Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 6375–6380
Marseille, 11–16 May 2020

ATC-ANNO: Semantic Annotation for Air Traffic Control with Assistive Auto-Annotation

Marc Schulder[*†], Johannah O'Mahony[*], Yury Bakanouski[*], Dietrich Klakow[*]
[*] Spoken Language Systems, Saarland University, Germany
† Institute for German Sign Language, University of Hamburg, Germany
marc.schulder@uni-hamburg.de, omahony@coli.uni-saarland.de,
yury.bakanouski@gmail.com, dietrich.klakow@lsv.uni-saarland.de

Abstract

In air traffic control, assistant systems support air traffic controllers in their work. To improve the reactivity and accuracy of the assistant, automatic speech recognition can monitor the commands uttered by the controller. However, to provide sufficient training data for the speech recognition system, many hours of air traffic communications have to be transcribed and semantically annotated. For this purpose we develop the annotation tool ATC-ANNO. It provides a number of features to support the annotator in their task, such as auto-complete suggestions for semantic tags, access to preliminary speech recognition predictions, syntax highlighting and consistency indicators. Its core assistive feature, however, is its ability to automatically generate semantic annotations. Although it is based on a simple hand-written finite state grammar, it is also able to annotate sentences that deviate from this grammar. We evaluate the impact of different features on annotator efficiency and find that automatic annotation allows annotators to cover four times as many utterances in the same time.

Keywords: Annotation Tool; Controlled Languages; Air Traffic Control; Assisted Annotation

1. Introduction

Air traffic control (ATC) communications is a challenging domain for automatic speech recognition (ASR) (Hamel et al., 1989; Cordero et al., 2012; Shore et al., 2012). Very little training data for the ATC domain is publicly available, necessitating the recording, transcription and annotation of more data to allow ASR to be successfully integrated into ATC systems (Oualil et al., 2015).

We present ATC-ANNO, a tool for the transcription and semantic annotation of recordings of air traffic control communications. Transcribing ATC communications is challenging, as air traffic controllers speak extremely fast, due to the time-sensitive nature of their task. In addition they are usually non-native speakers and their utterances follow a specific information-dense phraseology.

Due to these challenges, annotators must have prior experience with ATC communications. Unfortunately, this means that appropriate candidates usually have no experience in transcription or annotation of natural language data. They are domain experts, but novice annotators. We therefore try to support them particularly in more technical tasks, like writing XML tags for the semantic layer of the annotation, while keeping the interface as focussed as possible.

ATC-ANNO provides a number of assistive features. These range from basic features, such as syntax highlighting and consistency checks between different annotation layers, to more immediate support, like XML tag auto-completion suggestions or access to preliminary speech recognition predictions. Our most advanced assistive feature is the automatic generation of semantic annotations, based on a grammar of common phrases encountered in air traffic control. While the grammar is written as a finite state automaton, our algorithm is able to react to unforeseen utterances by skipping individual words in an attempt to find the closest matching defined utterance.

We chose to develop our own annotation tool because existing tools were either too simplistic (e. g. lack of annotation layers) or too complex (i. e. steep learning curve, many unrequired features). Implementing our assistive features would also not have been easily possible, as most tools lack a sufficient plugin structure. Furthermore, many annotation tools, such as the widely used ELAN (Brugman and Russel, 2004), use a timeline view to create timestamped annotations for subsections of a longer recordings, e. g. individual sentences or words in a longer monologue. In our task recordings are already segmented because ATCs turn their microphone on and off for every individual transmission. Without the need for segmentation, such a timeline view bears little advantage, but slows down the annotation process significantly.

Designing our own tool allowed us to keep the user interface lightweight and focussed on the intended workflow. Development was also guided by the feedback of annotators using it for the work of Shore et al. (2012) and later for the AcListant project (see Ohneiser et al. (2014)).

In the remainder of this paper we provide a brief overview of air traffic control communications and the challenges it poses to speech recognition (Section 2), followed by a description of the annotation task (Section 3) and the ATC-ANNO tool itself (Section 4). In Section 5 we provide an evaluation of how much various features of ATC-ANNO improve annotator performance.

The core source code of ATC-ANNO is made **publicly available**.[1] It includes a video demonstration of the tool to make users familiar with its workflow and functions.

2. Background on Air Traffic Control

In this section we provide information about the Air Traffic Control domain that is required for the understanding of this tool. Section 2.1 describes why and how speech recognition is used in air traffic control. Section 2.2 gives a brief introduction on the phraseology of air traffic communications and how it differs from natural language.

[1] `https://doi.org/10.5281/zenodo.3698318`

Transcription Layer	lufthansa four romeo juliett reduce speed two five zero knots
Semantic Layer	`<s> <callsign> <airline>` lufthansa `</airline>` `<flightnumber>` four romeo juliett `</flightnumber>` `</callsign> <commands> <command="reduce">` reduce speed `<speed>` two five zero `</speed>` knots `</command> </commands> </s>`
Concept Layer	DLH4RJ REDUCE 250

Table 1: Example of different information layers covered by the annotation process (see Section 3).

2.1. Speech Recognition in Air Traffic Control

Air traffic controllers manage a given airspace by issuing commands to pilots via radio communications. Their task is to optimize the flight plan while ensuring the safety of all aircraft. To support them, digital planning systems are used. These assistant systems use radar information to suggest a sequence of commands. The final decision, however, lies with the air traffic controller. In cases where the controller deviates from the suggested sequence, the assistant system is slow to react to these unforeseen changes, as it cannot hear the transmission of the controller to the pilot and must rely on inferences made from radar information.
To eliminate this weakness, the AcListant project introduced an assistant system with automatic speech recognition capabilities.[2] Hearing and interpreting the commands uttered by the controller allows the system to update its world state assumptions immediately. This results in better suggestions, which improve the efficiency of the human controller (Helmke et al., 2017).
Air traffic control poses a number of challenges for ASR. Utterances are very domain-specific and while communication is conducted in English, most controllers are non-native speakers. Commands are uttered at high speed to accommodate the heavy workload (Oualil et al., 2015). Commercial off-the-shelf ASR systems perform very poorly under these conditions (Cordero et al., 2012). Instead, the ASR system has to be trained on ATC domain data. Publicly available ATC recordings are very sparse (the largest corpus, ATCOSIM (Hofbauer et al., 2008), has 10 hours), so more data had to be recorded, transcribed and annotated (Oualil et al., 2015).

2.2. Air Traffic Phraseology

Communication in air traffic control is mainly conducted in English, following the phraseology format of the International Civil Aviation Organization (ICAO) (Eurocontrol, 2011). This phraseology is a strictly defined subset of the English language. This would be advantageous for ASR, as language models can be small and well defined. However, over 25% of command utterances deviate from the phraseology (Oualil et al., 2015). While this means the language model has to be more extensive after all, the phraseology guidelines still provide a strong starting point.
According to ICAO phraseology, an ATC utterance consists of a callsign and one or more commands. The callsign is the unique identifier of a specific aircraft in the airspace. It consists of the name of the airline and an alphanumeric sequence that is pronounced using the NATO phonetic alphabet. For example, the callsign `DLH4RJ` (DLH being a shorthand for Deutsche Lufthansa) would be spoken as *"Lufthansa four romeo juliett"*. Commands must contain specific identifying vocabulary. Numbers must be given as individual digits, except when they are multiples of a hundred or a thousand.

(1) Lufthansa four romeo juliett reduce speed two five zero and descend to altitude four thousand.

Example (1) shows a transmission with two commands, instructing the airplane `DLH4RJ` to reduce its speed to 250 knots and its altitude to 4000 feet.

3. Annotation

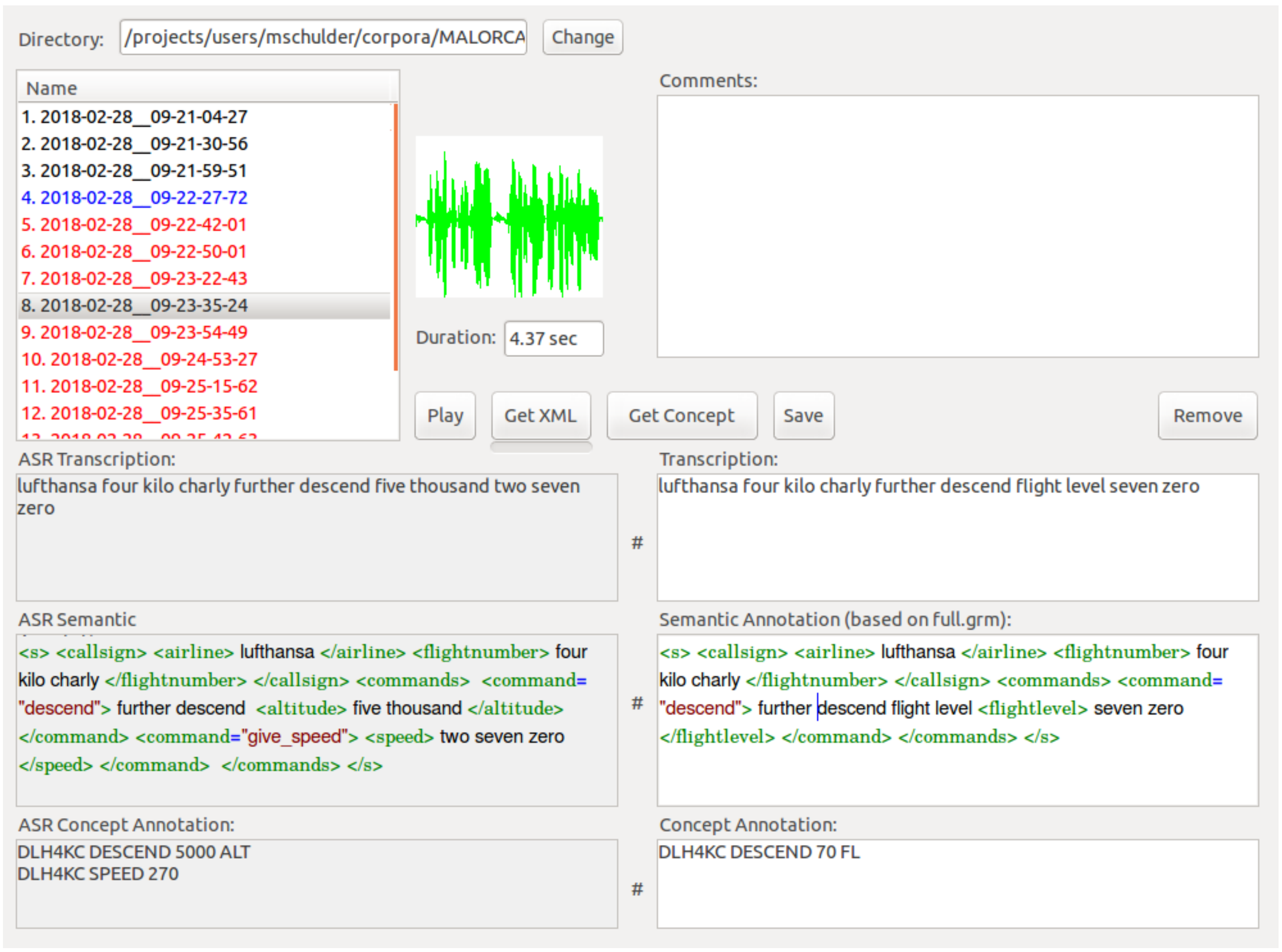
To allow ATC assistant systems to make use of the utterances recognised by ASR they need to be converted from natural language text to an unambiguous machine-interpretable format. To support conversion, the ASR text output is enhanced with semantic information.
To develop and evaluate each step of this system, all data gets annotated using the following three information layers:

1. **Transcription Layer:** The natural language text representation of the recorded utterance.
2. **Semantic Layer:** An extended version of the transcription layer, enhanced with XML tags indicating the semantic structure of the utterance as required for later processing.
3. **Concept Layer:** An abstract representation of the commands inferred from the semantic layer, to be processed by the assistant system.

An example of the three layers can be seen in Table 1. To annotate an utterance, the annotator starts by listening to its recording and then transcribes it. They then annotate the transcription with XML to indicate semantic content. Lastly, the abstract concepts are added. While these should in theory be inferrable from the semantic layer, annotators were instructed to use their own understanding of the domain to determine concepts instead. This way, flaws in the semantic formalism could be detected, as well as any pieces of information which were implicitly given.[3] For more information on how the annotation layers are used in training

[2]This approach has since been expanded upon by the follow-up projects AcListant-Strips and MALORCA.

[3]For example, on some occassions recordings would not contain a callsign when they were closely following a previous transmission. To correctly process such cases, a memory of the most recently used callsign was added to the system.



/projects/users/mschulder/corpora/MALORCA

1. 2018-02-28__09-21-04-27
2. 2018-02-28__09-21-30-56
3. 2018-02-28__09-21-59-51
4. 2018-02-28__09-22-27-72
5. 2018-02-28__09-22-42-01
6. 2018-02-28__09-22-50-01
7. 2018-02-28__09-23-22-43
8. 2018-02-28__09-23-35-24
9. 2018-02-28__09-23-54-49
10. 2018-02-28__09-24-53-27
11. 2018-02-28__09-25-15-62
12. 2018-02-28__09-25-35-61

Duration: 4.37 sec

ASR Transcription: lufthansa four kilo charly further descend five thousand two seven zero

Transcription: lufthansa four kilo charly further descend flight level seven zero

ASR Semantic: <s> <callsign> <airline> lufthansa </airline> <flightnumber> four kilo charly </flightnumber> </callsign> <commands> <command="descend"> further descend <altitude> five thousand </altitude> </command> <command="give_speed"> <speed> two seven zero </speed> </command> </commands> </s>

Semantic Annotation (based on full.grm): <s> <callsign> <airline> lufthansa </airline> <flightnumber> four kilo charly </flightnumber> </callsign> <commands> <command="descend"> further descend flight level <flightlevel> seven zero </flightlevel> </command> </commands> </s>

ASR Concept Annotation:
DLH4KC DESCEND 5000 ALT
DLH4KC SPEED 270

Concept Annotation:
DLH4KC DESCEND 70 FL

Figure 1: ATC-ANNO interface with example entry. The read-only text fields on the left are ASR predictions, while the text fields on the right are texts entered by the annotator. The # signs between text fields indicate that they do not contain equivalent information.

the ASR model and in interfacing with the assistance system, see Oualil et al. (2015) and Helmke et al. (2017) respectively.

4. Annotation Tool

Our annotation tool ATC-ANNO is designed to allow annotators to listen to recordings and annotate the three information layers outlined in Section 3. Figure 1 shows the graphical user interface, which consists of a file selector, seven text fields and six buttons. The three fields in the lower right quarter are fields for the annotator to enter their transcription, semantic annotation and concept annotation. To their left are equivalent fields to show ASR predictions, if available. The field in the upper right corner is used for miscellaneous comments.

Apart from buttons for basic functionality (directory selection, playing the recording, saving, and deleting irrelevant recordings), there are also buttons for the automatic generation of semantic and concept annotations. Once used, the auto-generation buttons turn into *undo* buttons in case the automatic prediction was not satisfactory.

ATC-ANNO is written in Python 2.7, using the wxPython package to display the graphical user interface. The finite state language model used by the automatic semantic annotation feature is generated using OpenFST.[4]

4.1. Assistive Features

We now introduce the assistive features. They are listed in increasing order of external resource requirements, such as domain-specific information or a pre-trained speech recognition model.

Syntax Highlighting: The XML markup is highlighted to ease readability and help spot broken tags. Color coding is different for project-specific tags and unknown tags.

Consistency Indicators: Signs between two text fields indicate whether the fields are consistent with each other or show differences. Consistency between the transcription and semantic annotation field is determined by stripping away the XML results in the transcription text and, therefore, requires no special information. Consistency between semantic and context annotation requires a working concept extraction algorithm. Annotator text and ASR prediction are considered consistent when texts of the same layer are identical.

[4] http://www.openfst.org

	Syntax Highlighting	Consistency Indicators	XML Suggestions	Concept Extraction	Semantic Auto-Annotation	ASR Prediction
Basic	✓	✓				
XML-Suggest	✓	✓	✓			
Auto-Annotate	✓	✓	✓	✓	✓	
ASR-Predict	✓	✓	✓	✓	✓	✓

Table 2: Overview of features available in different version of the annotation tool.

XML Suggestions: When the user starts entering markup, a list of suggestions for XML tags appears. As more characters are typed, the item selector automatically jumps to a possible completion tag. Suggestions contain not only the opening and closing tag of the given item, but also expected sub-items, e. g. `<command="reduce"> <speed> </speed> </command>` (see Table 1). This requires information about the tag set of the semantic layer.
Automatic concept extraction: Information is extracted from the semantic layer and reformatted as concepts (see Section 3). This requires the semantic tag set and relevant keywords within the tags (e. g. how a speed value can be expressed).
Automatic semantic annotation: The XML markup of the semantic layer is generated using a finite state language model, which contains the ICAO phraseology and common deviations, enhanced by semantic markup. For utterances found in the language model, the annotation feature functions as a normal finite state transducer whose input tape is a language model without markup (i. e. the transcription layer) and whose output layer is the same language model with markup (i. e. the semantic layer).
For utterances that deviate from the language model this feature attempts to find the most similar phrase in the model by skipping individual words. As such a search can have exponential complexity, we adapt Viterbi search (Viterbi, 1967) to efficiently find the path requiring the fewest skips. During computation the annotator is shown a progress bar as well as the currently best (partial) parse. In rare cases where the search takes too long due to phrases unrelated to the language model, the user can abort the process, at which point the system backtracks to a consistent partial annotation (i. e. only containing labels that have both an opening and closing tag).
ASR Prediction: If an ASR prediction is available, it is displayed in the left three read-only fields. Annotators can choose to accept the entire ASR prediction, copy it over to make corrections or reject it and perform the annotation by hand. This requires an ASR system trained on ATC data enhanced with semantic XML tags, as well as the automatic concept extraction algorithm.
The ASR system is not part of ATC-ANNO, but should rather be understood as a preprocessing step. For full functionality one should use an ASR model that also generates semantic tags. This rules out the use of preexisting models, such as those of closed commercial ASR systems. Such models can still be used to generate regular text predictions (sufficient recognition quality notwithstanding), but must then fall back on the automatic semantic annotation mechanism of ATC-ANNO to generate XML tags.

5. Evaluation

To evaluate the efficiency of the main assistive features outlined in Section 4.1, we added extensive usage logs and created four different versions of the annotation tool. Each consecutive version offers a growing number of features, but also higher requirements for external resources:

1. **Basic:** Has syntax highlighting and consistency indicators, but no active support for the annotation layers. Requires no domain- or project-specific information.

2. **XML-Suggest:** Adds XML tag suggestions to the features of the basic version. Requires the XML tag set to be predefined.

3. **Auto-Annotate:** Provides automatic semantic annotation and automatic concept extraction features on top of those of the *XML-Suggest* version. Requires a finite state grammar of common utterances and a concept extraction algorithm.

4. **ASR-Predict:** Adds the ASR prediction fields, resulting in the complete set of all available features. This requires the output from an ASR system that was already trained on other semantically annotated data. ASR output is generated separately before running ATC-ANNO.

On overview of the features available in each version of the tool can also be seen in Table 2. For *ASR-Predict* we use ASR outputs generated by *KALDI* (Povey et al., 2011). The ASR model was trained on a combination of data from the freely available *ATCOSIM* (Hofbauer et al., 2008) corpus and previously annotated recordings of the AcListant project. For more information on the model and its configuration, please see Oualil et al. (2015).
Throughout the development of ATC-ANNO, a number of annotators worked with it. However, most used it before usage logs were implemented. To get a sense of the learning curve required to use the tool, we chose to perform this evaluation with a new annotator who did not yet have experience with the tool or task. The annotator in question is a native speaker of English. In preparation of the evaluation they trained for several hours on a separate dataset. Their task was to annotate eight hours worth of data for each of the four tool versions, resulting in a total of 32 hours of work. The annotator was instructed to always work with a specific tool version for one hour and then switch to the next version, i. e. work with the *Basic* for one hour, then use *XML-Suggest* for an hour, etc.

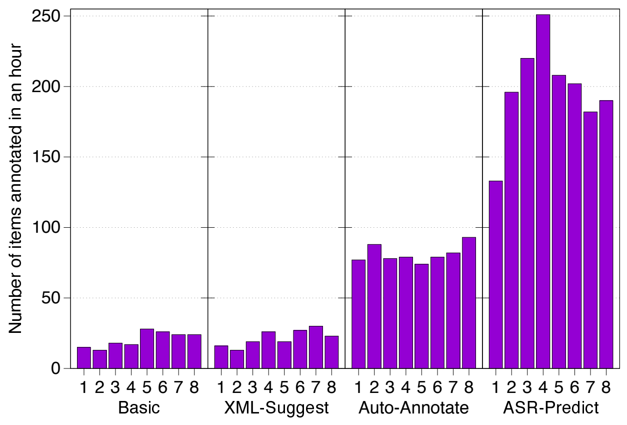


Figure 2: Number of items annotated per hour of work, separated by annotation tool version.

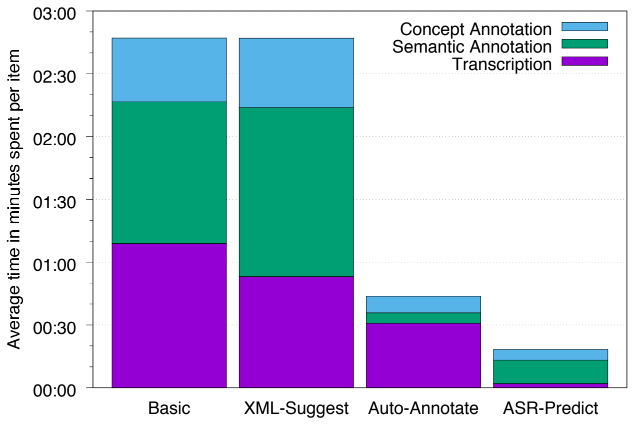


Figure 3: Average time spent annotating an item, separated by tool version. Each bar shows the total time as well as time spent on each of the subtasks (see Section 3).

Version	Annotated Items
Basic	165
XML-Suggest	173
Auto-Annotate	650
ASR-Predict	1 582

Table 3: Number of items annotated within 8 hours of use, separated by which version of ATC-ANNO was used.

5.1. Quantitative Results

Table 3 shows an overview of how many items were annotated in the allotted time for each version of the tool. At a total of 173 annotated items, *XML-Suggest* provides no clear improvement over *Basic* (165 items). Using *Auto-Annotate*, however, the annotator managed to cover 650 items, almost four times as many. For *ASR-Predict* this number is further doubled to 1582 items. Figure 2 provides a more detailed view of this, showing the hourly throughput of items. Hourly performance is quite stable for all versions, with the largest variance found in *ASR-Predict*. This is due to the large difference in speed between correct ASR output (requiring the annotator only to read it) and erroneous output (requiring manual corrections). Clusters of bad ASR output, such as were encountered in the first hour of *ASR-Predict*, can, therefore, have a negative impact on annotation speed.

Figure 3 shows the average time to annotate an item in each version, as well as time spent on each of the information layers. All time differences are statistically significant at $p < 0.05$ (two-tailed t-test), except total time and concept annotation for *Basic* vs *XML-Suggest*.

We can see that *Auto-Annotate* removes the need for manual annotation of the semantic layer almost entirely. The concept layer still costs slightly more time, as certain commands could not always be fully inferred from the semantic layer. Using *ASR-Predict*, manual transcription was almost never required, thanks to the high performance quality that the project-internal ASR model had reached by the time of our evaluation. Interestingly, semantic annotation required a bit more time than in *Auto-Annotate*, as the annotator tended to manually correct mistakes in the semantic layer, rather than using the automatic functions.

Another curious observation is that for *XML-Suggest*, annotation of the semantic layer is actually slower than in *Basic*. When working with *Basic*, the annotator would use the list of XML tags that had been provided during training. It turns out that copy-pasting tag structures from that list is slightly faster than using the dropdown menu in the ATC-ANNO interface.

Overall we found that, while XML suggestions were not a useful feature, both semantic auto-annotation and ASR predictions are immensely helpful, as long as their underlying resources are of high enough quality. This was the case for our evaluation, as the FST grammar had undergone multiple design iterations and the ASR had been trained on sufficient data and improved through use of context information (Oualil et al., 2015), reaching command error rates as low as 1.7% (Helmke et al., 2017).

In general, the features of ATC-ANNO grow together with the project for which they are used. Basic features like syntax highlighting and consistency indicators are useable from the start. Semantic auto-annotation becomes available once initial annotation rounds have provided experience with the data, allowing the creation of an FST grammar that models typical sentences. As annotation progresses, it can be speeded up by new iterations of the FST grammar design. Eventually, enough data is annotated to train preliminary ASR models and use their output as a starting point for annotation, creating a feedback loop that slowly shifts the task of the human annotator from full transcription to ASR output verification.

5.2. Qualitative Results

Due to the small number of annotators involved in the project, a user satisfaction study (e. g. QUIS (Harper and Norman, 1993)) was not feasible. However, multiple annotators remarked that writing XML markup was the most tedious part of the task. The semantic auto-annotation feature not only increased efficiency, but also improved user satisfaction. In the words of one annotator: *"It is now actually fun to do the annotations."*

6. Conclusion

We presented ATC-ANNO, a tool for semantic annotations of air traffic control communications. ATC-ANNO provides a number of assistive features for annotators, covering visual markers, markup suggestions, automatic annotation processes and preliminary annotation suggestions based on a speech recognition system.

We evaluated the most prominent of these features for how much they improve the efficiency of our annotation tool. Unsurprisingly, having high quality speech recognition can make annotation almost superfluous. More interestingly, when this is not available, the ability of our tool to semantically enhance sentences that are close (but not necessarily identical) to a given phraseology can significantly improve performance, increasing annotation speed by almost a factor of four.

The core source code of ATC-ANNO is publicly available under an open source licence. For new projects that seek to use ATC-ANNO, we recommend they start out with the *Basic* version during initial exploration of the data. Once an intuition for a common phraseology is given, some time should be spent on writing a grammar, which can then be used both for the *Auto-Annotate* version and as language model for early versions of the ASR. Once sufficient data is annotated, language models for ASR can be switched to n-gram models and *ASR-Predict* can be used for further annotation efforts.

7. Bibliographical References

Brugman, H. and Russel, A. (2004). Annotating Multimedia/Multi-modal Resources with ELAN. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 2065–2068, Lisbon, Portugal.

Cordero, J. M., Dorado, M., and de Pablo, J. M. (2012). Automated Speech Recognition in ATC Environment. In *Proceedings of the International Conference on Application and Theory of Automation in Command and Control Systems (ATACCS)*, pages 46–53, London, United Kingdom.

Eurocontrol. (2011). All Clear Phraseology Manual. Brussels, Belgium.

Hamel, C. J., Kotick, D., and Layton, M. (1989). Microcomputer System Integration for Air Control Training. Technical report, Naval Training Systems Center, Orlando, Florida.

Harper, B. D. and Norman, K. L. (1993). Improving User Satisfaction: The Questionnaire for User Interaction Satisfaction Version 5.5. In *Proceedings of the Annual Mid-Atlantic Human Factors Conference*, pages 224–228, Virginia Beach, Virginia, USA.

Helmke, H., Oualil, Y., and Schulder, M. (2017). Quantifying the benefits of speech recognition for an air traffic management application. In *Konferenz Elektronische-Sprachsignalverarbeitung*, pages 114–121, Saarbrücken, Germany.

Hofbauer, K., Petrik, S., and Hering, H. (2008). The ATCOSIM corpus of non-prompted clean air traffic control speech. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, Marrakech, Morocco.

Ohneiser, O., Helmke, H., Ehr, H., Gürlük, H., Hössl, M., Kleinert, M., Mühlhausen, T., Uebbing-Rumke, M., Oualil, Y., Schulder, M., Schmidt, A., Khan, A., and Klakow, D. (2014). Air Traffic Controller Support by Speech Recognition. In *Proceedings of the International Conference on Applied Human Factors and Ergonomics (AHFE)*, pages 492–503, Krakow, Poland.

Oualil, Y., Schulder, M., Helmke, H., Schmidt, A., and Klakow, D. (2015). Real-time integration of dynamic context information for improving automatic speech recognition. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Dresden, Germany.

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.

Shore, T., Faubel, F., Helmke, H., and Klakow, D. (2012). Knowledge-based word lattice rescoring in a dynamic context. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 1083–1086, Portland, Oregon, USA.

Viterbi, A. J. (1967). Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.