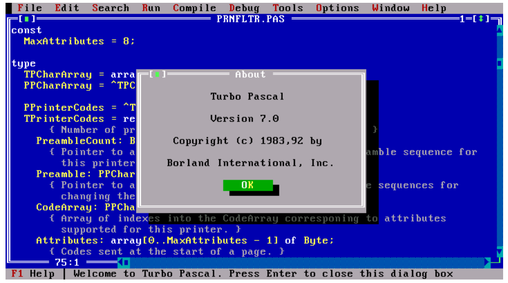
Building Conversational Speech Annotation Tool in Clojure

Paweł Stroiński | 18/19 September 2024 | Heart of Clojure

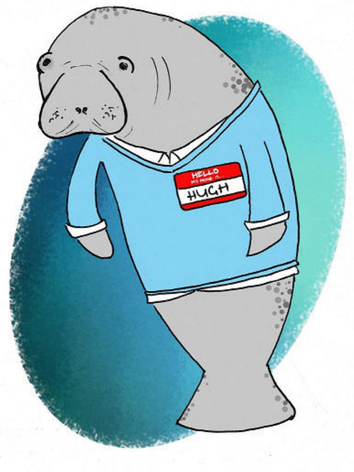
About me

- First programming language/compiler: Turbo Pascal.
- Dabbles in Clojure since 2015.
- Likes music, especially classical.



What we do

- In 2021, I joined **English Language iTutoring Ltd**, a small startup founded in 2014, and later bought by **Cambridge University Press & Assessment.**
- Integrated Learning and Assessment products for English language learners:
  - Write & Improve
  - Speak & Improve
  - Read & Improve
- Text and Speech auto-marking APIs;
  Data-annotation services (Manatee).
  - For the learning products above.
  - For Cambridge University Press & Assessment needs.

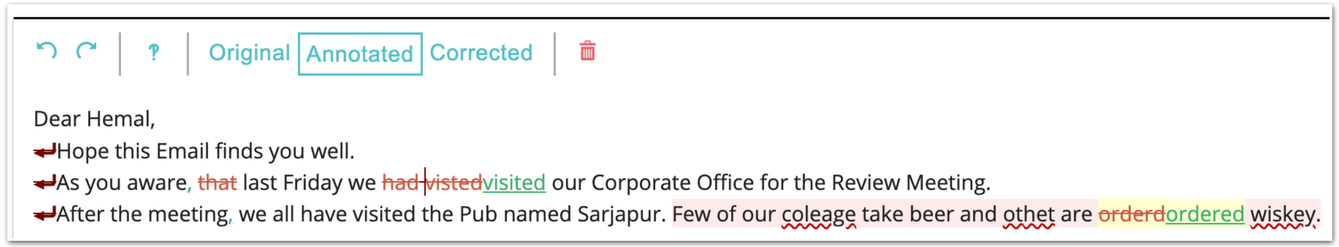


A bit of a background / timeline / credits

- Joined Manatee (annotation tools) team in 2021 when all other members (3 people) were leaving. **Inherited a bespoke text editor with annotation features written from scratch in ClojureScript by a previous employee who had already left the company.**
- Initial temporary help from a consultancy.
- The team grew to 3-4 permanent members.
- [The years covered by this talk are here.]
- I have recently moved on to another team to seek new challenges, and giving this talk to reflect on our adventures because the experience was something special!

Some types of annotation which we are doing

- Existing:
  - 🗣️ Transcription annotation
  - ✅ Grammatical error annotation in transcription
- New:
  - ✍️ Essay / text grammatical error annotation
  - 📖 Multi-Word-Expression marking
  - 👥 Conversational speech annotation
- Why we are doing this?
  - For the purpose of ML training in a language-learning context.



Text annotation demo

Hi Alex is good ~~know~~to know about you,

Thanks for the invitation, but I can´t go, ~~becouse~~because this weekend I will go with my family at the mount. it Is a plan that my sons prepared two months ago with my

Document data structure

- Script is a vector of tokens.
- Selection is relative to tokens in the script.

```
{:script    [{:word "we"       :type :word}
             {:word " "        :type :space}
             {:word "had"      :type :word  :edited? true}
             {:word " "        :type :space :edited? true}
             {:word "visted"   :type :word  :edited? true}
             {:word "visited"  :type :edit}]
 :selection {:start-index  3
             :end-index    3
             :start-offset 1
             :end-offset   1}}
```

Architecture of our bespoke text editor

Overview

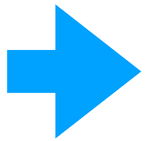
- A single `div` with `contenteditable`
-  Reagent
- Custom DOM event flow

Architecture of our bespoke text editor

Event flow (1/3)

- Stop all input events in `beforeinput` DOM event handler.
- Catch and process key presses in `keydown` DOM event handler.
- User events mapped to a Clojure map and handled by a pattern-matching multi-method (think `nubank/matcher-combinators` but customised).

```
{::type :key
 ::key  {:key  "a"
         :ctrl false
         :alt  false}}
```



```
(defmethod actions/→transformation
  {::actions/key  {:key  letter-or-number
                   :ctrl false
                   :alt  false}
   ::actions/type :key
```

Architecture of our bespoke text editor

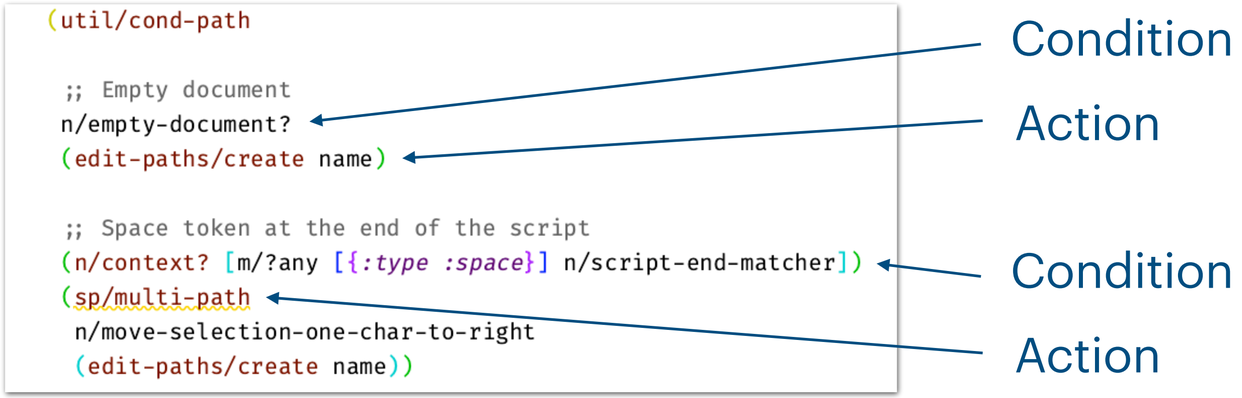
Event flow (2/3)

- Each method queues up an action to be executed. It is a `Specter Navigator`, but you can think of it as just a function. Its purpose is to modify the script and/or selection.
- An action can be very concrete and obvious, e.g. when clicking a specific button on a toolbar, or more dynamic when pressing a key, as it depends on the context.

Architecture of our bespoke text editor

Event flow (3/3)

- To find a concrete action in those dynamic cases, pattern matching is done on the selection, combined with other conditions, forming a rule base.



- As rules can be quite complex, it is helpful to sometimes modularise them, for example rules for caret selection and rules for multi-char selection.

Architecture of our bespoke text editor

DOM

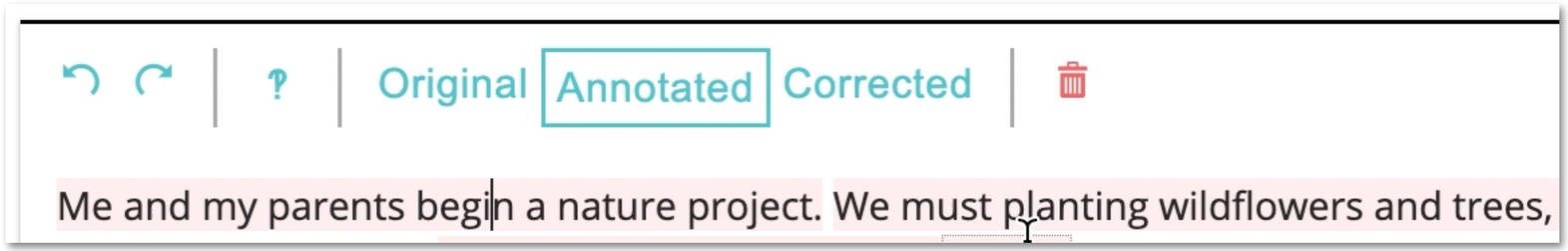
- A single `div` with `contenteditable` attribute.
  - A single `p` container.
    - `span` elements for each token.

- The document content (we call it a script) is simply mapped to hiccup on component render. Each token is converted to `[:span …]`.
- Render happens only after an edit action by a user or after external data change, i.e. a spell checker update. After each render, the selection is set as explained next.

Architecture of our bespoke text editor

Selection (intro)

- Selection is fully managed by the editor and updated after each edit action (render), as needed, to put the cursor in a correct position after edit. 🎬 A movie:



Architecture of our bespoke text editor

Selection (reading)

- User's changes to the selection are observed in DOM `selectionchange` event.
  - Flat (global) selection offsets are calculated using DOM's `Selection` and `Range` APIs, converting a range to a string and simply counting the characters.
  - The global offsets can be converted to token index / offset selection map by e.g. walking the script.

```
{:document-start-offset 0
 :document-end-offset   5}
```

```
{:start-index  0
 :end-index    0
 :start-offset 0
 :end-offset   5}
```

Architecture of our bespoke text editor

Selection (writing)

- In `reagent`'s `after-render` (`requestAnimationFrame`) we set the selection to what has been calculated.
  - Need to flatten the selection from our selection maintained on token index and string level to selection on a single string level.
  - Then use that flattened selection and DOM's standard `TreeWalker` (conveniently, it allows us to iterate over just text nodes), to find the text nodes and the offsets within them.
  - Apply the selection using DOM's `Selection` and `Range` APIs (replace current range with a new range which points to the nodes).

Usage of Specter in the editor

The 21st argument problem

There is a limit to the number of function arguments in ClojureScript (for functions with metadata), which upsets how the long `cond-paths` (with over 20 arguments) are compiled. We ended up with a macro to nest them.

Functions with metadata can not take more than 20 arguments

+1 vote

asked Jan 25, 2022 in ClojureScript by Arne Brasseur

It seems instances of MetaFn have a problem with high arity invocations.

```
(def foo ^:foo (fn [& args]) )

(foo 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
;; => nil

(foo 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
;;=> Execution error (Error) at (<cljs repl>:1). 1 is not ISeqable

(foo 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
;;=> Execution error (Error) at (<cljs repl>:1). Invalid arity: 22
```

Usage of Specter in the editor

The 21st closed-over local usage problem

We've pushed Specter once again (this time on JVM) when we had reached over 20 usages of locals which `cond-path` was closing over. We had just few locals, but each was used multiple times. They were compiled as positional arguments, reaching the limit in `IFn`. We ended up dividing that particular `cond-path` rule.

```
413      ...
414  Caused by: java.lang.RuntimeException: Can't specify more than 20 params
415   at clojure.lang.Util.runtimeException (Util.java:221)
416      ...
417      com.rpl.specter.impl$closed_code.invokeStatic (impl.cljc:625)
418      com.rpl.specter.impl$closed_code.invoke (impl.cljc:621)
419      com.rpl.specter.impl$eval5108$eval_PLUS___5116.invoke (impl.cljc:652)
420      com.rpl.specter.impl$mk_dynamic_path_maker.invokeStatic (impl.cljc:919)
421      com.rpl.specter.impl$mk_dynamic_path_maker.invoke (impl.cljc:912)
422      com.rpl.specter.impl$magic_precompilation.invokeStatic (impl.cljc:952)
423      com.rpl.specter.impl$magic_precompilation.invoke (impl.cljc:940)
```

Usage of Specter in the editor

⚖️ Pros and cons for this case (not a critique of Specter itself!)

✅ Pros

- A nice tool for rule bases in its `cond-path` navigator.
- Constraints what can be done, so forces code consistency, to some extent.
- (Potential) performance gain?

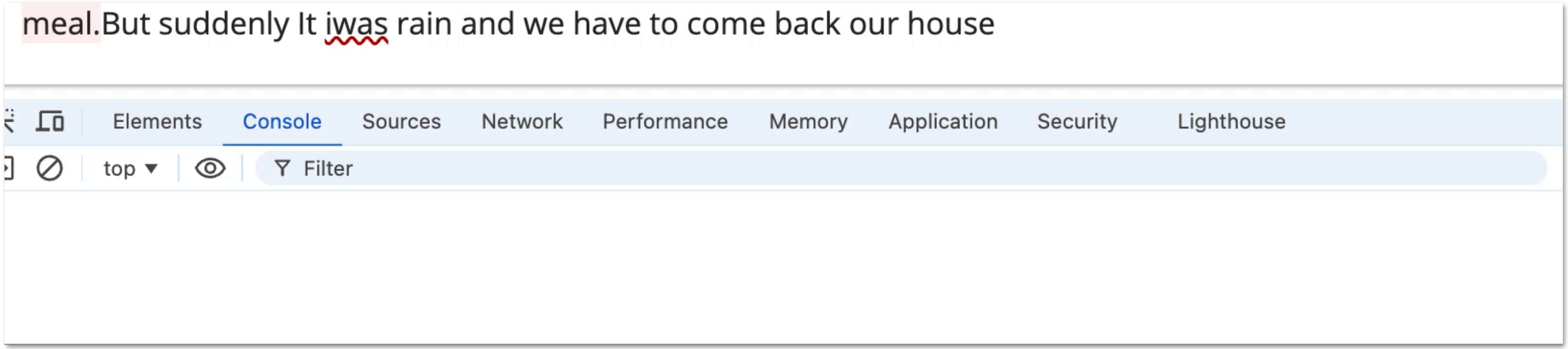
🔴 Cons

- A misfit in terms of domain, as script is a flat vector of simple maps, so Clojure sequence functions would suffice.
- A lot of confusion among developers who have not used Specter before. As every new tool, it comes with a cost.
- Specter API seems to attract writing macros, we ended up with just one, but it has to be used for almost everything, and this further constrains Specter usage.
- Inconsistencies between JVM and JS (of mixed causes) and the 21st arg problems. I think those problems highlight that our usage was atypical.

Given all of the above, it seems that a simple usage of Clojure functions, which are easier composable, would make more sense.

Debugging tooling

🎬 Introspecting rule base and selected tokens



The editor - what has helped (1/3)

The importance of a good model

- Script as a simple vector of maps, each representing a token.
- Selection as a simple map of numbers works well.
- Pure Clojure data, history for free.

The editor - what has helped (2/3)

The importance of tests

- Confidence when amending complex code.
- Very quick feedback (unit tests).
- Made easy by a good model. Easy domain to test: input script + user action = output script.

```
(testing "caret between an edited word and an edit prepends to the edit"
  (run-test-text
   {:script [{:word "a"  :type :word :edited? true}
             {:word "c"  :type :edit}]
    :selection (caret 0 1)}
   (key "b")
   {:script [{:word "a"  :type :word :edited? true}
             {:word "bc" :type :edit}]
    :selection (caret 1 1)}))
```

The editor - what has helped (3/3)

The importance of good documentation (docstrings)

- Don't have to parse the tricky macro-heavy code multiple times.
- Can build on understanding of simpler concepts to understand more complex ones.
- Maybe not as necessary in a typical Clojure code, which is easy to read, especially with good names, but sometimes still really useful.
- IDE helps to quickly look up meaning of a mysterious predicate in a rule base, thanks to using docstrings rather than comments.

```
(defn context?
  "A predicate
  all match, re
  * two tokens
  * token(s) cu
  * two tokens
```

Property-based testing to the rescue

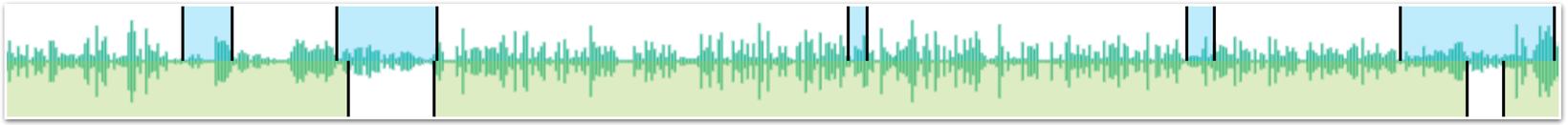
```clojure
(def preserve-script-prop
  (prop/for-all [init-offsets caret-or-selection-offsets
                 actions (gen/vector action)]
    (→ (transform-script input-script init-offsets actions)
       :script
       unannotate-script
       (= unannotated-input-script))))
```

- **Heisenbug:** Very rarely, some tokens from the original script were truncated after annotation.
- **Property:** An annotated script, after removal of the annotation, should be the same as the original.
- **Solution:** `clojure.test.check` (2M trials ~ 10 hours on the JVM).
- **Diagnosis:** Time was the factor. If multiple editor actions were batched before a render, the way they updated selection between runs was inconsistent with a non-batched execution, when selection was set and then read from the browser.

Part II: Annotating entire audio conversations

Concepts

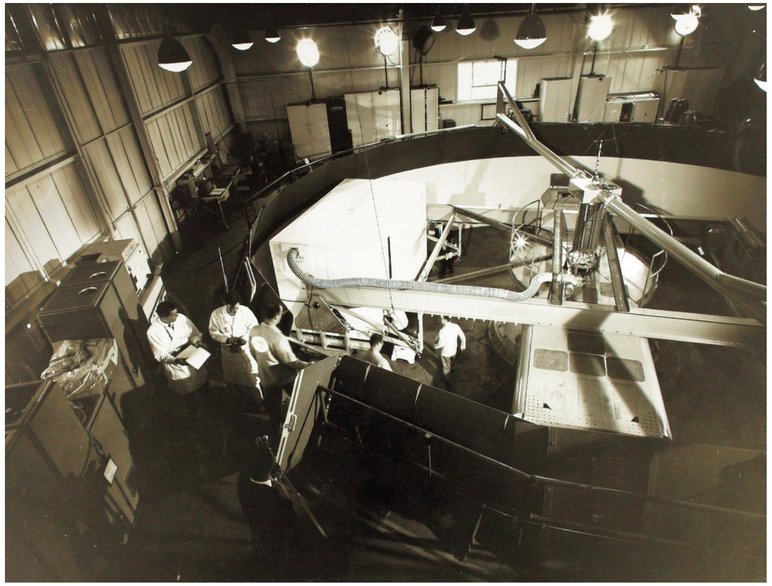
- Speaker diarisation *is the process of partitioning an audio stream containing human speech into homogeneous segments according to the identity of each speaker.* [Wikipedia]



- Segments are sections spoken by a given person.
  - If there is a different speaker, it has to be a separate segment.
  - If there is a long pause in the conversation, it can begin a separate segment.
  - Segments can overlap each other, but we don’t want to have more simultaneous segments than the number of speakers.

Motivation and context

- Training ML models to be better at speaker diarisation and transcribing conversations, in order to use them for:
  - Auto-marking conversational exams.
  - Conversational features in apps for learners.
- In our current data, we have just two speakers (**candidate** and **interlocutor**), which is a good start, but later we might also need to handle three speakers, to support all these use cases.



Combination of two ML tools providing initial data

Disclaimer: not my area of expertise!

- `pyannote-audio` is a neural networks toolkit which provides the **diarisation** data (timestamps and speaker assignment of segments). It comes with pre-trained models.
  - It detects silent bits, then homogeneous segments, and gives each one a speaker.
- `Whisper` is a speech recognition foundational model which provides the **transcription** with timestamps, by predicting the most likely sequence of tokens.
  - The off-the-shelf version is not ideal for faithfully transcribing learner's English, as it smooths out over mistakes or disfluencies. Also not ideal for overlapping speech.
- Both models can be fine-tuned by further training.
- We use the two models independently of each other, and then superimpose segment timestamps on top of a transcription obtained from the entire audio.

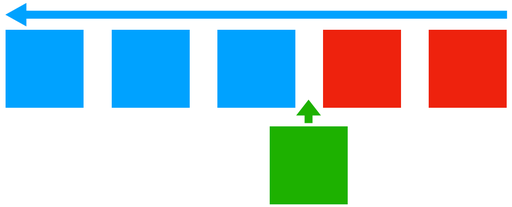
Initial data from both ML models (early versions) combined

Good case and tricky case

Speaker	Start	Duration	Word	Confidence
CAN	13.20	0.20	GOOD	0.92
CAN	13.40	0.26	AFTERNOON	0.89
CAN	14.38	0.40	HOW	0.97
CAN	14.78	0.16	MAY	0.98
CAN	14.94	0.04	I	0.98
CAN	14.98	0.20	HELP	1.00
CAN	15.18	0.14	YOU	0.99
INT	16.50	0.44	WELL	0.70
INT	17.08	0.08	I	0.99
INT	17.16	0.14	HAVE	0.91
INT	17.30	0.22	JUST	0.98
INT	17.52	0.54	FINISHED	0.98
INT	18.06	0.28	A	0.95
INT	18.34	0.24	COURSE	1.00

Speaker	Start	Duration	Word	Confidence
CAN	509.38	0.32	TO	1.00
CAN	509.70	0.18	IT	1.00
CAN	510.38	0.00	SO	0.95
INT	510.38	0.00	SO	0.95
CAN	510.38	0.76	I	0.45
INT	510.38	0.76	I	0.45
CAN	511.14	0.18	HAVEN'T	0.94
CAN	511.32	0.32	ACTUALLY	0.99
CAN	511.64	0.74	TRIED	0.66

(Post-)segmentation algorithm

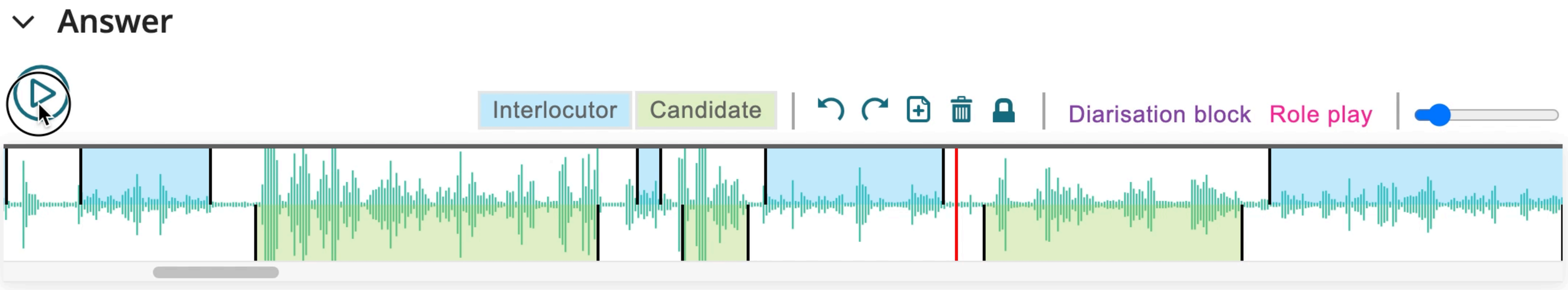


1. Keep adjacent tokens from the same speaker together: For each token to add, walk the previous result from right to left. If the token to add is not sufficiently close to the walked token or not from the same speaker, shelve the walked token for later and go left. Otherwise, insert between the walked token and shelved tokens.
2. Partition sequence of tokens each time there is a different speaker or a considerably long gap.
3. Make a new segment from each partition.

```
[{:start      13.20
  :end        16.10
  :speaker    "CAN"
  :transcript [{:type      :word
               :word      "GOOD"
               :start     13.20
               :duration 0.20}
              … ]}
 {:start      16.50
  :end        30.21
  :speaker    "INT"
  :transcript [ … ]}]
```

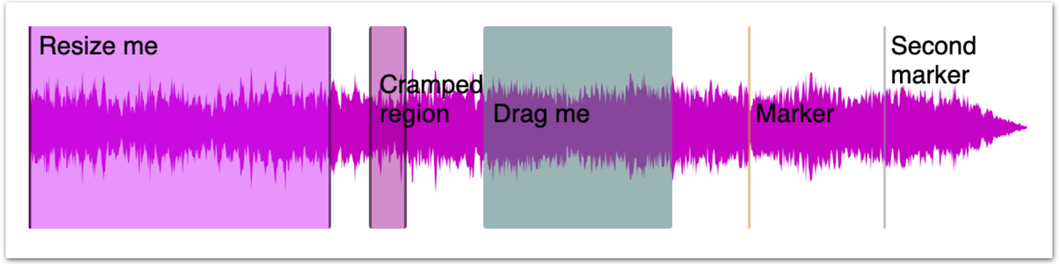
Demo

Quick overview of main UI parts



Wavesurfer.js

- Audio visualisation library for interactive waveforms.
- Imperative, mutable API, but (of course) workable from ClojureScript.
- Plenty of plugins, including for regions.



Wavesurfer.js challenges

Maintaining history in a mutable library

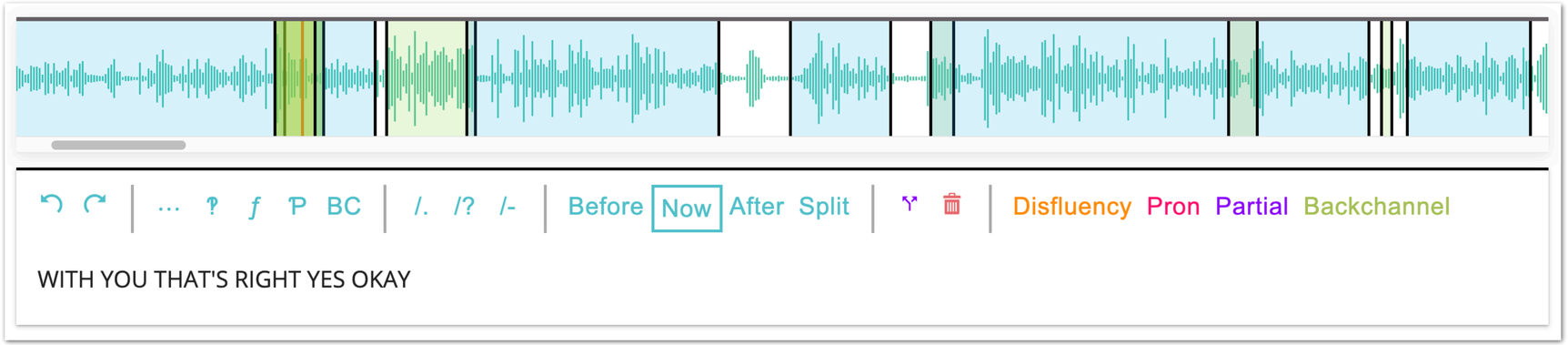
```
{"8f4a3254-dfa1-478a-8912-afa55f750a85" {:start 91.24  :end 100.22 :speaker "CAN"}
 "6edf1bba-5d29-471b-bc6d-074fa90e48df" {:start 242.24 :end 242.68 :speaker "INT"}
 "52e36eae-44b4-4514-9051-c5a9fae2caa1" {:start 474.86 :end 478.86 :speaker "CAN"}}
```

- A Clojure map describing segments as the source of truth. It's a map of maps, where the key is the segment ID.
- Updated when user is interacting with segments on the waveform. Separate 'live' and 'stable' states to independently record updates in-progress (such as resize).
- When we have a new 'stable' state after a user interaction, we add the previous one to the `undo` 📜 history vector. This is where having a recorded 'stable' state is useful.
- When user clicks ↩️ Undo, we remember the current state in the `redo` history vector, and load all regions from the peek of the `undo` history vector.

Wavesurfer.js challenges

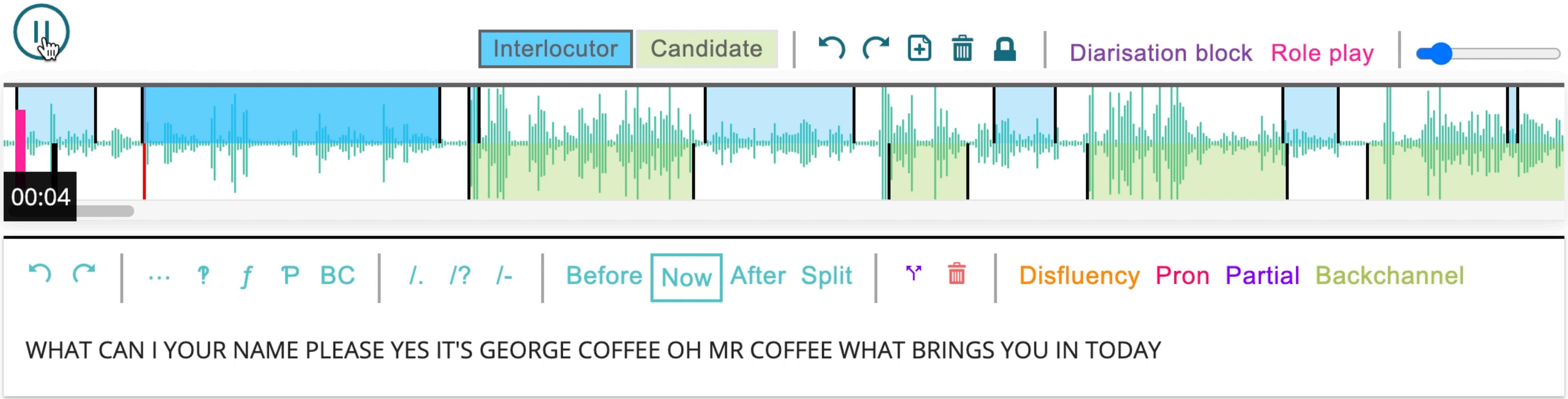
Representing overlapping speech

- It took us a while to realise that the ‘easy’ solution wasn’t optimal at all.
- Thankfully, the display of regions could be customised via CSS.



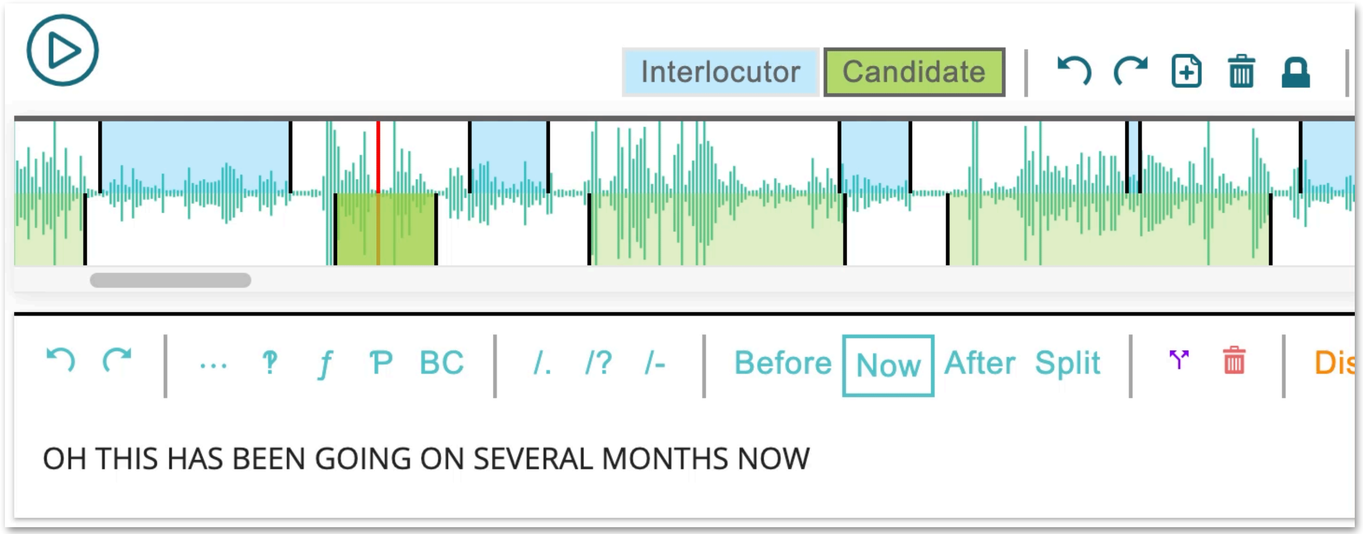
Correcting speaker assignment

- The ML we are using doesn’t always detect a speaker turn if it happens for a short while. Even if the transcription is accurate, it’s assigned to a wrong speaker. We found a solution by focusing on the transcription instead of segments. 🎬 A movie:



Multiple local undo histories as opposed to global undo

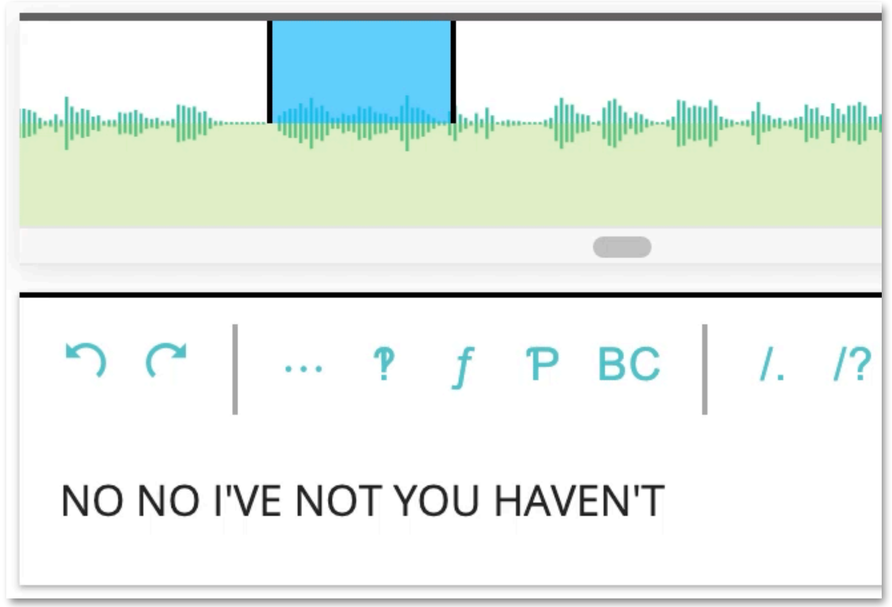
🎬 A movie:



- Pro: It allows annotators to go back to the individual segment's transcriptions and make use of local undo history independently of other segments.
- Con: It's not a common UI pattern and thus could be confusing.
- Con: Ideally, undoing a segment action should automatically undo transcription changes caused by it. As of now, the user has to manually click Undo.

Keeping user's changes to transcription while user is resizing segment (1/2)

🎬 A movie:



Keeping user's changes to transcription while user is resizing segment (2/2)
Clojure is great for this stuff

```
(loop [output []
       user-script (remove-spaces user-script)
       original-script (remove-spaces original-script)]
  (cond
    (empty? user-script)
    (→ (into output original-script)
        restore-spaces-v)

    (tokens-match? (first user-script) (first original-script))
    (recur (conj output (first user-script))
           (rest user-script)
           (rest original-script))

    (user-token? (first user-script))
    (recur (conj output (first user-script))
           (rest user-script)
           original-script)

    (some (partial tokens-match? (first user-script)) original-script)
    (recur (conj output (first original-script))
           user-script
           (rest original-script))

    :else
    (recur output
           (rest user-script)
           original-script)))
```

1. Loop through user’s current transcription and an ‘original’ (input) transcription inferred from the new segment location/speaker and ML data. If ran out of user’s transcription, just add the reminder of the ‘original’ to the output. Otherwise, consider head tokens in both sequences:
2. If they are about the same thing, add the user’s token to the output and skip over both tokens.
3. Otherwise, if the user's token has been added manually by them, add it, but don't skip over the ‘original’ token.
4. Otherwise, if any of the remaining ‘original’ tokens is about the same thing as the user’s head token, add the ‘original’ head token, but don't skip over the user’s token.
5. Failing that, don't add anything, just skip over the user’s token.

Thanks

- Diane Nicholls (Head of Data & Annotation & Product at ELiT) for letting me work on this stuff.
- Cambridge researchers, especially:
  Dr Øistein Andersen, Prof. Mark Gales, and Dr Kate Knill.
- Fellow developers who have worked on the application, especially:
  Adrian, Alex, Andrea, Daniel, Dave, Donavan, Jamie, Jon, Matt, Matthew, Peter, and Yannis.
  (I have listed only developers who I was lucky to work with, and one additional, the original author of the editor.)

Summary

- Working on an annotation software is a very gratifying experience, as the working material is a human output such as essays or conversations. If we add to this an interactive application written in Clojure and ClojureScript, the fun is only multiplied.