Handover and Integration Report

Table of Contents

Overview

This document provides information that may aid in the integration process of our Graphical Tool into C-LARA, as well as some suggestions for how C-LARA developers may want to extend the tool in the future.

Details on how to run the project are available on the README file in the main branch of the GitHub repository, as well as a breakdown of documentation, changes, testing, and other relevant information.

Workflow of Project

Current Format

The current workflow of our tool is as follows:

1. Go to the Upload Image page (Image set overview)
2. Select a single image to upload and press “Upload”
   a. Upload additional images using the same method
   b. Delete any uploaded image as desired using “Delete”
3. Press “Proceed” to go to the Image and Word Manager page
4. Click on an image to add words associated with the selected image
   a. Press “Add Word” to add a word
   b. Press “Edit” to edit the word
   c. Press “Delete” to delete the word
5. Press “Map” to go to the Mapping Tool for the chosen word
6. Begin the mapping process by clicking on a part of the image
   a. Click on subsequent points of the image to create the rest of the mapping border

a. Click on subsequent points of the image to create the rest of the mapping border
   b. Click on the first selected point to close the the border
   c. Click and drag any points to adjust the border
7. Click “Save” to save the current mapping
8. Click “Back” to redirect to the Image and Word Manager page, where you can select another word to map
9. Click “Back to Upload Page” to redirect to the Upload Image page
10. Click “Generate JSON File” to open a JSON file with the current progress of the mapping

JSON File Format

Current Format

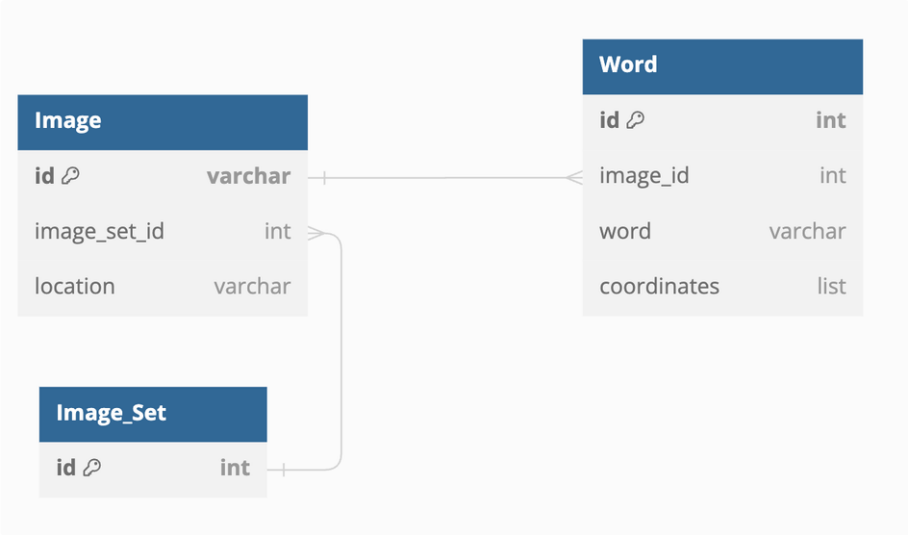
In the Integration with C-LARA document, we were provided with the following ‘Associated Areas’ metadata format to add the coordinates of the annotations produced by our graphical tool:

```
{  "image_id": "... image name ...",
   "segments": [[{"item": "... word ...", "coordinates": ... list of coordinate pairs or null ...},
                 ... more items like the one above ...
         {"item": "SPEAKER-CONTROL", "coordinates": ... list of coordinate pairs or null ...},
         {"item": "TRANSLATION-CONTROL", "coordinates": ... list of coordinate pairs or null ...}]
           ]
}
```

However, our output format follows this form:

```
{  "image_id": "... image name ...",
   "segments": [{"item": wordString, "coordinates": ... list of coordinate pairs or null ...},
                 ... more items like the one above ...
         ]
}
```

This is derived from our current Database Model, as shown below:



At this point, our program does not account for the inclusion of sentences, nor does it allow words to be grouped into sentences as specified in the suggested metadata format. This is because the old JSON file format from the LARA Picture Book Example did not clearly specify the inclusion of sentences, so we decided to omit that from our frontend design and interface.

However, to aid in the integration and extension of our tool into C-LARA, we included the three following classes in our Database to the `models.py` file in the `PostApp` folder:

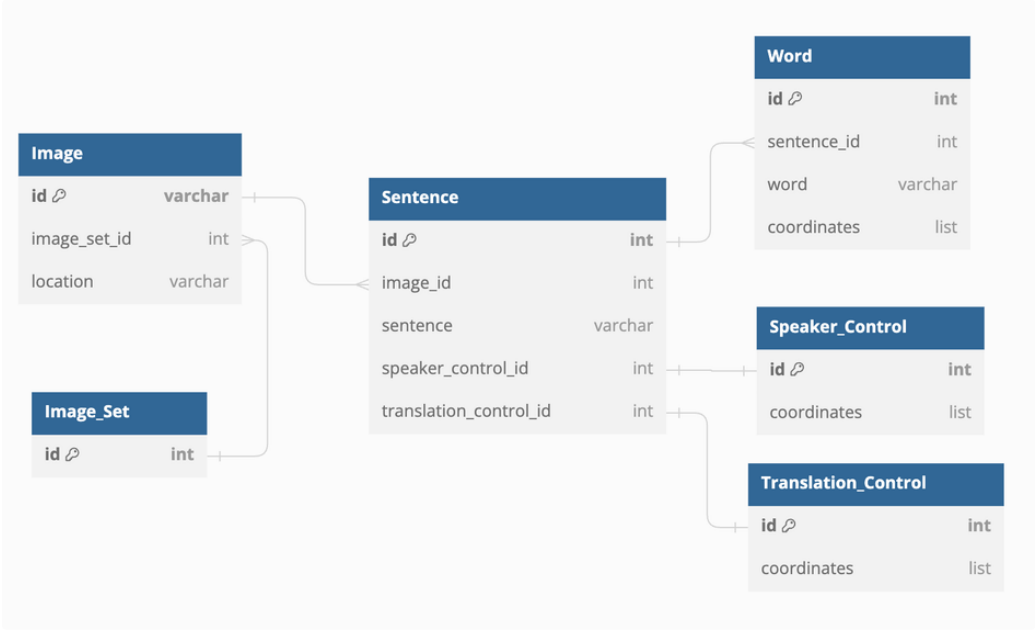
- `TranslationControl`
- `SpeakerControl`
- `Sentence`

Similar to the `Word` class, the `TranslationControl` and `SpeakerControl` classes contain a `coordinates` attribute where a list of coordinate pairs can be stored. The Sentence class also contains the attributes `translationControlID` and `translationControlID` which have foreign keys to instances of `TranslationControl` and `SpeakerControl`.

Suggested Changes

The `Sentence` class also contains the attribute `imageID`, which contains a foreign key to an `Image` instance. To adjust the database to fit the desired metadata format, simply change the `imageID` attribute in the `Word` class to `sentenceID`. This would also need to be updated in the `get` function in the `PostView` class (found in the `views.py` file), which processes the data from the graphical tool into the desired JSON file output format.

Here is the database model with the suggested changes:



These changes would still need to be implemented in the Frontend webpages by C-LARA developers, as there is currently no option for inputing sentences, nor for mapping a Translation control or Speaker control button to the image.

Form of APIs

Based on the current workflow of our tool, we have some suggestions for how it could interact with C-LARA in terms of the form of APIs.

API to post an annotation task

The current implementation of our tool is self-contained, so the user would have to redirect to our webpage to create an image set from scratch, and then manually upload each image and assign words.

If the tool is to be integrated directly into the workflow of C-LARA, where the text constructor would upload images and add associated words during the creation of a C-LARA project instead of using our tool, we would suggest bypassing the Upload Page entirely and adding functions that retrieve information from an uninstantiated Associated Areas JSON file, and the image and word data from the JSON file be

processed into the Image and Word Manager page. The “Edit”, “Delete”, and “Add Word” buttons could be deleted and instead just have the “Map” button next to each word.

API to retrieve the results of an annotation task

As discussed in the *Workflow* section of this report, our Graphical Tool contains a button on the Upload Page called “Generate JSON File” that redirects the user to the JSON file with saved annotation data in its current state, which they can copy and use as they wish. It also contains a “Save” button on the Mapping Page that inputs the mapping data for that word into the JSON file.

If the tool is the remain self-contained and not integrated directly into the C-LARA workflow, the “Generate JSON File” button could remain as is, or it could be changed to instead download a JSON file which could be imported manually into C-LARA.

If the tool is to be integrated directly into the workflow of C-LARA, the “Save” button could instead simply pass the mapping data for that image into C-LARA’s internal database, and the “Generate JSON File” button could be ignored or removed.

Features In Progress

At the moment, there are some features our team is still working on implementing and bugfixing. These will hopefully be present in future releases, but are currently either visible as buttons with no integrated functionality or are yet to be implemented into the existing interface.

Upload Page

We are working on an implementation that allows an uninstantiated Associated Areas JSON File with retrievable images and words to be processed into our tool, as described in the *Form of APIs* section of this report.

Image and Word Manager Page

We are working on an implementation that allows the user to input sentences associated with images and input words associated with the sentences, as described in the *JSON File Format* section of this report.

Mapping Tool Page

Our current version of the Mapping Tool only includes the implementation of a Lasso tool, but our frontend team is working on implementing a Pen tool that allows for more precise mappings. If this tool is integrated in the future, the user will be able to select the desired tool by pressing each button and complete the mapping.

The following buttons are present on the interface for future implementation, but are still in the process of being fixed and tested:

- Lasso tool (Enables the current implementation of the tool)
- Pen tool
- Pen tool color options
- Erase tool
- Undo
- Redo
- Clear All

Other Possible Extensions

“Show annotated image set”

Early in the design stage, we envisioned a feature that allowed the text constructor to preview the current progress of their mapping from the perspective of a C-LARA user in the format of a completed C-LARA text. We believe this would be a useful quality-of-life addition for text

constructors as it would allow them identify and fix any errors or gaps left in their outlines and mapping. However, due to time constraints, we were unable to implement this feature, and decided it would be better to be extended by C-LARA developers who have access to the interface code and can more readily generate a mirror of the interface.

Multiple Image Sets

In our current implementation, the `ImageSet` class in the database is not utilised by the tool—any image uploaded is assumed to be part of the same image set, and the `imageSetID` attribute in the `Image` class is left as `null`. C-LARA developers are encouraged to make use of this class by allowing text constructors to, for example, create several LARA texts concurrently with different images associated with them. In this case, several instances of `ImageSet` would need to be instantiated, and any uploaded images would need to be associated with the correct object ID.

Alternatively, we had the idea of using this class for a drafting system, in which text constructors can create separate annotated image sets (all linked to their account) and save their progress to return to at a later date.

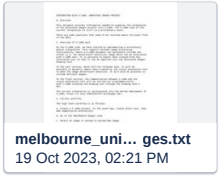
Segment Anything: Automatic Border Detection

During both of the sprint review meetings, we discussed the potential inclusion of a feature that could generate a mapping by automatically detecting the border of an object in an image.

SegmentAnything is a public library that enables this functionality. It detects an object in an image stores the surface coordinates of the object's mapping. To integrate it into our tool, we believe it would be possible to write a function to interpret the surface coordinates by finding the borders and using those as the edges of a polygon to be passed into the output JSON file.

There are guides on the GitHub that explain how to use the library, but it is currently still quite complicated and expensive to implement so we would suggest keeping an eye on future releases.

References



melbourne_uni… ges.txt
19 Oct 2023, 02:21 PM

GitHub - facebookresearch/segment-anything: The repository provides code for running inference with the SegmentAnything Model (SAM), links for downloading the trained model checkpoints, and example notebooks that show how to use the model.