Dynamic Authoring of Audio with Linked Scripts

ABSTRACT

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces - Graphical user interfaces

Author Keywords
Audio recording, scripting, transcript-based editing

INTRODUCTION
Audio recordings of speech are a common form of communication that are prevalent in a variety of contexts, including podcasts, audio books, e-lectures and voice-overs for narrated videos. Creating such audio recordings typically involves three main tasks: writing a script, recording the speech, and editing the recorded audio. While authors typically start by writing at least a rough script of what they plan to record, in practice, the process of creating the final audio rarely involves a simple linear progression through these steps. A more common workflow is to move back and forth between writing/editing the script, recording subsets of the speech, and editing together portions of multiple recorded takes.

For example, consider the case of recording the audio for an online lecture. After writing some notes to use as a rough script, the lecturer records a few takes and listens to the speech. She decides that one of the concepts requires a more detailed explanation, so she edits her notes, re-records the relevant speech, and merges the new recording into the final audio. Such updates may also happen in response to feedback from viewers after the lecture is published online. Similarly, when authoring a voice-over for a video, the initial recording may not align perfectly with the visual footage (e.g., some spoken explanations may be too short or too long for the corresponding video clips). Thus, the user may need to modify the script and re-record certain sections of the speech. In general, the process of recording and editing together the speech often reveals issues that require going back to edit portions of the script.

Unfortunately, most existing tools for authoring speech recordings do not facilitate this back and forth workflow. Typically, users write and edit the script in a text editing environment and then record and edit the audio in a standard waveform editing tool. The important point is that the written script and recorded audio are treated as completely separate entities. This separation introduces several workflow problems. When the user records the speech, any deviations from the initial written text (either intentional or not) are not reflected in the script. Thus, evaluating the reocrding to decide what takes to choose or what script modifications are necessary requires careful scrubbing through the audio to find the relevant parts. In addition, once the user chooses a particular version of the speech to include, the script no longer matches the speech, which complicates any subsequent edits to the content. Finally, if the user decides to modify a portion of the script, they must figure out what subset to re-record to ensure that the new recording can be merged in without creating audio artifacts (e.g., replacing a single word in a recorded sentence is hard to do since the word may blend seamlessly with the adjacent words).

To address these challenges, we present VoiceScript an interface that supports script writing, speech recording, and audio editing in a unified way. Our key idea is to maintain the notion of a *master script* that always reflects the current state of the project, including unrecorded, recorded and edited portions of the script. To help users maintain a consistent master script, VoiceScript provides semi-automated tools for merging recorded takes into the master script and visualizations that indicate what portions of the script need to be recorded (or re-recorded) in response to edits to the script. The combination of these features enables users to move back and forth between script editing, speech recording and audio editing in a seamless fashion.

Summarize results and user evaluation findings.

RELATED WORK
Adobe Story [2], FinalDraft [6] and Celtx [5] are examples of professional software dedicated to script writing. They support collaboration, automatic formatting, navigation and planning for future production, but they treat the script as a text document that is essentially separate from the recordings. In fact, in our preliminary interview of lay and professional audio producers, we found that many of them use general-purpose document editors like Google Docs [8] or Microsoft Word [11] to prepare their scripts.

At the recording and editing stage, Adobe Audition [1], Avid ProTools [4], GarageBand [7] and Audacity [3] are among popular digital audio workstations (DAWs). Video editing software such as Adobe Premier [] or ScreenFlow [**?**] are also commonly used. These tools allow users to edit audio by manipulating waveforms in a multi-track timeline interface. They also provide a wide variety of low-level signal processing functions. However, since they are designed to serve as general-purpose audio production systems, they include many features that are not directly relevant for creating audio narratives whose main content is speech. Hindenburg Systems [9] develops tools that are specifically targeted for

audio narratives. Still, they are primarily concerned only with the audio and they do not deal with the script directly.

Recently, several researchers have explored using audio transcripts to support text-based navigation and editing of audio. Whittaker and Amento [16] demonstrate that users prefer editing voicemail through its transcript instead of its waveform. Inspired by similar intuition, Casares et al. [13] and Berthouzoz et al. [12] enable video navigation and editing through time-aligned transcripts. Rubin et al. [15] extend this approach to audio narratives and propagate edits in the transcript text to the corresponding speech track. These systems all focus on editing the audio on the assumption that all the scripting and recording is done beforehand. *Narration Coach* developed by Rubin et al. supports an iterative narration recording process, but instead it assumes that the user has a fixed input script. Its focus, providing capture-time feedback to improve speech performance, is also very different from ours. Our work also takes advantage of text-based navigation and editing, but unlike these systems, we support an dynamic workflow where both the audio recordings and the underlying script can be continuously updated.

KEY OBSERVATIONS

To learn about current practices and challenges for recording audio narratives, we interviewed ten professional lecturers and two video producers who regularly created audio recordings as part of online lectures that are published on platforms, including YouTube, Udacity, EdX and MITx. Following are several key insights we gained from the pilot interviews.

Scripts play a major role during recording. All of the lecturers prepared written materials about what they were going to speak before they started recording. The format and level-of-details of these scripts varied. For instance, one lecturer used his lecture slides containing images and a list of bullet points as his script. Another lecturer typed a thorough word-for-word transcription of what he was going to say in a text document. Another person used a handwritten notes for an outline. In all cases, while they were recording, they kept the scripts within their view and depended on them to guide their speech.

Scripts evolve through the recording process. In many cases, the initial scripts were rough or incomplete. Only two out of the ten lecturers we interviewed prepared a word-for-word script before the recording. The majority of them used lecture slides or handwritten notes containing a rough outline of what they were going to record. They used these outlines as guides and improvised most of the speech. One of the lecturers did an initial recording from the outline, and then used that to flesh out the script even more before recording additional takes. Even when a thorough word-for-word script was prepared beforehand, the recording often did not follow the script exactly. While recording, the speaker sometimes remembered and added more details, or found a more natural way of saying a written sentence. In some cases, a major change was made to the script after a long period of time since the recording had happened. For example, one lecturer noted that he periodically revisited and re-recorded parts of lectures to include up-to-date examples.

So, the script (i.e. what they planned to record) developed along with the audio (i.e. what they actually recorded). A few people actually edited the written script to reflect this development, while in most cases it was only reflected in the audio. This is partly due to the fact that changing the script and changing the audio is a completely separate task in current workflows. Since editing the written script is additional work, if what they care about is the final audio, people usually do not take the trouble of correcting the script to match the audio .

The final track is created by cutting and merging multiple recordings. Most users recorded multiple takes, and then edited them in an audio editing software to produce the final recording. Many of them noted that aligning the waveforms of the multiple takes, finding the best take of a given part, and then cutting and joining them seamlessly were very time consuming and tedious tasks.

AUTHORING INTERFACE

Key Features

Motivated by these insights, we developed the VoiceScript interface to support a dynamic workflow for script writing and audio recording/editing. Our interface is built on two key features.

Text-based representation of audio. Previous work **??** have shown that text-based representation of audio greatly facilitates navigation and editing. Moreover, since scripts are also composed in text, a text view of the audio helps to link and unify the script with the audio. Our system uses automatic speech recognition to transcribe the audio recordings in real-time, and represent each take with a verbatim transcript. The final track, composed from these takes, is also represented as text. Hence, the task of editing and combining the audio recordings becomes akin to merging text documents.

Master-script linking script and audio. Unlike existing tools, our interface does not make a distinction between the script and the final audio track. Instead, our interface includes a *master-script* document that represents both the script (i.e. what the user planned to record) and the transcript of the final track (i.e. what the user actually recorded). The master-script evolves throughout the authoring process, as the user records new takes and merge parts of it in, adds or deletes text to be recorded, or edits recorded text.

Interface and Usage Scenario

The rest of the section describes the interface using an example scenario of how a user might create an audio recording from the beginning.

In Figure **??**capture of interface with outline in master script, the user begins by writing on the master-script an outline of points to record. The text appears in light grey to indicate that these parts have not been recorded yet. At this stage, the master-script is like an ordinary word document or script.

Once the user starts recording, the audio is transcribed in real time and a verbatim text corresponding to each take appears in a separate transcript document tab. Each transcript is time-aligned with the corresponding recording, so the user can

quickly navigate to specific part of the audio by clicking on a word in the transcript. Explain track change view. If the *compare-view* is turned on, our system aligns segments of the transcript to corresponding segments in the master-script, (in this case the points in the original outline), and shows them side-by-side (Figure **??**)capture of aligned view. A segment in the transcript that does not correspond to any part of the master-script (e.g. where the speaker improvised)is highlighted in yellow.

The user can *accept* an audio segment into the final track by clicking on a button next to each transcript segment. If there was a corresponding segment in the master-script, the transcript segment replaces it. If the audio was improvised the transcript segment is simply inserted into the master-script. The text of the accepted segment appears in darker color in the master-script to indicate that it has been recorded.

After the user records multiple takes, in addition to each of the transcripts, the *all tab* provides a summary of all of the takes. For each segment in the master script, it displays corresponding transcript segments, this time from all of the audio takes. A drop-down button next to a transcript segment indicates that there are multiple versions (or takes) of the segment. Clicking on the button opens a list showing the alternative versions (Figure **??**) capture of dropdown showing alternate takes. The user can listen to any of these takes and select one without having to search through individual takes.

If a part of the master-script has not been recorded in any of the takes, it is highlighted in red. In this way, the user can easily keep track of what has been already recorded, and what still needs to be recorded. After recording and accepting segments to the final track, the master-script contains both recorded and unrecorded text. The final track includes only the darker, recorded text.

During any point in the process, the user can edit the master-script like a text document. For example, the user can simply insert more text to record (which appears in light grey), or make changes to unrecorded text to flesh out the original outline or change the wording of a particular sentence. If the user deletes a recorded word from the text, it will be deleted from the final track. The user can also correct the transcription of a recorded word without affecting the underlying audio. When the user edits a recorded word (without completely deleting it), the word is italicized and marked red to warn the user that it may no longer match the underlying audio. If the edit was made to correction a transcription error and the word does indeed match the audio, the user can manually mark the word as *clean* which returns it to a normal dark font. If the edit was intended to change the content, the user can use the mark as a reminder to re-record that portion of the audio.

In order to produce the final recording, the user iterates back and forth between all of these operations, editing the master-script, recording audio takes, comparing alternative takes and accepting audio segments into the final track. The beauty of our interface is that it supports a wide range of workflows for different users and scenarios. For instance, instead of starting with a written outline, the user can begin with an empty master-script and simply start recording, then use the initial recording as an outline. The user can also record the entire script in a single take, or work on a single section at a time.

ALGORITHMIC METHODS

Our authoring interface relies on audio transcription and text alignment algorithms to link the master-script to the audio recordings.

Transcribing the audio recording

We use IBM Speech to Text Service [10] to obtain a verbatim transcript of each audio recording in real-time. The service outputs a time stamp for each word indicating its start and end time within the audio. It also segments the transcript into *utterances* where each utterance is separated by a longer silent gap in the speech (longer than 500 ms). While automatic speech recognition is imperfect, we have found that the results were accurate enough for the purpose of alignment (below) and for users to understand the transcript. Users can also have the option of correcting transcription errors without altering the underlying audio.

Aligning the transcript to the master-script

Once we have a verbatim transcript of an audio take, we compute the global word-to-word alignment between the transcript and the master script using the Needleman-Wunsch (NW) algorithm [14]. NW allows for insertions and deletions, which accounts for differences in the two texts for example, due to loose scripts, or inaccurate speech or transcription. We use the global alignment results to compute a co-segmentation of the two texts.

In order to display corresponding parts in the master-script and the transcript side-by-side, we need to partition the texts into comparable segments. For example, text differencing and merging tools usually treat each line of text as a single segment. Ideally, our segments would respect natural boundaries such as sentence punctuations, line breaks and pauses in the audio. We would also like to separate parts of the transcript that agrees with the master-script (i.e. planned speech) from parts that do not (i.e. improvised speech). We designed a scoring function that optimizes for these requirements and use an iterative algorithm to co-segment the two texts. We first explain the algorithm and then describe the scoring function in detail.

Iterative co-segmentation The segmentation of the master-script depends on the segmentation of the transcript and vice versa. We propose an iterative approach, where the algorithm alternates between optimally segmenting the master-script and the transcript independently using the result from one to segment the other. We initialize the segment boundaries at punctuations (.!?:;) in the unrecorded text and longer silent gaps ($>$ 500ms) in the recorded text. In practice, we found that two iterations were sufficient to converge to a solution.

For each optimization step, we use the classic optimal line-breaking algorithm by Knuth and Plass []. Given the text as a sequence of n words $T = \{w_0, \ldots, w_n\}$, the algorithm finds the optimal set of inter-word boundaries that break the text into segments. We refer to the boundary between w_i and w_{i+1} as b_i. The algorithm iterates through each word,

and for each w_i computes and records the optimal set of text segments S_i for words up to b_i, along with the total score $E(S_i)$ of this partial solution. To determine the optimal partial solution for w_i, it considers each previous boundary b_j $(j < i)$, and evaluates two possible ways of segmenting the text $T_{ji} = \{w_{j+1}, \ldots, w_i\}$: 1) appending T_{ji} to the last segment in S_j, or 2) forming a new text segment with T_{ji}. The algorithm selects the better (lower) of the two scores for T_{ji} and add it to $E(S_j)$ to obtain the total score for the proposed segmentation. After considering all candidate boundaries b_j, the partial solution with the minimum segmentation score is taken. Once the algorithm iterates through all the words, S_n gives the optimal set of segments for the entire text.

Scoring function The dynamic programming algorithm described above requires a scoring function (E) that evaluates the goodness of candidate text segments. We define this scoring function based on three terms:

1. *Punctuation and silent gaps*: We prefer segment boundaries after sentence punctuations, and in case of recorded text, where there is a longer silence gap. Placing cuts at silent gaps allows audio segments from different takes or different parts of a single take to be joined seamlessly. More precisely, we define the boundary score (e_b) for a single text segment $T_{ji} = \{w_{j+1} \ldots, w_i\}$ as:

$$e_b(T_{ji}) = \begin{cases} 1.0, \text{ if } w_i \text{ is unrecorded w punctuation (.!?:;)} \\ -1.0 \text{ if } w_i \text{ is unrecorded w/o punctuation} \\ t_{gap}(w_i) \text{ if } w_i \text{ is recorded} \end{cases} \quad (1)$$

   where $t_{gap}(w)$ is the silence gap in seconds after a recorded word, w, and is equal to 1.0 for w is at the end of the recording.

2. *Global alignment*: We try to separate transcript segments that have a counterpart in the master-script from those that do not (planned vs. improvised), and vice versa (recorded vs. not recorded). We utilize the global alignment output from the Needleman-Wunsch (NW) algorithm. For each word w_i in one text (T) NW outputs a mapping ($M_{TT'}$) to the other text (T'), and vice versa. For instance, $M_{TT'} = \{m_0, \ldots, m_n\}$ where m_i is the index of the word in T' that matches w_i. $m_i < 0$ if the word has no match. We prefer text segments that have the proportion of matching words close to 0 or 1. The alignment score (e_a) for a single text segment T_{ji} is:

$$e_a(T_{ji}) = 2 \times \left| \sum_{n=j+1}^{i} match(w_n)/(i-j) - 1/2 \right| \quad (2)$$

   where $match(w_i)$ is 1 or 0, depending on whether $m_i > 0$ or not (whether the word has a match or not).

3. *Consistency with the other text*: Since the end goal is to align the segments from both texts, we would like the segment boundaries from one text to align with the segment boundaries in the other text. Let $S' = \{s'_0, \ldots, s'_n\}$ be the segmentation of text T', where s'_i is the index of the segment that word w'_i belongs to. Given this segmentation and the mapping of T to T' from NW ($M_{TT'}$), the consistency score (e_c) for a text segment T_{ji} is:

$$e_c(T_{ji}) = \begin{cases} 1.0, \text{ if } s'_{m_i} \neq s'_{m_k} \text{ for the smallest } k > i, m_k > 0 \\ -1.0 \text{ otherwise} \end{cases} \quad (3)$$

   s'_{m_i} is the index of the segment which w'_{m_i} belongs to and likewise for s'_{m_k}, where w'_{m_k} is the closest word after w'_{m_i} that has a match in T.

We combine these terms into a single scoring function e as follows.

$$e(T_{ji}) = e_a(T_{ji}) + e_b(T_{ji}) + 0.5 e_c(T_{ji}) \quad (4)$$

The goodness score for a set of text segments S is:

$$E(S) = \sum_{T_{ji} \in S} e(T_{ji}) - |S| \quad (5)$$

where $|S|$ is the number of segments and is a regularization term.

The final output of our algorithm is a segmentation of the two texts from the master-script and the transcript.

How to match these segments.

We use this output in the *compare-view* to display matching segments side-by-side and to overwrite a segment of the master-script when the user accepts a matching transcript segment.

RESULTS

Voiceover for the we made and collaborative user scneario

INFORMAL USER EVALUATION

Table of usage statistics from both studies.

To gauge the utility of our interface, we conducted an informal evaluation with two users (U1 and U2). We started each session with a 10-minute demonstration of our interface. We gave them a short article about a technical subject and asked them to create an explanatory audio recording using our authoring interface. Users were allowed to go back to the article during the authoring process or to take notes on the master-script, but they were discouraged from recording the article by reading it out loud. We examined their workflow, and the number/type of features they used. We also solicited written qualitative feedback about the authoring experience at the end of the session. Each session lasted about 40 minutes.

U1 created a recording about the article *What is a decibel?* from howstuffworks.com [], and U2 created a recording about *How lasers work* from David Macaulay's illustrated book, *The Way Things Work* []. Overall, the results from the study were extremely encouraging. Both users successfully produced a complete audio recording summarizing the articles, taking advantage of many of our interface features (Table **??**).

Interestingly, each user adapted a very different workflow. U1 started by writing a complete outline as a list of main ideas. For each take, U1 recorded a few points in the outline, merged them into the master-script then continued to record the next point on a separate take. On the other hand, U2 wrote part

of the outline, recorded that portion, and moved on to write the outline of the next part. The different workflows could be due to personal preference, or to the fact that U2's article was organized into a clear step-by-step explanation whereas U1's article flowed like a continuous narrative. In either case, our interface was able support both users' workflow.

Both users took advantage of the master-script to go back and forth between scripting and audio recording. After recording and merging in all the points to the final track, U1 noticed a mistake in one of the examples (instead of saying *140 decibels*, U1 had said *40* decibels). U1 corrected the corresponding recorded text in the master-script, which was consequently marked red. U1 re-recorded and replaced this part by reading out the edited master-script. U2 wrote a very rough outline for the first part of the article. Then U2 used the transcript of the first take to refine the script, and recorded a second take. For the second part of the article, U2 wrote a much more detailed script beforehand. U2 noted, *"For the introduction, I had a pretty good idea of what I wanted to say, so it saved me time to use only bullet points. [For the second part] I wrote full sentences, as I was not familiar with all the technical details and it would have been more difficult to improvise. I enjoyed being able to use the master-script in both ways."*

Both users offered strong positive feedback about our authoring interface, and said they would use it for creating speech recordings. They were most enthusiastic about the integration of the script and the final track in the master-script view, and the ability to align the master-script to the transcripts. U2 wrote, *"Writing the outline on the same interface and having that integrated with the audio was most helpful."* U1 said that the alignment, *"helped to keep track of what pieces of information was already recorded and which ones were still needed."* Participants were also impressed with the quality of the merged recording. U1 wrote, *"I was surprised how the final recording from the multiple takes was seamless."* Speech recognition pretty good.

COMPARATIVE STUDY

We also conducted a pilot study to compare our interface with a state-of-the-art transcript-based speech editing interface [15]. We recruited four participants, none of whom had experience using text-based audio editing systems. We gave them a script with bullet points outlining a mini lecture on a science subject (e.g. *gravity* and *dark matter*) and two audio takes roughly corresponding to that script. Their task was to cut and merge the two takes to produce a recording that contained all the contents listed in the script and only those contents. The two takes were similar, but both takes had some missing content from the outline and one had some extra content. So, the participants had to choose parts from each take and combine them to get the final result. We encouraged the users to focus on having the complete content rather than the details of the audio quality (e.g. tempo, diction, flow of speech etc.).

In Rubin et al.'s system (referred to as *Interface-R* hereafter), as in our interface, users can edit the transcript like a text document using operations such as copy-and-paste, insert or delete, and the edits are propagated to the audio. The system also detects alternate takes of the same sentence and groups them for users to select between them. However, unlike VoiceScript, Interface-R is geared for editing pre-recorded audio, and does not support scripting or real-time recording. In fact, it only allows editing one recording at a time. To simulate multiple takes, we took advantage of their multi-column interface, so that each take appeared in a separate column one after the other as if they were spoken by two different speakers. In VoiceScript, the script was contained in the master-script, whereas in Interface-R, we gave users a hardcopy of the script.

Each participant completed the task twice with different outlines, once using our interface and the other using Interface-R. The subject of the outline and the order of the interface was counter-balanced. We examined the time they spent editing, the number/type of functions they used, and the quality of the final recording (Table **??**). After the session, participants gave written qualitative feedback about the two interfaces. In total, each session lasted 1 hour.

Each of the four participants preferred VoiceScript over Interface-R for the given task, and noted they would use our interface to edit audio recordings. Every participant also completed the task faster using our interface (7.4 vs 9.9 min Interface-R).

Explain usage

All of the participants commented on how the master-script helped them in the task. One person noted that in VoiceScript *"the master-script linked the two takes into one comprehensive view that made the editing a lot simpler, [while in Interface-R] I felt like I had to consider two separate documents and combine them manually."* Another person preferred the VoiceScript interface for the task of editing the audio content, and commented that Interface-R would be useful to make detailed edits to the speech flow. 3 out of the 4 participants also found the color-coded visualization of the master-script (recorded vs. unrecorded) and the transcript (planned vs. improvised) very helpful. A participant wrote about the *compare-view* feature that *"The alignment was great. I felt like almost half of my work was already completed before I began."*

CONCLUSIONS

ACKNOWLEDGMENTS

REFERENCES

1. 2016a. Adobe Audition. `http://www.adobe.com/products/audition.html`. (April 2016). Accessed: 2016-04-02.
2. 2016b. Adobe Story. `https://story.adobe.com/en-us/`. (Apr 2016). Accessed: 2016-04-02.
3. 2016. Audacity. `http://www.audacityteam.org/`. (April 2016). Accessed: 2016-04-02.
4. 2016. Avid Protools. `http://www.avid.com/en/pro-tools`. (April 2016). Accessed: 2016-04-02.
5. 2016. Celtx. `https://www.celtx.com/index.html`. (April 2016). Accessed: 2016-04-02.
6. 2016. Final Draft. `https://www.finaldraft.com/`. (April 2016). Accessed: 2016-04-02.

7. 2016. GarageBand. http://www.apple.com/mac/garageband/. (April 2016). Accessed: 2016-04-02.
8. 2016. Google Docs. https://www.google.com/docs/about/. (April 2016). Accessed: 2016-04-02.
9. 2016. Hindenburg Journalist Pro. http://hindenburg.com/products/hindenburg-journalist-pro. (April 2016). Accessed: 2016-04-02.
10. 2016. IBM Speech to Text Service. https://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/doc/speech-to-text/. (April 2016). Accessed: 2016-04-02.
11. 2016. Micorsoft Word. https://products.office.com/en-us/word. (April 2016). Accessed: 2016-04-02.
12. Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. 2012. Tools for placing cuts and transitions in interview video. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 67.
13. Juan Casares, A Chris Long, Brad A Myers, Rishi Bhatnagar, Scott M Stevens, Laura Dabbish, Dan Yocum, and Albert Corbett. 2002. Simplifying video editing using metadata. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*. ACM, 157–166.
14. A general method applicable to the search for similarities in the amino acid sequence of two proteins. 1970. *Journal of molecular biology* 48, 3 (1970), 443–453.
15. Steve Rubin, Floraine Berthouzoz, Gautham J Mysore, Wilmot Li, and Maneesh Agrawala. 2013. Content-based tools for editing audio stories. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 113–122.
16. Steve Whittaker and Brian Amento. 2004. Semantic speech editing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 527–534.