Building a Corpus for Cockpit Voice Recorder Transcripts

in the project seminar

Computational Natural Language Systems

RVS-OCC-01-06

RVS, Faculty of Technology, University of Bielefeld

Oliver Hölz* Thomas Hettenhausen†

October 23, 2001

Contents

1 Overview 3

2 Analysis of the CVR transcripts 4
  2.1 Sentence/expression structure 4
  2.2 Tokens 4
  2.3 Semantic structure 5

3 Implementation 8
  3.1 Languages we used 8
    3.1.1 XML 8
    3.1.2 Perl 8
  3.2 Design of the XML structure 8
  3.3 Design of the Perl script 9

4 Example analysis 11
  4.1 Original transcript 11
  4.2 XML source 11
  4.3 XML-to-screen output 14
  4.4 Summary 14

A The grammar behind the XML structure 17

B The Document Type Definition (DTD) 18

C List of keywords 19

*oliver.hoelz@uni-bielefeld.de
†thettenh@techfak.uni-bielefeld.de
D  Contexts  20
E  Flight phases  21
F  Source code of the Perl script  22
1 Overview

Cockpit Voice Recorder (CVR) transcripts are a valuable tool when studying aviation incidents. It is not always necessary to have the original audio recording, since often it suffices to be able to understand the process of the talks between involved persons.

In their original (released) format, the CVR transcripts are only little formatted. They are usually plain text files encoded in ASCII. They consist of lines, each line containing both information about the speaker of the message, the time when it was spoken and the message itself. In most cases, this information is separated only by blanks.

Our object is to provide an adequate corpus format for these transcripts for automated processing, making it easier to annotate, search and compare CVR transcripts automatically. This format enables us to help potential users of the corpus (e.g. [2]).
2 Analysis of the CVR transcripts

CVR transcripts are an important instrument in developing aviation safety. In their original plain text format the form of structure these documents have is that each line consists of three parts: a time code, a speaker and the actual transcribed message (see [5], [6] and [7]). Usually these parts are only separated by a blank, and sometimes not even this basic “standard” is used.

This kind of processing does not work very well with automated processing. Comparing them, annotating them, searching them, finding sequences and more with a computer system is not possible. It also means that the contained information which might be important for analyzing accidents is rather inaccessible.

2.1 Sentence/expression structure

As mentioned above, CVR transcripts are usually separated into lines. All lines contain a time code, a speaker and the message itself. The lines are consecutively ordered by the time code. For the speaker roles we identified different schemes for certain situations. In intra-cockpit communication, two schemes can be distinguished. The first scheme uses speaker roles such as “Captain”, “First Officer”, “Pilot-In-Command” [6].

Example:

11:58:05  Capt  Altimeter 1014, MDA is 173, Set.
11:58:08  F/O  1014, MDA 173 set right, V bugs.

The second one simply enumerates the different recording sources (e.g. head-phones) in use [7].

Example:

18:08:38  CAM-2  a little rudder.
18:08:39  CAM-1  all right.

Another scheme is utilized in transcripts of radio communication between a tower and different airplanes. Here the official calling signs identify the speaker [5].

Example:

16:14:39  ASA261  K Alaska two sixty one say again the frequency one two zero five two
16:15:03  R30  Uh Alaska two sixty one twenty six fifty two

Sometimes the last scheme is mixed with one of the first two if required.

The messages themselves can be made up of one or more sentences, although they are not always punctuated correctly. Since these documents are transcripts of spoken language, the sentences are often elliptic and may contain utterances and unintelligible words.

2.2 Tokens

After analyzing the sentence/expression structure, we took a closer look at the token level. The transcripts contain many technical terms special to aviation. Often these are grouped with common terms such as numbers, alphabets or directions.
We built a list of keywords intuitively by manually searching the transcripts for these technical terms. After we had done so with an initial transcript, we checked more transcripts against our list. When a potential keyword was not already in the list, it was added. Due to this iterative algorithm, we soon had a relatively stable list of keywords. We completed this list with the aforementioned numbers, alphabets, and direction identifiers, and then took a closer look at this list.

Since one of our objects is to analyze the semantic structure of the dialog, we must be able to determine the topics of the individual sentences. Together with the authors of [1], we came up with a number of categories. These included e.g. flight phases, parts of planes or airports, weather conditions and changes of the state of the plane (heading, speed, altitude). We also created a list of possible flight phases, such as cruise, taxiing and descend (see appendices C and D).

The keywords in our list then were annotated with both the meaning of the token, the contexts that its use could indicate and the flight phases it could occur in, and sorted alphabetically.

The syntax for an entry in the keyword list is

\[ \text{keyword[meaning][context][flight phase]} \]

so it would look like this

\[ \text{runway[runway][radioNumerals][taxiing:takeoff:approach:descend:landing]} \]

In this example \textit{runway} is the word actually appearing in the transcript. It has the meaning \textit{runway}, indicating that the speaker is talking about runways. Its context is \textit{radioNumerals}, because runways are usually referred to by numbers, as in “runway 09”, so a numeral can be expected to follow in the transcript. The flight phases in this example are those dealing with the beginning and the end of a flight, because that is when the crew usually talks about assigned runways.

2.3 Semantic structure

Semantic structure of the dialog refers to the relations between the different speech acts in the dialog, as opposed to the semantic structure in a single speech act.

With the categorisation of keywords it becomes possible to narrow down the number of possible topics of a message or sentence. This made it interesting for us to analyze the semantic structure of the dialog, to figure out whether these topics indicate relationships between consecutive sentences.

These relations are usually of the kind request-and-response or similar dialog acts. We then began to determine typical signatures for these dialog acts and created a list of keyphrases that indicate whether the sentence they are part of are a request or a response. These keyphrases are not to be confused with the keywords as described in the last section. While the former usually consist of more than one word and indicate a certain speech act such as request, the latter are the technical terms found throughout the transcripts that help us determining the subject.

E.g. “Could you” is a typical keyphrase indicating a request, while “brakes” is a keyword in a technical context.
The transcripts we analyzed manually were the following (the first three transcripts are those we spent most of our time with):

- China Airlines Flight 676 (1998-02-16) (China 676)
- Airborne Express Flight 827 (1996-12-22) (AirEx 827)
- UN Flight KSV 3275 (1999-11-12) (KSV 3275)
- Vladivostokavia Flight 352 (2001-07-04) (Vladivostokavia 352)

We looked for the following dialog acts:

- Statement with positive confirmation (state +)
- Statement with negative confirmation (state -)
- Question and answer (q / a)
- Request and response(s) (req / res)
- Statement and response(s) (state / res)

This resulted in the following figures (in percent):

<table>
<thead>
<tr>
<th>Flight</th>
<th>state +</th>
<th>q / a</th>
<th>req / res</th>
<th>state -</th>
<th>state / res</th>
</tr>
</thead>
<tbody>
<tr>
<td>AirFrance 4590</td>
<td>22</td>
<td>0</td>
<td>67</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>UN Flight KSV 3275</td>
<td>20</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>China Air 676</td>
<td>30</td>
<td>11</td>
<td>57</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Alaska Airlines</td>
<td>34</td>
<td>17</td>
<td>45</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Airborne Express</td>
<td>19</td>
<td>19</td>
<td>31</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>Vladivostokavia</td>
<td>43</td>
<td>14</td>
<td>29</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>

When we looked at the dialogs a little closer, we noticed that although these acts are different in form, they are pragmatically indistinguishable for our purpose.

One reason for this is the fact that there is a well defined chain of command. In such an environment, a statement like “the brakes need to be checked” by the highest ranking officer aboard is pragmatically a request or command to a lower ranking officer.

Also, for all involved it is clear that the situation appears in a technical environment. So the occurring speech acts in the CVR transcripts are limited in their variety.

Here are some examples that we give to show that for our work it is ok to merge the categories:

- **Question and Answer:**

  1. (from China 676)
     CAP
     F/O  Do you call them?
     F/O  Yes.
     F/O  Tower, Dynasty 676, 3 miles on final. Confirm clear to land.
This question resulted in the first officer calling the tower, so it is actually a request to call the tower.

2. (from Alaska Airlines 261)
   R14  Do you see him up there high ahead and to your right?
   SKW5154  Ah we’re looking Skywest fifty one fifty four.
   Skywest is not really answering the question, but instead beginning with a new action as requested.

**Statement and positive confirmation:**

1. (from Alaska Airlines 261)
   A SA261  Our intention is to land at Los Angeles.
   R25  Roger. You’re cleared to Los Angeles airport via present position.
   The tower interprets the simple statement as a request, acknowledges it and grants the requested clearance.

2. (from Kosovo 3275)
   CTL  You’re number two to a much faster aircraft just ahead of you now.
   RDO  Okay
   This can be interpreted as the tower requesting that the pilot lines up according to the aircrafts’ speed.

Therefore, we will be merging the three categories of QuestionAndAnswer, StatementAndPositiveConfirmation and RequestAndResponse(s) into just one RequestAndResponse(s) category.

With the merged categories, we get the following figures (in percent):

<table>
<thead>
<tr>
<th>Flight</th>
<th>req / res</th>
<th>state -</th>
<th>state / res</th>
</tr>
</thead>
<tbody>
<tr>
<td>AirFrance 4590</td>
<td>89</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>UN Flight KSV 3275</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>China Air 676</td>
<td>98</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Alaska Airlines 261</td>
<td>96</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Airborne Express 827</td>
<td>69</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>Vladivostokavia 352</td>
<td>86</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>

When we add the corresponding numbers together, the overall share of that category in the total number of dialog acts calculates (statistically weighted) to approx. 91 percent. Thus we will only be analyzing RequestAndResponse(s) structures with our program.
3 Implementation

3.1 Languages we used

3.1.1 XML

The Extensible Markup Language (XML) is a subset of SGML (Standard Generalized Markup Language), apart from the possibility to construct a start- and endtag in the same brackets e.g. <document/>>, which is not permitted in SGML. The XML 1.0 specification is released as a recommendation by the World Wide Web Consortium [13]. Since then XML has become a universal syntax for defining non-proprietary document markup and data formats. The opportunity to design your own DTD (Document Type Definition) gives you a huge flexibility to use XML. This is one of the advantages of XML in contrast to HTML, just as the separation of content and styling, which is blurred in HTML.

The important case for our project is the provision of content management in XML. The styling component is secondary, but still - as opposed to HTML - mandatory. To show our results we chose CSS (Cascading Style Sheets), a presentation through XSL (Extensible Stylesheet Language) is also possible. By means of the stylesheets, formatting for different output media is possible, e.g. printer or screen. More important is the construction of a reasonable DTD, which must contain all the elements and attributes we will use to markup our CVR transcripts. Inside the XML document itself, you need to refer to the DTD to which the document is suited.

3.1.2 Perl

Perl stands for “Practical Extraction and Report Language”. It has become a general purpose programming language, but it is most widely used for serious text processing. Because of its regular expressions, Perl is very powerful in searching and manipulating texts, and this is the main advantage for us to use perl as our programming language.

3.2 Design of the XML structure

For a well formed XML structure a DTD which is based on a context free grammar is essential. This context free grammar describes the hierarchical structure of the XML tree.

The grammar which is the basis for the tree of our XML structure is derived intuitively from the structure of the original transcripts (see appendices A and B).

The whole transcript is comprised of lines. Each of these lines of the transcript contains three parts:

- a consecutive time code,
- the person who is the speaker (or in some cases a machine sounding a warning tone) and
- the spoken message itself.
Sometimes, two or more lines form a block because they belong to a common
dialog act. The actual message can contain more than one sentence, and each
of these sentences can have its own topic.

Therefore, our XML structure starts with the <transcript> tag. Branch-
ing from there are the lines of the original transcript marked with <line>,
combined into <block>. A <line> consists of the three parts <time>,
<speaker> and <message>.

| transcript |
| block |
| regres |
| line |
| OR |
| AND |

If a sentence contains one or more keywords, these are marked with the
<keyword> tag. It has a required attribute, pointing out the keyword’s
possible contexts it can occur in (see appendix D). The <sentence> tag
has a required attribute as well: its sentence type (question, exclamation or
statement).

We included a <topic> tag for further extensions: when a sentence covers
more than one semantic topic it can be divided into sections.

3.3 Design of the Perl script

Since we are working with transcripts of natural speech and want to markup
relations between sentences spoken in the course of the dialog, we have to load
the whole transcript into memory at the beginning of the execution of the script.
We cannot simply mark it line by line, as we maybe could with some other text.
This enables us to jump back to previously processed pieces of the transcript.

Therefore, we internally build an array containing all the lines of the original
transcript, with each original line being one element in the array. Each of these
list elements then gets analyzed for time and speaker information as well as possible
sentence separators. With this new data, the single element gets written back
to the original array, but now it is a hash with the keys time, speaker and
message, where message itself is an array of the separate sentences.

So, the original line

becomes a hash with the following pairs of key and values:

<table>
<thead>
<tr>
<th>time</th>
<th>11:58:28</th>
</tr>
</thead>
<tbody>
<tr>
<td>speaker</td>
<td>F/O</td>
</tr>
</tbody>
</table>
| message| 1. Fasten right.  

For a more insight view, check the annotations made in the perl script itself (see appendix F).
4 Example analysis

4.1 Original transcript

Below is the first 8 lines of the original ASCII transcript of AA261. The only structure in this format is the use of the tabs between time, speaker and message and the linebreak at the end of each line.

16:09:55 ASA261 Center Alaska two sixty one we are uh in a dive here
16:10:01 R30 Alaska two sixty one uh say again
16:10:03 ASA261 (unintelligible) pitch
16:10:06 R30 Alaska two sixty one say again sir
16:10:06 ASA261 Yeah we’re out of twenty six thousand feet
we’re in a vertical dive - not a dive yet -
but uh we’ve lost vertical control of our airplane
16:10:07 R30 Alaska two sixty one roger
16:10:28 ASA261 We’re at twenty three seven request uh -
yeah we’ve got it back under control there no we don’t
(unintelligible)
16:10:36 R30 Alaska two sixty one uh say the altitude
you’d like to uh remain at

4.2 XML source

After processing the ASCII transcript, you get the same transcript as before in XML, with added meta information. At first glance, this looks harder to understand than the original. After applying an appropriate style sheet and viewing it with a XML browser, though, it offers much more options in human readability.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<tranascript>
  <line>
    <time>16:09:55</time>
    <speaker>ASA261</speaker>
    <message>
      <sentence type="statement">
        <topic>Center Alaska <keyword context="radioNumeral">two</keyword>
        <keyword context="radioNumeral">sixty</keyword>
        one</keyword>
        we are uh in a <keyword context="altitudeAbsolute,altitudeChange">dive</keyword>
        here.</topic>
      </sentence>
    </message>
  </line>
</transcript>
```

11
<line>
<time>
16:10:01
</time>
<speaker>
R30
</speaker>
<message>
<sentence type="statement">
  <topic>Alaska <keyword context="radioNumeral">two</keyword>
  <keyword context="radioNumeral">sixty</keyword>
  <keyword context="radioNumeral">one</keyword> uh say again.</topic>
</sentence>
</message>
</line>

<line>
<time>
16:10:03
</time>
<speaker>
ASA261
</speaker>
<message>
<sentence type="statement">
  (unintelligible) pitch.</sentence>
</message>
</line>

<line>
<time>
16:10:01
</time>
<speaker>
R30
</speaker>
<message>
<sentence type="statement">
  <topic>Alaska <keyword context="radioNumeral">two</keyword>
  <keyword context="radioNumeral">sixty</keyword>
  <keyword context="radioNumeral">one</keyword> say again sir.</topic>
</sentence>
</message>
</line>

<line>
<time>
16:10:06
</time>
<speaker>
ASA261
</speaker>
Yeah we’re out of twenty six we’re in a vertical dive a not a dive yet but uh we’ve lost vertical control of our airplane.

Alaska two sixty one roger.

We’re at twenty three seven request uh yeah we’ve got it back under control there we don’t (unintelligible).
4.3 XML-to-screen output

This is a screenshot of the web browser Opera v5.11 displaying the above part of the transcript in the XML version using a rudimentary CSS style sheet.

4.4 Summary

The markup of the syntactic structure resp. \texttt{<time>}, \texttt{<speaker>}, \texttt{<message>}, \texttt{<sentence>}, \texttt{<keyword>} etc. works correctly for more than 95 per cent of the original lines of the ASCII texts. Especially the marking-up and categorizing of the keywords and sentences is an enormous improvement in working with Cockpit Voice Recordings. Now these transcripts can be searched, parsed and compared automatically. This is important in analyzing the activities in the cockpit and the radio communication between the involved parties.

Our plans for the future are improving the markup of the semantic structures resp. request-response dialog structure. A better style sheet in XSL should
ensure a superior visualization of the results. For the perl script, a better user interface is in work.
References


Appendix A  The grammar behind the XML structure

This context free grammar is derived intuitively from the structure of the original Cockpit Voice Recorders transcripts and is the basis for our XML DTD.

\[ G=\{N, T, R, S\} \]

\[ N=\{\text{TRANSCRIPT, BLOCK, REQRES, LINE, BREAK, TIME, MESSAGE, SENTENCE, KEYWORD, SPEAKER, TEXT, CAPTAIN, FIRSTOFFICER, TOWER, OTHERSPEAKER01, OTHERSPEAKER02, OTHERSPEAKER03, OTHERSPEAKER04, LETTERS, LETTER}\} \]

\[ T=\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, :, , , -\} \]

\[ R=\{(\text{TRANSCRIPT}:={}\text{BLOCK} | \text{TRANSCRIPT} \text{ BLOCK}) \]
\[ (\text{BLOCK}:={}\text{REQRES} | \text{LINE}) \]
\[ (\text{REQRES}:={}\text{LINE} | \text{REQRES} \text{ LINE}) \]
\[ (\text{LINE}:={}\text{TIME} \text{ SPEAKER} \text{ MESSAGE} | \text{BREAK}) \]
\[ (\text{MESSAGE}:={}\text{SENTENCE} | \text{MESSAGE} \text{ SENTENCE}) \]
\[ (\text{SENTENCE}:={}\text{TOPIC} | \text{SENTENCE} \text{ TOPIC}) \]
\[ (\text{TOPIC}:={}\text{KEYWORD} | \text{LETTERS} | \text{DOTS} | \text{KEYWORD} \text{ SENTENCE} | \text{LETTERS} \text{ SENTENCE} | \text{DOTS} \text{ SENTENCE}) \]
\[ (\text{SPEAKER}:={}\text{CAPTAIN} | \text{FIRSTOFFICER} | \text{TOWER} | \text{OTHERSPEAKER01} | \text{OTHERSPEAKER02} | \text{OTHERSPEAKER03} | \text{OTHERSPEAKER04}) \]
\[ (\text{BREAK}:={}\text{LETTERS}) \]
\[ (\text{TEXT}:={}\text{LETTERS}) \]
\[ (\text{TIME}:={}\text{LETTERS}) \]
\[ (\text{CAPTAIN}:={}\text{LETTERS}) \]
\[ (\text{FIRSTOFFICER}:={}\text{LETTERS}) \]
\[ (\text{TOWER}:={}\text{LETTERS}) \]
\[ (\text{OTHERSPEAKER01}:={}\text{LETTERS}) \]
\[ (\text{OTHERSPEAKER02}:={}\text{LETTERS}) \]
\[ (\text{OTHERSPEAKER03}:={}\text{LETTERS}) \]
\[ (\text{OTHERSPEAKER04}:={}\text{LETTERS}) \]
\[ (\text{KEYWORD}:={}\text{LETTERS}) \]
\[ (\text{DOTS}:={}\text{LETTERS}) \]
\[ (\text{LETTERS}:={}\text{LETTER} | \text{LETTERS} \text{ LETTER}) \]
\[ (\text{LETTER}:={}a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | : | , | -) \]
\} \]

\[ S=\{\text{TRANSCRIPT}\} \]
Appendix B  The Document Type Definition (DTD)

This XML DTD is designed on the basis of the context free grammar.

<!ELEMENT transcript (block+)>  
<!ELEMENT block (reqres|line)>  
<!ELEMENT reqres ((line+)|(reqres+)>))  
<!ELEMENT line ((time, speaker, message)|break)>  
<!ELEMENT message (sentence*)>  
<!ELEMENT sentence (topic*)>  
<!ELEMENT topic (keyword|PCDATA|dots)+>  
<!ELEMENT speaker (captain|firstofficer|tower|otherspeaker01|otherspeaker02|otherspeaker03|otherspeaker04)>  
<!ELEMENT break #PCDATA>  
<!ELEMENT text #PCDATA>  
<!ELEMENT time #PCDATA>  
<!ELEMENT captain #PCDATA>  
<!ELEMENT firstofficer #PCDATA>  
<!ELEMENT tower #PCDATA>  
<!ELEMENT otherspeaker01 #PCDATA>  
<!ELEMENT otherspeaker02 #PCDATA>  
<!ELEMENT otherspeaker03 #PCDATA>  
<!ELEMENT otherspeaker04 #PCDATA>  
<!ELEMENT keyword #PCDATA>  
<!ELEMENT dots #PCDATA>  

<!ATTLIST sentence type (interrogation|exclamation |statement) #REQUIRED "statement">  

<!ATTLIST keyword context (C_all|C_altitude|C_absoluteAltitude |C_relativeAltitude|C_altitudeChange|C_communication |C_radioFrequency|C_radioAlphabet|C_radioNumerals|C_confirmation |C_instruction|C_direction|C_north|C_west|C_south|C_east|C_left|C_right|C_up|C_down|C_directionChange |C_distance|C_absoluteDistance|C_relativeDistance|C_distanceChange|C_flightState|C_taxiing|C_takeoff|C_climb|C_cruise|C_approach |C_descend|C_landing|C_emergency|C_urgency|C_goaround|C_heading |C_absoluteHeading|C_relativeHeading|C_headingChange|C_measurement |C_angleMeasurement|C_altitudeMeasure|C_speedMeasure |C_heightMeasurement|C_distanceMeasurement|C_volumeMeasurement |C_timeMeasurement|C_none|C_object|C_flyingObject|C_groundObject |C_airport|C_airportObject|C_taxiway|C_runway|C_runwayObject |C_tower|C_plane|C_terrain|C_airTrafficControl|C_permission |C_cleared|C_notCleared|C_planePart|C_controllableSystem |C_staticSystem|C_position|C_absolutePosition|C_relativePosition |C_positionChange|C_mark|C_threshold|C_quality|C_speed |C_absoluteSpeed|C_relativeSpeed|C_speedChange|C_technical|C_weather |C_fog|C_ice|C_rain|C_snow|C_visibility|C_wind|C_windshear|F_taxiing |F_takeoff|F_climb|F_cruise|F_approach|F_descend|F_landing|F_goaround |F_emergency|F_urgency|F_all|F_none) #REQUIRED "C_all">
Appendix C  List of keywords

This list of keywords was built intuitively using an iterative algorithm.

AOA  eastbound  mayday  stabilizer
above  echo  mike  stand by
affirmative  eight  miles  standing by
airport  eighty  minutes  start
airspeedindicator  elevator  nine  takeoff
alpha  emergency  ninety  tango
altitude  engine  nifty  taxiway
approach  field  no  ten
approaching  fifty  nose  thirty
approved  five  north  three
autopilot  flaps  northbound  threshold
baseleg  flight level  november  throttles
below  fly  ok  thousand
block  flying  one  thrust lever
boots  follow  oscar  thrust
brake pressure  forward  pan  traffic
brake  four  papa  trim
bravo  forty  power  turn
bugs  Foxtrot  proceed  turning Point
call  frequency  quebec  turning
captured  fuel  rain  twenty
centerline  gear  raining  two
charlie  go around  rate of descend  understood
check  golf  re清除ed  uniform
checklist  heading  reduce  up
circuit  hold  reducing  vector
clearance  hotel  right  victor
cleared  hundred  romeo  visibility
climb  hydraulics  roger  weather
closer  inbound  rudder  west
compressor  igniter  runway end  westbound
contact  ignition  runway  whiskey
copy  increase  seven  wind
course  india  seventy  windshears
crosswind  juliett  shears  windshield wipers
decrease  kilo  showers  xray
degrees  knots  sierra  yankee
delta  landing  six  yaw dampers
descend  left  sixty  yes
descending  lima  south  zero
distance  lower  southbound  zulu
dive  maintain  speed
down  mark  speedbrake
east  marker  spoilers
Appendix D  Contexts

These contexts were determined together with the authors of [1] by analyzing the list of keywords.

\begin{itemize}
  \item C_all
  \item C_altitude
  \item C_absoluteAltitude
  \item C_relativeAltitude
  \item C_altitudeChange
  \item C_communication
  \item C_radioFrequency
  \item C_radioAlphabet
  \item C_radioNumerical
  \item C_confirmation
  \item C_instruction
  \item C_direction
  \item C_north
  \item C_nw
  \item C_nw
  \item C_west
  \item C_south
  \item C_sw
  \item C_se
  \item C_east
  \item C_left
  \item C_right
  \item C_up
  \item C_down
  \item C_directionChange
  \item C_distance
  \item C_absoluteDistance
  \item C_relativeDistance
  \item C_distanceChange
  \item C_flightState
  \item C_taxing
  \item C_takeoff
  \item C_climb
  \item C_cruise
  \item C_approach
  \item C_descend
  \item C_landing
  \item C_emergency
  \item C_urgency
  \item C_gear
  \item C_heading
  \item C_absoluteHeading
  \item C_relativeHeading
  \item C_headingChange
  \item C_measurement
  \item C_angleMeasurement
  \item C_altitudeMeasurement
  \item C_speedMeasurement
  \item C_heightMeasurement
  \item C_distanceMeasurement
  \item C_volumeMeasurement
  \item C_timeMeasurement
  \item C_airport
  \item C_airObject
  \item C_flyingObject
  \item C_groundObject
  \item C_taxiway
  \item C_runway
  \item C_runwayObject
  \item C_tower
  \item C_plane
  \item C_tan
t
  \item C_clear
  \item C_notClear
  \item C_planePart
  \item C_controllableSystem
  \item C_staticSystem
  \item C_position
  \item C_absolutePosition
  \item C_relativePosition
  \item C_positionChange
  \item C_mark
  \item C_threshold
  \item C_quality
  \item C_speed
  \item C_absoluteSpeed
  \item C_relativeSpeed
  \item C_speedChange
  \item C_technical
  \item C_weather
  \item C_fog
  \item C_ice
  \item C_rain
  \item C_snow
  \item C_visibility
  \item C_wind
  \item C_windshear
\end{itemize}
Appendix E  Flight phases

These flight phases were taken from the state machine as described in [1].

F_taxiing
F_takeoff
F_climb
F_cruise
F_approach
F_descend
F_landing
F_goaround
F_emergency
F_urgency
F_all
F_none
Appendix F  Source code of the Perl script

#!/perl

#BEGIN SUBROUTINES

# this subroutine gets the argument message (type: array of strings)
# elements of message are sentences, each gets split into words (token: whitespaces)
# and is compared with the elements in the keywordarray. if found, then marked up
# and written back into variable.
# finally returns a string containing the whole massage marked with everything.
sub markUpKeywords {
    my(@sentences) = @_;
    my($sentence, @words, $word, $keyword, $message);

    foreach $sentence (@sentences) {
        @words = split(/\b/, $sentence,);

        # marks the keyword with context included as attribute to the keyword tag
        foreach $keyword (keys %kWoL) {
            $sentence = "";
            foreach $word (@words) {
                if ($word =~ /$keyword/gi) {
                    @keywordattribsarray = @{$kWoL{$keyword}};

                    foreach $keywordattrib (@keywordattribsarray) {
                        if ($keywordattribs ne "") {
                            @keywordattribs = '$keywordattribs","'.$keywordattrib; }
                        else {
                            @keywordattribs = $keywordattrib; }
                    } 

                    $word = "<keyword context=""$keywordattribs"">$word."<\keyword>";
                    $keywordattribs = "";
                } $sentence = "$sentence.$word; 

            } $

        } if ($sentence = "<interrogation \/>") { $sentence = "<sentence
type="interrogation"/>".$sentence."\?</\topic><\sentence>"; }
        elsif ($sentence = "<exclamation \/>") { $sentence = "<sentence
type="exclamation"/>".$sentence."!</\topic><\sentence>"; }
else { $sentence = "<sentence type="statement">"<topic>".
    $sentence."/\t/topic/<\sentence>"; }
$sentence =~ s/<interrogation \/>|<exclamation \/>|<statement \/>//g;
$message = $message.$sentence;
}
push @message, $sentence;
#$message = "<message>".$message."<\message>";
return @message;
}

# builds up an hash out of our keywords
sub buildKeywordHash
{
  while(defined($keywordline=<KWIN>))
  {
    chomp($keywordline);
    @hashline = split(\[/, $keywordline, 2);
    $hashline[1] =~ s/\[\]/\]/g;
    @contexts = split(\://, $hashline[1],);
    $kwhol{$hashline[0]} = [@contexts];
  }
}

# reading in of command line arguments
sub readCmdLine
{
  foreach $cmdarg (@ARGV)
  {
    if ($cmdarg =~ /^-i/) # input
      {
        $inputfile = $cmdarg;
        $inputfile =~ s/-i//;
      }
    elsif ($cmdarg =~ /^-k/) # keywords
      {
        $keywordfile = $cmdarg;
        $keywordfile =~ s/-k//;
      }
    elsif ($cmdarg =~ /^-o/) # output
      {
        $outputfile = $cmdarg;
        $outputfile =~ s/-o//;
      }
    elsif ($cmdarg =~ /^-h/) # cmd line help
      {
        typeHelp();
        goto END;
      }


if ($inputfile eq "")
{
    print "\nNo inputfile specified.\nTerminating.\.\.\nUse -h for help.\n"
    goto END;
}
}

# reading in of the file config.cfg for global configuration
sub readConfig
{
    open(CFGIN, "<config.cfg");
    while(defined($configinput=<CFGIN>))
    {
        chomp($configinput);

        # determines which file to write to (just rename input to .xml or
        # specified on command line)
        if ($configinput =~ /generate_output_filename\: yes\)/i)
        {
            if ($outputfile eq "")
            {
                $outputfile = $inputfile;
                $outputfile =~ s/\s/\s/xml/i;
            }
        }
        elsif ($configinput =~ /keyword_file\:/gi)
        {
            $keywordfile = $configinput;
            $keywordfile =~ s/\s/\s/xml/i;
        }
        # determines the path to the style sheet
        elsif ($configinput =~ /stylesheet_path\:/gi)
        {
            $stylesheet = $configinput;
            $stylesheet =~ s/\s/\s/xml/i;
        }
    }
}

    close(CFGIN);
}

# checks whether the variables are set
sub variableCheck
{
    if ($keywordfile eq "")
    {

    }
print "\nNo keywordfile specified\nTerminating\nUse -h for help\n";
goto END;
}
elsif ($outputfile eq "")
{
    print "\nNo outputfile specified\nTerminating\nUse -h for help\n";
goto END;
}
elsif ($stylesheet eq "")
{
    $stylesheet = "transcript.css"
}

# prints the help to the console
sub typeHelp
{
    print "\n"
    print "Available options/switches:\n"
    print "\-i\": specifies input filename\n"
    print "\-o\": specifies output filename\n"
    print "\-k\": specifies keyword list filename\n"
    print "\n"
    print "Do not put blanks between switch and filename!"
}

# SCRIPT BODY

# FILEHANDLES
$keywordfile = "";
$inputfile = "";
$outputfile = "";
@transcript;
readCmdLine();
readConfig();

variableCheck();

open(KWIN, "<".$keywordfile); # open handle to keyword list
open(OUT, "">".$outputfile); # open handle to output file
open(IN, "<".$inputfile); # open handle to input ascii transcript

@transcript_in = <IN>;
@transcript_out;

foreach $line (@transcript_in)
{
    chomp ($line);
    $line =~ s/\./\+/g; # replaces dots with a tag
# create 3 strings out of line

($time, $speaker, $messagestring) = split(/\t/, $line, 3);
$time = "<time>".$time."</time>";
$speaker = "<speaker>".$speaker."</speaker>";

# add a tag to the punctuation
$messagestring =~ s/\?\?/interrogation \>/\?\?/g;
$messagestring =~ s/\!\!/exclamation \!/\!\!/g;
$messagestring =~ s/\./statement \>/\./g;

# splits the message into the single sentences
@sentencearray = split(/\?\!\!/\./\?\!\./\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?\?

26