interspeech 2008 incorporating SST 08

# A Data Format Enabling Interoperation of Speech Recognition, Translation and Information Extraction Engines: The GALE Type System

*John F. Pitrelli, Burn L. Lewis, Edward A. Epstein, Jerome L. Quinn, and Ganesh Ramaswamy*

IBM T.J. Watson Research Center, Yorktown Heights, NY, U.S.A.

{pitrelli,burn,eae,jlquinn,ganeshr}@us.ibm.com

10.21437/Interspeech.2008-459

## Abstract

Live interoperation of several speech- and text-processing engines is key to tasks such as real-time cross-language story segmentation, topic clustering, and captioning of video. One requirement for interoperation is a common data format shared across engines, so that the output of one can be understood as the input of another. The GALE Type System has been created to serve this purpose for interoperating language-identification, speaker-recognition, speech-recognition, named-entity-detection, translation, story-segmentation, topic-clustering, summarization, and headline-generation engines in the context of Unstructured Information Management Architecture. GTS includes types designed to bridge across the domains of these engines, for example, linking the text-only domain of translation to the time-domain types needed for speech processing, and the monolingual domain of information-extraction engines to the cross-language types needed for translation.

**Index Terms**: data typing, interoperation, speaker recognition, language identification, speech recognition, speech-to-text, entity detection, translation, story-boundary detection, summarization, headline generation, UIMA, NLP, STT, MT

## 1. Introduction

Recent advances in human language technologies facilitate the creation of growing aggregates of engines to confront multi-faceted tasks, such as transcribing, translating, and extracting information from video. While errors will compound across such an array of engines, improved accuracy enables consideration of such an approach for tasks which can tolerate some degree of error. Systems such as the GALE[1] Interoperability Demo (IOD) system [1] and IBM's Translingual Automatic Linguistic Extraction System (TALES) interoperate groups of five to 15 engines to perform such tasks. Here, interoperation refers to a single-point invocation of such a group of engines. These engines can include language identification, speaker recognition, speech recognition (speech-to-text or STT), named-entity detection (ED), machine translation (MT), story segmentation, topic clustering of stories, multi-story topic summarization, and generation of headlines from topic summaries and from story translations. Such a combination of engines enables language- and speaker-segmentation of audio, invocation of appropriate speech recognition and translation models to generate transcripts in the original and in the desired language time-aligned to the video, and categorization of the content by entity, by story, or by topic, displayed as headlines which are clickable to expose complete topic summaries and story translations. IOD illustrates an example of many of these capabilities at http://rosetta.watson.ibm.com:8888/iod/ .

One requirement for interoperation is a software framework capable of running such an aggregate of engines despite heterogeneous computing environments including operating system and programming language. Unstructured Information Management Architecture (UIMA) [2] [3] fulfills this role; a description of how IOD uses GTS with UIMA to interoperate remotely-located engines appears elsewhere [1].

Another requirement is a data format shared across engines, so that the output of one can be understood as the input of another. For example, STT's principal output is text, and so makes sense as input to MT, as depicted in Figure 1. However, in practice STT output is formatted as text label for time-spans of audio waveform. In contrast, MT in general knows nothing of audio and time; rather, it processes chunks of text unaware of their origin. Furthermore, the size of "chunk" suitable for processing by STT may not match the amount of content processed at once by MT. Therefore, some intermediate data transformation is necessary to interoperate STT and MT without subjecting either engine to major customization away from its core purpose. Figure 2 depicts an implementation of an STT-MT sequence. STT outputs data types natural to it, here, word (`AudioToken`) and sentence units (`SU`s) defined in terms of audio time spans. Next, a *data reorganization* module creates a *text view* of the data, which is a more appropriate format for text processing such as MT. While the original *audio view* represents the segment of content as an array of waveform samples vs. time, the text view represents it as a text string. UIMA keeps these associated views of the same content in a *Common Analysis Structure (CAS)*, and cross-reference types are used to maintain alignment between the views. Here, data reorganization consists of (1) creating a source-language text view by concatenating the strings from the `AudioTokens`; (2) creating a set of `Sentence` annotations which convert the `SU`s from spans of time to spans on the text view; and (3) creating `AudioXrefs` which explicitly map word spans in the text view back to the `AudioTokens` that generated them so that time alignments can be maintained. These types are intended to be used by any text-procesing engine to follow. To designate chunks to be translated at a time, another data reorganization module, Prepare MT, creates another data type, `Translatables`. Currently, these simply mirror the `Sentences` but could be defined by some other algorithm to segment the text string, for example, into longer units for MT engines which benefit from more context, or into shorter units for specialized translation engines such as for names.

Thus, different data types, such as `SU`s and `Sentences`, are needed to represent conceptually the same information but formatted in ways which are appropriate to the nature of the engines which input or output them. Thus, the interface for *e.g.* text-processing engines is identical whether content originated as text or was generated by STT processing audio. These data types also serve to standardize the formats of data created by multiple engines performing the same function, *e.g.* multiple STT engines, so that they may be used consistently by downstream engines *e.g.* applying system-combination techniques. These data types are part of the GALE Type System (GTS), created for such data regularization for interoperation of the set of engine functions listed above, in the context of UIMA. GTS also includes types designed to bridge across other var-

[1] http://www.darpa.mil/ipto/programs/gale/gale.asp

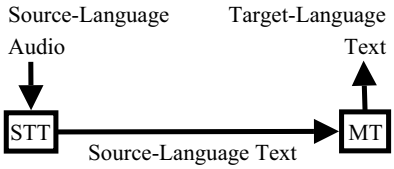


Figure 1: *Conceptual STT-MT sequence.*

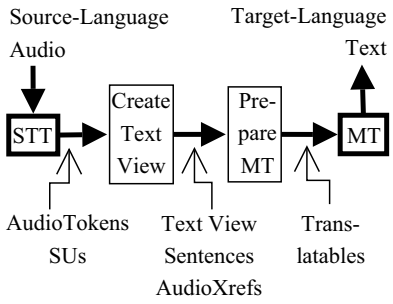


Figure 2: *STT-MT implementation. Bold boxes indicate engines; others are data reorganization modules.*

ied engine domains, such as linking the monolingual domain of information-extraction engines to the cross-language types needed for translation.

To facilitate interoperation, each engine is implemented as a UIMA annotator. Typically such an annotator consists of the core engine adapted to UIMA by means of a thin wrapper. The wrapper converts GTS-formatted input data into the engine's native form, invokes the engine, and then converts the engine's output into GTS form. These conversions are simple because the types are designed to fit the engines' inherent functions. For example, `AudioToken` represents a word emitted by STT, and has a string attribute `spelling` to hold the word, and numerical attributes `begin` and `end` for the begin and end timepoints for the word in the audio signal. A wrapper simply creates `AudioToken` objects from the engine's native word-output format.

It should be noted that GTS must be combined with usage conventions, such as which engines must/may create which types/attributes, in order to constitute a complete data-flow design for an aggregate of engines. Hence the sections which follow describe GTS, while any particular engine combination employing GTS would define such usage conventions.

GTS is defined in a set of UIMA XML descriptors. UIMA annotators access the types as Java classes or as structures in Java, C++, Perl, and Tcl. Thus, class diagrams are convenient for depicting the various sets of types in the following sections.

## 2. Global Types

Some types, such as those carrying metadata pertaining to an entire document, are kept in a global view intended to be accessed by all engines. These type classes, diagrammed in Figure 3, derive from UIMA's base class for CAS data types, `TOP` (not shown), through a subclass `Global`. One is `Parameter`, used to pass parameters to and among engines. A typical instance of `Parameter` is `SessionID`, which history-maintaining engines such as topic clustering use to distinguish different users' data. The main global types are the `DocID`s, which carry metadata about a particular segment of content (a "document"). Documents crawled from the Web carry the original URLs from which they were crawled as well as a local URL

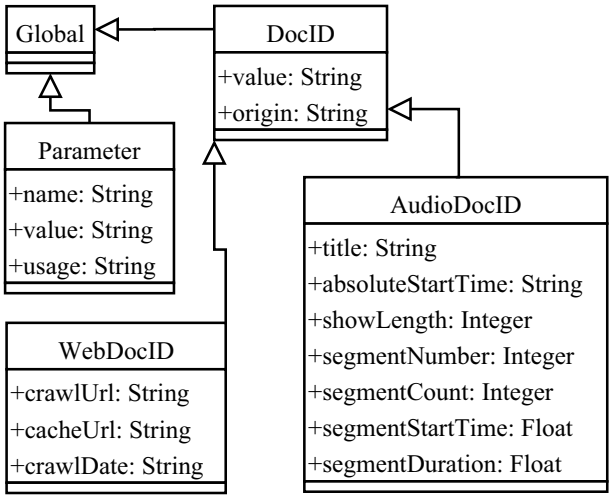


Figure 3: *Selected global types.*

on which they are cached, plus the date they were crawled, in a `WebDocID`. Audio and video content, such as from broadcast news, use `AudioDocID` to keep track of such information as show title, time the show's broadcast began, length of the show, number of segments into which the show has been segmented, which of those segments is represented by the current CAS, start time of this segment relative to the start of the show, and duration of the segment.

## 3. Audio Types

The base GTS type for labeling audio is `AudioSpan`, which enables an engine to specify an event spanning from a begin time to and end time on an audio waveform. An engine can "sign" an object using the `componentId` string attribute. Event type is represented by choice of subtype, as diagrammed in Figure 4. `SU` and `PU` represent sentence- and paragraph-like units. A language- or dialect-identification engine typically creates `AudioLanguageID` objects, labeling the `language` of a time span's speech. This type also provides for indicating a measure of the `confidence` of the labeling. A speaker-recognition engine would typically create `SpeakerID` objects, which provide the attribute `globalSpeakerLabel` for identifying time spans with known speakers. Another attribute, `localSpeakerId`, allows the engine to distinguish spans of time containing speech by the same vs. by differing speakers without requiring previously-known speaker identities. `SpeakerId` also contains a slot for any of a fixed set of `GenderEnum` strings, such as "female", "male" or "unknown".

`AudioToken` is the basic unit of speech-recognition output, usually representing a single word. `AudioFragment` allows for categorization of spans of audio with `AudioTypeEnums` specifying strings such as speech, non-speech, etc., `AudioSubTypeEnums` enabling labels such as "speech+music" and "speech+noise", and `AudioQualityEnums` such as "wideband" and "telephone".

Many types have `componentId` and `confidence` attributes for purposes analogous to those described here; for brevity they will be omitted in the sections below.

## 4. Text-Token Types

Types for labeling spans of text are intended for use by text-processing engines such as entity detection, translation, and story-boundary detection. GTS derives these types from

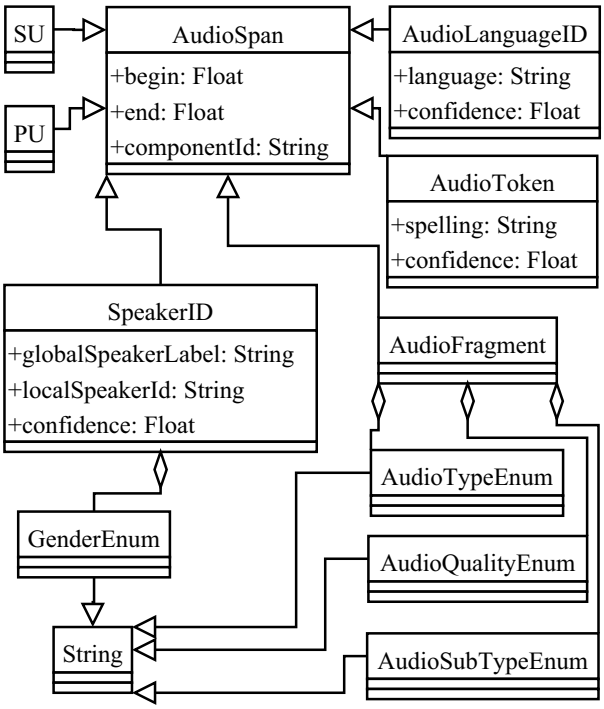


Figure 4: *Selected types for labeling audio.*

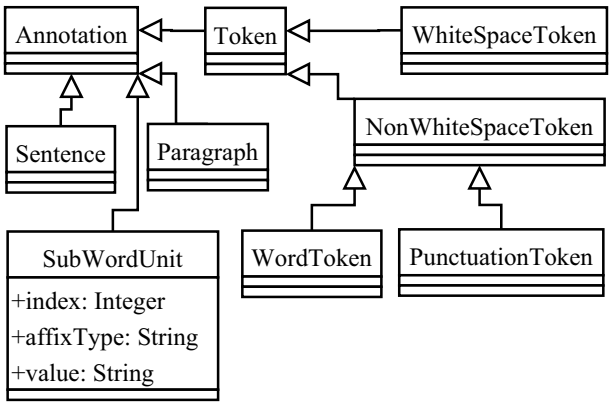


Figure 5: *Types for labeling text tokens.*

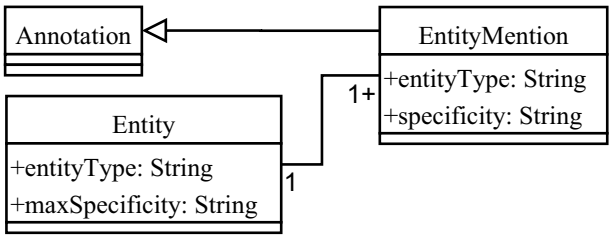


Figure 6: *Selected types for labeling entities and mentions.*

UIMA's `Annotation` type, which provides two attributes, `begin` and `end`, integers which represent indices in the string of a text view. As diagrammed in Figure 5, many GTS subtypes of `Annotation` do not specify additional attributes, as the name of the type serves to label the portion of the view text spanned by `begin` and `end`. `Token`s specify spans of text to be a word (`WordToken`), punctuation (`PunctuationToken`), or characters which may separate these (`WhiteSpaceToken`). `WordToken`s may be further analyzed into `SubWordUnit`s, which may represent a morphemic decomposition by specifying an `index` numbering the morphemes across the word, an `affixType` specifying whether the sub-word is a prefix, stem or suffix, and a string `value` representing the canonical form of the sub-word unit. Finally, GTS provides types for longer units, `Sentence` and `Paragraph`. Like other text-token types, these label a span of characters in a text view, as distinguished from `SU` and `PU` which label a span of time in an audio view.

## 5. Entity Types

GTS provides several pairs of types representing mentions of entities, and grouping of multiple mentions of the same entity, as shown in Figure 6. An `EntityMention` in any language's text view marks a span of text containing one mention of an entity. Attributes include `entityType` to categorize the mentioned entity as, *e.g.*, a person, an organization, a location, etc., and `specificity` to categorize how the entity is specified, such as named, nominal or pronominal. In the case of an entity-detection engine performing within-document co-reference analysis, the `Entity` type ties together `EntityMentions`. For example, "George Bush", "the president", and "he" may be three mentions all referring to the same entity. In an `Entity`, `maxSpecificity` is the most-specific of the `specificity` attributes of the associated `EntityMentions`, while the `entityType` is the most detailed. `Entity` is not an `Annotation` because it is not tied to a single range of text characters or even a single text view; rather, it is generally associated with multiple strings via its associated `EntityMentions`.

Similarly, `EventMention` and `Event` (not shown) have `eventTypes` such as "legal" or "violence", and `RelationMention` and `Relation` represent relations between entities. A `RelationMention` associates two `EntityMentions`, with the `Relation` object logging what type of relation it is, for example, "part of" or "located at".

## 6. Translation Types

Machine translation is represented primarily by two types, `Translatable` for input and `TranslationResult` for output, as depicted in Figure 7. `Translatable` specifies a span of characters on a source-language text view which should be submitted as a chunk to a translation engine. In the case of multiple special-purpose translation engines, *e.g.* a name translator, `preferredComponent` can be used to specify an engine best suited to do the translation. Engines generally produce one `TranslationResult` corresponding to each `Translatable`, containing the same `begin` and `end` values as it represents the translation of the same string. `TranslationResult`'s `value` is the target-language string output by the translation engine. In addition, engines may create `Alignments` representing correspondence between substrings of the source- and target-language text. Each `Alignment` associates a span from `targetBegin` to `targetEnd` in the `value` string of its `TranslationResult` with an array of `Token`s in the source-language view.

## 7. Story-Processing Types

Segmentation of content into meaningful chunks, such as story segmentation of broadcast news, enables a variety of further processing, such as topic clustering, summarization, and headline generation. Figure 8 shows types related to such functionality. `StoryBoundary` marks points in a text view which have been identified by a story-boundary-detection engine as

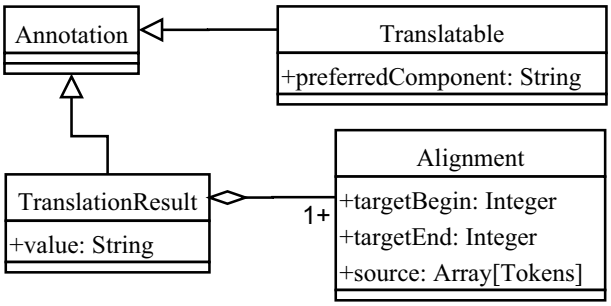


Figure 7: *Types for machine translation.*

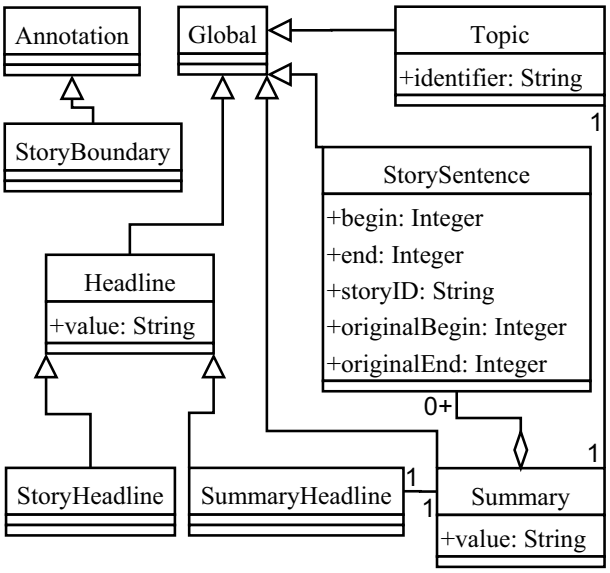


Figure 8: *Types for story-level processing.*

possible story boundaries. Typically `begin` will match `end`, or they will span only white space, as this `Annotation` is intended to separate texts rather than span them. These objects would be used to re-segment content into new CASes spanning one story apiece. Thus the remaining story-processing types are `Global` in that they contain metadata pertaining to an entire CAS. `Topic` enables a topic-clustering engine to give stories in the same cluster the same `identifier`, which a multi-document summarization engine may use to generate a `Summary` of all stories in the cluster. The summary itself is stored as the `Summary`'s `value` attribute, and optionally the summarizer may log the origin of the summary's sentences using `StorySentence` objects. Attributes `begin` and `end` record the position of a sentence in the `Summary`'s `value` string, while `storyID` holds the `value` of the `DocID` of the CAS containing the original sentence, and `originalBegin` and `originalEnd` identify the location in the story text. `Headline` provides a slot `value` for a headline string; currently GTS provides for two distinct types, `StoryHeadline` derived from the current story, and `SummaryHeadline` derived from the current summary.

## 8. View-Cross-Reference Types

As explained above, UIMA provides for views, sparing text-processing engines from needing to be adapted to their inputs' origins as audio processed by speech recognition, and sparing monolingual engines from dealing with multiple languages. The links among these views are represented by the types depicted in Figure 9. `AudioXref` is an `Annotation`; that is, it covers a span of characters in a text view, and its attribute is an `AudioSpan` in an audio view. `AudioXrefs` are typically created by a data reorganization module post-processing STT to create text views before text-processing engines are invoked. Similarly, `TranslationResultXrefs` are created when forming a target-language-text view from MT results; these annotate spans of characters in this view with a link to a `TranslationResult` annotation in the source-language-text view, thus representing alignment between these two views at the level of the `Translatable`. (The `Alignment` objects introduced above enable retrieval from the `TranslationResult` of a finer-grained alignment between the languages' texts.) These links enable *e.g.* target-language captions to be linked through source-language text back to time spans in source-language video. Finally, three cross-reference types like `EntityMentionXref` link target-language mention annotations back to mentions annotated in the source language.

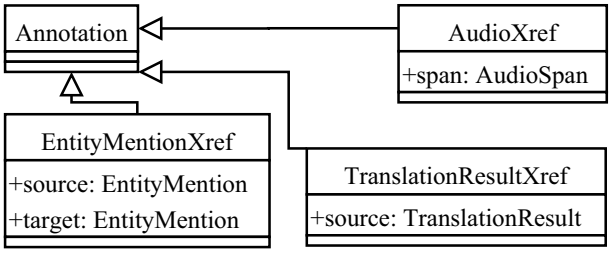


Figure 9: *Selected types for cross-referencing across views.*

## 9. Conclusions

Interoperating large aggregates of natural-language-processing engines requires establishing a shared data format to enable one engine's output to be understood as the next engine's input. Complicating this task is the differences among the domains of the engines processing the data – speech vs. text, translating vs. monolingual. The GALE Type System has enabled interoperation of many types of engines including language identification, speaker recognition, speech recognition, named-entity detection, translation, story segmentation, topic clustering, summarization, and headline generation. We are working to donate GTS into the open-source Apache UIMA project, where we hope the community will enhance it with additional functionality.

## 10. Acknowledgements

The authors thank Sasha Caskey, Martin Franz, Nanda Kambhatla, Daniel Kiecza, Francis Kubala, Scott McCarley, Leiming Qian, Salim Roukos, Todd Ward, and Jian-Ming Xu for their contributions to GTS, building on the data exchange formats used in IBM's TALES system and in the BBN Translingual Audio Monitoring Component. This work was supported in part by DARPA under contract HR0011-06-2-0001.

## 11. References

[1] J. F. Pitrelli, B. L. Lewis, E. A. Epstein, M. Franz, D. Kiecza, J. L. Quinn, G. Ramaswamy, A. Srivastava, and P. Virga, "Aggregating Distributed STT, MT, and Information Extraction Engines: The GALE Interoperability-Demo System", *Proc. Interspeech 2008*.

[2] D. Ferrucci and A. Lally, "UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment", *Nat. Lang. Eng.*, v. 10, no. 3-4, pp. 327–348, 2004.

[3] Apache UIMA: http://incubator.apache.org/uima