Advanced GATE Applications

Topics covered

- This module is about adapting ANNIE to create your own applications, and to look at more advanced techniques within applications
  - Using different gazetteers
  - Using conditional applications
  - Section-by-section processing
  - Using multiple annotation sets
  - Separating useful content in a document
  - Schema Enforcer
  - Using Groovy
  - Modular Pipelines

Using different gazetteers

Why?

- The standard gazetteer in ANNIE only performs exact matching against the text
- An entry in a gazetteer list must match the word exactly in the text (with the exception of capitalisation issues depending on if the case-sensitive parameter is switched on)
- But what if we want to match a plural word in the text with a singular word in the gazetteer?
- Or different forms of a verb (says, saying, say, said etc.)
- It would be nice not to have to specify alternative forms of each word in the lists
- Luckily, we have ways to do this

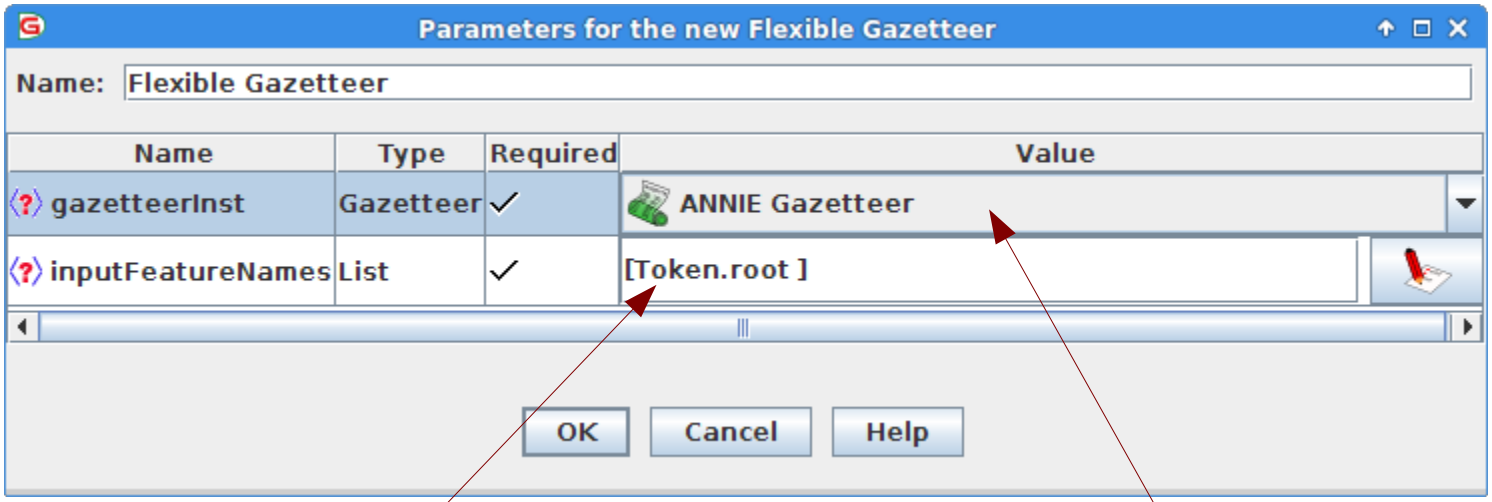
Advanced Gazetteers

- There are several different gazetteers which let you do more complex matching
  - **Flexible Gazetteer**: enables matching against features on an annotation (typically the Token's root feature)
  - **Feature Gazetteer**: enables matching against features on an annotation, but also enables adding/removing annotations and features when a match is found
  - **Extended Gazetteer**: as for the flexible gazetteer, but also provides features for more powerful matching of partial words, annotating prefixes and suffixes, and more versatile handling of word boundaries and white space.
  - **BWP Gazetteer**: approximate gazetteer based on Levenshtein Edit Distance for strings, aiming to handle text with noise and errors

Flexible Gazetteer

- Found in the Tools plugin
- Requires a regular gazetteer to be loaded - this should not be in the pipeline, however
- Load-time parameters let you specify:
  - the regular gazetteer to use
  - the annotations and features to match on
- The typical use for this is to match against the root form of a word (e.g. dogs -> dog; laughing -> laugh)
- To do this, you need to specify Token.root as the annotation and feature to match on
- You also need to make sure you have run the morphological analyser first, so you have root features on your Tokens

Flexible gazetteer load-time parameters



Choose the annotation name and feature that you want to match on

Select a gazetteer that you have loaded, e.g. the default ANNIE one

Hands-on with flexible gazetteer

- Load ANNIE
- Load the Tools plugin
- Create a new Flexible Gazetteer, and select Token.root as the input Feature name
- Select the ANNIE gazetter as the gazetteer instance to use
- Create a new morphological analyser
- Go to the ANNIE application and add the morphological analyser and flexible gazetteer to the pipeline after the POS tagger
- Remove the ANNIE gazetteer from the application (but don't remove it from GATE!)
- Try it on some text!

Extended Gazetteer

- Found in the StringAnnotation plugin Plugin Repository "Additional Plugins from the GATE Team" or download from https://github.com/johann-petrak/gateplugin-StringAnnotation
- Faster loading, uses much less memory than regular gazetteer
- Needs annotations that identify words and whitespace
- Can limit annotating to just within containing annotations
- Same PR can be used for direct matching of document text or indirect matching of feature values
- Can specify whether to match at the beginning and/or the end of words separately
- Can use (gzip) compressed list files (.lst.gz)

Init parameters

- **caseSensitive**: false if case should be ignored for matching
- **configFile URL**: specify the definition/config file – similar to the "listsURL" parameter on the ANNIE gazetteer
- **caseConversionLanguage**: Specify the language to use for converting characters to upper case when case-insensitive matching (e.g. ß→SS for de) . Default is en (English)
- **gazetteerFeatureSeparator**: same as for the ANNIE gazetteer
- => no encoding parameters, list files have to be UTF-8 encoded

Run-time parameters

- **containingAnnotationType**: if an annotation type is given, then matching is done only fully within the span of such annotations. E.g. DocumentContent, Sentence.
- **longestMatchOnly**: if set to true, then only the longest match is used and all shorter matches are ignored.
- **matchAtWordEndOnl**y: if true, then the end of a match can only occur at the end of a word annotation. Typically set to true.
- **matchAtWordStartOnly**: if true, then the start of a match can only occur at the start of a word annotation. Typically set to true.
- **textFeature**: feature of the word annotation to match on (as for FlexibleGazetteer). Typically left empty or set to root.

More run-time parameters

- **outputAnnotationType**: in case you want to change the name of the annotation to be created on a match (instead of Lookup)
- **spaceAnnotationType**: the annotation type that identifies space between words, default is SpaceToken.
- **splitAnnotationType**: the annotation type that identifies positions in the document that should not be crossed by matches, default is Split.
- **wordAnnotationType**: type of annotations that define the word boundaries of the text that should be used for matching or if matching by feature is used, the annotations containing the feature. Typically set to Token.

Extended gazetteer cache files

- When a gazetteer is first loaded from a .def file, then the ExtendedGazetteer will create a new gazetteer cache file.
- This cache file has the same name as the .def file but with a file extension ".gazbin" instead of ".def".
- When the gazetteer gets loaded and such a cache file exists, the cache file will be loaded instead of the original files.
- NOTE: if a cache file exists, it will always be used, no matter if the .def or any .lst file has been changed in the meantime. If you update the gazetteer, make sure you select “Remove cache and re-initialise” in the GUI

Feature gazetteer

- Found in the StringAnnotation plugin
- Enables adding/removing annotations/features when a match is found
  - For example, if tokens have a root feature and there is a gazetteer list that has as a feature the frequencies of English word roots in some corpus, the "add features" action can be used to enrich the token annotations with word frequencies.
  - **filter annotations**: if there is a gazetteer of stopwords, the string or root feature of existing token annotations can be matched and the "remove annotation" action can be used to remove these annotations if a stopword is matched.

Init parameters

- exactly the same as for the ExtendedGazetteer
- Note: this gazetteer uses the cache, .def and .lst files in exactly the same way as the ExtendedGazetter. If the ExtendedGazetteer and/or FeatureGazetteer load from the same files using the same Init-parameters, only one shared copy is used in memory.

Feature gazetteer run-time parameters

- **containingAnnotationType**: If an annotation type is given, then matching is done only within the span of such annotations.
- **InputAnnotationSet**: the set that contains the annotations to be updated, if annotations are updated
- **matchAtStartOnly**: if true, then a match must be found at the start of the value of the feature, if false, a match may start anywhere.
- **matchAtEndOnly**: if true, then a match must be found that ends at the end of the value of the feature, if false, a match may end anywhere.
- **outputAnnotationType**: in case you want to change the name of the annotation to be created on a match, if annotations are created (instead of Lookup)

More run-time parameters

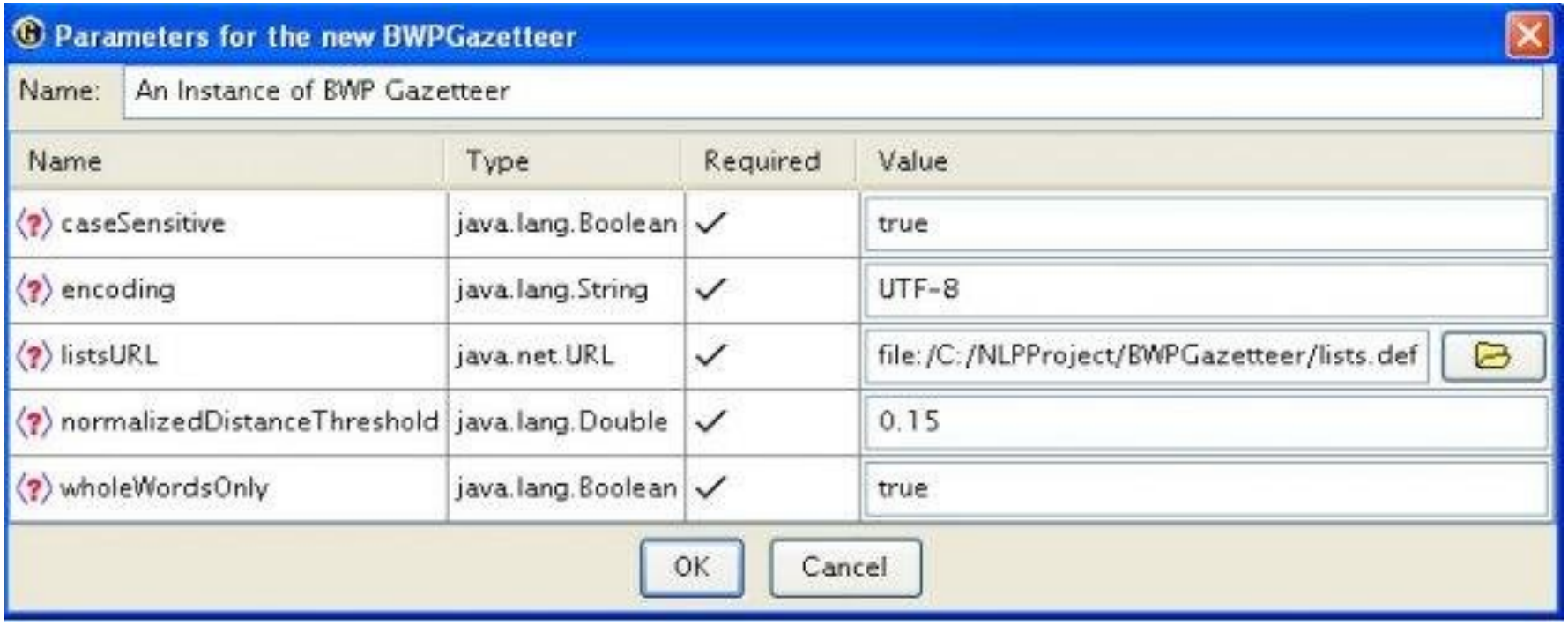
- **wordAnnotationType**: the annotation type that is used for matching. For example Token or Lookup.
- **textFeature**: the name of a feature of the word annotation which is used for matching, e.g. root or id
- **processingMode**: select an option from:
  - **AddFeatures**: add all features from the def file or the gazetteer entry which are not already present in the annotation
  - **OverwriteFeatures**: overwrite all features (if any existing) from the def file or the gazetteer entry with the new values
  - **RemoveAnnotation**: delete the annotation from the input AS
  - **AddNewAnnotation**: add a new annotation to the output AS
  - **KeepAnnotation**: keep annotation if match, else remove

Gazetteer features

- All GATE gazetteers allow arbitrary feature values to be associated with particular entries in a single list
- ANNIE does not use this capability, but to enable it for your own gazetteers, set the optional gazetteerFeatureSeparator parameter
- We advise setting the value of this feature to the tab character (\t) so that it is never confused with part of a list entry
- It also means you can directly use tab-separated value (.tsv) files from your favourite spreadsheet editor
- You do not have to provide the same features for every line in the file, e.g. you can provide extra features for some lines in the list but not others.
- Use the “+cols” option in the GATE gazetteer editor to add new sets of features and values

BWP gazetteer

- Does not come with GATE, but can be loaded as a plugin
- Download from http://sourceforge.net/projects/bwp-gazetteer/



Conditional Processing

What is conditional processing?

- In GATE, you can set a processing resource in your application to run or not depending on certain circumstances
- You can have several different PRs loaded, and let the system automatically choose which one to run, for each document.
- This is very helpful when you have texts in multiple languages, or of different types, which might require different kinds of processing
- For example, if you have a mixture of German and English documents in your corpus, you might have some PRs which are language-dependent and some which are not
- You can set up the application to run the relevant PRs on the right documents automatically.

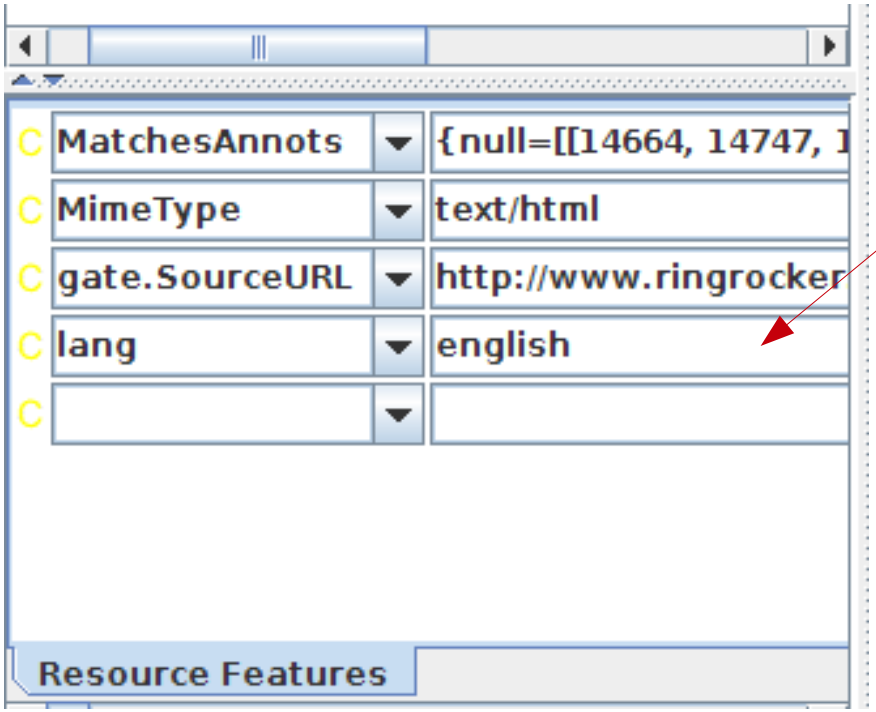
Conditional processing with different languages

- Suppose we have a corpus with documents in German and English, and we only want to process the English texts.
- First we must distinguish between the two kinds of text, using a language identification tool
- For this we can use TextCat, which is a GATE plugin
- We use this (in default mode) to add a feature on each document, telling us which language the document is in
- Then we run a conditional processing pipeline, that only runs the subsequent PRs if the value of the language feature on the document is English
- The other documents will not be processed

Hands-on with multilingual corpora (1)

- Create a new corpus in GATE and populate it with the two documents found in corpus/rar-english-german-corpus/
- Select iso-8859-1 as the encoding when you populate the corpus
- You should have one English and one German document loaded
- Load ANNIE
- Load the Language Identification plugin and load the TextCat Language Identification PR
- Add TextCat to the end of the ANNIE application and run it on the corpus
- You should get some sensible annotations for the English document and some slightly less sensible ones for the German one

Check the language of the documents



- Click on a document
- In the bottom left corner is the document features pane
- TextCat will add a language feature here

What if we want to process the German too?

- If we want to process both German and English documents with different resources, we have a couple of options
  1. We can just call some language-specific PRs conditionally, and use the language-neutral PRs on all documents
  2. We can call different applications from within the main application
- The following two hands-on exercises demonstrate the difference between these

Hands-on with multilingual apps (1)

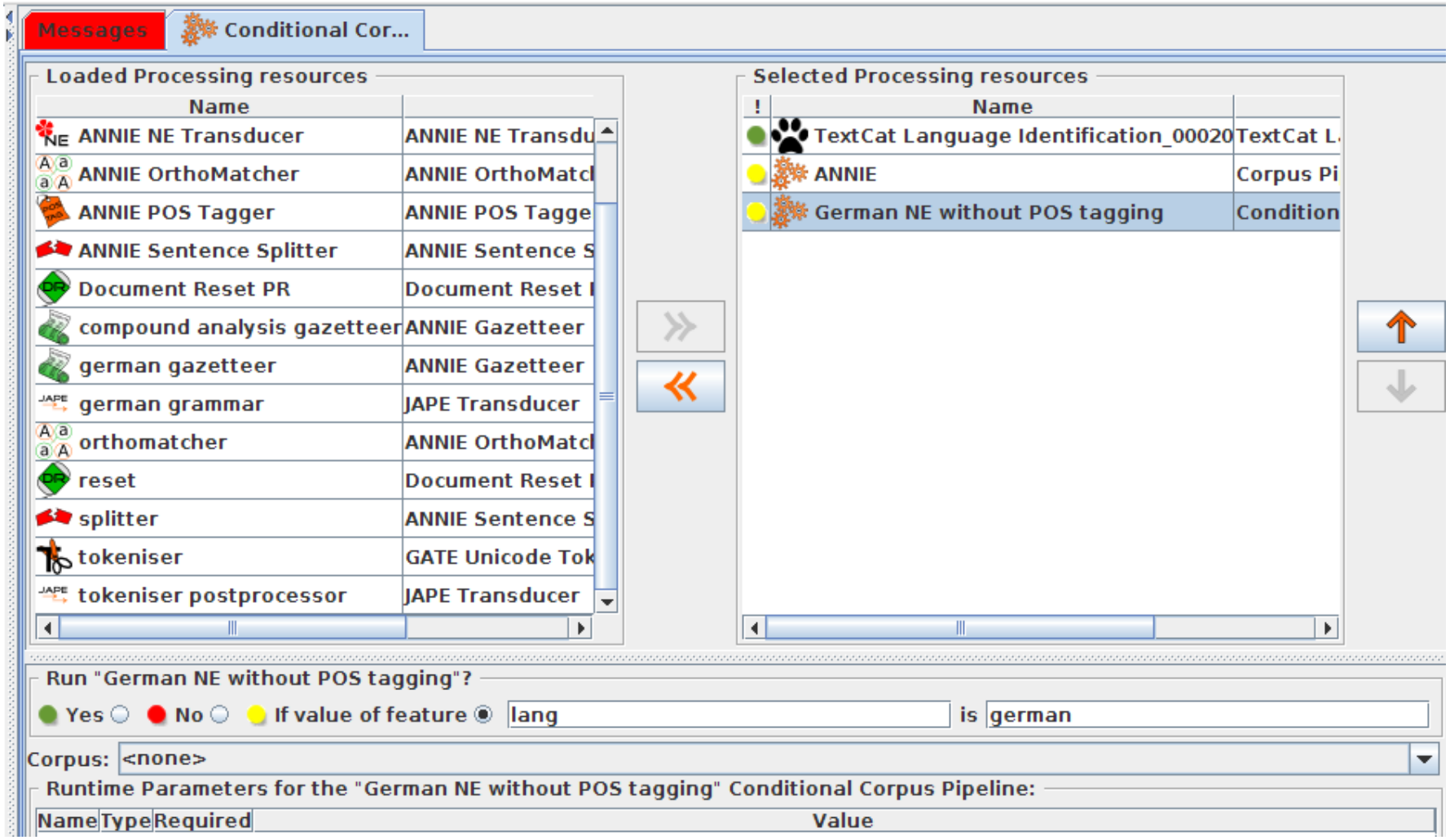
- Load the application annie+german.gapp
- Look at the various PRs in the app: some are set to run on English documents, some on German ones, and some on all documents
- Run the application on your corpus
- The German document should now be annotated with German NEs and the English document with English ones
- There will be some mistakes (we're not using a German POS tagger here so results are weaker than usual)

Hands-on with multilingual apps (2)

- Close recursively all applications you have loaded in GATE (keep the corpus)
- Load ANNIE with defaults
- Load the Lang_German plugin
- Load the German IE application from “Ready-made applications”
- Create a new conditional corpus pipeline
- Load a TextCat PR and add it to the new pipeline created
- Add the ANNIE and German **applications** to the pipeline (in either order) after the TextCat

  Set ANNIE to run on English documents and the German app to run on German ones
- Save the main application and run it on your corpus

Your application should look like this



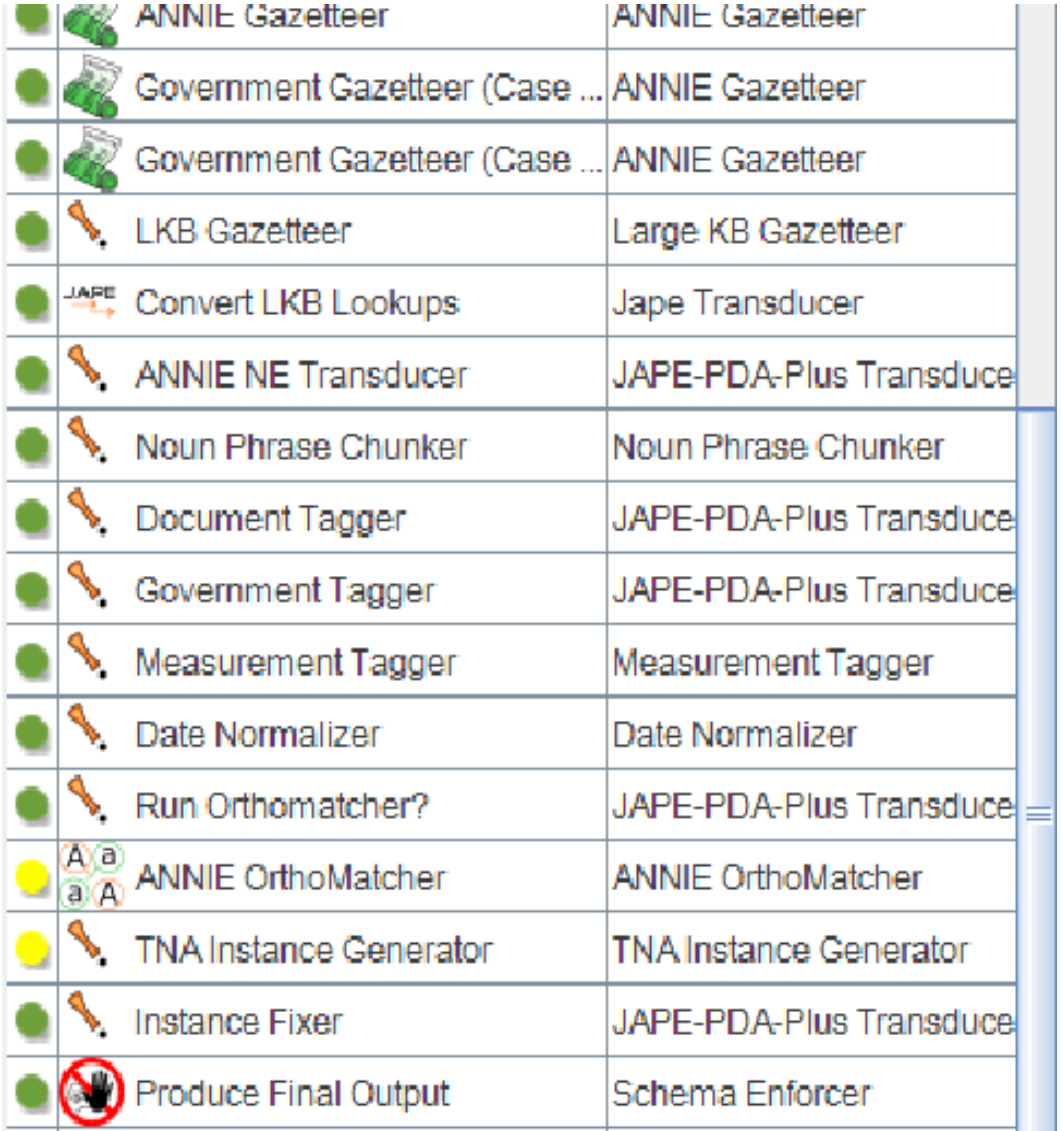
Other uses for conditional processing

- Processing degraded text along with normal text
- For degraded text (e.g. emails, ASR transcriptions) you might want to use some case-insensitive PRs
- Another use is in combination with different kinds of files (HTML, plain text etc) which might require different processing
- More about this later....

Another example

- In one application we developed, we found a problem when running the Orthomatcher (co-reference) on certain texts where there were a lot of annotations of the same type.
- To solve this issue, we first checked to see how many annotations of each were present in a document
- If more than a certain number were present, we added a document feature indicating this
- We then set the orthomatcher to only run on a document which did not contain this feature.

Application



Grammar to check number of annotations

If there are more than 200 annotations of one type, don't run the orthomatcher

```
Rule: CheckAnnotations
({Person}|{Organization}|{Location})
-->
{
AnnotationSet annots = inputAS.get("Person");
if (annots.size() > 200) {
doc.getFeatures().put("runOrthomatcher","false");
return;}
...
doc.getFeatures().put("runOrthomatcher","true");
}
```

Section by Section Processing: the Segment Processing PR

What is it?

- PR which enables you to process labelled sections of a document independently, one at a time
- Useful for
  - very large documents
  - when you want annotations in different sections to be independent of each other
  - when you only want to process certain sections within a document

Processing large documents

- If you have a very large document, for example an entire patient record concatenated into a single GATE document, you may not be able to load it all into memory at once
- One solution is to chop it up into smaller documents and process each one separately, using a datastore to avoid keeping all the documents in memory at once
- But this means you then need to merge all the documents back afterwards
- The Segment Processing PR does this all in one go, by processing each labelled section separately

Processing Sections Independently

- Another problem with large documents can arise when you want to handle each section separately
- You may not want annotations to be co-referenced across sections, for instance if a web page has profiles of different people with similar names
- Using the Segment Processing PR enables you to handle each section separately, without breaking up the document
- It also enables you to use different PRs for each section, using a conditional controller
- For example, some documents may have sections in different languages

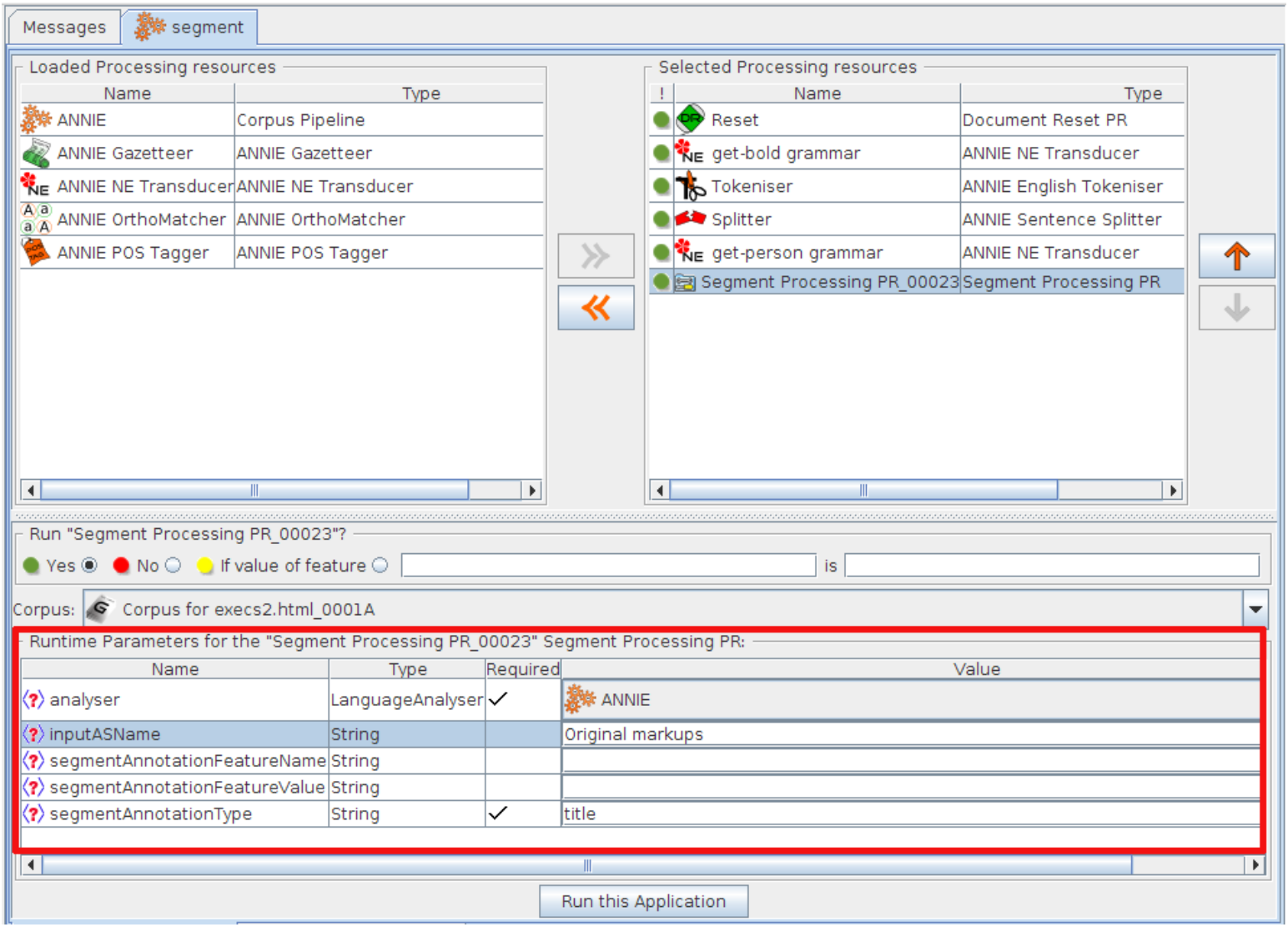
Getting rid of the junk

- Another very common problem is that some documents contain lots of "junk" that you don't want to process, e.g. HTML files contain javascript or contents lists, footers etc.
- There are a number of ways in which you can do this: you may need to experiment to find the best solution for each case
  - **Segment Processing** PR enables you to only process the section(s) you are interested in and ignore the junk
  - Using the **AnnotationSetTransfer** PR, though this works slightly differently
  - Using the **Boilerpipe** PR - this works best for ignoring standard kinds of boilerplate

How does it work?

- The PR is part of the Alignment Plugin
- A new application needs to be created, containing the Segment PR
- The PR then takes as one of its parameters, an instance of the application that you want to run on the document (e.g. ANNIE)
- You can add other PRs before or after the Segment PR, if you want them to run over the whole document rather than the specified section(s)

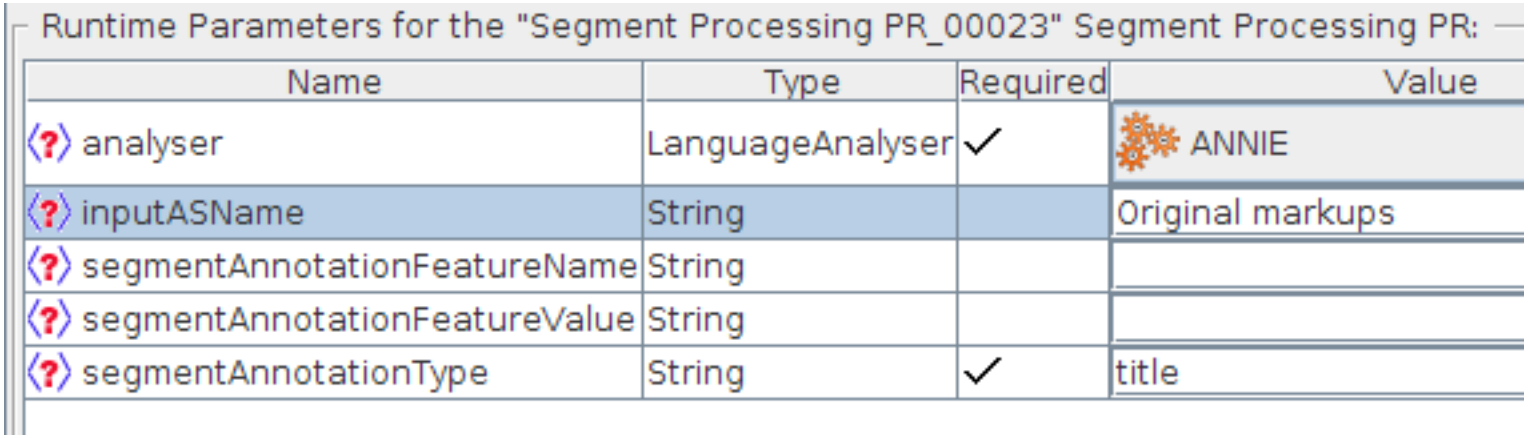
Running ANNIE on a title segment



- Application contains a Segment Processing PR

- Segment Processing PR calls ANNIE application

Segment Processing Parameters



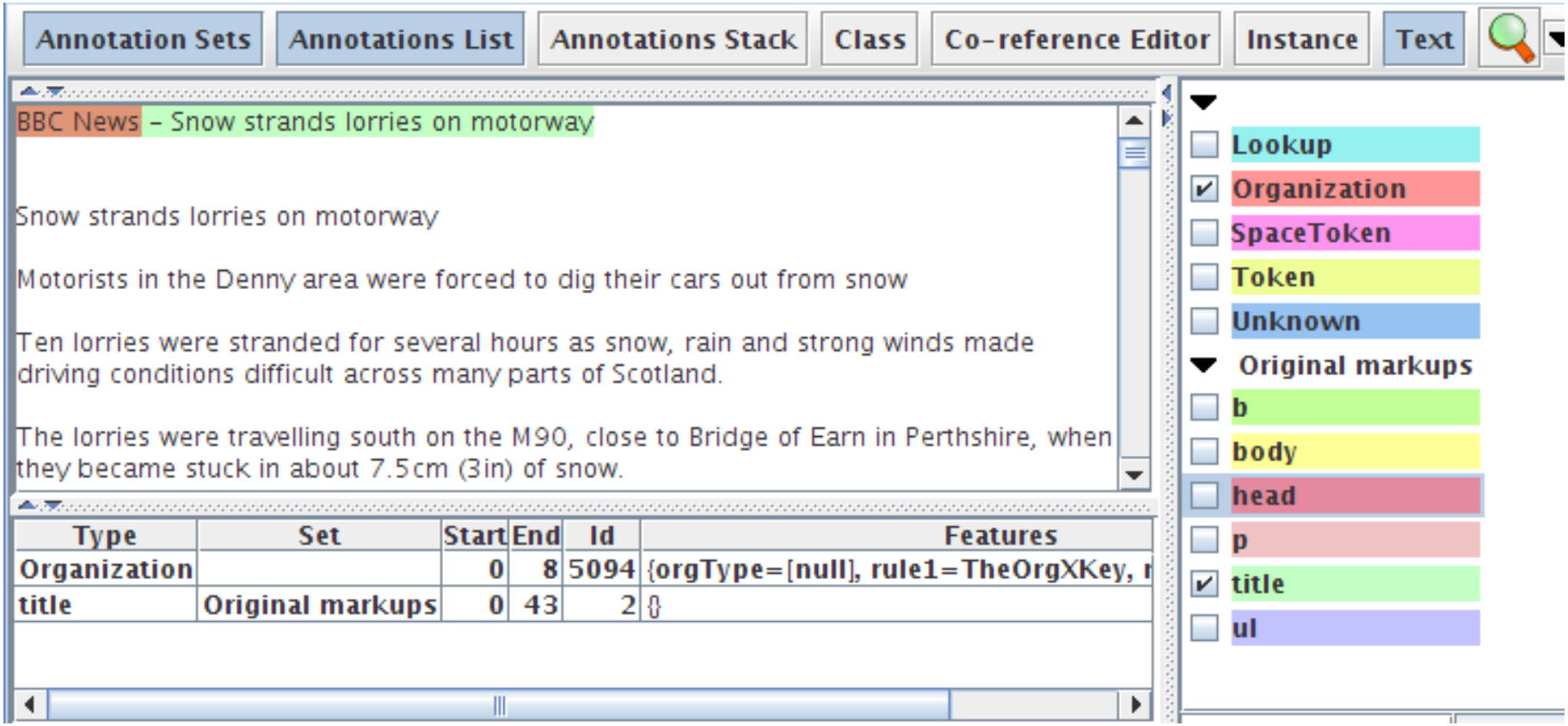
Runtime Parameters for the "Segment Processing PR_00023" Segment Processing PR:

Name	Type	Required	Value
analyser	LanguageAnalyser	✓	ANNIE
inputASName	String		Original markups
segmentAnnotationFeatureName	String		
segmentAnnotationFeatureValue	String		
segmentAnnotationType	String	✓	title

- Segment Processing PR calls the ANNIE application

- ANNIE is set to run only on the text covered by the span of the “title” annotation in the Original markups annotation set

Annotation Result



- Green shading shows the title, which spans the section to be annotated
- The only NE found is the Organization “BBC News” in the title
- Tokens in the rest of the document are not annotated

Using multiple annotation sets

Annotation Set Transfer

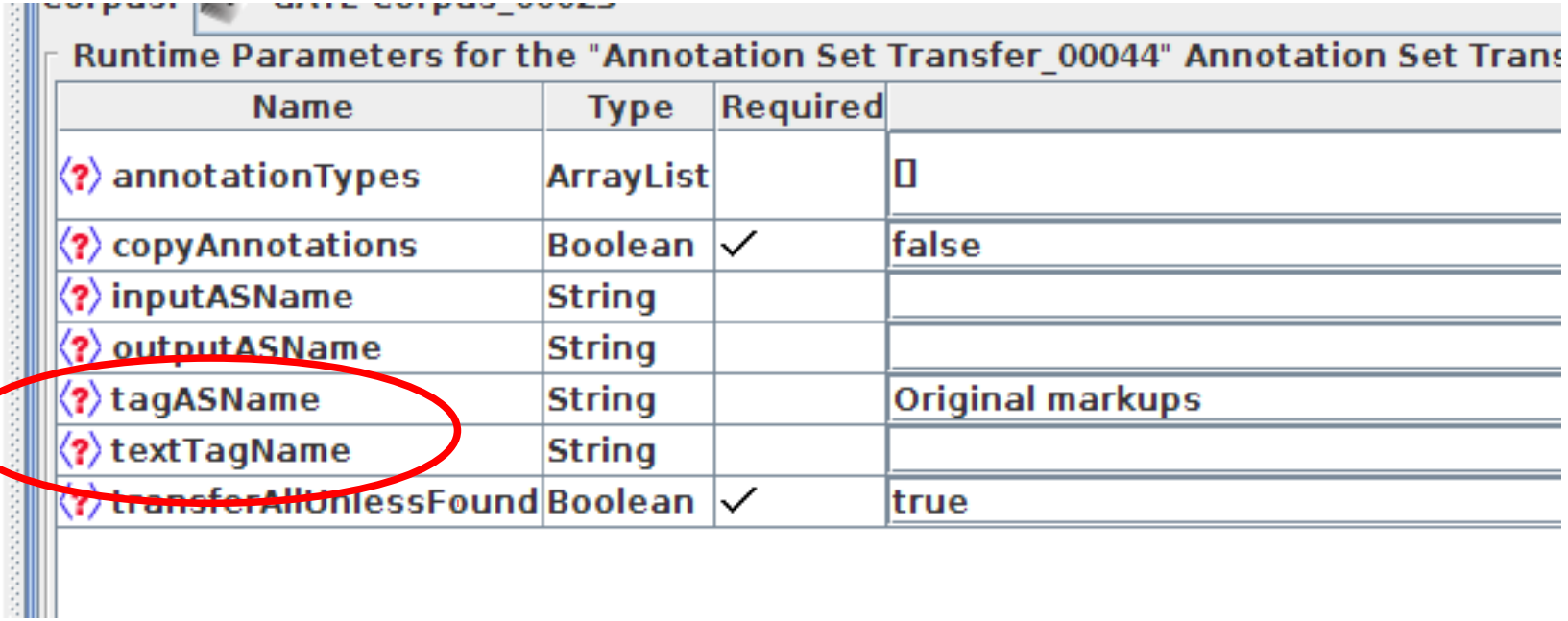
- This PR enables copying or moving annotations from one set to another
- This is useful for backing up annotations before processing, or where a PR wants everything in the same set (ML PRs)
- For example, you might want to move all the gold standard annotations from Default to Key annotation set

Transferring annotations

The annotations remain the same, they're just stored in a different set

Delimiting a section of text

- Another use is to delimit only a certain section of text in which to run further PRs over
- We can move the annotations we want to work on into a different set, and work on that set
- Use the tagASName and textTagName to specify the covering annotation (for example “body” in the “Original markups” annotation set gives the body text of html documents)



Hands-on Exercise

- Objective: move all the annotations from the Default set into the Key set
- Clear GATE of all previous documents, corpora, applications and PRs
- Load document self-shearing-sheep-marked.xml and create an instance of an AST (you may need to load the Tools plugin)
- Have a look at the annotations in the document
- Add the AST to a new application and set the parameters to move all annotations from Default to Key
- Make sure you don't leave the originals in Default!
- Run the application and check the results

Schema Enforcer

- When creating an application, you often end up with lots of annotations and features which are not needed in the final output
- If pushing the output into a MIMIR index, or if storage space is an issue, it's particularly important to get rid of these
- You can tidy up the output using the AnnotationSetTransfer PR to move selected annotation types to a new set, but there's still the problem of the features
- Schema Enforcer PR will ensure that annotations and features will only appear in the final output set if they adhere strictly to the annotation schemas used
- Straightforward to use - load Schema Tools plugin and just list the schemas to be used in the runtime parameters (they must be loaded in GATE already)

Modular Pipelines

- With a normal application (corpus pipeline) you can load other applications as sub-components, as we have seen
- The problem with this is that when you make changes to any of these sub-components and then save your main application, the original application is not saved.
- So if you want to use these sub-components separately, you have to remember to save separately any changes to them.
- The modular pipelines method gets round this by saving the individual applications separately.
- It's not part of GATE, but you can download the plugin from

  https://github.com/johann-petrak/gateplugin-ModularPipelines

  or from the Plugin Repository “Additional Plugins from the GATE Team”

How to use it

- Load the Modular Pipelines plugin
- Create a new Parametrized Corpus Controller from the Applications menu
- Load an application (sub-pipeline) by creating a new Processing Resource of type “Pipeline”
- Select a .gapp file as the value of PipelineFileURL in the loadtime parameters
- This will load the application into GATE
- Add the pipeline to your Parametrized Corpus Controller application
- Add more sub-pipelines or PRs as you wish