The AMIDA Automatic Content Linking Device: Just-in-Time Document Retrieval in Meetings

Andrei Popescu-Belis[1], Erik Boertjes[2], Jonathan Kilgour[3], Peter Poller[4], Sandro Castronovo[4], Theresa Wilson[3], Alex Jaimes[1], and Jean Carletta[3]

[1] IDIAP Research Institute
Centre du Parc, P.O. Box 592
CH-1920 Martigny, Switzerland
andrei.popescu-belis@idiap.ch
[2] TNO ICT, Brassersplein 2
NL-2612 CT Delft, The Netherlands
[3] HCRC and CSTR, University of Edinburgh
2 Buccleuch Place, Edinburgh, EH8 9LW, UK
[4] DFKI GmbH, Stuhlsatzenhausweg 3
D-66123 Saarbruecken, Germany

Abstract. The AMIDA Automatic Content Linking Device is a just-in-time document retrieval system that constantly retrieves items from a repository and displays them to a participant or to all of them. The repository includes meeting related documents together with excerpts from previous meetings of the group. The device can be used online during a meeting, but also offline, integrated in a meeting browser. Its main components and their communication are described: the Document Bank Creator, the Indexer, the Query Aggregator, and the User Interface. Results and feedback for a first version of the system are then outlined, together with plans for future development within the AMIDA project. (*Demo proposal.*)

Key words: just-in-time retrieval, meeting assistants, meeting processing, online document retrieval.

1 Objectives

Participants in a meeting often mention documents containing facts that are currently discussed, but only a few of them are at hand. Searches could be performed in a document management system for the right piece of information, but the participants in a meeting usually do not have the time to perform such operations frequently during a meeting. Even when browsing through the recording of a previous meeting, users do not have the time to search for additional information related to that meeting, though this would help them to better understand or locate information in the meeting.

Therefore, a system that could relate ongoing discussions to potentially relevant documents or other pieces of information, i.e. an Automatic Content Linking Device (ACLD), would be very valuable in at least two application scenarios:

1. *Just-in-time retrieval* [1–3]: participants to a meeting are constantly given suggestions about documents (including excerpts of previous meetings) that are potentially relevant to the ongoing discussion. Participants are free to ignore them, or to start using them to enhance the discussion, e.g. with figures, precise facts, or decisions that were made in previous meetings.
2. *Document/speech alignment for meeting browsers* [4, 5]: users of a meeting archive can view the recordings of previous meetings augmented with related documents, regardless of whether the participants to the meeting referred to them explicitly or not. This can be essential for meetings whose main purpose is to discuss a long document, e.g. a report, and might provide a quicker understanding of the meeting context.

Conceptually, the content linking mechanism is the same in both cases – only the resources that are available and the constraints of producing real-time results are different. Therefore, in the current implementation of the ACLD, a meeting from the AMI Meeting Corpus [6] is replayed (here, ES2008d as a case study) and as it plays, the device indicates relevant documents and pseudo-documents (segments from previous meetings of the series, i.e. ES2008a-c). The highlighting changes as the meeting proceeds to show those past documents that are most relevant at each point, which the user can access in various ways.

2 Overall Architecture

The Automatic Content Linking Device performs searches at regular intervals, within a database of documents that can be prepared before a meeting, including for instance reports, emails that were exchanged, minutes and presentations from past meetings, etc. Pseudo-documents such as fragments of previous meetings that are related to the current one are also included.

The search criterion is constructed based on the terms that were recognized automatically from the meeting discussion (some pre-specified terms can receive greater weight) and the results are presented as a list of document names ordered by relevance (this list can be empty if no document matches enough the words that were recognized). A persistence mechanism helps documents that are often retrieved to remain some time at the top of the list. A user interface offers the participants quick access to the content of the documents that are retrieved, if they need to search them for valuable information.

These functionalities are supported by a number of modules that exchange data through a subscription-based client/server architecture called "the Hub" [7]. The architecture of the ACLD is shown in Figure 1, while the main components are first outlined below and then described in the following subsections.

Document Bank Creator (DBC): gathers documents that are of potential interest for an upcoming meeting – this is done semi-automatically from the MMM server in the current implementation.

Document Indexer (DI): creates an index over the document bank of the current meeting.

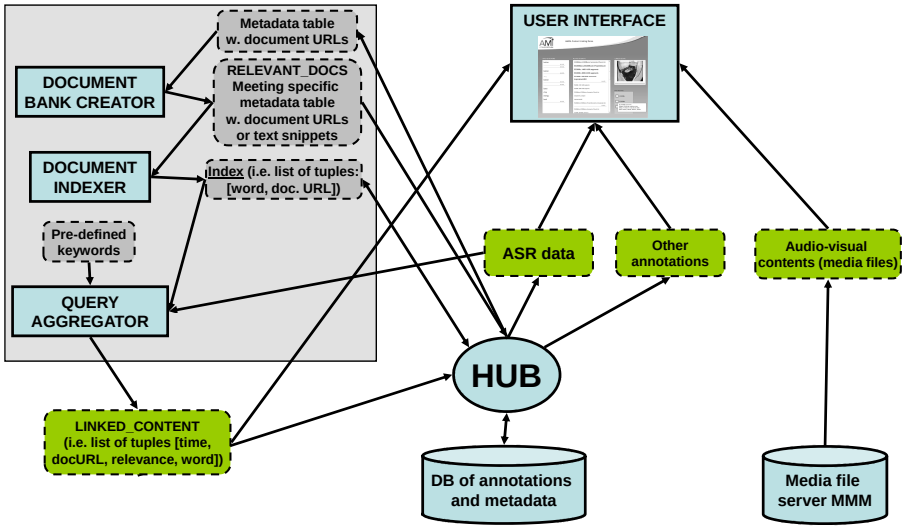

Fig. 1. Architecture of the AMIDA Automatic Content Linking Device.

Query Aggregator (QA): performs document searches at regular time intervals, using words and terms that are recognized automatically from the meeting discussion. Uses also a list of pre-specified keywords which, when they are detected in speech, receive more weight within the query. Produces a list of document names, ordered by relevance, based on the search results and on a persistence model explained below.

User Interface (UI): displays results from the QA and offers quick access to TXT/HTML and source versions of documents.

3 Components of the Automatic Content Linking Device

3.1 User Interface (UI)

We start the description of the ACLD with the User Interface, as this allows a better understanding of the functioning of the system. A snapshot of the UI is shown in Figure 2.

In an ideal scenario, the only information given to the UI is the identifier of a completed meeting to display, or some form of connection to an ongoing meeting in an instrumented meeting room [8]. This allows the interface to retrieve via the Hub all the pointers to the related media, and to subscribe to all the annotations that will be displayed, including the annotation produced by the

Fig. 2. Snapshot of AMIDA ACLD's user interface over meeting ES2008d. Left column shows recognized keywords with timing, central column shows relevant documents (with font size coding for importance), and bottom-right box shows a list of the previous meetings of the series, giving access to their contents.

Query Aggregator. In demonstrations, it is more convenient for repeatability reasons to use a completed meeting – in the present version, we use ES2008d, the fourth meeting of the ES2008 series in the AMI Corpus [6] – hence, a number of metadata variables are hard-coded into the UI.

The UI displays at any given moment in a meeting (or place in a transcript) the N most relevant documents, based on the *Linked_Content* data structure it receives from the QA, which contains information about the document URLs, their types and relevances, meeting time and detected keywords. This list is constantly updated as the meeting proceeds. The interface offers the users several possibilities for interaction with documents, depending on each document type. For a meeting fragment, hovering over its label displays its extractive summary (obtained on-the-fly via the Hub), while clicking on the label displays the ASR transcript. For documents, clicking on their label displays their content in a text window, from where a formatted version in HTML can also be obtained. The source document can also be retrieved, but opening it with its dedicated program (often MS Word or Powerpoint) slows considerably the viewing process.

3.2 Document Bank Creator (DBC)

The Document Bank Creator is run offline before a meeting to create the bank of documents and pseudo-documents that will be searched over during the meting. This is a preparation task, which generates a metadata layer with pointers to documents, or copies documents in a separate folder, in preparation for the Document Indexer. In a more user-friendly scenario for the future, the module could determine automatically the potentially relevant documents based on the project or series the meeting belongs to.

The DBC includes documents, fragments of previous meetings (200 word long, created from the ASR of the previous meetings), slides, and emails. It accepts heterogeneous file formats, and extracts text from them using calls to the proprietary software that created the files. In the process, the module also generates HTML versions of each document, which are easier and quicker to visualize than the original MS Office versions.

3.3 Document Indexer (DI)

The Document Indexer uses the text version of the files associated to the current meeting by the DBC to construct an index, i.e. a data structure that optimizes word-based search over the document set, which can become quite large over time. The index can also be conceived of as a new annotation layer, represented logically as a list of tuples (meeting, keyword, doc_type, URL), where the URLs are used as unique identifiers of the documents. The DI uses a state-of-the-art system, Apache Lucene (in its Perl implementation called Plucene), using all words as keywords and building an optimized index using word stemmers and the TF*IDF weighing scheme.

As the index is a permanent layer of information concerning the documents related to a meeting, it could be stored in a declarative format in the Hub's main database, from where it could be retrieved at the beginning of the demo by the Query Aggregator, which is constantly using it. Alternatively, in the present implementation, the index is accessed directly by the Aggregator as a set of files in native Plucene format.

3.4 Query Aggregator (QA)

The Query Aggregator periodically extracts from the speech of a given meeting a list of keywords that are mentioned, using the ASR (or even, for development purposes, a manual transcript). The aggregator gets the words via the Hub and processes them in batches of fixed size (currently every 30 seconds), extracting keywords from them. Then, it combines them to build a query, and searches the index to retrieve relevant documents, which are sent as new annotations to the Hub, from where they can be used by the UI to display the document labels and give access to them. This task has thus a similar goal as speech/document alignment [9, 10], except that "alignment" is viewed here as the construction of sets of relevant documents for each meeting segment, and not only as finding

the document that the segment "is about" – the retrieval techniques that are employed are therefore different too.

A number of pre-specified keywords are detected as they come through the ASR, and their importance is increased when doing the search, using Plucene's boosting mechanism (the weight of the boosting is set at 5 times the weight of non-boosted words). However, the QA works also without a list of boosted terms. A specific list was defined by the user-study group for the meeting series under study, and at present it contains words or expressions such as 'cost', 'energy', 'component', 'case', 'chip', 'interface', 'button', 'L_C_D', 'material', 'latex', 'wood', 'titanium', and so on (for a total of about 30 words). In addition, the words from the ASR or transcript are filtered for stopwords (a list of about 80 words was defined), so that mostly content words are used for search.

The QA performs document search by matching the keywords from the ASR (with stopwords removed) with those from the index constructed above and returns the most relevant set of documents at that given moment, more specifically a list of tuples such as (meeting, time, keyword, relevance, doc_type, pointer). It is useful to include in this annotation the keywords that were matched (i.e. the ones that helped to retrieve the specific document) as well as a relevance score produced by Plucene to allow the interface to sort the relevant documents as needed. This new annotation layer, called *Linked_Content*, is sent via the Hub and stored in the Hub's database, to which the UI subscribes.

To avoid inconsequent results from one 30-second time frame to another, due to the fact that words vary considerably and therefore search results vary as well, a *persistence mechanism* was defined. This mechanism modifies the current relevance scores for each document returned by Plucene considering those from the previous time frame. If t_n denotes the current time frame and t_{n-1} the previous one, and if $r(t_n, d_k)$ is the relevance of document d_k computed by Plucene, then the updated relevance $r'(t_n, d_k)$ computed using the smoothing mechanism, is $r'(t_n, d_k) = r(t_n, d_k) + \alpha * r'(t_{n-1}, d_k)$, where α is a smoothing factor. Roughly, a larger value of α denotes a larger persistence – but α should be set below 1, because if $\alpha > 1$ then $r(t_n))$ keeps increasing even if the document is no longer retrieved by Plucene. In our experiments, a typical value of $\alpha = 0.8$ was used.

Additionally, a filtering mechanism deletes the least relevant of the documents sent to the UI. In any case, at most N documents are returned (here, $N = 6$), but fewer documents may be returned if there is a large difference in (updated) relevance between the highest and the lowest ones. More precisely, the QA maintains internally a list of all documents that were retrieved, sorted by decreasing relevance. From this list, the QA sends at most N to the UI, with two additional constraints: (1) documents with relevance below a threshold (typically 0.2) are never sent; (2) the list of results is truncated where relevance decreases sharply, typically when $r'(t_n, d_{k+1}) \leq 0.5 * r'(t_n, d_k)$.

An offline version of the QA generates static XML and HTML views for past meetings, which are used for debugging and for evaluation. The HTML view shown in Figure 3 displays on the left the ASR of the current meeting segment t_n, and on the right up to six most relevant documents (in HTML

version) with their computed relevances $r'(t_n, d_k)$. The keywords are highlighted in pink, both within the meeting transcript and in the documents. The words from the transcript are highlighted (in blue) only in the documents when they are found, otherwise the entire meeting segment (except the stopwords) would be highlighted, which is not very informative. The upper frame of the window allows to select any segment of the current meeting based on its timing in seconds.

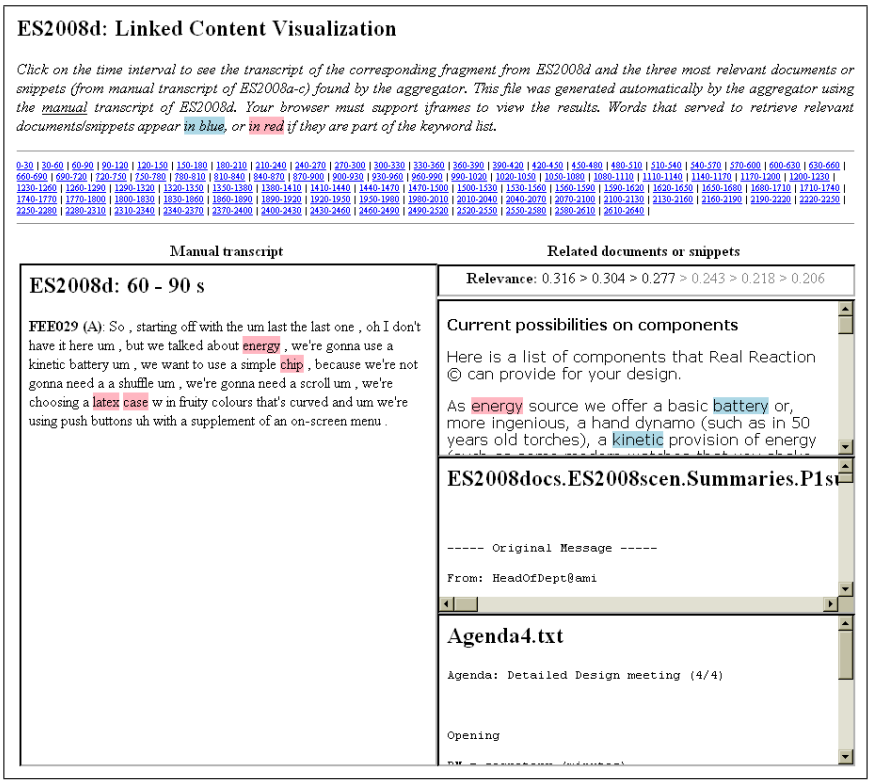

Fig. 3. HTML view of the offline output of the Query Aggregator.

4 Implementation

The current version of the AMIDA Automatic Content Linking Device was demonstrated starting February 2008. Both the UI and the QA are implemented using two components: a Java front-end ensuring communication with the Hub (as consumer or producer-and-consumer) and a separate part in a different programming language – the UI itself is in Flash, and the QA is in Perl.

The demo runs on a single Windows PC or over a network, and other platforms will be considered later. The main software prerequisite is the Hub itself,

which requires a MySQL database with one table (for timed triples) and Java JDK/SDK 1.5 or later. To run the QA, Perl and the Plucene module are required. Compilation of all source files is centrally managed by a `build.xml` file in the top level directory of the repository, which requires the `Ant` software. A number of variables can be set by modifying the initial lines of `build.xml`. The same `build.xml` file can also execute the following groups of actions required to start the ACLD on meeting ES2008d (assuming all sources were compiled):

1. Start the Hub, clean the table of triples, then refill its content via the Hub to the state that holds after meetings ES2008a-c but before ES2008d.
2. Start the QA Java front end which subscribes to the Hub, which calls the Perl script on demand.
3. Start the UI Java front end which also subscribes to the Hub, and the Flash GUI connected to it by sockets.
4. Stream the automatically-recognized words for ES2008d through the Hub.

5 Evaluation, Feedback and Future Plans

The execution tests of the ACLD have been satisfactory: the communication between the modules using the Hub works smoothly, and the logs show proper connection, sending and receival of annotation triples. The documents that are retrieved contain the expected words and keywords – this was checked on the static HTML representation produced by the QA (Figure 3). All the functionalities offered by the UI over these documents are available. The nature of Perl scripting makes it easy to change many of the parameters of the QA, even while the system is running, which allows experimenting with various values of the persistence and filtering model, and with different lists of keywords and stopwords.

5.1 Performance Evaluation

The *performance evaluation* of the ACLD application is the topic of future work. One can of course test the performance of the retrieval system in terms of precision and recall, but this requires the definition of a ground truth document set for each time interval of a meeting, which is the main difficulty for such an evaluation. Three approaches of the ACLD evaluation problem are planned.

To construct ground truth data, which can be used to evaluate automatically the ACLD every time its parameters are changed, an experiment must be set up, in which subjects are shown a meeting transcript (segmented into 30-second segments) and a set of documents, and are asked to associate to each segment of the meeting the documents that they believe are the most related to that segment. The main challenge of this approach is to demonstrate acceptable inter-coder agreement. To make the task more tractable, subjects could work only on a subset of documents each, and the final result would be based on the integration of a large number of subjects' responses.

Conversely, users could watch (part of) a meeting and judge the relevance of each document returned by the ACLD, using a dedicated interface similar to the BET wrapper [11]. Of course, this cannot measure the 'silence' of the ACLD, but only its 'noise' or 'purity'.

The ACLD should also be tested *in use* on the participants to an ongoing meeting, by measuring how often they consult the documents found by the ACLD. The results would be very informative to end-users, but such an experiment is costly and must be repeated every time the system is changed.

5.2 Feedback from Potential Users

The ACLD was demonstrated to potential industrial partners, and to a review committee, and received very positive verbal evaluation, as well as useful feedback and suggestions for future work, which can be grouped into three categories. The participants found that both online and offline application scenarios are promising, as well as individual and group uses.

The **graphical layout of the interface** could be improved by allowing a larger part of the screen to be used for displaying the documents, using larger overviews of each document, and discarding past documents more quickly. This would also help to reduce the number of mouse clicks required to access the content of documents. Color-coding the document types and displaying their relations to the meeting ASR would also improve user experience.

Another line of suggestions concerns the **document repository**, which can be extended in various ways. The repository could include documents from larger sets, which are not entirely known to users, so that the interface brings new knowledge into a meeting. These sets could be private, personalized and better structured. A significant extension of the repository would also include a list of websites, but this should be limited to avoid too much potential noise in the results.

A number of **additional functionalities** were suggested. For instance, keeping a record of the documents that were consulted during a meeting might help users who want to go back to them after the meeting. Detecting similarities with previous discussions would help alerting users that they already had this discussion before. Finally, retrieval could be improved by including a relevance feedback mechanism for the returned documents, by representing keywords in a structured manner, and by using word sense disambiguation to improve the precision of the retrieval.

Acknowledgments

This work was supported by the European IST Programme through the AMIDA Integrated Project FP6-0033812.

References

1. Hart, P.E., Graham, J.: Query-free information retrieval. IEEE Expert: Intelligent Systems and Their Applications **12**(5) (1997) 32–37

2. Budzik, J., Hammond, K.J.: User interactions with everyday applications as context for just-in-time information access. In: International Conference on Intelligent User Interfaces, New Orleans, LA (2000)
3. Henziker, M., Chang, B.W., Milch, B., Brin, S.: Query-free news search. World Wide Web: Internet and Web Information Systems **8** (2005) 101–126
4. Rhodes, B.J., Maes, P.: Just-in-time information retrieval agents. IBM Systems Journal **39**(3-4) (2000) 685–704
5. Franz, A., Milch, B.: Searching the web by voice. In: Coling 2002 (19th International Conference on Computational Linguistics), Taipei (2002) 11–15
6. Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., Lathoud, G., Lincoln, M., Lisowska, A., McCowan, I., Post, W., Reidsma, D., Wellner, P.: The AMI Meeting Corpus: A preannouncement. In Renals, S., Bengio, S., eds.: Machine Learning for Multimodal Interaction II. LNCS 3869. Springer-Verlag, Berlin/Heidelberg (2006) 28–39
7. AMIDA: Commercial component definition. Deliverable 7.2, AMIDA Integrated Project IST033812 (Augmented Multi-party Interaction with Distance Access) (November 2007)
8. Moore, D.J.: The IDIAP Smart Meeting Room. Communication 02-07, IDIAP Research Institute (July 2002)
9. Mekhaldi, D., Lalanne, D., Ingold, R.: From searching to browsing through multimodal documents linking. In: ICDAR 2005 (8th International Conference on Document Analysis and Recognition), Seoul (2005) 924–928
10. Popescu-Belis, A., Lalanne, D.: Reference resolution over a restricted domain: References to documents. In: ACL 2004 Workshop on Reference Resolution and its Applications, Barcelona (2004) 71–78
11. Popescu-Belis, A., Baudrion, P., Flynn, M., Wellner, P.: Towards an objective test for meeting browsers: the BET4TQB pilot experiment. In Popescu-Belis, A., Bourlard, H., Renals, S., eds.: Machine Learning for Multimodal Interaction IV. LNCS 4892. Springer-Verlag, Berlin/Heidelberg (2008) 108–119