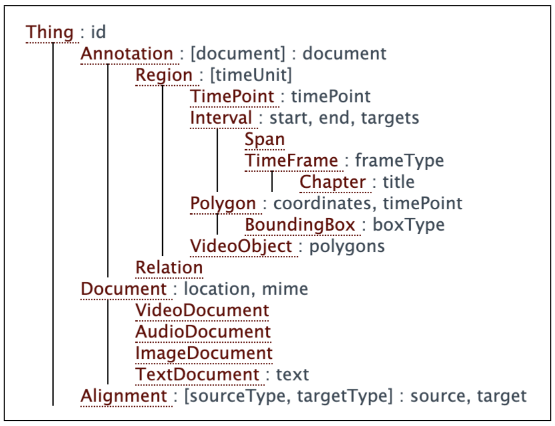
All applications use the MMIF format[6] to store annotations created by the processing tools. The format is a JSON-LD format based on the LAPPS Interchange Format (LIF) for text processing tools (Verhagen et al., 2015), which separated the primary text source from the annotations on the text. MMIF does the same but also allows for image, audio and video data. Primary data of all types are stored in the MMIF file as references to files and those files are considered read-only.

Annotations are added in views (sometimes referred to as annotation layers), which contain metadata and a list of annotations. Metadata include the creator of the view, a specification of the contents of a view in terms of what annotation types are contained in it, the creation date, and an error message if the application failed. Each annotation in the annotation list either references another annotation or is anchored to primary data by character offset, image coordinates, or time segments and time points. An application can add any number of views to a MMIF file, but views previously generated by other applications cannot be altered by the application.

```
{
  "@type": "http://mmif.clams.ai/vocabulary/TimeFrame",
  "properties": {
    "id": "f1",
    "start": 12,
    "end": 2385,
    "frameType": "bars-and-tones"
  }
}
```

Figure 2: Example annotation

All annotations have a type, which refers to a definition in the CLAMS vocabulary, which is part of the MMIF specifications[7], and a dictionary of properties, as shown in the example in Figure 2, which depicts a time frame with start and end offsets in milliseconds, where the time unit used is defined in the metadata of the view. Note that for many MMIF fragments in this paper we take some liberties and divert a bit from the MMIF standard, this is both for explanatory and space purposes.



```
Thing : id
    Annotation : [document] : document
        Region : [timeUnit]
            TimePoint : timePoint
            Interval : start, end, targets
                Span
                TimeFrame : frameType
                    Chapter : title
            Polygon : coordinates, timePoint
                BoundingBox : boxType
            VideoObject : polygons
        Relation
    Document : location, mime
        VideoDocument
        AudioDocument
        ImageDocument
        TextDocument : text
    Alignment : [sourceType, targetType] : source, target
```

Figure 3: The CLAMS Vocabulary

The vocabulary is purposely kept as small as possible and new categories are only added when the need arises, the screenshot of the vocabulary in Figure 3 actually shows the entire vocabulary. But application developers are free to define their own categories and use the `@type` field to point to their own definition. In fact, note there there are no categories in the CLAMS vocabulary that seem relevant to text annotations, this is because for text annotations we refer to categories already defined in the LAPPS vocabulary (Ide et al., 2014b).

Since all applications depend on reading and creating MMIF representations we have created a MMIF Software Development Kit (SDK) for application developers, which is available as a Python package.[8]

[1]https://github.com/lowerquality/gentle.

[2]https://github.com/notAI-tech/fastPunct.

[3]A slate is a board showing the identifying details of a take of a motion picture or TV program, which is held in front of the camera at its beginning.

[4]https://github.com/tesseract-ocr.

[5]https://spacy.io.

[6]Our data interchange format shares its acronym with a W3C open standard, Multimodal Interaction Framework by the W3C Multimodal Interaction Working Group. W3C-MMIF is a specification for Web accessibility on systems that have multiple modes of input and output devices (keyboard, haptic, voice, etc). W3C-MMIF is based on an XML-based markup language, Extensible MultiModal Annotation (EMMA), for the representation and interpretation of user input and production of system output. On the other hand, our MMIF format is a JSON-based specification for representing manual and automatic annotation on audiovisual multimedia data. Hence our MMIF and W3C-MMIF are completely different creatures.

[7]The current version as of April 2022 is available at https://mmif.clams.ai/0.4.0.

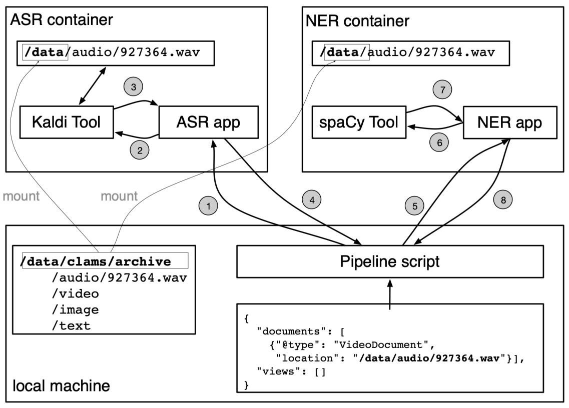
[8]https://pypi.org/project/mmif-python.

2.3. Processing Pipelines

It is perfectly viable to start up a single application with, say, the Tesseract OCR tool, and then programatically or via a command line[9] send MMIF files with POST request to that application. However, in many cases archivist's needs require a pipeline of applications. One of the pipelines that has been generally useful in our work is the following:

$$\text{Segmenter} \rightarrow \text{ASR} \rightarrow \text{post-ASR} \rightarrow \text{NER}$$

In this section we will run through an example of how applications interact in a pipeline, in particular we look at how a text processing tool is grafted upon an audio processing tool. For brevity's sake, we will take a shortened version of the pipeline above with just speech recognition and entity extraction. Consider the pipeline as represented in some more detail in Figure 4.



```
{
  "documents": [
    {"@type": "VideoDocument",
     "location": "/data/audio/927364.wav"}],
  "views": []
}
```

Figure 4: Simple CLAMS Pipeline

We have a MMIF file on the local machine that refers to primary data in the documents list. The pipeline script is part of the Pipeline Runner, which will be described in a little more detail below, but for now it is enough to know that the Pipeline script handles the interactions with the applications. The script fires off an HTTP POST request to the first application in the pipeline (step 1 in the diagram). Since it is the application that deals with the data, the MMIF file points at standard data locations on the Docker containers (by convention the `/data` directory), which is mounted to the data on the local machine. Note that while the MMIF file is attached to the request, no primary data are handed in.
The first application wraps the Kaldi speech recognition tool. It lifts the needed data from the MMIF file and the mounted primary sources and hands those data to Kaldi (step 2), which will process them on the container and hand the result back to the application (step 3), which will fold the results into the MMIF file in a new view and hand the view back to the pipeline script that sent the HTTP request (step 4).

[9]Using the curl command-line utility for transferring data to and from a server.

```
{
  "@type": "http://mmif.clams.ai/vocabulary/TextDocument",
  "properties": {
    "id": "td1",
    "text": "this is jim lehrer for the news hour" }
},
{
  "@type": "http://mmif.clams.ai/vocabulary/Alignment",
  "properties": {
    "id": "a1", "source": "d1", "target": "td1" }
}
```

Figure 5: Kaldi output – the text document

The new view will have a text document as an annotation type, which is aligned with the entire audio or video document that it was created from, as shown in Figure 5. Note that *TextDocument* is both a document type and an annotation type. As a document type it would be part of the multimedia primary sources that the MMIF document was created for, and as an annotation type it would be added to a view as the result of processing. But as we will see below this annotation can be input to further processing. This by the way is one of the reasons that annotations in views cannot be altered once they have been added because it could give rise to inconsistencies and the MMIF SDK ensures that it is impossible to do this.

```
{
  "@type": "http://vocab.lappsgrid.org/Token",
  "properties": {
    "id": "t1",
    "word": "this",
    "start": 0, "end": 4, "document": "td1" }
},
{
  "@type": "http://mmif.clams.ai/vocabulary/TimeFrame",
  "properties": {
    "id": "tf1",
    "start": 50, "end": 460 }
},
{
  "@type": "http://mmif.clams.ai/vocabulary/Alignment",
  "properties": {
    "id": "a2", "source": "tf1", "target": "t1" }
}
```

Figure 6: Kaldi output – tokens and time frames

In addition to the text document and its alignment to a primary source the ASR app also adds tokens and time frames in the audio that the tokens are aligned to, as shown in Figure 6.
Once the pipeline script gets these results back it takes the updated MMIF file and sends it off to the second application in the pipeline (step 5), which in our example wraps the spaCy named entity recognizer. At this point it is worthwhile to point out that although the two applications apply in sequence, the second application knows nothing about the pipeline and has no knowledge of what happened before. It does however know

what kind of data it wants to run on since the metadata associated with an application specify input and output requirements and the app uses the input requirements to mine the MMIF file for annotations in views that match the requirements. It then uses those views to extract the information that the embedded tools needs. One of the requirements of all language processing tools is that there be text document to process (and potentially the tool may require certain annotations to be available for those text documents. Since the spaCY NER app only requires text documents, it will look for text documents in both the primary sources and in each view, and then apply the spaCy tool to each of them apply, add results to a set of views in the MMIF file, and return the updated MMIF file (steps 6-8). Note that while the tool would have had access to the primary audio file, it was not instructed to do so by the NER app.

2.3.1. Workflow Management Components

Beyond using Unix shell scripts to run applications and manage pipelines, we have two more user-friendly ways to do this, the Pipeline Runner and the CLAMS Appliance.

```
data: ${PWD}/examples/data

pipeline:
  image: clams-pipeline
  container: pipeline

services:
  - asr:
      image: clams-kaldi-asr
      container: pipeline_asr
  - spacy:
      image: clams-spacy-nlp
      container: pipeline_spacy
```

Figure 7: Pipeline specification

The Pipeline Runner allows the user to define a workflow of CLAMS apps. It assumes that there is a prebuilt container image available for each application, which is a reasonable requirement since each CLAMS app is bundled with a Docker specification file that has all instructions needed to build the image. The user then defines a specification as in Figure 7, which gives the names of the images we want to use, the names of the running containers created from the images, and the local mount point into the `/data` directory on the containers. The Pipeline Runner then provides two Python scripts to start and run a pipeline. The first script uses the Docker compose command to start the containers and the second script takes care of managing the processing, as previously exemplified in Figure 4. The run script also allows addition of parameters for individual services, which could be specified when running the script or when creating the specification in Figure 7.

The CLAMS appliance is a tool that creates a CLAMS-Galaxy instance. Galaxy (Giardine et al., 2005) provides a web-based graphical user interface that allows users to import data, construct complex workflows, and explore and visualize the metadata generated by applying workflows to their data. The Galaxy platform was originally developed for genomic research, but has successfully been used for the deployment and integration of NLP tools (Ide et al., 2016), we extend it to multimodal data.

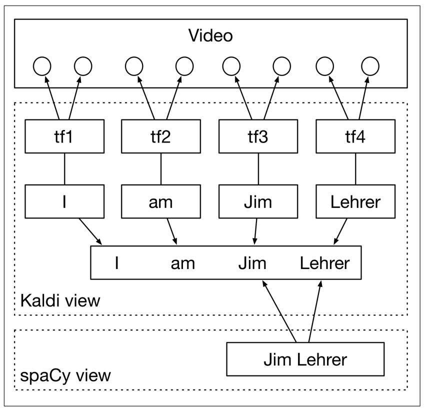


Figure 8: Linking a named entity to a video time frame

Configuring a CLAMS instance is done via a YAML file and the appliance will generate a container orchestration script (currently with the Docker compose command). The appliance configuration takes care of the local directory structure issue mentioned before. Users can also configure the appliance to include only select CLAMS apps that users need for their workflow. Within an appliance instance, workflow creation and manipulation is done via the Galaxy GUI, which means that upon deployment of the instance, users are guided to access Galaxy via a web browser of their choice to access all apps and start experimenting with workflows. Once a user is satisfied with the workflow they came up with after the experiments, they can use the CLAMS Pipeline Runner for running it in batch mode for a larger collection of media.

2.4. Summarization and Visualization

MMIF is designed to allow a variety of processing tools to add annotations using a common format. And because annotations are in the end always anchored to locations in the primary data you can view a MMIF file as a set of graphs rooted in primary data locations and therefore all graphs are connected to each other. However, the connections are often not explicit and it needs some traversing to connect, for example, a named entity to a time frame in an audio or video stream. Consider the graph in Figure 8, which schematically shows the result of ASR and NER processing.

The Kaldi app generates tokens from the video stream and aligns each token to a time frame that points to a begin and end time point in the video. It also creates a text document from the tokens, which is by default aligned to the entire video. The spaCy app takes the created text document, generates its own tokenization (not shown in the figure) and extracts a named entity that it anchors to begin and end character positions in the text. But note that the MMIF structure does not provide a direct path from the entity to a time frame in the video and that we have to some work to make that explicit, which in the worst case may include generating alignments between the named entity and the tokens.

Moreover, MMIF is not what we would call a compact format and MMIF files can easily be millions of megabytes in size (which is still orders of magnitudes smaller than the primary data). In addition, they contain a lot of redundancy. For example, the Tesseract OCR application samples images from the video, typically one for every second, and then generates text from those images. For cases when some text is projected on the screen for a while, for example with credits on a slate, some name can be on the screen for many seconds. So if the name *Bill Clinton* was on screen for 10 seconds we will have a MMIF file with 10 text bounding boxes and ten results of the OCR, which includes alignments from the text to the bounding boxes and then for each of the texts two tokens and one named entity, for a grand total of 60 annotations.[10]

To deal with the complexity of MMIF and allow users to easily inspect the results of processing, we have built summarizers and visualizers. All applications mentioned in section 2.1 are MMIF producers, that is, they take MMIF as input and return MMIF as output, where the output is the same as the input modulo added views. Summarizers and visualizers are MMIF consumers, which are applications that take MMIF as input and create another format as output.
We have two summarizers, one that take the MMIF annotations and creates a condensed graph and one that produces the PBCore format which is used by the AAPB. But here we focus on the four visualizers that we have implemented so far: closed captions on video files, time segments on videos, highlights of text spans in text documents, and rectangles on images.
Closed captions. The ASR app generates tokens, time frames and alignments between them. We use those to create a WebVTT file, a World Wide Web Consortium standard for displaying timed text[11], and then use the HTML <video>, <source> and <track> tags to display the transcript of the video as it plays.

[10]You may have noticed in Figure 3 that CLAMS does have the notion of a video object, which in this case might soften the pain a bit by replacing 10 bounding boxes with one video object, but we hesitate throwing out the results of any processing and using a video object does not deal with all redundancies.

[11]https://www.w3.org/TR/webvtt1/.

Time segments. The SMPTE Bar detection app and the Slate detection app generate time segments. The Universal Viewer visualization tool converts a MMIF file that contains time segment annotations to an IIIF manifest[12] which is then displayed using the Universal Viewer.[13]
Text highlights. To highlight named entities we use the spaCy named entity visualizer, which creates an HTML document with colored highlights for a variety of entity types.
Rectangles on images. To visualize bounding boxes we use a Python string template that we feed the image and a set of bounding boxes. Javascript code in the template takes the coordinates from the bounding boxes and then superimposes rectangles on the image.

The visualization code traverses the MMIF file looking for primary data and annotation content to visualize. Each type of annotation that it knows how to visualize will then be put in a separate tab, as the example in Figure 9 shows.

2.5. Advancement since the prototype

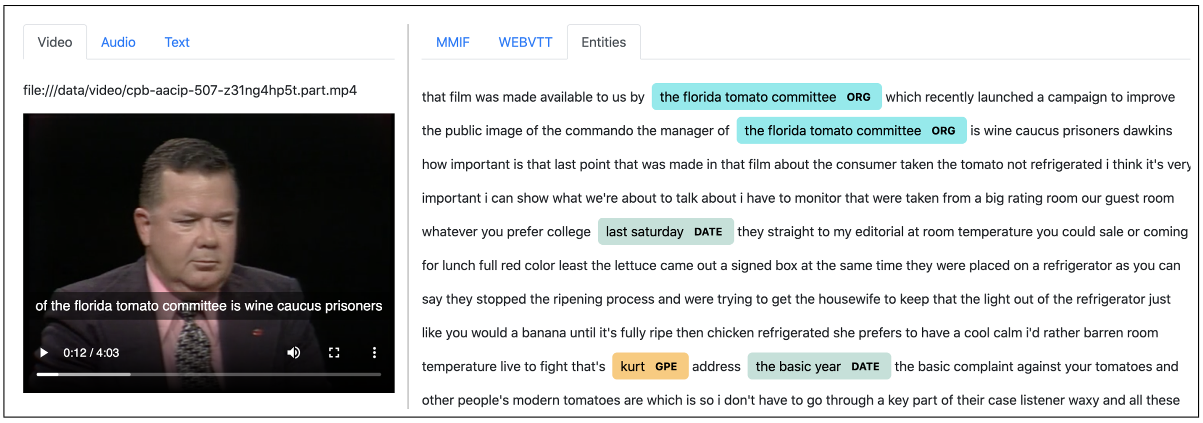
In this section we touch upon the improvements made for CLAMS relative to the prototype version in (Rim et al., 2019). These improvements can be summarized as follows:

- Added a few more applications to the line up: forced alignment, slate parsing, post-ASR.
- A more definite metadata specification for apps and tools that helps their interoperability in pipelines.
- Further developed the MMIF format and designed the interaction between text processing applications and image, audio and video processing applications.
- Created the Pipeline Runner and the CLAMS Appliance as workflow managers. The latter was available in an embryonic form but depended on manually setting up Galaxy.
- Streamlined visualizations and made them less ad hoc. Included visualization of named entities (and other text spans) and bounding boxes.
- Added parameter passing and error handling to the platform.
- Made containerization a more integral part of the platform.

And of course we also went through many cycles of code review and refactoring and kept adding to the documentation.

[12]https://iiif.io/.

[13]https://universalviewer.io/.


Video | Audio | Text

file:///data/video/cpb-aacip-507-z31ng4hp5t.part.mp4

of the florida tomato committee is wine caucus prisoners

0:12 / 4:03

MMIF | WEBVTT | Entities

that film was made available to us by the florida tomato committee ORG which recently launched a campaign to improve the public image of the commando the manager of the florida tomato committee ORG is wine caucus prisoners dawkins how important is that last point that was made in that film about the consumer taken the tomato not refrigerated i think it's very important i can show what we're about to talk about i have to monitor that were taken from a big rating room our guest room whatever you prefer college last saturday DATE they straight to my editorial at room temperature you could sale or coming for lunch full red color least the lettuce came out a signed box at the same time they were placed on a refrigerator as you can say they stopped the ripening process and were trying to get the housewife to keep that the light out of the refrigerator just like you would a banana until it's fully ripe then chicken refrigerated she prefers to have a cool calm i'd rather barren room temperature live to fight that's kurt GPE address the basic year DATE the basic complaint against your tomatoes and other people's modern tomatoes are which is so i don't have to go through a key part of their case listener waxy and all these

Figure 9: Example visualization of a MMIF file with video, audio and text primary data. The pane on the left contains the video with the transcript shown as closed captions, the entities from the transcript are on the right.

3. The AAPB

The American Archive for Public Broadcasting (AAPB) is a collaboration of the Library of Congress and GBH, a public broadcasting foundation in Boston, Massachusetts. It aims to preserve for posterity the most significant public television and radio programs of the past 60 years. Video assets have been digitized and made accessible on the web where users can search the collection.[14] The user experience would improve if more metadata can be brought into the search index. One problem here is the uneven availability of metadata for many of the videos. Hence GBH and the CLAMS team have set out to extract metadata from the video, in particular, from the perspective of GBH, there was clear value in the following:

- Finding the beginning of a video and skipping the bars and tone section that can span minutes at the beginning of the video.
- Create transcripts for all videos and mine the transcripts for metadata.
- Add metadata from slates, rolling credits and other text elements on screen.

Having the text extracted via OCR or ASR allows us to run language processing tools like named entity recognition and use the results to enrich the metadata. To that end we have experimented with a variety of workflows, two of the most useful ones are:

- Segmenter → ASR → post-ASR → NER
- Bars-and-tones → Slate detection → ASR → post-ASR → NER

We report here on the first two components of the second workflow, partially because they were developed in-house.

[14]https://americanarchive.org/

Many video assets in the AAPB begin with a period of SMPTE Bars, also called ”bars and tone”. These segments have historically been used for calibration of equipment. On the AAPB website, these segments lead to a negative user experience, as users must skip over these segments or wait for the content of the program to begin. In order to improve the user experience, we have developed a CLAMS application to detect SMPTE Bars. The bars detection application uses the structural similarity index measure (SSIM) to produce a similarity score to an image of SMPTE bars. The timeframe annotation produced by the application can then be used to begin the video at the point where the bars segment has ended. This functionality is currently being added to the website.

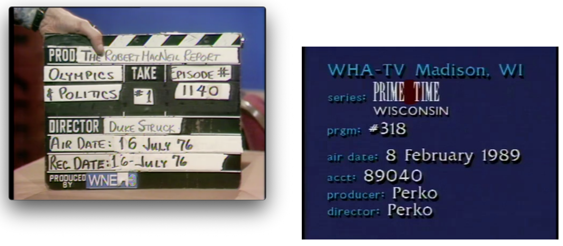


Figure 10: Example slates

Slates are segments of a video during which production related metadata is shown on screen. The prototypical slate is a physical clapperboard, such as the one shown on the left in Figure 10. The slate on the right is a digital slate. The slates vary in format and in the particular information they contain, but they all serve the purpose of recording information about the production that follows. Within the AAPB there are a variety of programs from different public television stations. Often the slates used by an individual program or by all the programs from an individual station follow a template that may vary in content, color, and font over time. We have approached extracting metadata from slates with a pipeline consisting of slate detection, text component detection, and OCR.

Archivists from GBH annotated 301 videos with timestamps of when slates begin and end. These annotations were used to sample slate and non-slate frames from the videos. The frames were used to train a classifier where 20% of the frames were held out for evaluation. The detection model consists of a pre-trained Resnet-50 backbone with two fully-connected layers. The model was trained with the Adam optimizer for 20 epochs using a dropout of .2 and an initial learning rate of 0.003. The model was evaluated against the test set and correctly classified 98% of the frames.

After detecting slates, the next task in our slate processing pipeline is text localization, generating bounding boxes surrounding text in the frame. We evaluate text localization performance against our dataset by calculating the MAP at different IOU and maximum detection thresholds. One limitation of using this evaluation for our task is that it unfairly punishes algorithms that split text regions into more fine grained elements than were annotated. This issue is illustrated with the following example. Given our annotation strategy, the text in 10 is annotated with one bounding box and transcribed as “Air Date”. When comparing the performance of two text localization algorithms, an algorithm that detects “Air” and “Date” as two separate components will have a lower score than an algorithm that detects “Air Date” as a single component.

We compare the performance of the EAST (Zhou et al., 2017) pretrained text detection model, and a finetuned Faster-RCNN model (Ren et al., 2017). We fine-tuned a pretrained Faster-RCNN model with Resnet-50 backbone for 5 epochs. The classification head of the model was trained with 2 output classes: background and text. We used a stochastic gradient descent optimiser with a learning rate of 0.005, momentum of 0.9 and weight decay of 0.0005. In Table 1 we show the performance of this model evaluated against our test set and compare these results to the results from the EAST model.

Method	Max Detections	Conf.	Avg. Precision	Avg. Recall
EAST	10		0.001	0.015
EAST	100		0.001	0.020
FRCNN	10	.5	0.709	0.756
FRCNN	100	.5	0.709	0.761
FRCNN	10	.9	0.497	0.527
FRCNN	100	.9	0.707	0.759

Table 1: Text Localization Results

These results show that text components in slates can be effectively identified by the Faster-RCNN model. Although the EAST model identifies the text, the grouping of characters does not correspond to the keys and values we would like to extract from the frame. In order to effectively use the EAST model, it would be necessary to introduce another step in the pipeline to group related text boxes.

4. Related work

On the computational linguistics side, our work is inspired in part by long-standing popular NLP tool-chaining platforms like UIMA (Ferrucci et al., 2009) and GATE (Cunningham et al., 2013). More recently, web-based platforms such as the LAPPS Grid (Ide et al., 2014a) and WebLicht (Hinrichs et al., 2010) not only offer tool repositories of various levels of state-of-the-art NLP tools for textual data, such as CoreNLP (Manning et al., 2014) and OpenNLP (OpenNLP, 2017), but also implement open source SDKs for tool developers to promote adoption. These workflow engines can operate different tools that are separately developed but can interact because of common data interchange formats. In particular, our format is based on LIF (Verhagen et al., 2015), the format used by the LAPPS Grid. Additionally the LAPPS Grid defines a semantic linked data vocabulary that ensures semantic interoperability (Ide et al., 2014b) and we borrow from that as well.

On the multimedia side of things, MALACH (Oard et al., 2002) was one of the early studies that used computational linguistics tools to build an automatic metadata extraction system. In MALACH, oral history recording data was processed through ASR and NLP pipelines that extracted relevant information for cataloging. In prototyping its World Service Archive (Raimond et al., 2014), the BBC developed COMMA, a metadata extraction and linked data-based interlinking system for public radio broadcasts. Its outcome is now in use by the BBC (BBC, 2015), however it is not publicly available.

More recently, the EU funded MiCO project (Aichroth et al., 2015) aimed at accomplishing a media analysis platform for multimodal media that supports customized workflows leveraging on assorted open and closed source content analysis tools. An interoperability layer, MiCO Broker, was developed based on RDF and XML structures to chain different tools.

The format is a JSON-LD format based on the LAPPS Interchange Format (LIF) for text processing tools (Verhagen et al., 2015), which separated the primary text source from the annotations on the text. MMIF does the same but also allows for image, audio and video data. Primary data of all types are stored in the MMIF file as references to files and those files are considered read-only.

The Audiovisual Metadata Platform Pilot Development (AMPPD) project (Dunn et al., 2018) has worked to enable more efficient generation of metadata to support discovery and use of digitized and born-digital audio and moving image collections. AMPPD is noteworthy as it is designing and developing a platform that exploits chains of automated tools and human-in-the-loop to generate and manage metadata at institutional scale.

5. Conclusion and Future Plans

We have described the CLAMS project and its current use to enhance metadata for the AAPB. The results of our processing have already been shown useful to the AAPB, but much remains to be done to make CLAMS a better asset to archivists and librarians. In the course of our work and our collaboration with GBH it has become clear that several enhancements to the platform would be very beneficial.

On the developer side we believe that while CLAMS makes running pipelines relatively easy, it can still be made more user-friendly. We want to make it even easier to experiment with workflows by providing an App Directory that stores all relevant information, including location of pre-built container images, and then make it almost trivial to select components and create a local working CLAMS platform that embodies the workflows created.

One aspect we have only touched upon recently is in designing workflows that include manual components. For example, we run ASR over the audio stream of the video and then feed the result into post-processing and named entity recognition. However, we will experiment with how much adding results of manual fixes will enhance downstream processing.

One problem with many of the tools is that they do not necessarily translate well to a new set of videos. We have found that for some tools it would be easy to adapt the tools by providing some samples of correct outcomes, for example by identifying what slates look like for a new data set.

We are always working on adding new tools, including tools for topic segmentation (chaptering), text extraction (recognizing rolling credits and other text on screen), more language processing tools (high on our wish list is to improve linking of entities to authority files), and new visualizations (for example by extending the spaCy entity visualizer to other categories).

Finally, we are currently writing up detailed evaluation plans for all the components we use. In the initial stages of CLAMS the focus was on building the platform and integrating third-party processing tools, we have some evaluations but in many cases these are somewhat informal. Even though we have not developed many of the embedded tools ourselves, it is still important to identify what tools are bottlenecks in overall quality of the metadata.

6. Acknowledgements

This work has been supported by two grants from the Andrew W. Mellon Foundation. We also like to thank our collaborators at GBH Boston, in particular Karen Cariani, Casey Davis Kaufman, Timothy Lepczyk, Kevin Carter, Samantha Driscoll, Henry Neels and Andrew Myers.

7. Bibliographical References

Aichroth, P., Weigel, C., Kurz, T., Stadler, H., Drewes, F., Björklund, J., Schlegel, K., Berndl, E., Perez, A., Bowyer, A., et al. (2015). Mico-media in context. In *2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–4. IEEE.

BBC. (2015). COMMA - BBC R & D. `https://www.bbc.co.uk/rd/projects/comma`. Accessed: 2019-02-20.

Cunningham, H., Tablan, V., Roberts, A., and Bontcheva, K. (2013). Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLoS computational biology*, 9(2):e1002854.

Dunn, J. W., Hardesty, J. L., Clement, T., Lacinak, C., and Rudersdorf, A. (2018). Audiovisual Metadata Platform (AMP) Planning Project: Progress Report and Next Steps. Technical report, Indiana University.

Ferrucci, D., Lally, A., Verspoor, K., and Nyberg, E. (2009). Unstructured information management architecture (UIMA) version 1.0. OASIS Standard, mar.

Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., et al. (2005). Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10):1451–1455.

Hinrichs, E., Hinrichs, M., and Zastrow, T. (2010). Weblicht: Web-based LRT services for german. In *Proceedings of the ACL 2010 System Demonstrations*, pages 25–29. Association for Computational Linguistics.

Ide, N., Pustejovsky, J., Cieri, C., Nyberg, E., Wang, D., Suderman, K., Verhagen, M., and Wright, J. (2014a). The Language Application Grid. In *LREC2014*, Reykjavik, Iceland, may. European Language Resources Association (ELRA).

Ide, N., Pustejovsky, J., Suderman, K., and Verhagen, M. (2014b). The Language Application Grid Web Service Exchange Vocabulary. In *OIAF4HLT@COLING*.

Ide, N., Pustejovsky, J., Suderman, K., Verhagen, M., Cieri, C., and Nyberg, E. (2016). The Language Application Grid and Galaxy. In *LREC 2016*, pages 51–70.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Oard, D. W., Demner-Fushman, D., Hajič, J., Ramabhadran, B., Gustman, S., Byrne, W. J., Soergel, D., Dorr, B., Resnik, P., and Picheny, M. (2002). Cross-language access to recorded speech in the malach project. In *International Conference on Text, Speech and Dialogue*, pages 57–64. Springer.

OpenNLP. (2017). Apache OpenNLP. `https://`

opennlp.apache.org/. Accessed: 2019-02-20.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. (2011). The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December. IEEE Catalog No.: CFP11SRW-USB.

Raimond, Y., Ferne, T., Smethurst, M., and Adams, G. (2014). The BBC world service archive prototype. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:2–9.

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, Jun.

Rim, K., Lynch, K., and Pustejovsky, J. (2019). "computational linguistics applications for multimedia services". In *Proceedings of the 3rd Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 91–97, Minneapolis, USA, June. Association for Computational Linguistics.

Verhagen, M., Suderman, K., Wang, D., Ide, N., Shi, C., Wright, J., and Pustejovsky, J. (2015). The LAPPS Interchange Format. In *Revised Selected Papers of the Second International Workshop on Worldwide Language Service Infrastructure - Volume 9442*, WLSI 2015, page 33–47, Berlin, Heidelberg. Springer-Verlag.

Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., and Liang, J. (2017). East: An efficient and accurate scene text detector. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul.