The MedKAT Pipeline

Table of Contents

Chapter 1. MedKAT Overview

1.1. MedKAT/P Overview

MedKAT/P (Medical Knowledge Analysis Tool) is a tool tailored to the medical/pathology domain, containing components for extracting cancer-specific characteristics from unstructured text. It is based on Natural Language Processing (NLP) principles, and contains both rule-based and machine-learning based components. MedKAT/P is built on the open source Unstructured Information Management Architecture (UIMA) framework[1] , and consists of a set of modules (annotators), each having a configuration file in XML format. In general terms, annotators mark up an unstructured textual document, inserting "annotations" that can be associated with a particular piece of text or which can be container objects for other annotations. A subsequent annotator can read and process all previously created annotations. The execution sequence, or pipeline, of annotators is also described in a configuration file. Configuration files can be modified with any text (XML) editor, though there is Eclipse[2] -based tooling to ease this task.

Functionally, the MedKAT/P pipeline can be broken into several sets of components:

- *Document ingestion*: annotators that determine document structure and extract implicit meaning from that structure.
- *General natural language processing*: components for tokenization, sentence discovery, part-of-speech tagging, and shallow parsing
- *Concept finding*: components that determine concepts based on specified terminology or patterns and determines negation
- *Relation finding*: components that populate the Cancer Disease Knowledge Model (CDKM, see Section 2.1, "The Cancer Disease Knowledge Model" [15]) and resolves co-references

1.1.1. Document Ingestion

Document ingestion identifies sections, subsections, etc., into annotations and simultaneously adds derived information, such as the number of sections and subsections, header information, and correlations between disjoint pieces of text describing the same tissue specimen. The derived information is based both on textual labels and visual (formatting) cues within the document.

1.1.2. General Natural Language Processing

The pipeline can use any tokenizer that has been written as a UIMA annotator. For optimal performance, all textual resource used within the pipeline, such as terminologies

[1] http://incubator.apache.org/uima/
[2] http://www.eclipse.org/

and ontologies, are expected to be tokenized in the same fashion as the documents that are analyzed.

The determination of sentence boundaries in the provided pipeline is done using OpenNLP, and then additional annotators are executed to adjust previously determined sentence annotations to take into account the structure of medical documents and their implied meaning. Examples of potential issues for a general sentence detector are list and header processing, parenthesis processing and non-standard use of punctuation symbols. Similarly, part-of-speech (POS) tagging is done using the OpenNLP POS tagger, and then a few domain-specific tokens are corrected.

In several of the algorithms the context plays an integral part. Context is defined here as the range of text within a document used to determine the semantic meaning of a word or phrase. To determine context, the pipeline uses the OpenNLP shallow parser.

1.1.3. Concept Finding

Concept finding is one of the most critical components in our system. It maps textual mentions to terminology concepts to create codified information. This task is directed by ConceptMapper[3], which creates candidate matches between concept structures based on a terminology and unstructured text. A complete description of ConceptMapper is provided in the document *The ConceptMapper Annotator*. The mapping of textual mentions to concepts depends on the final intended application, and for our purposes, ConceptMapper has been configured to generate many potential matches, allowing for overlapping and interwoven results, and a subsequent set of rule-based annotators filter out overgenerated matches depending numerous criteria. For example, the snippet “hepatic flexure of the colon” can be codified using ICD-O as “hepatic” (C22.0), “colon” (C18.9) and “hepatic flexure of the colon” (C18.3). Note that the ICD-O entry for C18.3 actually reads as “hepatic flexure of colon” (note the absence of “the” before “colon”), which demonstrates why exact string matching is not sufficient to codify unstructured text. ConceptMapper finds all possible mappings between the terminology and the free text (C22.0, C18.9, C18.3), and the subsequent filters mark the ones to be ignored due to a potential term subsumption (C22.0, C18.9). Of course, while using the longest match heuristic would avoid the need for such filtering in this simplified case, there are more complex examples where that is not sufficient.

The “hepatic flexure of the colon” example described earlier showed the necessity for some filtering. For example, one such filtering rule is the subsumption rule, which specifies whether contained annotations, e.g. “hepatic” should be exposed or not. Other filters mark annotations based on particular values of one of their attributes and another removes duplicate and identical annotations. One filter discovers subsumed generic anatomical sites or histologies and marks them. There is also a filter that handles nominal ellipsis. Consider for example the phrase “Colon, ascending and transverse.” One interpretation is that it refers to two sites (ascending colon and transverse colon),

[3]ConceptMapper is in the process of being released as part of the Apache UIMA Sandbox. The source will be included as part of this MedKAT package until aforementioned release is complete.

another interpretation is that the physician described three sites, colon, ascending colon and transverse colon, as denoted by the use of the comma, and this is the interpretation that is assumed by MedKAT/P.

MedKAT/P also contains regular expression-based annotators which, in conjunction with a terminology, discover textual mentions describing dimensions and sizes, dates, number of excised and positive lymph nodes and stage.

The negation detector is a generalized algorithm as reported in [ChapmanEtAl2001]. Negation trigger words (e.g. "no," "ruled out") are specified in a user modifiable dictionary. The trigger words become the anchors around which negated phrases are discovered within a user specified window. Here we report on results assuming that the window is a sentence. Within the predefined window, the algorithm examines generalized noun phrases starting to the right of the negation keyword. If none is found, then it continues examining phrases to the left. When a generalized noun phrase is found, all semantic entities are marked as negated.

1.1.4. Relation Finding

The next step in the pipeline is to discover the relationship between the appropriate leaf classes (e.g. histology, anatomical sites, size, grade) to populate container classes such as the primary and metastatic tumor classes, the lymph node class and the gross description part class. The relations between the classes are "contains" and "is-part-of." There is a common methodology in filling container classes, coupled with certain class specific rules as outlined in the next paragraphs. We will first outline the common methodology applied to instantiate the primary and metastatic tumor classes, the lymph node class and the gross description part class and then provide more specific details and examples. At this point in the processing, the Concept Finding portion of the pipeline has already identified the leaf classes of anatomical site, histology, grade value, stage and dimension and the container class size.

The first step is to determine which section of a document should be considered for instantiating a container class. For instance, in general gross description part classes are populated only from the gross description section of a pathology report. The tumor and lymph node classes take information from the final diagnosis section. The relation between class and document section is specified in the configuration file associated with the annotator responsible for discovering a particular class. Second, certain classes are categorized according to multiple criteria (e.g., primary vs. metastatic vs. benign, tumor size vs. margin size). Third, we determine which mentions refer to each other (i.e., are co-referring). Fourth, we determine which instances of classes (e.g. histology or anatomical site) should be considered candidates for populating each of the container classes (e.g. primary tumor class or lymph node class). In the final fifth step container classes are merged or split according to class specific rules.

In step two described above, some classes are categorized. One categorization labels classes as positive or negated with respect to a particular class. A class whose negation attribute is set to true by the negation detector is negated with respect to all classes. An example of a negated histology is the phrase "tumor free," where tumor is a histology that our negation algorithm (previously described) marked as such. A class which is negated with respect to a particular class is referred to as excluded with respect to a class. Exclusion states that an instance of a class can be part of only a single container class. For instance, an anatomical site mentioned as part of an invasion class is excluded from consideration for filling a tumor class. Anatomical sites are categorized into originating sites, lymph nodes, invasion sites and other sites. Histology classes are categorized as metastatic and non-metastatic. Size mentions are categorized as tumor sizes and other sizes. Our categorization algorithm is based on a set of trigger phrases (specific to a particular categorization) and the noun phrase hierarchy previously described. For each class instance to be categorized, the algorithm checks whether an appropriate trigger word is co-occurrent with the mention attribute of the class. Co-occurrence is defined based on the noun hierarchy, which means that noun phrases, noun phrase lists and prepositional noun phrases are inspected in turn. In addition, ICD-O codes are used for categorization of histologies and anatomical sites.

Although pronominal anaphora resolution is not required for analysis of pathology reports, co-reference resolution is critical in populating the CDKM. Co-reference is based on codes associated to a concept, such as ICD-O. The methodology for discovering co-referenced generic histology classes is similar to pronoun resolution [KennedyBoguraev1996]. For each histology `H`, examine each generic histology `GH` mentioned after `H` and which is categorized equivalently to `H`. Only generic histologies `GH` occurring between `H` and a subsequent equally categorized histology `H1` are considered. The set of generic histologies `GH` is co-referenced with `H`. The following examples may clarify the algorithm some more. Let `HM1` and `HM2` be two metastatic histologies, `HN1` a non-metastatic histology, and `GHM1`, `GHM2` and `GHN1` be generic metastatic and non-metastatic histologies occurring in the following sequence:

```
HM1 GHM1 GHM2 HN1 GHM3 GHN1 HM2.
```

Example 1.1. Sample dictionary entry

Here `GHM1` `GHM2` and `GHM3` will be co-referenced with `HM1` and `GHN1` with `HN1` respectively.

Besides generic rules for populating class instances, we used class-specific rules as well. For example, the gross description part class may contain one or more anatomical sites and a size. Processing of the document starts with an initial anatomical site within the gross description section and continues with all the other anatomical sites within the same context (i.e. hierarchy of noun phrases) until either a size is found, or the hierarchy is exhausted. At this point, the size expression is parsed to determine whether a single size or a range of sizes was specified. In the latter case, two gross description part classes are instantiated, both having the same anatomical sites but different sizes. This is a class specific implementation of step (5) of the general algorithm.

The primary and metastatic tumor classes are populated simultaneously by a single *TumorModelAnnotator*. The assumption is that tumor classes are populated with information within a user-defined portion of the document – the Tumor Context *TC*. The algorithm iterates through multiple steps. First it identifies all non-negated histologies within *TC* . Second, for all identified histologies, it examines the noun phrase containing the histology for all occurrences of any of these three classes: anatomical sites, grade values and sizes and associates them with the histology. It is noteworthy that each of these classes can be associated with only a single histology and hence, once an association is found, it is removed from further consideration. Third, for histologies missing one or more of these associations, step two is repeated, but for noun phrase lists instead of noun phrases. Fourth, step two is repeated for any histology missing any associations within the context of a sentence. Ultimately, tumor classes which have co-referenced histologies are merged into a single instance. Classes which refer to the same exact anatomical site(s), grade value(s) and sizes and differ only in the histologies are merged as well. An artifact of pathology reports is that anatomical sites are at times implied to be the same as the sites mentioned in the gross description. To account for this, for any non-negated histology that has no anatomical site associations, we extend the context to the gross description part of the document. For the particular pathology reports used for this study, as is the norm, the first sentence of the tumor context TC is considered to be the gross description. The final step is to instantiate the tumor classes based on the categorization of the histology and anatomical sites. It is important to consider all histologies (including benign ones) and all anatomical sites in the process to identify associations correctly, but neither benign histologies, nor histologies with an associated anatomical site that is a lymph node, are considered for primary or metastatic tumor classes. The category of the remaining histologies (metastatic or non-metastatic) determines which type of tumor class is instantiated.

Lymph nodes classes have attributes that are anatomical sites, histologies, and lymph node expressions (*LNE*). In particular, only anatomical sites which have been categorized as lymph nodes (*AS-L*) are considered. *LNE* 's describe either the general state (positive/ negative) of the lymph nodes or provide more detail in terms of number of positive lymph nodes and excised nodes (from which the state is deduced). For each *AS-L* , the algorithm for instantiating lymph node classes determines the histologies and *LNE* 's co-occuring with the anatomical site (*AS-L*) in the same sentence. If they are not found, the context is expanded to sentences within the same section. A set of rule-based filters are applied to derive the correct associations, taking the categorization of histologies and anatomical sites into positive and negative classes into account.

To populate the gross description part class, we introduced two new syntactic structures – the ParenthesisSeparatedNoun-phrase (*PSN*) and the ParenthesisPhrase (*PPH*). A sequence of a noun phrase, a parenthesis, a noun phrase followed by another parenthesis is called a *PSN* . Any expression enclosed with matching parenthesis is a *PPH* . We define a hierarchy of syntactic constructs consisting of the following levels: noun phrase, *PSN* , generalized noun phrase and *PPH* . The algorithm for populating a gross description part examines the syntactic hierarchy in order, with noun phrases being at level 0 and *PPH* being at level 4. If anatomical site(s) and size expression(s) co-occur in the same syntactic structure, one or more gross description part classes are instantiated. The number of instantiated classes depends on the type and number of size expressions found. If an anatomical site AS occurs without a size expression within a syntactic structure, a set of rules determines whether the AS should be associated with an already existing gross description part class or a new class be instantiated. The rules depend on the lexical ordering of the anatomical site and size mentions.

Chapter 2. The Cancer Disease Knowledge Model

2.1. The Cancer Disease Knowledge Model

In this section, we describe our extensible knowledge model for storing cancer characteristics and their relations, including temporal information and inference (see Figure 2.1, "Cancer Disease Model" [15]). We refer to this model as the Cancer Disease Knowledge Model (CDKM). Each node in the model is referred to as a class. Each class can have multiple attributes which can be filled with individual values of a given type, e.g. strings, integers, or other classes. Subsequent figures describe some of the classes in more detail. We propose to use the CDKM as the formalism to record a patient's disease state, track disease progression and draw inferences on outcome in conjunction with available structured information.

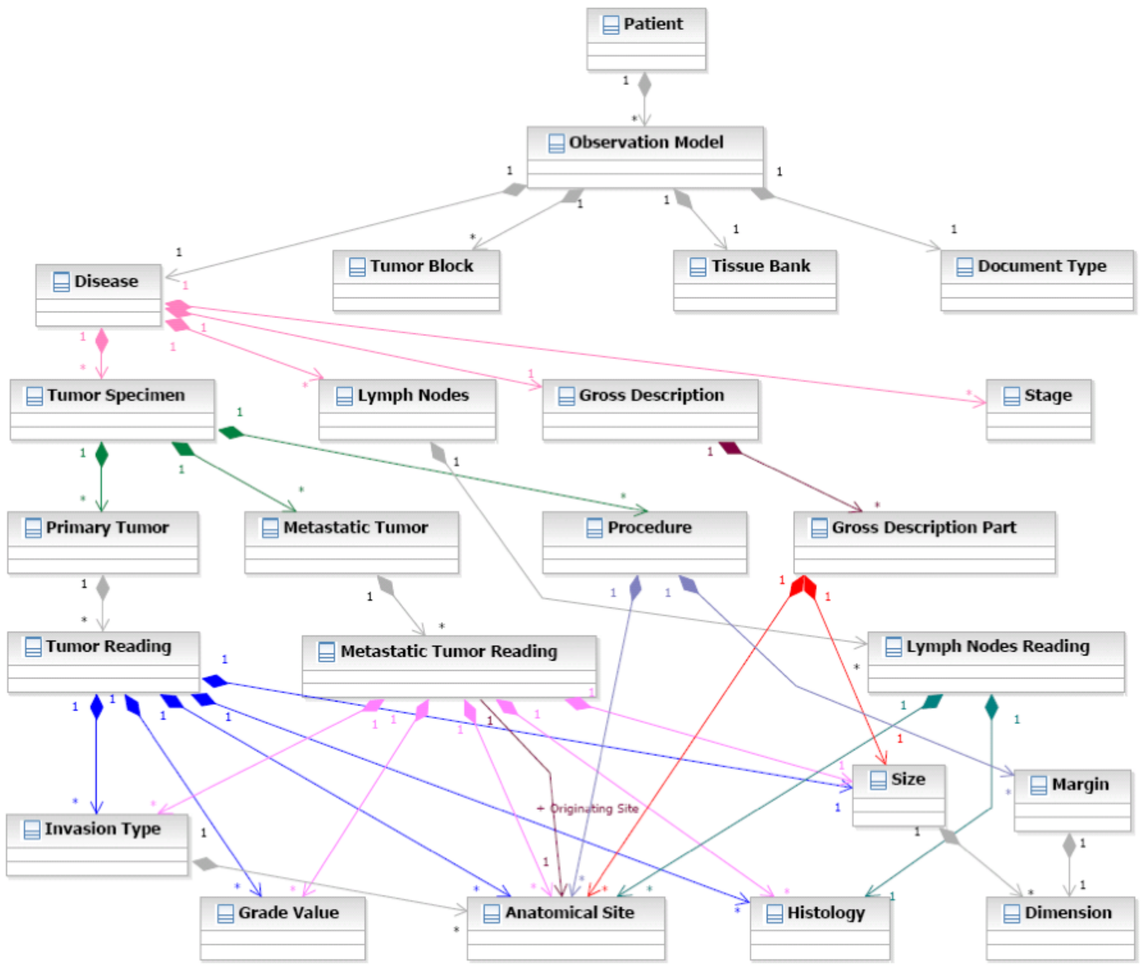

Figure 2.1. Cancer Disease Model

Classes whose attributes are only values are referred to as leaf classes. Our model has five leaf classes which describe cancer characteristics: anatomical site, histology, grade value, dimension and stage and three other leaf classes: document type and tumor block and tissue bank. Classes whose attributes are either values or other classes are referred to as container classes. Each leaf class can be thought of as a named entity with associated specific attributes. Figure 2.2, "Anatomical Site" [16] shows details for the Anatomical Site class.

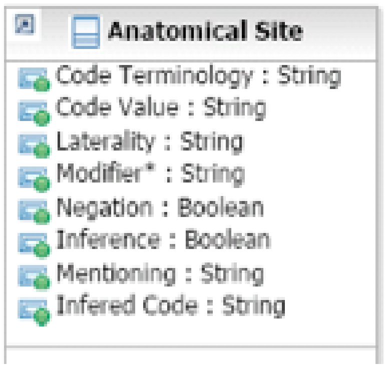

Figure 2.2. Anatomical Site

Here, anatomical site attributes specify the code terminology and code value associated with the attribute mentioning whose value is extracted from the text. Other attributes are laterality, negation and modifiers. An asterisk next to an attribute label indicates that multiple instances of an attribute can be specified. In addition, the anatomical site leaf class—like many other classes in the model—has attributes to specify whether a particular instance of a class contains inferred values. For instance, the text may refer to lymph nodes—code value LN—but from the context one could infer that mesenteric lymph nodes—code value MLN—were described. In this case, an instance of the anatomical site class would have the string LN in the Code Value attribute, the string MLN in the Inferred Code attribute and the Inference attribute set to true.

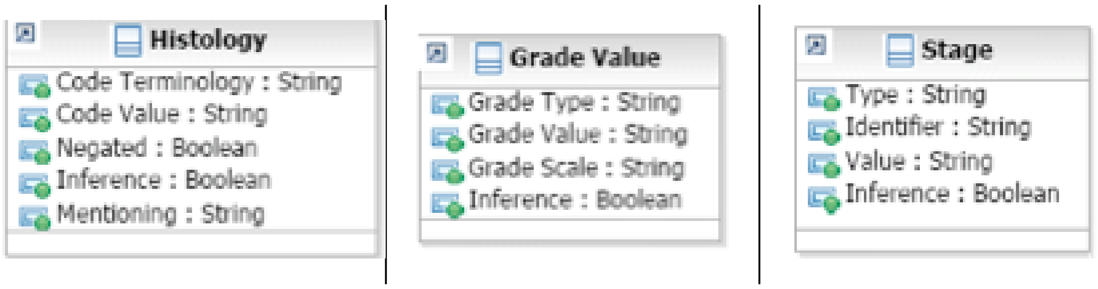

Figure 2.3. Histology, Grade Value and Stage Classes

Figure 2.3, "Histology, Grade Value and Stage Classes" [16] shows three other leaf classes capturing cancer disease characteristics. Histology and Grade Value are attributes of several container classes, such as primary and metastatic tumor. The stage of a cancer is either mentioned explicitly within a pathology reports or can be derived from other information from the primary tumor, the lymph node status, and the occurrence or absence of metastases.

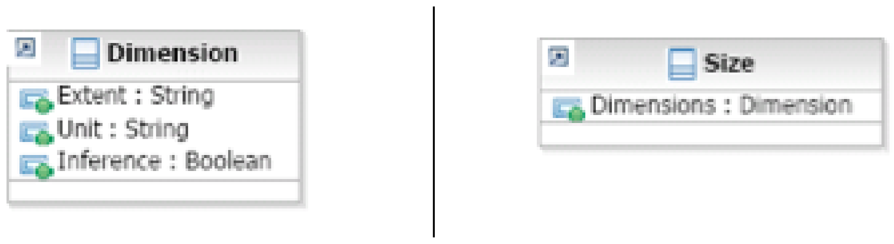

Figure 2.4. Dimension and Size Classes

Instances of the Dimension class can describe a measurement in a single dimension, such as linear extent or a weight (Figure 2.4, "Dimension and Size Classes" [17]). The container class Size has multiple attributes, each of which can be filled by a Dimension class.

There are additional leaf classes, as can be found in detail in [CodenEtal2009]

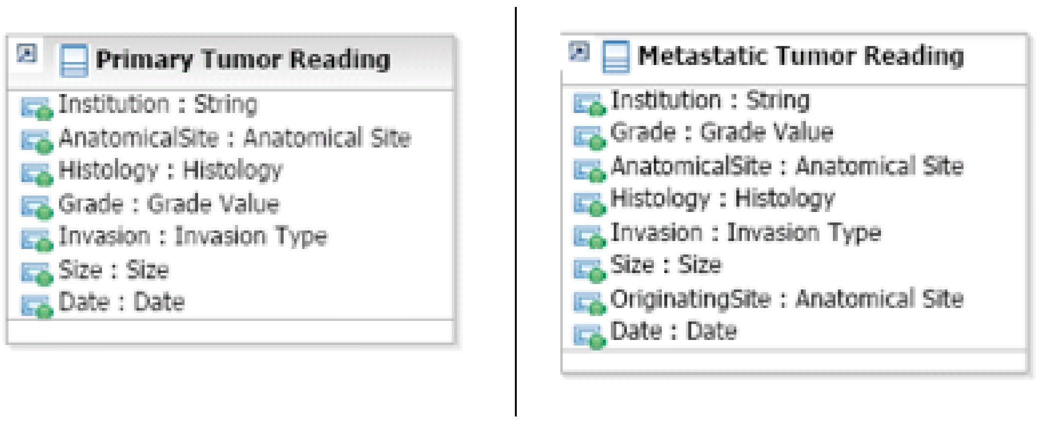

Figure 2.5. Primary and Metastatic Tumor Reading Classes

The primary and metastatic tumor reading classes depicted in Figure 2.5, "Primary and Metastatic Tumor Reading Classes" [17] are examples of container classes in the model. A tumor reading class contains the following attributes: histology, anatomical site, size, date and invasion type Invasion Type Class is not currently filled by MedKAT. In addition, the institution where the analysis on the tissue sample was performed and its date are attributes of a tumor reading class. The metastatic tumor reading class specifies two anatomical sites: originating and metastatic.

Figure 2.1, "Cancer Disease Model" [15] shows that a tumor class (primary or metastatic) can contain multiple instances of tumor reading classes, capturing the notion of multiple interpretations of the same tissue sample. For instance, two doctors in the same or different institutions can reach different conclusions about the type and severity (e.g., histology, grade) of the disease based on the same tissue sample. Different interpretations are not that common in pathology reports, and based on some preliminary observations seem to be rather common in clinical notes.

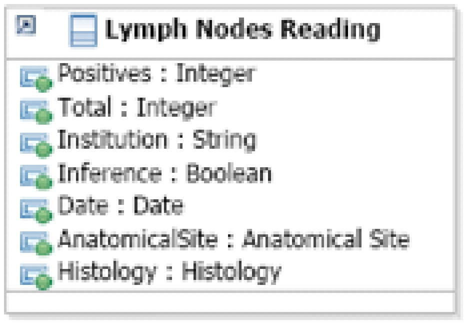

Figure 2.6. Lymph Node Reading Class

Figure 2.6, "Lymph Node Reading Class" [18] describes the Lymph Node Reading class. Noteworthy attributes are the number of Positive nodes and Total number of lymph nodes excised. Similarly to the tumor classes, a Lymph Nodes class can contain multiple lymph nodes reading classes.

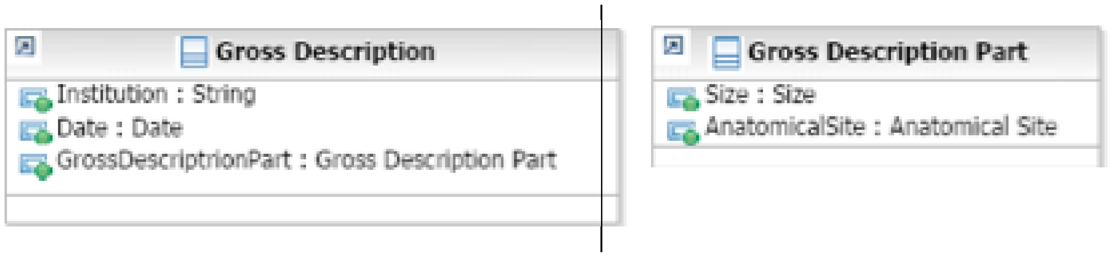

Figure 2.7. Gross Description Classes

The Gross Description classes are shown in Figure 2.7, "Gross Description Classes" [18]. The Gross Description Part classes describe each excised tissue sample, whereas the institution where the procedure was performed and the date are associated with the Gross Description class.

Over the course of time, unfortunately, a patient can have multiple disease episodes; each episode is captured in an observation model, which can have time stamps or sequence numbers associated with it. In general, a single pathology report does not reflect multiple episodes; however a single clinical note often describes the patient's disease progression.

There is more to the CDKM, as described in [CodenEtal2009]. The CDKM is easily extended by adding additional concepts and relations. Such models, instantiated from textual sources have multiple use cases: examples include identification of cohorts of patients who have similar disease progression or summarization of disease progression of a single patient from multiple reports.

Chapter 3. The ConceptMapper Annotator

3.1. Introduction

ConceptMapper is a highly configurable, high performance dictionary lookup tool, implemented as a UIMA (Unstructured Information Management Architecture) component. Using one of several matching algorithms, it maps entries in a dictionary onto input documents, producing UIMA annotations.

3.2. Using ConceptMapper

ConceptMapper was designed to provide highly accurate mappings of text into controlled vocabularies, specified as dictionaries, including the association of any necessary properties from the controlled vocabulary as part of that mapping. Individual dictionary entries could contain multiple terms (tokens), and ConceptMapper can be configured to allow multi-term entries to be matched against non-contiguous text. It was also designed to perform fast, and has been easily able to provide real-time results even with multi-million entry dictionaries.

Lookups are token-based, and are limited to applying within a specific context, usually a sentence, though this is configurable (e.g., a noun phrase, a paragraph or other NLP-based concept).

3.3. Functionality

There are many parameters to configure all aspects of ConceptMapper's functionality, in terms of:

- processing the dictionary
- the way input documents are processed
- the availability of multiple lookup strategies
- its various output options

3.3.1. Dictionaries

The requirements on the design of the ConceptMapper dictionary were that it be easily extensible and that arbitrary properties could be associated with individual entries. Additionally, the set of properties could not be fixed, but rather customizable for any particular application.

The structure of a ConceptMapper dictionary is quite flexible and is expressed using XML (see Example 3.1, “Sample dictionary entry” [20]). Specifically, it consists of a set of entries, specified by the <token> XML tag, each containing one or more variants (synonyms), the text of which is specified using by the "base" feature of the <variant> XML tag. Entries can have any number of associated properties, as needed. Individual variants (synonyms) inherit features from their parent token (i.e., the canonical form), but can override any or all of them, or even add additional features.

In the following sample dictionary entry, there are 6 variants, and according to the rules described earlier, each inherits the all attributes from the canonical form (canonical, CodeType, CodeValue, and SemClass), though the variants "colonic" and "colic" override the value of the POS (part of speech) attribute:

```
<token canonical="colon, nos"
       CodeType="ICDO" CodeValue="C18.9"
       SemClass="Site" POS="NN">
 <variant base="colon, nos"/>
 <variant base="colon"/>
 <variant base="colonic" POS="JJ" />
 <variant base="colic" POS="JJ" />
 <variant base="large intestine" />
 <variant base="large bowel" />
</token>
```

Example 3.1. Sample dictionary entry

The result of running ConceptMapper are UIMA annotations, and there are two configuration parameters that are used to map the attributes from the dictionary (see AttributeList [ ]) to features of UIMA annotations (see FeatureList [ ]).

The entire dictionary is loaded into memory, which, in conjunction with an efficient data structure, provides very fast lookups. As stated earlier, dictionaries with millions of entries have been used without any performance issues. The obvious drawback to storing the dictionary in memory is that large dictionaries require large amounts of memory; this is partially mitigated by the fact that the dictionary is implemented as a UIMA shared resource (see DictionaryFile [ ]). This means that multiple annotators, such as multiple instances of ConceptMapper that are set up using different parameters, can all access it without having to load it more than once. The dictionary loader is specified in the external resource section of the descriptor, and is expected to implement the interface `org.apache.uima.conceptMapper.support.dictionaryResource.DictionaryResource`. Two implementations are included in the distribution, `org.apache.uima.conceptMapper.support.dictionaryResource.DictionaryResource_impl`, the standard implementation, which loads an XML version of a dictionary, and `org.apache.uima.conceptMapper.support.dictionaryResource.CompiledDictionaryResource_impl` which loads a pre-compiled version, for faster loading. The compiler is supplied as `org.apache.uima.conceptMapper.dictionaryCompiler.CompileDictionary`, which takes two arguments, a ConceptMapper analysis engine descriptor that loads the dictionary using the standard dictionary loader, and the name of the output file into which to write the compiled dictionary.

3.3.2. Dictionary Entry Tokenization

Input documents are processed on a token-by-token basis, so it is important that the dictionary entries are tokenized in the same way as the input documents. To accomplish this, ConceptMapper allows any UIMA analysis engine to be specified as the tokenizer for the dictionary entries. See parameter TokenizerDescriptorPath [ ] for details.

3.3.3. Input Document Processing

As stated earlier, input documents are processed on a token-by-token basis. Tokens are processed one span (e.g., a sentence or a noun phrase) at a time. Token annotations are specified by the parameter TokenAnnotation [ ], while span annotations are specified by the parameter SpanFeatureStructure [ ]. By default, all tokens within a span are considered, and it is the text associated with each token that is used for lookups. ConceptMapper can also be configured to consider tokens differently:

- Case sensitive or insensitive matching. See the parameter caseMatch [ ]
- Stop words: ignore token during lookup if it appears in given stop word list. See the parameter StopWords [ ]
- Stemming: a stemmer can be specified to be applied to the text of the token. In practice, the stemmer could be a standard stemmer providing the root form of the token, or it could perform other functions, such as abbreviation expansion or spelling variant replacement. See the parameter Stemmer [ ]
- Use a token feature instead of the token's text. This is useful for cases where, for example, spelling or case correction results need to be applied instead of the token's original text. See the parameter TokenTextFeatureName [ ]
- skip tokens during lookups based on particular feature values, as described below

The ability to skip tokens during lookups based on particular feature values makes it easy to skip, for example, all tokens with particular part of speech tags, or with some previously computed semantic class. For example, given the text below in Example 3.2, "Sample Input Text" [21]:

```
Infiltrating mammary carcinoma
```

Example 3.2. Sample Input Text

Assume each word is a token that has a feature SemanticClass, and that feature for the token "mammary" contains the value "AnatomicalSite", while the tokens "Infiltrating" and "carcinoma" do not. It is then possible to configure ConceptMapper to indicate that tokens that have a particular feature, in this case SemanticClass, equal to one of a set of values, in this case "AnatomicalSite", should be excluded when performing dictionary lookups (see parameters ExcludedTokenClasses [ ] and

ExcludedTokenTypes [ ]). By doing this, for the purposes of dictionary lookup, the example text would effectively appear to be:

```
Infiltrating carcinoma
```

Example 3.3. Result of Token Skipping

In addition to the set of feature values that indicate their associated token are to be excluded during lookup, there are also configuration parameters that can be used to specify a set of feature values for inclusion (see parameters IncludedTokenClasses [ ] and IncludedTokenTypes [ ]). The algorithm for selecting annotations to include during lookup is as follows:

```
if there is an includeList but no excludeList
include annotation if feature value in includeList

else if there is an excludeList
exclude annotation if feature value in excludeList

else
include annotation
```

Example 3.4. Token Selection Algorithm

This provides a simple way to restrict the selection of pre-classified tokens, whether that pre-classification is done via previous instances of ConceptMapper or some altogether different annotator. See TokenTextFeatureName [ ]

3.3.4. Lookup Strategies

The actual dictionary lookup algorithm is controlled by three parameters. One specifies token-order independent lookup (OrderIndependentLookup [ ]). For example, a dictionary entry that contained the variant:

```
<variant base='carcinoma, infiltrating'/>
```

would also match against any permutation of its tokens. In this case, assuming that punctuation was ignored, it would match against both “infiltrating carcinoma” and “carcinoma, infiltrating”. Clearly, this particular setting must be used with care to prevent incorrect matches from being found, but for some domains it enables the use of a more compact dictionary, as all permutations of a particular entry do not need to be enumerated.

Another parameter that controls the dictionary lookup algorithm toggles between finding only the longest match vs. finding all possible matches (FindAllMatches [ ]). For the text:

```
... carcinoma, infiltrating ...
```

If there was a dictionary entry for “carcinoma” as well as the entry for “carcinoma, infiltrating”, this parameter would control whether only the latter was annotated as a result or both would be annotated. Using the setting that indicates all possible matches should be found is useful when subsequent disambiguation is required.

The final parameter that controls the dictionary lookup algorithm specifies the search strategy (SearchStrategy [ ]), of which there are three. The default search strategy only considers contiguous tokens (not including tokens from the stop word list or otherwise skipped tokens, as described above), and then begins the subsequent search after the longest match. The second strategy allows for ignoring non-matching tokens, allowing for disjoint matches, so that a dictionary entry of

```
    A C
```

would match against the text

```
    A B C
```

This can be used as alternative method for finding “infiltrating carcinoma” over the text “infiltrating mammary carcinoma”, as opposed to the method described above, wherein the token “mammary” had to have been have been somehow pre-marked with a feature and that feature listed as indicating the token should be skipped. On the other hand, this approach is less precise, potentially finding completely disjoint and unrelated tokens as a dictionary match. As with the default search strategy, the subsequent search begins after the longest match.

The final search strategy is identical to the previous, except that subsequent searches begin one token ahead, instead of after the previous match. This enables overlapped matching. As with the setting that finds all matches instead of the longest match, using this setting is useful when subsequent disambiguation is required.

3.3.5. Output Options

Output is in the form of new UIMA annotations. As previously discussed, the mapping from dictionary entry attributes to the result annotation features can also be specified. Given the fact that dictionary entries can have multiple variants, and that matches could contain non-contiguous sets of tokens, it can be useful to be able to be able to know exactly what was matched. There are two parameters that can be used to provide this information. One allows the specification of a feature in the output annotation that will be set to the string containing the matched text. The other can be used to indicate a feature to be filled with the list of tokens that were matched. Going back to the example in figure 2, where the token “mammary” was skipped, the matched string would be set to "Infiltrating carcinoma" and the matched tokens would be set to the list of tokens "Infiltrating" and "carcinoma".

Another output control AE descriptor parameter can be used to specify a feature of the resultant annotation to be set to contain the span annotation enclosing the matched token. Assuming, for example, that the spans being processed are sentences, this provides a convenient way to link the resultant annotation back to its enclosing sentence.

It is also possible to indicate dictionary attributes to store back into each of the matched tokens. This provides the ability for tokens to be marked with information regarding what it was matched against. Going back to the example in figure 2, one way that the SemanticClass feature of the token "mammary" could have been labeled with the value "AnatomicalSite" was using this technique: a previous invocation of ConceptMapper had "mammary" as a dictionary entry, that entry had the SemanticClass feature with the value "AnatomicalSite", and SemanticClass was listed as an attribute to write back as a token feature. If, instead of "mammary" the match was against a multi-token entry, then each of the multiple tokens would have that feature set.

3.4. Configuration Parameters

Detailed description of all configuration parameters:

- `TokenizerDescriptorPath`: *[Required]*String

  Path to tokenizer Analysis Engine descriptor, which is used to tokenize dictionary entries.

- `TokenAnnotation`: *[Required]*String

  Type of feature structure representing tokens in the input CAS.

- `SpanFeatureStructure`: *[Required]*String

  Type of feature structure that corresponds to spans of data for processing (e.g. a sentence) in the input CAS.

- `AttributeList`: *[Required]*Array of Strings

  List of attribute names for XML dictionary entry record. Must correspond to parallel list FeatureList [ ].

- `FeatureList`: *[Required]*Array of Strings

  List of feature names for ResultingAnnotationName [ ]. Must correspond to parallel list AttributeList [ ].

- `caseMatch`: *[Required]*String

  Specifies the case folding mode. The following are the allowable values:

    - `ignoreall` - fold everything to lowercase for matching
    - `insensitive` - fold only tokens with initial caps to lowercase
    - `digitfold` - fold all (and only) tokens with a digit
    - `sensitive` - perform no case folding

- `StopWords`: *[Optional]* Array of Strings

  A list of words that are always to be ignored in dictionary lookups.

- `Stemmer`: *[Optional]* String

  Name of stemmer class to use before matching. *Must* implement the `org.apache.uima.conceptMapper.support.stemmer` interface and have a zero-parameter constructor. If not specified, no stemming will be performed.

- `TokenTextFeatureName`: *[Optional]* String

  Name of feature of token annotation that contains the token's text. If not specified, the token's covered text will be used.

- `TokenClassFeatureName`: *[Optional]* String

  Name of feature used when doing lookups against IncludedTokenClasses [ ] and ExcludedTokenClasses [ ]. Values contained in this feature are of type String. This parameter is mandatory if either IncludedTokenClasses [ ] or ExcludedTokenClasses [ ] are specified. See Example 3.4, "Token Selection Algorithm" [22] for a description of how these are used during lookup.

- `TokenTypeFeatureName`: *[Optional]* String

  Name of feature used when doing lookups against IncludedTokenTypes [ ] and ExcludedTokenTypes [ ]. Values contained in this feature are of type Integer. This parameter is mandatory if either IncludedTokenTypes [ ] or ExcludedTokenTypes [ ] are specified See Example 3.4, "Token Selection Algorithm" [22] for a description of how these are used during lookup.

- `IncludedTokenTypes`: *[Optional]* Array of Integers

  Type of tokens to include in lookups (if not supplied, then all types are included except those specifically mentioned in ExcludedTokenTypes [ ])

- `ExcludedTokenTypes`: *[Optional]* Array of Integers

  Type of tokens to exclude from lookups (if not supplied, then all types are excluded except those specifically mentioned in IncludedTokenTypes [ ], unless IncludedTokenTypes [ ] is not supplied, in which case none are excluded)

- `IncludedTokenClasses`: *[Optional]* Array of Strings

  Class of tokens to include in lookups (if not supplied, then all classes are included except those specifically mentioned in ExcludedTokenClasses [ ])

- `ExcludedTokenClasses`: *[Optional]* Array of Strings

  Class of tokens to exclude from lookups (if not supplied, then all classes are excluded except those specifically mentioned in IncludedTokenClasses [ ], unless IncludedTokenClasses [ ] is not supplied, in which case none are excluded).

- `OrderIndependentLookup`: *[Optional]* Boolean

  If "True", token (as specified by TokenAnnotation [ ]) ordering within span (as specified by SpanFeatureStructure [ ]) is ignored during lookup (i.e., "top box" would equal "box top"). Default is False.

- `SearchStrategy`: *[Optional]* String

  Specifies the dictionary lookup strategy. The following are the allowable values:

  - `ContiguousMatch` - longest match of contiguous tokens (as specified by TokenAnnotation [ ]) within enclosing span (as specified by SpanFeatureStructure [ ]), taking into account included/excluded items (see IncludedTokenTypes [ ], ExcludedTokenTypes [ ], IncludedTokenClasses [ ] and ExcludedTokenClasses [ ]). DEFAULT strategy

  - `SkipAnyMatch` - longest match of not-necessarily contiguous tokens (as specified by TokenAnnotation [ ]) within enclosing span (as specified by SpanFeatureStructure [ ]), taking into account included/excluded items (see IncludedTokenTypes [ ], ExcludedTokenTypes [ ], IncludedTokenClasses [ ] and ExcludedTokenClasses [ ]). Subsequent lookups begin in span after complete match. *Implies* order-independent lookup (see OrderIndependentLookup [ ]).

  - `SkipAnyMatchAllowOverlap` - longest match of not-necessarily contiguous tokens (as specified by TokenAnnotation [ ]) within enclosing span, (as specified by SpanFeatureStructure [ ]) taking into account included/excluded items (see IncludedTokenTypes [ ], ExcludedTokenTypes [ ], IncludedTokenClasses [ ] and ExcludedTokenClasses [ ]). Subsequent lookups begin in span after next token. *Implies* order-independent lookup (see OrderIndependentLookup [ ]).

- `FindAllMatches`: *[Optional]* Boolean

If True, all dictionary matches are found within the span specified by SpanFeatureStructure [ ], otherwise only the longest matches are found.

- `ResultingAnnotationName`: *[Optional]*String

  Name of the annotation type created by this TAE.

- `ResultingEnclosingSpanName`: *[Optional]*String

  Name of the feature in the ResultingAnnotationName [ ] that will be set to point to the span annotation that encloses it (i.e. its sentence)

- `ResultingAnnotationMatchedTextFeature`: *[Optional]*String

  Name of the feature in the ResultingAnnotationName [ ] that will be set to the string that was matched in the dictionary. This could be different that the annotation's covered text if there were any skipped tokens in the match.

- `MatchedTokensFeatureName`: *[Optional]*String

  Name of the FSArray feature in the ResultingAnnotationName [ ] that will set to the set of tokens matched.

- `TokenClassWriteBackFeatureNames`: *[Optional]*Array of Strings

  Names of features in the ResultingAnnotationName [ ] that should be written back to a token from the matching dictionary entry, such as a POS tag.

- `PrintDictionary`: *[Optional]*Boolean

  If True, print dictionary after loading. Default is False.

- `DictionaryFile`: *[Dictionary Resource]*Boolean

  Dictionary file resource specification. Specify class name for dictionary loader, then bind to name of file containing dictionary contents to be loaded.

Chapter 4. Pipeline Contents

4.1. MedKATSentenceDetector

4.1.1. Description

Detect sentence boundaries and create sentence annotations that span these boundaries. Uses the OpenNLP MaxEnt Sentence Detector. Create annotation of type `uima.tt.SentenceAnnotation` on sentence boundaries.

4.1.2. Location

MedKAT_NLP/descriptors/analysis_engine/primitive/MedKATSentenceDetector.xml

4.1.3. Parameters

Table 4.1. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
Modelfile	String	Yes	No	Filename of the model file.	../OpenNLP/models/english/sentdetect/EnglishSD.bin.gz
SentenceType	Sring	Yes	No	Type of annotations that are to be created at the sentence boundaries	uima.tt.SentenceAnnotat

4.1.4. Capabilities

Table 4.2. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	No	Yes	uima.tt

4.2. MedKATTokenizer

4.2.1. Description

Tokenize the text and create token annotations that span the tokens. The tokenization is performed using the OpenNLP MaxEnt tokenizer, which tokenizes according to the Penn Tree Bank tokenization standard. In general, tokens are separated by white space, but punctuation marks (e.g., ".", ",", "!", "?", etc.) and apostrophe'd endings (e.g., "'s", "'nt", etc.) are separate tokens. Create annotation of type `uima.tt.TokenAnnotation` on token boundaries .

4.2.2. Location

MedKAT_NLP/descriptors/analysis_engine/primitive/MedKATTokenizer.xml

4.2.3. Parameters

Table 4.3. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset value
Modelfile	String	Yes	No	Filename of the model file.	../OpenNLP/models/english/tokenize/EnglishTok.bin.gz
SentenceType	Sring	Yes	No	Type of annotation that specifies sentences	uima.tt.SentenceAnnotation
TokenType	Sring	Yes	No	Type of annotations that are to be created at the token boundaries	uima.tt.TokenAnnotation

4.2.4. Capabilities

Table 4.4. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt

Name	Input	Output	Namespace
TokenAnnotation	No	Yes	uima.tt

4.3. SentenceShortener

4.3.1. Description

Reset sentence boundaries to exclude trailing spaces.

4.3.2. Location

MedKATp/descriptors/analysis_engine/primitive/SentenceShortener.xml

4.3.3. Parameters

Table 4.5. Parameters

Name	Type	Mandatory	Multi-valued	Description	
SentenceType	String	Yes	No	Type name of annotations that specify a sentence	uima.tt.SentenceAnnotati

4.3.4. Capabilities

Table 4.6. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	Yes	uima.tt

4.4. SectionFinder

4.4.1. Description

Divide document into sections based on parameter settings.

4.4.2. Location

MedKATp/descriptors/analysis_engine/primitive/SectionFinder.xml

4.4.3. Parameters

Table 4.7. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
sectionHeadingStrings	String	Yes	Yes	section heading strings which should be found	"FINAL DIAGNOSIS", "GROSS DESCRIPTION"
sectionAnnotations	String	Yes	Yes	name of annotations to be inserted	"org.ohnlp.medkat.taes.sectionFin "org.ohnlp.medkat.taes.sectionFin

4.4.4. Capabilities

Table 4.8. Capabilities

Name	Input	Output	Namespace
SectionAnnotation	No	Yes	org.ohnlp.medkat.taes.sectionFinder

4.5. SubsectionDetector

4.5.1. Description

Divide document sections into sub-sections based on parameter settings.

4.5.2. Location

MedKATp/descriptors/analysis_engine/primitive/SubsectionDetector.xml

4.5.3. Parameters

Table 4.9. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
Patterns	String	Yes	Yes	Regex patterns which indicate	"^\s*(\d{1,2}(?:,\s*\d{1,2})*)\)", "^\s*(\d{1,2}

Name	Type	Mandatory	Multi-valued	Description	Preset values
				subsection. MUST have subsection number as first captured group in pattern, with possible subsequent subsection number as second captured group	(?:,\s*\d{1,2})*)\.\s+", "^\s*(\d{1,2}(?:,\s*\d{1,2})*):\sSP\:\s+", "^\s*SP\:\s+", "^\s*Part\s(\d{1,2}(?:,\s*\d{1,2})*):\s+", "^\s*\#(\d{1,2}(?:,\s*\d{1,2})*)\.\s+", "^\s*(\d{1,2}(?:,\s*\d{1,2})*):\s+", "^\s*BI\:\s+", "^\s*(\d{1,2})(-)(\d{1,2})\)", "^\s*(\d{1,2})(-)(\d{1,2})\.\s+", "^\s*(\d{1,2})(-)(\d{1,2}):\sSP\:\s+", "^\s*Part\s(\d{1,2})(-)(\d{1,2}):\s+", "^\s*\#(\d{1,2})(-)(\d{1,2})\.\s+", "^\s*(\d{1,2})(-)(\d{1,2}):\s+"

4.5.4. Capabilities

Table 4.10. Capabilities

Name	Input	Output	Namespace
SectionAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFinder
SubHeading	No	Yes	org.ohnlp.medkat.taes.subsectionDetector
MaxSubsectionIndicator	No	Yes	org.ohnlp.medkat.taes.subsectionDetector

4.6. SubSubsectionDetector

4.6.1. Description

Divide document sub-sections into sub-sub-sections based on parameter settings.

4.6.2. Location

MedKATp/descriptors/analysis_engine/primitive/SubSubsectionDetector.xml

4.6.3. Parameters

Table 4.11. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
subSubSectionAnnotations	String	Yes	Yes	Array of SubSubsection annotation types that are to be created when parallel array item from parameter subSubSectionAnnotationLabels is found in a document.	
subSubSectionAnnotationLabels	String	Yes	Yes	Array of SubSubsection labels that specify the	

Name	Type	Mandatory	Multi-valued	Description	Preset values
				beginning of a new sub-subsection. Parallel array with parameter subSubSectionAnnotations.	
PrimarySectionAnnotations	String	Yes	Yes	Array of annotation type names specifying primary sections.	"org.ohnlp.medkat.taes.s "org.ohnlp.medkat.taes.s
SecondarySectionAnnotations	String	No	Yes	Array of annotation type names specifying secondary sections.	
subSubSectionConcepts	String	No	Yes	Array of concept names, such as from the that may be used to assign semantics to a subsubsection (e.g., "HistologicGrade"). This is used only when there may be some need to relate a subsubsection to a particular dictionary concept	

Name	Type	Mandatory	Multi-valued	Description	Preset values
				for further processing. This parameter contains a when parallel array item with parameter subsubsectiondetector.param.subSubSectionAnno	

4.6.4. Capabilities

Table 4.12. Capabilities

Name	Input	Output	Namespace
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectionDetector
SubSubsection	No	Yes	org.ohnlp.medkat.taes.subSubsectionDete

4.7. SyntacticUnitFinder

4.7.1. Description

Create annotations that enclose matching parentheses, as "(" and ")" or "[" and "]" or "{" and "}".

4.7.2. Location

MedTKATp/descriptors/analysis_engine/primitive/SyntacticUnitFinder.xml

4.7.3. Parameters

Table 4.13. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
TokenAnnotati	String	Yes	No	Type name of a token annotation.	uima.tt.TokenAnnotation

Name	Type	Mandatory	Multi-valued	Description	Preset values
DataBlockFeatureStructure	String	Yes	No	Feature structure used to delimit blocks of data (e.g. a sentence).	uima.tt.SentenceAnnotati

4.7.4. Capabilities

Table 4.14. Capabilities

Name	Input	Output	Namespace
ParagraphAnnotation	Yes	No	uima.tt
SentenceAnnotation	Yes	No	uima.tt
TokenAnnotation	Yes	No	uima.tt
SyntacticUnit	No	Yes	org.ohnlp.medkat.taes.syntacticU

4.8. NewLineSentenceAnnotator

4.8.1. Description

Creates annotations at location of the "\n" character.

4.8.2. Location

MedKATp/descriptors/analysis_engine/primitive/NewLineSentenceAnnotator.xml

4.8.3. Parameters

None.

4.8.4. Capabilities

Table 4.15. Capabilities

Name	Input	Output	Namespace
NewlineSentenceAnnotation	No	Yes	org.ohnlp.medkat.taes.textdocpa

4.9. TextDocParser

4.9.1. Description

Finds the Diagnosis and Gross Description section annotations and splits it into subsections and possibly a bullet list entries.

4.9.2. Location

`MedKATp/descriptors/analysis_engine/primitive/TextDocParser.xml`

4.9.3. Parameters

Table 4.16. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
subheadLeaders	String	No	No	Tags that indicate subsections in Gross Description.	"A.", "B.", "C.", "D.", "E.", "F.", "G."

4.9.4. Capabilities

Table 4.17. Capabilities

Name	Input	Output	Namespace
SubHeading	Yes	Yes	org.ohnlp.medkat.taes.subsectionDetector
SectionAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFinder
NewlineSentenceAnnotation	Yes	No	org.ohnlp.medkat.taes.textDocParser.suba
SyntacticUnit	No	Yes	org.ohnlp.medkat.taes.syntacticUnitFinde
DocumentAnnotation	No	Yes	uima.tcas
DiagnosisAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFinder
BulletListAnnotation	No	Yes	org.ohnlp.medkat.taes.bulletList

4.10. SentenceBreakDetector2

4.10.1. Description

Finds sentences that overlap sections and breaks them into 2 smaller sentence annotations. This annotator detects sentences that cross section boundaries, and breaks those sentences into smaller ones within each section. It deletes the original annotation and leaves only the new ones. Uses only annotations specified in the descriptor file.

4.10.2. Location

MedKATp/descriptors/analysis_engine/primitive/SentenceBreakDetector.xml

4.10.3. Parameters

Table 4.18. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset Values
SentenceClass	String	Yes	No	Major class that is to be broken up if it crosses boundaries of other subsections.	uima.tt.SentenceAnnotati
SectionClasses	String	Yes	Yes	List of section types that are to be checked for sentence crossings.	"org.ohnlp.medkat.taes.s "org.ohnlp.medkat.taes.se "org.ohnlp.medkat.taes.s "org.ohnlp.medkat.taes.b
SubsectionClass	String	Yes	No	Base class for subsections to be broken into sentences by line break. Used in synoptic reports.	org.ohnlp.medkat.taes.su

4.10.4. Capabilities

Defined by parameters.

4.11. ParenSentenceCombiner

4.11.1. Description

Combines any sentence annotations within parentheses into a single sentence. This is designed for use with Mayo pathology documents where short phrases, which may include periods, are parenthesized. It combines them. It will not do what you want if there are two actual sentences within the parentheses.

4.11.2. Location

MedKATp/descriptors/analysis_engine/primitive/ParenSentenceCombiner.xml

4.11.3. Parameters

Table 4.19. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
SentenceType	String	No	No	Type for sentences to be combined. If not specified, default is `uima.tt.SentenceAnnotation`	uima.tt.SentenceAnnotation

4.11.4. Capabilities

Table 4.20. Capabilities

Name	Input	Output	Namespace
SyntacticUnit	Yes	No	org.ohnlp.medkat.taes.syntacticUnitFinde

4.12. MedKATNLP_RunSP

4.12.1. Description

An aggregate to parse the document and create phrasal and clausal annotations over the text. Uses the OpenNLP MaxEnt parser. Assigns POS tags to tokens as part of the processing.

4.12.2. Location

MedKAT_NLP/descriptors/analysis_engine/aggregate/MedKATNLP_RunSP.xml

4.12.3. Included Descriptors

Table 4.21. Included Descriptors

Location	Name
MedKAT_NLP/descriptors/analysis_engine/primitive/MedKATPOSTagger.xml	MedKATPOSTagger
MedKAT_NLP/descriptors/analysis_engine/primitive/MedKATParser	MedKATParser
MedKAT_NLPBase/descriptors/POSAdapterAnnotator.xml	POSAdapterAnnotator

4.12.4. Parameters

Table 4.22. Parameters

Name	Type	Mandatory	Multi-valued	Description	Overrides	Preset Value
POSAdapterClassNames	String	No	Yes	Class name for POS adapters. The class is instantiated to perform necessary	POSAdapterAnnotator/ POSAdapterClassNames	org.ohnlp.medkat.oper org.ohnlp.medkat.oper

Name	Type	Mandatory	Multi-valued	Description	Overrides	Preset Value
				modification to previously generated annotations		

4.12.5. Capabilities

Table 4.23. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt
TokenAnnotation	Yes	No	uima.tt
TokenAnnotation:pennTag	No	Yes	uima.tt
AdjPAnnotation	No	Yes	uima.tt
ClauseAnnotation	No	Yes	uima.tt
NPAnnotation	No	Yes	uima.tt
NPListAnnotation	No	Yes	uima.tt
PhraseAnnotation	No	Yes	uima.tt
PPAnnotation	No	Yes	uima.tt
TCAnnotation	No	Yes	uima.tt
VGAnnotation	No	Yes	uima.tt

4.13. MedKATPOSTagger

4.13.1. Description

Assigns part of speech tags to tokens using the OpenNLP MaxEnt part of speech tagger. Requires that sentence and token annotations have been created in the CAS. Updates the POS field of each token annotation with the part of speech tag.

4.13.2. Location

MedKAT_NLP/descriptors/analysis_engine/primitive/MedKATPOSTagger.xml

4.13.3. Parameters

Table 4.24. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
Modelfile	String	Yes	No	Filename of the model file.	../OpenNLP/models/english/parser/tag.bin.gz
TokenType	Sring	Yes	No	Type of annotations that are to be created at the token boundaries	uima.tt.TokenAnnotation
SentenceType	Sring	Yes	No	Type of annotation that specifies sentences	uima.tt.SentenceAnnotati
POSFeature	Sring	Yes	No	A name of a feature in annotation representing tokens that stores POS information	pennTag

4.13.4. Capabilities

Table 4.25. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt
TokenAnnotation	Yes	No	uima.tt
TokenAnnotation:pennTag	No	Yes	uima.tt

4.14. MedKATParser

4.14.1. Description

Parse the document and create phrasal and clausal annotations over the text.
Uses the OpenNLP MaxEnt parser. This analysis engine takes a parameter called "ParseTagMapping" which maps each parse tag to a syntax annotation type. The parse tags come from the standard Penn Tree Bank phrase and clause tags (produced by the OpenNLP parser), and each syntax annotation type must be defined in the type system and have a corresponding JCas Java class.

4.14.2. Location

MedKAT_NLP/descriptors/analysis_engine/primitive/MedKATParser.xml

4.14.3. Parameters

Table 4.26. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
ModelDirectory	String	Yes	No	Directory that contains model files.	../OpenNLP/models/english/parser
UseTagDictionary	Boolean	No	No	Indicator if tag dictionary to be used.	false
CaseSensitiveTagDictionary	Boolean	No	No	Indicator if used tag dictionary is case sensitive.	false
BeamSize	Integer	No	No		NONE
AdvancePercentage	Float	No	No		NONE
ParseTagMappings	String	Yes	Yes	Map between tags and annotation type to be created	"S,uima.tt.ClauseAnnotation", "SBAR,uima.tt.TCAnnotation", "SBARQ,uima.tt.ClauseAnnotatio "SINV,uima.tt.ClauseAnnotation", "SQ,uima.tt.ClauseAnnotation", "ADJP,uima.tt.AdjPAnnotation",

Name	Type	Mandatory	Multi-valued	Description	Preset values
				for the tagged text fragments	"ADVP,uima.tt.PhraseAn "CONJP,uima.tt.PhraseAn "FRAG,uima.tt.PhraseAn "INTJ,uima.tt.PhraseAnn "LST,uima.tt.NPListAnno "NAC,uima.tt.PhraseAnn "NP,uima.tt.NPAnnotatio "NX,uima.tt.PhraseAnno "PP,uima.tt.PPAnnotation "PRN,org.ohnlp.medkat.t "PRT,uima.tt.PhraseAnno "QP,uima.tt.PhraseAnnot "RRC,uima.tt.ClauseAnn "UCP,uima.tt.PhraseAnno "VP,uima.tt.VGAnnotatio "WHADJP,uima.tt.AdjPA "WHAVP,uima.tt.PhraseA "WHNP,uima.tt.NPAnno "WHPP,uima.tt.PPAnnota "X,uima.tt.PhraseAnnota
POSTagReplacements	String	No	Yes	Replacements for POS tags generated by external taggers	"CS,before,CC", "CS,if,IN", "CS,when,WRB", "CS,whether,WRB", "CS,,RB", "NP,,NNP", "NPS,,NNPS", "PP,,PRP", "PP$,,PRP$", "AUX,,VB", "AUXD,,VBD", "AUXG,,VBG", "AUXN,,VBN", "AUXP,,VBP", "AUXZ,,VBZ"
TokenType	String	Yes	No	Type of annotations that are to be created at the token boundaries	uima.tt.TokenAnnotation

Name	Type	Mandatory	Multi-valued	Description	Preset values
SentenceType	String	Yes	No	Type of annotation that specifies sentences	uima.tt.SentenceAnnotation
POSFeature	String	Yes	No	A name of a feature in annotation representing tokens that stores POS information	pennTag

4.14.4. Capabilities

Table 4.27. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt
TokenAnnotation	Yes	No	uima.tt
AdjPAnnotation	No	Yes	uima.tt
ClauseAnnotation	No	Yes	uima.tt
NPAnnotation	No	Yes	uima.tt
NPListAnnotation	No	Yes	uima.tt
PhraseAnnotation	No	Yes	uima.tt
PPAnnotation	No	Yes	uima.tt
TCAnnotation	No	Yes	uima.tt
VGAnnotation	No	Yes	uima.tt

4.15. ConceptMapperInitalPass

4.15.1. Description

Map dictionary entries onto input documents using ConceptMapper. See Chapter 3, *The ConceptMapper Annotator* [19] for detailed documentation of ConceptMapper.

4.15.2. Location

MedTKATp/descriptors/analysis_engine/primitive/ConceptMapperInitalPass.xml

4.15.3. Parameters

Table 4.28. Parameters

Name	Preset values
TokenizerDescriptorPath	../OpenNLP_Pipeline/descriptors/analysis_engine/aggregate/OpenNLPSentenceDetectorAndTokenizer.xml
TokenAnnotation	uima.tt.TokenAnnotation
SpanFeatureStructure	uima.tt.SentenceAnnotation
AttributeList	canonical AttributeType AttributeValue SemClass POS key parent
FeatureList	DictCanon AttributeType AttributeValue SemClass POS key parent
caseMatch	ignoreall
StopWords	of in

Name	Preset values
	with and
Stemmer	
TokenTextFeatureName	
TokenClassFeatureName	SemClass
TokenTypeFeatureName	
IncludedTokenTypes	
ExcludedTokenTypes	
IncludedTokenClasses	Grade HistologicGrade NuclearGrade
ExcludedTokenClasses	
OrderIndependentLookup	true
SearchStrategy	SkipAnyMatchAllowOverlap
FindAllMatches	true
ResultingAnnotationName	org.ohnlp.medkat.taes.conceptMapper.DictTerm
ResultingEnclosingSpanName	enclosingSpan
ResultingAnnotationMatchedTextFeature	matchedText
MatchedTokensFeatureName	matchedTokens
TokenClassWriteBackFeatureNames	POS SemClass
PrintDictionary	false
DictionaryFile	file:dict/initialDict_base.xml

4.15.4. Capabilities

Table 4.29. Capabilities

Name	Input	Output	Namespace
TokenAnnotation	Yes	Yes	uima.tt
SentenceAnnotation	Yes	No	uima.tt
ParagraphAnnotation	Yes	No	uima.tt
DictTerm	No	Yes	org.ohnlp.medkat.taes.conceptM
DocumentAnnotation	No	Yes	uima.tcas

4.16. ConceptMapperSecondPass

4.16.1. Description

Map dictionary entries onto input documents using ConceptMapper. See Chapter 3, *The ConceptMapper Annotator* [19] for detailed documentation of ConceptMapper.

4.16.2. Location

MedTKATp/descriptors/analysis_engine/primitive/ConceptMapperSecondPass.xml

4.16.3. Parameters

Table 4.30. Parameters

Name	Preset values
TokenizerDescriptorPath	../OpenNLP_Pipeline/descriptors/analysis_engine/aggregate/OpenNLPSentenceDetectorAndTokenizer.xml
TokenAnnotation	uima.tt.TokenAnnotation
SpanFeatureStructure	uima.tt.SentenceAnnotation
AttributeList	canonical AttributeType AttributeValue SemClass POS

Name	Preset values
	key parent
FeatureList	DictCanon AttributeType AttributeValue SemClass POS key parent
caseMatch	ignoreall
StopWords	of in with and
Stemmer	dict/medTermStems.txt
TokenTextFeatureName	
TokenClassFeatureName	SemClass
TokenTypeFeatureName	
IncludedTokenTypes	
ExcludedTokenTypes	
IncludedTokenClasses	
ExcludedTokenClasses	Grade Site Metastatic Invasive
OrderIndependentLookup	true

Name	Preset values
SearchStrategy	SkipAnyMatchAllowOverlap
FindAllMatches	true
ResultingAnnotationName	org.ohnlp.medkat.taes.conceptMapper.DictTerm
ResultingEnclosingSpanName	enclosingSpan
ResultingAnnotationMatchedTextFeature	matchedText
MatchedTokensFeatureName	matchedTokens
TokenClassWriteBackFeatureNames	POS
	SemClass
PrintDictionary	false
DictionaryFile	file:dict/mainDict_augmented.xml

4.16.4. Capabilities

Table 4.31. Capabilities

Name	Input	Output	Namespace
TokenAnnotation	Yes	Yes	uima.tt
SentenceAnnotation	Yes	No	uima.tt
ParagraphAnnotation	Yes	No	uima.tt
DictTerm	No	Yes	org.ohnlp.medkat.taes.conceptM
DocumentAnnotation	No	Yes	uima.tcas

4.17. ConceptMapperLymph

4.17.1. Description

Map dictionary entries onto input documents using ConceptMapper. See Chapter 3, *The ConceptMapper Annotator* [19] for detailed documentation of ConceptMapper.

4.17.2. Location

MedTKATp/descriptors/analysis_engine/primitive/ConceptMapperLymph.xml

4.17.3. Parameters

Table 4.32. Parameters

Name	Preset values
TokenizerDescriptorPath	../OpenNLP_Pipeline/descriptors/analysis_engine/aggregate/OpenNLPSentenceDetectorAndTokenizer.xml
TokenAnnotation	uima.tt.TokenAnnotation
SpanFeatureStructure	uima.tt.SentenceAnnotation
AttributeList	canonical AttributeType AttributeValue SemClass POS key parent
FeatureList	DictCanon AttributeType AttributeValue SemClass POS key parent
caseMatch	ignoreall
StopWords	
Stemmer	
TokenTextFeatureName	
TokenClassFeatureName	SemClass
TokenTypeFeatureName	

Name	Preset values
IncludedTokenTypes	
ExcludedTokenTypes	
IncludedTokenClasses	
ExcludedTokenClasses	
OrderIndependentLookup	true
SearchStrategy	ContiguousMatch
FindAllMatches	false
ResultingAnnotationName	org.ohnlp.medkat.taes.conceptMapper.DictTerm
ResultingEnclosingSpanName	enclosingSpan
ResultingAnnotationMatchedTextFeature	matchedText
MatchedTokensFeatureName	matchedTokens
TokenClassWriteBackFeatureNames	POS SemClass
PrintDictionary	false
DictionaryFile	file:dict/lymph.xml

4.17.4. Capabilities

Table 4.33. Capabilities

Name	Input	Output	Namespace
TokenAnnotation	Yes	Yes	uima.tt
SentenceAnnotation	Yes	No	uima.tt
ParagraphAnnotation	Yes	No	uima.tt
DictTerm	No	Yes	org.ohnlp.medkat.taes.conceptM
DocumentAnnotation	No	Yes	uima.tcas

4.18. ConceptMapperNegator

4.18.1. Description

Map dictionary entries onto input documents using ConceptMapper. See Chapter 3, *The Concept Mapper Annotator* [19] for detailed documentation of ConceptMapper.

4.18.2. Location

MedTKATp/descriptors/analysis_engine/primitive/ConceptMapperNegator.xml

4.18.3. Parameters

Table 4.34. Parameters

Name	Preset values
TokenizerDescriptorPath	../OpenNLP_Pipeline/descriptors/analysis_engine/aggregate/OpenNLPSentenceDetectorAndTokenizer.xml
TokenAnnotation	uima.tt.TokenAnnotation
SpanFeatureStructure	uima.tt.SentenceAnnotation
AttributeList	canonical SemClass POS
FeatureList	DictCanon SemClass POS
caseMatch	ignoreall
StopWords	
Stemmer	
TokenTextFeatureName	
TokenClassFeatureName	SemClass
TokenTypeFeatureName	
IncludedTokenTypes	

Name	Preset values
ExcludedTokenTypes	
IncludedTokenClasses	
ExcludedTokenClasses	
OrderIndependentLookup	true
SearchStrategy	ContiguousMatch
FindAllMatches	false
ResultingAnnotationName	org.ohnlp.medkat.taes.conceptMapper.NoTerm
ResultingEnclosingSpanName	enclosingSpan
ResultingAnnotationMatchedTextFeature	
MatchedTokensFeatureName	
TokenClassWriteBackFeatureNames	POS
	SemClass
PrintDictionary	false
DictionaryFile	file:dict/neg.xml

4.18.4. Capabilities

Table 4.35. Capabilities

Name	Input	Output	Namespace
TokenAnnotation	Yes	Yes	uima.tt
SentenceAnnotation	Yes	No	uima.tt
ParagraphAnnotation	Yes	No	uima.tt
NoTerm	No	Yes	org.ohnlp.medkat.taes.conceptM
DocumentAnnotation	No	Yes	uima.tcas

4.19. ConceptMapperTumorOrigin

4.19.1. Description

Map dictionary entries onto input documents using ConceptMapper. See Chapter 3, *The ConceptMapper Annotator* [19] for detailed documentation of ConceptMapper.

4.19.2. Location

MedTKATp/descriptors/analysis_engine/primitive/ConceptMapperTumorOrigin.xml

4.19.3. Parameters

Table 4.36. Parameters

Name	Preset values
TokenizerDescriptorPath	../OpenNLP_Pipeline/descriptors/analysis_engine/aggregate/OpenNLPSentenceDetectorAndTokenizer.xml
TokenAnnotation	uima.tt.TokenAnnotation
SpanFeatureStructure	uima.tt.SentenceAnnotation
AttributeList	canonical SemClass POS
FeatureList	DictCanon SemClass POS
caseMatch	ignoreall
StopWords	
Stemmer	
TokenTextFeatureName	
TokenClassFeatureName	SemClass
TokenTypeFeatureName	
IncludedTokenTypes	
ExcludedTokenTypes	
IncludedTokenClasses	
ExcludedTokenClasses	
OrderIndependentLookup	true
SearchStrategy	ContiguousMatch

Name	Preset values
FindAllMatches	false
ResultingAnnotationName	org.ohnlp.medkat.taes.conceptMapper.OriginTerm
ResultingEnclosingSpanName	enclosingSpan
ResultingAnnotationMatchedTextFeature	matchedText
MatchedTokensFeatureName	
TokenClassWriteBackFeatureNames	POS SemClass
PrintDictionary	false
DictionaryFile	file:dict/origin.xml

4.19.4. Capabilities

Table 4.37. Capabilities

Name	Input	Output	Namespace
TokenAnnotation	Yes	Yes	uima.tt
SentenceAnnotation	Yes	No	uima.tt
ParagraphAnnotation	Yes	No	uima.tt
OriginTerm	No	Yes	org.ohnlp.medkat.taes.conceptM
DocumentAnnotation	No	Yes	uima.tcas

4.20. DictTermFilters

4.20.1. Description

Set of filters to mark named entities variously, to prevent further processing of them. Markers are defined as an integer mask, with the following values:

Type	Mask Value	Description
Negated	1	Term has been negated. See DrNo [78]
Ignored	2	Term should be ignored. See IgnoredTermFilter [62].

Type	Mask Value	Description
Duplicate	4	Term is duplicate. See DuplicateTermFilter [75]
Subsumed	8	Term has been negated subsumed within another. See SimpleSubsumptionFilter [59] and SubsumptionFilter [76]
Superfluous	16	Term is superfluous. Currently unused.
Modifier	32	Term is a modifier of another term. See ModifierTermFilter [60]
Contains Disallowed	64	Term contains some other disallowed term. See CommaAndDisallowedFilter [61]"
Metstatic	256	Term has been marked as metastatic.

4.20.2. Location

MedTKATp/descriptors/analysis_engine/aggregate/DictTermFilters.xml

4.20.3. Parameters

None.

4.20.4. Capabilities

None.

4.20.5. Delegates

- SimpleSubsumptionFilter [59]
- ModifierTermFilter [60]
- CommaAndDisallowedFilter [61]
- IgnoredTermFilter [62]
- DuplicateTermFilter [75]

- SubsumptionFilter [76]

4.21. SimpleSubsumptionFilter

4.21.1. Description

Mark entities as "Subsumed", if necessary (see DictTermFilters for description of markers).

Entities containing commas are not processed. Otherwise, entities are marked as Subsumed if they have fewer tokens than another term and begin and end indices either match or fall between another entity's begin and end indices.

4.21.2. Location

MedTKATp/descriptors/analysis_engine/primitive/SimpleSubsumptionFilter.xml

4.21.3. Parameters

Table 4.38. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
EnclosingSpan	String	Yes	No	Class of span to process (e.g., sentence, noun phrase, etc.).	uima.tt.NPAnnotation
AllowedMarkersMask	Integer	No	No	Markers allowed to be set in entities to be processed. Bit mask, as defined in DictTermFilters	257
SemanticClasses	String	Yes	Yes	Semantic classes of terms (assumes DictTerm annotations) upon which to operate.	Site Diagnosis

4.21.4. Capabilities

Table 4.39. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt

4.22. ModifierTermFilter

4.22.1. Description

Mark entities as "Modifier", if necessary (see DictTermFilters for description of markers).

If more than one term in span, and the span has no commas, mark all but last one as a modifier.

4.22.2. Location

MedTKATp/descriptors/analysis_engine/primitive/ModifierTermFilter.xml

4.22.3. Parameters

Table 4.40. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
EnclosingSpan	String	Yes	No	Class of span to process (e.g., sentence, noun phrase, etc.).	uima.tt.NPAnnotation
AllowedMarkersMask	Integer	No	No	Markers allowed to be set in entities to be processed. Bit mask, as defined in DictTermFilters	257
SemanticClasses	String	Yes	Yes	Semantic classes of terms	Site

Name	Type	Mandatory	Multi-valued	Description	Preset values
				(assumes DictTerm annotations) upon which to operate.	

4.22.4. Capabilities

Table 4.41. Capabilities

Name	Input	Output	Namespace
DictTerm	Yes	No	org.ohnlp.medkat.taes.conceptM

4.23. CommaAndDisallowedFilter

4.23.1. Description

Mark entities as "Subsumed", if necessary (see DictTermFilters for description of markers).

Marks terms as "Disallowed" if they contain a term that is:

- Marked as a "Modifier" (unless the term is a complete term unto itself, e.g., "sigmoid colon")
- One from a set of [Ignored, Duplicate, Modifier, ContainsDisallowedTerm] and also contains a comma

4.23.2. Location

MedTKATp/descriptors/analysis_engine/primitive/CommaAndDisallowedFilter.xml

4.23.3. Parameters

Table 4.42. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
EnclosingSpan	String	Yes	No	Class of span to process (e.g., sentence, noun phrase, etc.).	uima.tt.NPAnnotation

Name	Type	Mandatory	Multi-valued	Description	Preset values
AllowedMarkersMask	Integer	No	No	Markers allowed to be set in entities to be processed. Bit mask, as defined in DictTermFilters	257
SemanticClasses	String	Yes	Yes	Semantic classes of terms (assumes DictTerm annotations) upon which to operate.	Site Diagnosis
TokenClass	String	Yes	No	Class name of token annotations.	uima.tt.TokenAnnotation

4.23.4. Capabilities

Table 4.43. Capabilities

Name	Input	Output	Namespace
DictTerm	Yes	No	org.ohnlp.medkat.taes.conceptMapper

4.24. IgnoredTermFilter

4.24.1. Description

Mark entities as "Ignored", if necessary (see DictTermFilters for description of markers).

Entities are marked as Ignored if they have multiple tokens or if their code value matches an entry in the IgnoredTermCodes parameter.

4.24.2. Location

MedTKATp/descriptors/analysis_engine/primitive/IgnoredTermFilter.xml

4.24.3. Parameters

Table 4.44. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
EnclosingSpan	String	Yes	No	Class of span to process (e.g., sentence, noun phrase, etc.).	uima.tt.SentenceAnnotati
AllowedMarkersMask	Integer	No	No	Markers allowed to be set in entities to be processed. Bit mask, as defined in DictTermFilters	257
SemanticClasses	String	Yes	Yes	Semantic classes of terms (assumes DictTerm annotations) upon which to operate.	Site
IgnoredTermCodes	String	Yes	Yes	Codes of DictTerm's AttributeValue features to marked as Ignored.	m8000/0 m8001/0 m8005/0 m8010/0 m8011/0 m8040/0 m8050/0 m8051/0 m8052/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8053/0
					m8060/0
					m8096/0
					m8100/0
					m8101/0
					m8102/0
					m8103/0
					m8110/0
					m8120/0
					m8121/0
					m8130/0
					m8140/0
					m8146/0
					m8147/0
					m8149/0
					m8150/0
					m8151/0
					m8160/0
					m8161/0
					m8170/0
					m8190/0
					m8191/0
					m8200/0
					m8202/0
					m8204/0
					m8210/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8212/0
					m8220/0
					m8221/0
					m8230/0
					m8240/0
					m8250/0
					m8251/0
					m8260/0
					m8261/0
					m8264/0
					m8270/0
					m8271/0
					m8272/0
					m8280/0
					m8281/0
					m8290/0
					m8300/0
					m8310/0
					m8313/0
					m8321/0
					m8322/0
					m8323/0
					m8324/0
					m8325/0
					m8330/0
					m8333/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8334/0
					m8336/0
					m8361/0
					m8370/0
					m8371/0
					m8372/0
					m8373/0
					m8374/0
					m8375/0
					m8380/0
					m8381/0
					m8390/0
					m8391/0
					m8392/0
					m8400/0
					m8401/0
					m8402/0
					m8403/0
					m8404/0
					m8405/0
					m8406/0
					m8407/0
					m8408/0
					m8409/0
					m8410/0
					m8420/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8440/0
					m8441/0
					m8443/0
					m8450/0
					m8453/0
					m8454/0
					m8460/0
					m8461/0
					m8470/0
					m8471/0
					m8480/0
					m8503/0
					m8504/0
					m8505/0
					m8506/0
					m8550/0
					m8560/0
					m8561/0
					m8580/0
					m8587/0
					m8600/0
					m8601/0
					m8602/0
					m8610/0
					m8630/0
					m8631/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8641/0
					m8650/0
					m8660/0
					m8670/0
					m8671/0
					m8680/0
					m8683/0
					m8700/0
					m8711/0
					m8712/0
					m8713/0
					m8720/0
					m8722/0
					m8723/0
					m8725/0
					m8726/0
					m8727/0
					m8728/0
					m8730/0
					m8740/0
					m8750/0
					m8760/0
					m8761/0
					m8770/0
					m8771/0
					m8772/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8780/0
					m8790/0
					m8800/0
					m8810/0
					m8811/0
					m8812/0
					m8813/0
					m8815/0
					m8820/0
					m8823/0
					m8824/0
					m8825/0
					m8826/0
					m8830/0
					m8831/0
					m8832/0
					m8840/0
					m8842/0
					m8850/0
					m8851/0
					m8852/0
					m8854/0
					m8856/0
					m8857/0
					m8860/0
					m8861/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8862/0
					m8870/0
					m8880/0
					m8881/0
					m8890/0
					m8891/0
					m8892/0
					m8893/0
					m8894/0
					m8895/0
					m8900/0
					m8903/0
					m8904/0
					m8905/0
					m8930/0
					m8932/0
					m8935/0
					m8936/0
					m8940/0
					m8959/0
					m8965/0
					m8966/0
					m8967/0
					m8982/0
					m8983/0
					m8990/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m9000/0
					m9010/0
					m9011/0
					m9012/0
					m9013/0
					m9014/0
					m9015/0
					m9016/0
					m9020/0
					m9030/0
					m9040/0
					m9050/0
					m9051/0
					m9052/0
					m9053/0
					m9054/0
					m9055/0
					m9080/0
					m9084/0
					m9090/0
					m9100/0
					m9103/0
					m9110/0
					m9120/0
					m9121/0
					m9122/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m9123/0
					m9125/0
					m9130/0
					m9131/0
					m9132/0
					m9141/0
					m9142/0
					m9150/0
					m9160/0
					m9170/0
					m9171/0
					m9172/0
					m9173/0
					m9174/0
					m9175/0
					m9180/0
					m9191/0
					m9200/0
					m9210/0
					m9220/0
					m9221/0
					m9230/0
					m9241/0
					m9252/0
					m9262/0
					m9270/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m9271/0
					m9272/0
					m9273/0
					m9274/0
					m9275/0
					m9280/0
					m9281/0
					m9282/0
					m9290/0
					m9300/0
					m9301/0
					m9302/0
					m9310/0
					m9311/0
					m9312/0
					m9320/0
					m9321/0
					m9322/0
					m9330/0
					m9340/0
					m9363/0
					m9370/0
					m9373/0
					m9390/0
					m9413/0
					m9490/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m9491/0
					m9492/0
					m9493/0
					m9501/0
					m9502/0
					m9507/0
					m9510/0
					m9530/0
					m9531/0
					m9532/0
					m9533/0
					m9534/0
					m9535/0
					m9537/0
					m9540/0
					m9541/0
					m9550/0
					m9560/0
					m9562/0
					m9570/0
					m9571/0
					m9580/0
					m9582/0

4.24.4. Capabilities

Table 4.45. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt

4.25. DuplicateTermFilter

4.25.1. Description

Mark entities as "Duplicate", if necessary (see DictTermFilters for description of markers).

Entities with the same AttributeValue and begin and end indices are considered duplicates.

4.25.2. Location

MedTKATp/descriptors/analysis_engine/primitive/DuplicateTermFilter.xml

4.25.3. Parameters

Table 4.46. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
EnclosingSpan	String	Yes	No	Class of span to process (e.g., sentence, noun phrase, etc.).	uima.tt.SentenceAnnotat
AllowedMarkersMask	Integer	No	No	Markers allowed to be set in entities to be processed. Bit mask, as defined in DictTermFilters	257
SemanticClasses	String	Yes	Yes	Semantic classes of terms (assumes	Site Diagnosis

Name	Type	Mandatory	Multi-valued	Description	Preset values
				DictTerm annotations) upon which to operate.	

4.25.4. Capabilities

Table 4.47. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt

4.26. SubsumptionFilter

4.26.1. Description

Mark entities as "Subsumed", if necessary (see DictTermFilters for description of markers).

Mark duplicates (duplicates are defined as having equal begin, end and semantic class)

Entities containing commas are not processed if their SemClass is specified in the CommaOverridesSubsumption parameter. Otherwise, entities are marked as Subsumed if they have fewer tokens than another term and begin and end indices either match or fall between another entity's begin and end indices, depending on the setting of the TokensMatchCriterion parameter.

4.26.2. Location

MedTKATp/descriptors/analysis_engine/primitive/SubsumptionFilter.xml

4.26.3. Parameters

Table 4.48. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
EnclosingSpan	String	Yes	No	Class of span to process (e.g., sentence, noun phrase, etc.).	uima.tt.SentenceAnnotation

Name	Type	Mandatory	Multi-valued	Description	Preset values
AllowedMarkersMask	Integer	No	No	Markers allowed to be set in entities to be processed. Bit mask, as defined in DictTermFilters	257
SemanticClasses	String	Yes	Yes	Semantic classes of terms (assumes DictTerm annotations) upon which to operate.	Site Diagnosis
TokensMatchCriterion	String	Yes	No	indicates whether any or all tokens need to match to qualify for subsumption. Possible values are: • NoMatchingTokensRequired (default) • AtLeastOneTokenRequiredToMat • AllTokensRequiredToMatch	AtLeastOneTokenRequir
CommaOverridesSubsumption	String	Yes	Yes	Semantic classes of terms (assumes DictTerm annotations) upon which to operate.	Site

4.26.4. Capabilities

Table 4.49. Capabilities

Name	Input	Output	Namespace
SentenceAnnotation	Yes	No	uima.tt

4.27. DateFinder

4.27.1. Description

Finds and annotates dates.

4.27.2. Location

`MedTKATp/descriptors/analysis_engine/primitive/DateFinder.xml`

4.27.3. Parameters

None.

4.27.4. Capabilities

Table 4.50. Capabilities

Name	Input	Output	Namespace
DateAnnotation	No	Yes	org.ohnlp.medkat.taes.support.dateFinder
SCRDate	No	Yes	org.ohnlp.medkat.scr.types

4.28. DrNo

4.28.1. Description

Using negation terms (as specified by "NoTermType" annotations) in conjunction with noun-phrase chunking information, find negated tokens and mark them as such (current implementation assume tokens are of type "org.ohnlp.medkat.taes.conceptMapper.DictTerm"). Initially, tokens are considered following the negation term, and if none are found, then prior to the negation term. See negation algorithm [11] for more details.

4.28.2. Location

`MedTKATp/descriptors/analysis_engine/primitive/DrNo.xml`

4.28.3. Parameters

Table 4.51. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
NoTermType	String	Yes	No	Type name of annotations that indicate negation.	org.ohnlp.medkat.taes.co
NoTermEnclosingSpanFeature	String	Yes	No	Name of feature within specified NoTermType annotation used to delimit negation processing (e.g. a sentence).	enclosingSpan
SemanticClassesToApplyNegation	String	No	Yes	Name of feature within specified NoTermType annotation used to delimit negation processing (e.g. a sentence).	Diagnosis Metastatic Invasion Lymph Site

4.28.4. Capabilities

Table 4.52. Capabilities

Name	Input	Output	Namespace
NoTerm	Yes	No	org.ohnlp.medkat.taes.conceptM
DictTerm	Yes	No	org.ohnlp.medkat.taes.conceptM

Name	Input	Output	Namespace
NPCombinedAnnotation	Yes	No	org.ohnlp.medkat.taes.npMerger

4.29. DimensionFinder

4.29.1. Description

Finds and annotates sizes or ranges of sizes, up to three dimensions, including unit expressions. Units are limited to 'cm' and 'mm'

4.29.2. Location

MedTKATp/descriptors/analysis_engine/primitive/DimensionFinder.xml

4.29.3. Parameters

None.

4.29.4. Capabilities

Table 4.53. Capabilities

Name	Input	Output	Namespace
DateAnnotation	No	Yes	org.ohnlp.medkat.taes.dimensionAnnotat
DateAnnotation	No	Yes	org.ohnlp.medkat.taes.dimensionAnnotat
DateAnnotation	No	Yes	org.ohnlp.medkat.taes.dimensionAnnotat
DateAnnotation	No	Yes	org.ohnlp.medkat.taes.dimensionAnnotat
DateAnnotation	No	Yes	org.ohnlp.medkat.taes.dimensionAnnotat
SCRDimension	No	Yes	org.ohnlp.medkat.scr.types
SCRSize	No	Yes	org.ohnlp.medkat.scr.types

4.30. DiagnosisCoreferencer

4.30.1. Description

Finds co-referring diagnoses, as defined in the discussion of the co-referencing algorithm [12], above. Makes two assumptions:

- ICD-O is coding system used.

- Using default MedKAT typesystem, with DictTerm annotations representing named entities.

A new version of this should be created without these assumptions.

4.30.2. Location

MedTKATp/descriptors/analysis_engine/primitive/DiagnosisCoreferencer.xml

4.30.3. Parameters

Table 4.54. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
NewAnnotationName	String	Yes	No	Fully qualified name of annotation to create as a coreference annotation.	org.ohnlp.medkat.taes.co
NewAnnotationFeatureName	String	Yes	No	Name of feature in annotation specified by parameter NewAnnotationName that is used to store coreferring elements.	elements
NewAnnotationSectionNumberFeatureName	String	Yes	No	Name of feature in annotation specified by parameter NewAnnotationName that is used to store the section number of the newly created annotation.	subsectionNumber

Name	Type	Mandatory	Multi-valued	Description	Preset values
SemClass	String	Yes	No	Value of SemClass feature of org.ohnlp.medkat.taes.conceptMapper.DictTerm annotation that must be matched for all coreferenced items. NOTE: this should be changed so that there is no dependency on a specific typesystem!.	Site
SectionAnnotations	String	Yes	Yes	Sections within which to perform coreferencing.	org.ohnlp.medkat.taes.sectionFin org.ohnlp.medkat.taes.sectionFin org.ohnlp.medkat.taes.sectionFin org.ohnlp.medkat.taes.sectionFin org.ohnlp.medkat.taes.sectionFin
GenericTermCodes	String	No	Yes	Sections within which to perform coreferencing.	m8000/0 m8000/1 m8000/3 m8000/6 m8000/9 m8001/0 m8001/1 m8001/3 m8010/0

Name	Type	Mandatory	Multi-valued	Description	Preset values
					m8010/2 m8010/3 m8010/6
ExcludedSubSubsections	String	No	Yes	Subsubsections to ignore during coreferencing.	org.ohnlp.medkat.taes.s
AnnotatorName	String	No	No	Name of this annotator, for logging messages.	DiagnosisCoreferencer

4.30.4. Capabilities

Table 4.55. Capabilities

Name	Input	Output	Namespace
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectio
DictTerm	Yes	No	org.ohnlp.medkat.taes.concept M
CoreferringDiagnoses	No	Yes	org.ohnlp.medkat.taes.coreferen

4.31. SiteCoreferencer

4.31.1. Description

Finds co-referring sites, as defined in the discussion of the co-referencing algorithm [12], above, above. Makes two assumptions:

- ICD-O is coding system used.
- Using default MedKAT typesystem, with DictTerm annotations representing named entities.

A new version of this should be created without these assumptions.

4.31.2. Location

MedTKATp/descriptors/analysis_engine/primitive/SiteCoreferencer.xml

4.31.3. Parameters

Table 4.56. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
NewAnnotationName	String	Yes	No	Fully qualified name of annotation to create as a coreference annotation.	org.ohnlp.medkat.taes.coreference
NewAnnotationFeatureName	String	Yes	No	Name of feature in annotation specified by parameter NewAnnotationName that is used to store coreferring elements.	elements
NewAnnotationSectionNumberFeatureName	String	Yes	No	Name of feature in annotation specified by parameter NewAnnotationName that is used to store the section number of the newly created annotation.	subsectionNumber
SemClass	String	Yes	No	Value of SemClass feature of org.ohnlp.medkat.taes.conceptMapper.DictTerm annotation that must be matched	Site

Name	Type	Mandatory	Multi-valued	Description	Preset values
				for all coreferenced items. NOTE: this should be changed so that there is no dependency on a specific typesystem!.	
SectionAnnotations	String	Yes	Yes	Sections within which to perform coreferencing.	org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s
ExcludedSubSubsections	String	No	Yes	Subsubsections to ignore during coreferencing.	org.ohnlp.medkat.taes.s
AnnotatorName	String	No	No	Name of this annotator, for logging messages.	SiteCoreferencer

4.31.4. Capabilities

Table 4.57. Capabilities

Name	Input	Output	Namespace
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectio
DictTerm	Yes	No	org.ohnlp.medkat.taes.concept M
CoreferringSites	No	Yes	org.ohnlp.medkat.taes.coreferen

4.32. GradeDetector

4.32.1. Description

Create annotations that enclose matching parentheses, as "(" and ")" or "[" and "]" or "{" and "}".

4.32.2. Location

MedTKATp/descriptors/analysis_engine/primitive/GradeDetector.xml

4.32.3. Parameters

Table 4.58. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
TokenAnnotation	String	Yes	No	Type name of a token annotation.	uima.tt.Tokenannotation
SentenceAnnotationType	String	Yes	No	Type name of a sentence annotation.	uima.tt.SentenceAnnotation
UnknownMaxValueIndicator	String	No	No	Value to use of no grade scale defined. Default is "0".	"Unspecified"
PrimarySectionAnnotations	String	Yes	Yes	Type name of document sections to process.	org.ohnlp.medkat.taes.sectionFin
ExcludedSubSubsections	String	No	Yes	Type name of document sub-subsections to NOT process.	org.ohnlp.medkat.taes.subSubsec
GleasonsGrade	String	Yes	No	SemClass attribute associated	GleasonsGrade

Name	Type	Mandatory	Multi-valued	Description	Preset values
				with grade named entity annotation for Total Gleasons Grade (promary + secondary).	
PrimaryGleasonGrade	String	Yes	No	SemClass attribute associated with grade named entity annotation for Primary Gleason's Grade.	GleasonsGrade
SecondaryGleasonGrade	String	Yes	No	SemClass attribute associated with grade named entity annotation for Secondary Gleasons Grade.	GleasonsGrade
MaxToLookBeyond	Integer	Yes	No	How many tokens beyond grade term to look for grade level	GleasonsGrade

4.32.4. Capabilities

Table 4.59. Capabilities

Name	Input	Output	Namespace
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectionDetector
DictTerm	Yes	Yes	org.ohnlp.medkat.taes.conceptMapper
GradeAnnotation	No	Yes	org.ohnlp.medkat.taes.gradeDetector

4.33. SCRNamedEntityTypeConverter

4.33.1. Description

Converts instances of internal MedKAT named entity types to SCRxxx external types. The conversion is performed for anatomical sites, diagnoses and generic named entities. Instances of internal types that contain coreferenced objects are also converted to the corresponding external types.

4.33.2. Location

MedKATp/descriptors/analysis_engine/primitive/SCRNamedEntityTypeConverter.xml

4.33.3. Parameters

None

4.33.4. Capabilities

Table 4.60. Capabilities

Name	Input	Output	Namespace
DictTerm	Yes	No	org.ohnlp.medkat.taes.conceptMapper
CorefAnnotation	Yes	No	org.ohnlp.medkat.taes.coreferencer
SCRanatomicalSite	No	Yes	org.ohnlp.medkat.scr.types
SCRCoreference	No	Yes	org.ohnlp.medkat.scr.types
SCRHistologicalDiagnosis	No	Yes	org.ohnlp.medkat.scr.types
SCRNamedEntity	No	Yes	org.ohnlp.medkat.scr.types

4.34. SizeLocationRegExAnnotator

4.34.1. Description

Matches regular expressions in document text.

4.34.2. Location

MedKATp/descriptors/analysis_engine/primitive/SizeLocationRegExAnnotator.xml

4.34.3. Parameters

Table 4.61. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
Patterns	String	No	Yes	Regular expression patterns to match. The language is that supported by Java 1.4..	\d\so \'\sclock \saxis| \d\d\so \'\sclock \saxis|\d \d\:\d \d\saxis| \d\:\d \d\saxis| \d\d\so \'\sclock| \d\d\sO \'\sCLOCK| \d\so \'\sclock| \d\sO \'\sCLOCK| \d\so \'clock| \d\d\so \'clock|\d \d:\d\d| \d\:\d\d (?i)(?: [<>]?(?:(?: \d+(?:\. \d*)?|\.\d

Name	Type	Mandatory	Multi-valued	Description	Preset values
					+)\s*[cm]m))|(?:\d+(?:\.\d*)?|\.\d+)\s*x\s*(?:\d+(?:\.\d*)?|\.\d+)(?:\s*x\s*(?:\d+(?:\.\d*)?|\.\d+))?\s*(?:[cm]m)? cm|CM|mm|MM \d+/\d+|number\s*of\s*metastatic\s*nodes:[123456789][0123456789]* __NUMBER__
PatternFiles	String	No	Yes	Names of files containing patterns to match, using the same pattern language as for the Patterns configuration paramter. Either Patterns or PatternFiles may be	

Name	Type	Mandatory	Multi-valued	Description	Preset values
				specified, but not both.	
TypeName	String	Yes	Yes	Names of CAS Types to create for the patterns found. The indexes of this array correspond to the indexes of the Patterns or PatternFiles arrays. If a match is found for Patterns[i] or for any pattern in PatternFile[i], it will result in an annotation of type TypeNames[i].	org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s org.ohnlp.medkat.taes.s
ContainingAnnotationTypes	String	No	Yes	Names of CAS Input Types within which annotations should be created.	
AnnotateEntireContainingAnnotation	Boolean	No	No	When the ContainingAnnoationTypes parameter is specified, a value of true for this parameter	

Name	Type	Mandatory	Multi-valued	Description	Preset values
				will cause the entire containing annotation to be used as the span of the new annotation, rather than just the span of the regular expression match. This can be used to "classify" previously created annotations according to whether or not they contain text matching a regular expression.	

4.34.4. Capabilities

Table 4.62. Capabilities

Name	Input	Output	Namespace
LocationExpression	No	Yes	org.ohnlp.medkat.taes.sizeLocationRegEx
SizeExpression	No	Yes	org.ohnlp.medkat.taes.sizeLocationRegEx
UnitExpression	No	Yes	org.ohnlp.medkat.taes.sizeLocationRegEx
LymphLevelExpression	No	Yes	org.ohnlp.medkat.taes.sizeLocationRegEx
NumberExpression	No	Yes	org.ohnlp.medkat.taes.sizeLocationRegEx

4.35. Disambiguator

4.35.1. Description

Disambiguates between different size types.

4.35.2. Location

MedKATp/descriptors/analysis_engine/primitive/Disambiguator.xml

4.35.3. Parameters

Table 4.63. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
spanAnnotationName	String	No	Yes	spans to be checked	uima.tt.SentenceAnnotati
nppAnnotationName	String	Yes	No	Type name of prepositional noun phrases	uima.tt.NPPAnnotation
npAnnotationName	String	Yes	No	Type name of noun phrases	uima.tt.NPAnnotation
siteAnnotationName	String	Yes	No	Type name of anatomical sites	org.ohnlp.medkat.scr.typ
ppAnnotationName	String	Yes	No	Type name of prepositional phrases	uima.tt.PPAnnotation
ExcludingPrepositions	String	No	Yes	Prepositions excluded	from, to

4.35.4. Capabilities

Table 4.64. Capabilities

Name	Input	Output	Namespace
DimensionSetAnnotation	Yes	No	org.ohnlp.medkat.taes.dimensionAnnotat
NPCombinedAnnotation	Yes	No	org.ohnlp.medkat.taes.npMerger
MarginAnnotation	No	Yes	org.ohnlp.medkat.taes.disambiguator
OtherDimensionAnnotation	No	Yes	org.ohnlp.medkat.taes.disambiguator
SizeDimensionAnnotation	No	Yes	org.ohnlp.medkat.taes.disambiguator
TumorSizeAnnotation	No	Yes	org.ohnlp.medkat.taes.disambiguator

4.36. LymphStatus

4.36.1. Description

Parses lymph nodes expressions and populates appropriate attributes in annotations of the corresponding type set by parameters.

4.36.2. Location

MedKATp/descriptors/analysis_engine/primitive/LymphStatus.xml

4.36.3. Parameters

Table 4.65. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
LymphLevelExpressionName	String	No	No	Type name of annotation that specify lymph nodes expressions	org.ohnlp.medkat.taes.sizeLocatio
NumberName	String	No	No	Type name of annotation that specify numeric expressions	org.ohnlp.medkat.taes.sizeLocatio

Name	Type	Mandatory	Multi-valued	Description	Preset values
SentenceClass	String	Yes	No	Class name of sentence annotations	uima.tt.SentenceAnnotat

4.36.4. Capabilities

Table 4.66. Capabilities

Name	Input	Output	Namespace
LymphLevelExpression	Yes	Yes	org.ohnlp.medkat.taes.sizeLocat
DictTerm	Yes	No	org.ohnlp.medkat.conceptMappe
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectio
DiagnosisAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFir
SectionAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFir
NumberExpression	Yes	No	org.ohnlp.medkat.taes.sizeLocat
DateAnnotation	Yes	No	org.ohnlp.medkat.taes.support.d

4.37. NPMerger

4.37.1. Description

For related NP annotations of different types produces a single annotation that includes all relevant pieces.

4.37.2. Location

MedKATp/descriptors/analysis_engine/primitive/NPMerger.xml

4.37.3. Parameters

None

4.37.4. Capabilities

Table 4.67. Capabilities

Name	Input	Output	Namespace
NPAnnotation	Yes	No	uima.tt

Name	Input	Output	Namespace
NPListAnnotation	Yes	No	uima.tt
NPPAnnotation	Yes	No	uima.tt
NPCombinedAnnotation	Yes	No	org.ohnlp.medkat.taes.npMerger

4.38. LymphNodesAnnotator

4.38.1. Description

Creates annotations that contain information about lymph nodes described in the report. (See Lymph nodes model algorithm [13] for the algorithm overview.

4.38.2. Location

MedKATp/descriptors/analysis_engine/primitive/LymphNodesAnnotator.xml

4.38.3. Parameters

Table 4.68. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
DiagnosisTypes	String	Yes	Yes	Type names of annotations representing diagnoses.	org.ohnlp.medkat.scr.types.SCRHi
SiteTypes	String	Yes	Yes	Type names of annotations representing anatomical sites	org.ohnlp.medkat.scr.types.SCRA
UndefinedNodeCount	Integer	Yes	No	Numeric value that is assigned to appropriate attributes of lymph node model,	999999

Name	Type	Mandatory	Multi-valued	Description	Preset values
				when the number of the nodes is not specified in the report and cannot be inferred.	
SentenceClass	String	Yes	No	Class name of annotations that represent sentences	uima.tt.SentenceAnnotat

4.38.4. Capabilities

Table 4.69. Capabilities

Name	Input	Output	Namespace
SyntacticUnit	Yes	No	org.ohnlp.medkat.taes.syntacticU
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsection
LymphLevelExpression	Yes	No	org.ohnlp.medkat.taes.sizeLocat
DictTerm	Yes	No	org.ohnlp.medkat.concept Mapp
DiagnosisAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFir
SCRLymphNodesReading	No	Yes	org.ohnlp.medkat.scr.types
SCRLymphNodes	No	Yes	org.ohnlp.medkat.scr.types

4.39. GrossDescriptionAnnotator

4.39.1. Description

Creates annotations that contain information about gross description parts described in a report. (See Gross description model algorithm [13] for the algorithm overview.

4.39.2. Location

MedKATp/descriptors/analysis_engine/primitive/GrossDescriptionAnnotator.xml

4.39.3. Parameters

Table 4.70. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
SizeTypeNames	String	Yes	Yes	Type names of annotations representing gross description sizes	org.ohnlp.medkat.scr.types.SCRSi
SiteTypeNames	String	Yes	Yes	Type names of annotations representing anatomical sites.	org.ohnlp.medkat.scr.types.SCRA
FragmentsFeatureNames	String	Yes	Yes	A name of an attribute that contain textual fragments of anatomical sites	Fragments
SentenceClass	String	Yes	No	Class name of annotations that represent sentences	uima.tt.SentenceAnnotation
TokenClass	String	Yes	No	Class name of annotations that represent tokens	uima.tt.TokenAnnotation
NPClass	String	Yes	No	Class name for annotation that	uima.tt.NPAnnotation

Name	Type	Mandatory	Multi-valued	Description	Preset values
				represent noun phrases	
NPListClass	String	Yes	No	Class name for annotation that represent list of noun phrases	uima.tt.NPListAnnotatio
NPPClass	String	Yes	No	Class name for annotation that represent prepositional noun phrases	uima.tt.NPPAnnotation
NPSClass	String	Yes	No	Class name for annotation that represent possessive noun phrases	uima.tt.NPSAnnotation

4.39.4. Capabilities

Table 4.71. Capabilities

Name	Input	Output	Namespace
SyntacticUnit	Yes	No	org.ohnlp.medkat.taes.syntacticU
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectio
SCRSize	Yes	No	org.ohnlp.medkat.scr.types
SCRAnatomicalSite	Yes	No	org.ohnlp.medkat.scr.types
NPSAnnotation	Yes	No	uima.tt

Name	Input	Output	Namespace
NPPAnnotation	Yes	No	uima.tt
NPListAnnotation	Yes	No	uima.tt
NPAnnotation	Yes	No	uima.tt
NPCombinedAnnotation	Yes	No	org.ohnlp.medkat.taes.npMerger
GrossDescriptionAnnotation	Yes	No	org.ohnlp.medkat.taes.sectionFinder
SCRGrossDescriptionPart	No	Yes	org.ohnlp.medkat.scr.types
SCRGrossDescription	No	Yes	org.ohnlp.medkat.scr.types
ParenSeparatedNPAnnotation	No	Yes	org.ohnlp.medkat.taes.grossDescription

4.40. TumorModelAnnotator

4.40.1. Description

Creates annotations that contain information about primary and metastatic tumors. (See tumor model algorithm [12]) for the algorithm overview.

4.40.2. Location

MedKATp/descriptors/analysis_engine/primitive/TumorModelAnnotator.xml

4.40.3. Parameters

Table 4.72. Parameters

Name	Type	Mandatory	Multi-valued	Description	Preset values
SectionAnnotations	String	Yes	Yes	Type name of document sections to process.	org.ohnlp.medkat.taes.sectionFin
ExcludedSubSubsections	String	No	Yes	Type name of document sub-subsections to NOT process.	org.ohnlp.medkat.taes.subSubsec
AnnotatorName	String	No	No	Name of this annotator,	TumorModelAnnotator

Name	Type	Mandatory	Multi-valued	Description	Preset values
				for logging messages.	
CoreferenceAnnotationType	String	Yes	No	Type name of this coreference annotations.	org.ohnlp.medkat.scr.typ
CoreferenceFeature	String	Yes	No	Name of feature of coreference annotations that contains the coreferring annotations.	Elements
ExcludingPrepositions	String	Yes	Yes	Prepositions that indicate an anatomical site in its prep phrase is not to be linked to the diagnosis as site.	from through into to at
tumorSizeTypeName	String	Yes	No	Name of type of Anatomical Site annotations.	org.ohnlp.medkat.taes.di
siteTypeName	String	Yes	No	Name of type of size annotations.	org.ohnlp.medkat.scr.typ
siteTerminologyFeatureName	String	Yes	No	Name of feature of Anatomical Site annotations that specifies the name	Terminology

Name	Type	Mandatory	Multi-valued	Description	Preset values
				of the terminology coding system.	
siteCodeFeatureName	String	Yes	No	Name of feature of Anatomical Site annotations that specifies its code within the specified terminology coding system.	Code
siteCorefsFeatureName	String	Yes	No	Name of feature of Anatomical Site annotations that contains coreferring Anatomical Sites.	Coreferences
siteNegationFeatureName	String	Yes	No	Name of feature of Anatomical Site annotations that specifies whether this Anatomical Site entity has been negated.	Negation
diagnosisTypeName	String	Yes	No	Name of type of Histological	org.ohnlp.medkat.scr.types.SCRHi

Name	Type	Mandatory	Multi-valued	Description	Preset values
				Diagnosis annotations.	
diagnosisTerminologyFeatureName	String	Yes	No	Name of feature of Histological Diagnosis annotations that specifies the name of the terminology coding system.	Terminology
diagnosisCodeFeatureName	String	Yes	No	Name of feature of Histological Diagnosis annotations that specifies its code within the specified terminology coding system.	Code
diagnosisCorefFeatureName	String	Yes	No	Name of feature of Histological Diagnosis annotations that contains coreferring Histological Diagnosis.	Coreferences
diagnosisNegationFeatureName	String	Yes	No	Name of feature of Histological Diagnosis annotations that specifies	Negation

Name	Type	Mandatory	Multi-valued	Description	Preset values
				whether this Histological Diagnosis entity has been negated.	

4.40.4. Capabilities

Table 4.73. Capabilities

Name	Input	Output	Namespace
SubHeading	Yes	No	org.ohnlp.medkat.taes.subsectionDetector
DictTerm	Yes	No	org.ohnlp.medkat.taes.conceptMapper
SCRHistologicalDiagnosis	Yes	No	org.ohnlp.medkat.scr.types
SCRAnatomicalSite	Yes	No	org.ohnlp.medkat.scr.types
SCRCoreference	Yes	No	org.ohnlp.medkat.scr.types
CoreferringDiagnoses	No	Yes	org.ohnlp.medkat.taes.coreferencer
PrimaryDiagnosis	No	Yes	org.ohnlp.medkat.taes.diagnosisTypeDete
OtherDiagnosis	No	Yes	org.ohnlp.medkat.taes.diagnosisTypeDete
MetastaticDiagnosis	No	Yes	org.ohnlp.medkat.taes.diagnosisTypeDete
LymphDiagnosis	No	Yes	org.ohnlp.medkat.taes.diagnosisTypeDete
DiagnosisBase	No	Yes	org.ohnlp.medkat.taes.diagnosisTypeDete
SCRPrimaryTumorReading	No	Yes	org.ohnlp.medkat.scr.types
SCRPrimaryTumor	No	Yes	org.ohnlp.medkat.scr.types
SCRMetastaticTumorReading	No	Yes	org.ohnlp.medkat.scr.types
SCRMetastaticTumor	No	Yes	org.ohnlp.medkat.scr.types

Bibliography

[ChapmanEtAl2001] W.W. Chapman, W. Bridewell, P. Hanbury, G.F. Cooper, and B.G. Buchanan. Copyright © 2001 Journal of Biomedical Informatics. *A simple algorithm for identifying negated findings and diseases in discharge summaries*. 34. 301-310.

[KennedyBoguraev1996] C. Kennedy and B. Boguraev. Copyright © 1996 COLING. *Anaphora for Everyone: Pronominal Anaphora Resolution without a Parser*. 113-118.

[CodenEtal2009] Anni Coden, Guergana Savova, Igor Sominsky, Michael Tanenblatt, James Masanz, Karin Schuler, James Cooper, Wei Guan, and Piet C. de Groen. Copyright © 2009 Journal of Biomedical Informatics. *Automatically extracting cancer disease characteristics from pathology reports into a Disease Knowledge Representation Model*. `http://dx.doi.org/10.1016/j.jbi.2008.12.005`