# Information Extraction with GATE

**Kalina Bontcheva** (University of Sheffield)
kalina@dcs.shef.ac.uk

**[Work on GATE has been supported by various EPSRC, AHRC, and EU grants]**

These slides: **http://www.dcs.shef.ac.uk/~kalina/gate-course/ie-gate.ppt**



## Structure of today's lecture

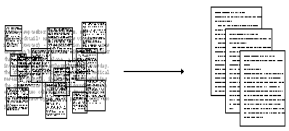
- What is Information Extraction
- Developing and evaluating IE systems
- ANNIE: GATE's IE Components
- ANNIC: Finding patterns in corpora
- GATE Programming API
- Advanced JAPE Rule writing





## IE is not IR

IR pulls **documents** from large text collections (usually the Web) in response to specific keywords or queries. You analyse the **documents**.



IE pulls **facts** and **structured information** from the content of large text collections. You analyse the **facts**.





## IE for Document Access

- With traditional query engines, getting the facts can be hard and slow
  - Where has the Queen visited in the last year?
  - Which places on the East Coast of the US have had cases of West Nile Virus?
- Which search terms would you use to get this kind of information?
- How can you specify you want someone's home page?
- IE returns information in a structured way
- IR returns documents containing the relevant information somewhere (if you're lucky)





## IE as an alternative to IR

- IE returns knowledge at a much deeper level than traditional IR
- Constructing a database through IE and linking it back to the documents can provide a valuable alternative search tool.
- Even if results are not always accurate, they can be valuable if linked back to the original text





## Example System: HaSIE

- Application developed with GATE, which aims to find out how companies report about health and safety information
- Answers questions such as:

"How many members of staff died or had accidents in the last year?"

"Is there anyone responsible for health and safety?"

"What measures have been put in place to improve health and safety in the workplace?"

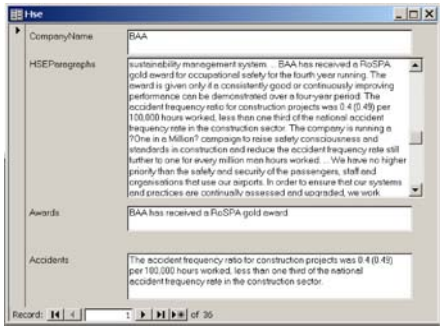
# HASIE

- Identification of such information is too time-consuming and arduous to be done manually
- IR systems can't cope with this because they return whole documents, which could be hundreds of pages
- System identifies relevant sections of each document, pulls out sentences about health and safety issues, and populates a database with relevant information





# HASIE








# Named Entity Recognition: the cornerstone of IE

- Identification of proper names in texts, and their classification into a set of predefined categories of interest
- **Persons**
- **Organisations** (companies, government organisations, committees, etc)
- **Locations** (cities, countries, rivers, etc)
- **Date** and **time** expressions
- Various other types as appropriate





# Why is NE important?

- NE provides a foundation from which to build more complex IE systems
- **Relations** between NEs can provide tracking, ontological information and scenario building
- **Tracking** (co-reference) "Dr Smith", "John Smith", "John", he"
- **Ontologies** "Athens, Georgia" vs "Athens, Greece"





# Two kinds of approaches

**Knowledge Engineering**

- rule based
- developed by experienced language engineers
- make use of human intuition
- require only small amount of training data
- development can be very time consuming
- some changes may be hard to accommodate

**Learning Systems**

- use statistics or other machine learning
- developers do not need LE expertise
- require large amounts of annotated training data
- some changes may require re-annotation of the entire training corpus





# Typical NE pipeline

- **Pre-processing** (tokenisation, sentence splitting, morphological analysis, POS tagging)
- **Entity finding** (gazeteer lookup, NE grammars)
- **Coreference** (alias finding, orthographic coreference etc.)
- **Export** to database / XML

## An Example

Ryanair announced yesterday that it will make Shannon its next European base, expanding its route network to 14 in an investment worth around €180m. The airline says it will deliver 1.3 million passengers in the first year of the agreement, rising to two million by the fifth year.

- **Entities**: Ryanair, Shannon
- **Mentions**: it=Ryanair, The airline=Ryanair, it=the airline
- **Descriptions**: European base
- **Relations**: Shannon base_of Ryanair
- **Events**: investment(€180m)





## Structure of today's lecture

- ✓ What is Information Extraction
- Developing and evaluating IE systems
- ANNIE: GATE's IE Components
- ANNIC: Finding patterns in corpora
- GATE Programming API
- Advanced JAPE Rule writing





## System development cycle

1. Collect corpus of texts
2. Define what is to be extracted
3. Manually annotate gold standard
4. Create system
5. Evaluate performance against gold standard
6. Return to step 3, until desired performance is reached





## Corpora and System Development

- "Gold standard" data created by manual annotation
- Corpora are divided typically into a training, sometimes testing, and unseen evaluation portion
- Rules and/or ML algorithms developed on the training part
- Tuned on the testing portion in order to optimise
  - Rule priorities, rules effectiveness, etc.
  - Parameters of the learning algorithm and the features used
- Evaluation set – the best system configuration is run on this data and the system performance is obtained
- No further tuning once evaluation set is used!





## Annotation Guidelines

- People need clear definition of what to annotate in the documents, with examples
- Typically written as a guidelines document
- Piloted first with few annotators, improved, then "real" annotation starts, when all annotators are trained
- Annotation tools require the definition of a formal DTD (e.g. XML schema)
  - What annotation types are allowed
  - What are their attributes/features and their values
  - Optional vs obligatory; default values





## Annotation Schemas (1)

```
<?xml version="1.0"?>
<schema
   xmlns="http://www.w3.org/2000/10/XMLSchema">
   <!-- XSchema deffinition for email-->
   <element name="Email" />
</schema>
```

# Annotation Schemas (2)

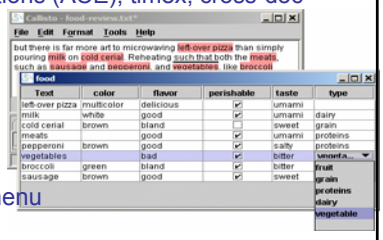
```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2000/10/XMLSchema">
    <!-- XSchema deffinition for token-->
    <element name="Address">
      <complexType>
        <attribute name="kind"  use="optional">
        <simpleType>
          <restriction base="string">
              <enumeration value="email"/>
              <enumeration value="url"/>
              <enumeration value="phone"/>
              <enumeration value="ip"/>
              <enumeration value="street"/>
              <enumeration value="postcode"/>
              <enumeration value="country"/>
              <enumeration value="complete"/>
                  </restriction> ...
```





# Callisto (callisto.mitre.org)

- Has specialised plugins for different annotation tasks:
  - NE tagging, relations (ACE), timex, cross-doc coref,...
- Attributes:
  - boolean
  - list of values
  - free text/values
  - defined in XML
- Right-click action menu







http://mmax2.sourceforge.net/

Figure 1: MMAX2 Main Window

- Handles(brackets) mark annotation boundaries
- Clicking on handles highlights the relevant part and shows their attributes
- Right click shows a list of actions, such as deletion of markables





Showing attributes in MMAX2

Figure 3: MMAX2 Attribute Window

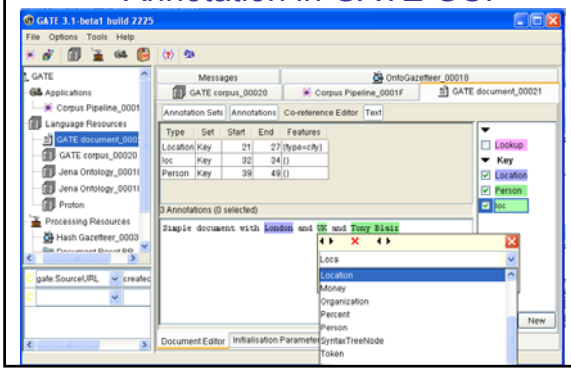
- Attributes with several possible values are shown as radio buttons/check boxes
- When there are more values – drop-down menu

- Creating new Annotations
  - Select then with mouse by holding and dragging
  - A popup appears with items “Create X” for each annotation type X defined in the annotation schema
  - All default values of attributes (defined in schema) will be set





# Annotate Gold Standard – Manual Annotation in GATE GUI





# Annotation in GATE GUI

- Adding annotation sets
- Adding annotations
- Resizing them (changing boundaries)
- Deleting
- Changing highlighting colour
- Setting features and their values

## Hands-On Exercise (4)

- Create Key annotation set
  - Type Key in the bottom of annotation set view and press the New button
- Select it in the annotation set view
- Annotate all email addresses with Email annotations in the Key set
- Save the resulting document as xml





## Performance Evaluation

**2 main requirements:**

- **Evaluation metric**: mathematically defines how to measure the system's performance against human-annotated gold standard
- **Scoring program**: implements the metric and provides performance measures
  - For each document and over the entire corpus
  - For each type of annotation





## Evaluation Metrics

- Most common are Precision and Recall
- **Precision** = correct answers/answers produced
- **Recall** = correct answers/total possible correct answers
- Trade-off between precision and recall
- **F1** (balanced) Measure = 2PR / 2(R + P)
- Some tasks sometimes use other metrics, e.g. cost-based (good for application-specific adjustment)





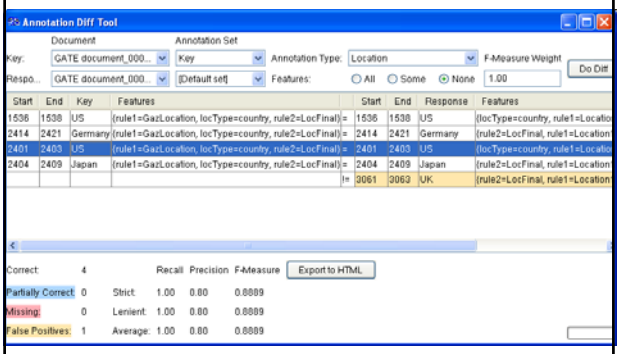
## AnnotationDiff

- Graphical comparison of 2 sets of annotations
- Visual diff representation, like tkdiff
- Compares one document at a time, one annotation type at a time
- Gives scores for precision, recall, F_measure etc.





## Annotation Diff








## Hands On Exercise (5)

- Open the annotation diff (Tools menu)
- Select the email document
- Key must contain the manual annotations
- Response: the set containing those created by our email rule
- Check precision and response
- See the errors

# Corpus Benchmark Tool

- Compares annotations at the corpus level
- Compares all annotation types at the same time, i.e. gives an overall score, as well as a score for each annotation type
- Enables regression testing, i.e. comparison of 2 different versions against gold standard
- Visual display, can be exported to HTML
- Granularity of results: user can decide how much information to display
- Results in terms of Precision, Recall, F-measure





# Corpus structure

- Corpus benchmark tool requires particular directory structure
- Each corpus must have a **clean** and **marked** sub-directory
- **Clean** holds the unannotated version, while **marked** holds the marked (gold standard) ones
- There may also be a **processed** subdirectory – this is a datastore (unlike the other two)
- Corresponding files in each subdirectory must have the same name





# How it works

- Clean, marked, and processed
- Corpus_tool.properties – must be in the directory where build.xml is
- Specifies configuration information about
  - What annotation types are to be evaluated
  - Threshold below which to print out debug info
  - Input set name and key set name
- Modes
  - Storing results for later use
  - Human marked against already stored, processed
  - Human marked against current processing results
  - Regression testing – default mode





# Corpus Benchmark Tool

View Attachment: temp.html

**ABC19980430.1830.0858.sgm**

| Annotation Type | Precision | Recall | Annotations |
|---|---|---|---|
| Annotation type: Organization | 1.0<br>Precision increase on human-marked from 0.75 to 1.0 | 0.75<br>Recall increase on human-marked from 0.375 to 0.75 | MISSING ANNOTATIONS in the automatic texts: **ABC**: *[2849,2852]*<br>SPURIOUS ANNOTATIONS in the automatic texts:<br>PARTIALLY CORRECT ANNOTATIONS in the automatic texts: |
| Annotation type: Person | 0.9444444444444444<br>Precision increase on human-marked from 0.8947368421052632 to 0.9444444444444444 | 0.9444444444444444 | |
| Annotation type: GPE | 1.0 | 1.0<br>Recall increase on human-marked from 0.8571428571428571 to 1.0 | |





# Demonstration

- Corpus Benchmark Tool





# Structure of today's lecture

- ✓ What is Information Extraction (IE)?
- ✓ Developing and evaluating IE systems
- ANNIE: GATE's IE Components
- ANNIC: Finding patterns in corpora
- GATE Programming API
- Advanced JAPE Rule writing

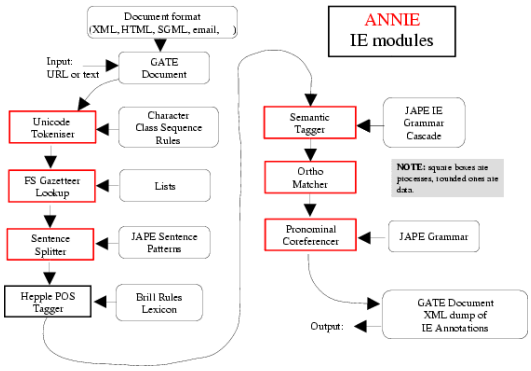
# What is ANNIE?

- ANNIE is a vanilla information extraction system comprising a set of core PRs:
  - Tokeniser
  - Sentence Splitter
  - POS tagger
  - Gazetteers
  - Named entity tagger (JAPE transducer)
  - Orthomatcher (orthographic coreference)





# Core ANNIE Components








# Creating a new application from ANNIE

- Typically a new application will use most of the core components from ANNIE
- The tokeniser, sentence splitter and orthomatcher are basically language, domain and application-independent
- The POS tagger is language dependent but domain and application-independent
- The gazetteer lists and JAPE grammars may act as a starting point but will almost certainly need to be modified
- You may also require additional PRs (either existing or new ones)





# DEMO of ANNIE and GATE GUI

- Loading ANNIE
- Creating a corpus
- Loading documents
- Running ANNIE on corpus





# Unicode Tokeniser

- Bases tokenisation on Unicode character classes
- Language-independent tokenisation
- Declarative token specification language, e.g.:

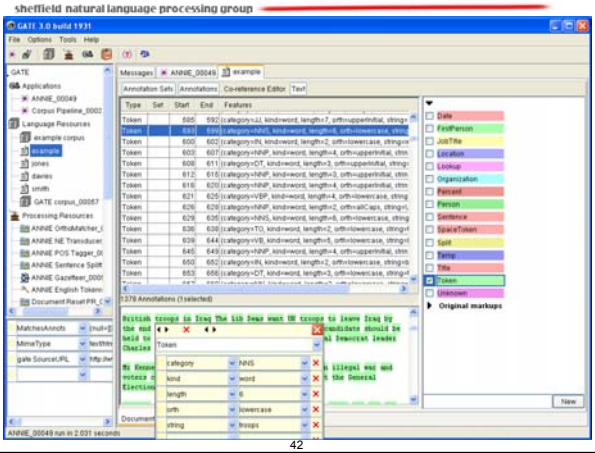
```
"UPPERCASE_LETTER" LOWERCASE_LETTER"* >

Token; orthography=upperInitial; kind=word
```

Look at the ANNIE English tokeniser and at tokenisers for other languages (in plugins directory) for more information and examples





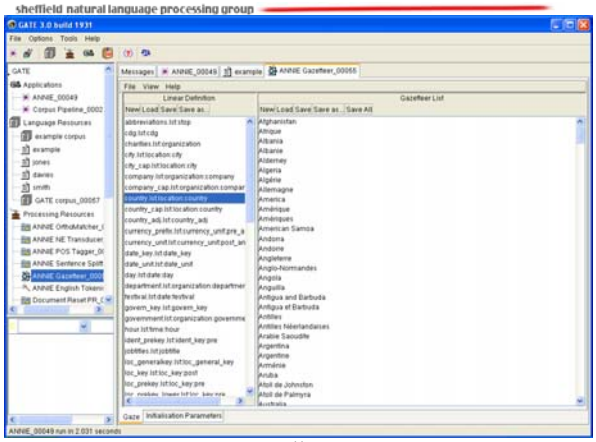


# Gazetteers

- Gazetteers are plain text files containing lists of names (e.g rivers, cities, people, …)
- Information used by JAPE rules
- Each gazetteer set has an index file listing all the lists, plus features of each list (majorType, minorType and language)
- Lists can be modified either internally using Gaze, or externally in your favourite editor
- Generates Lookup results of the given kind

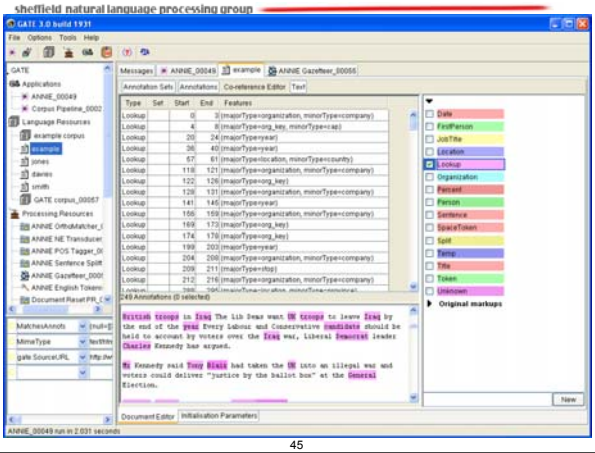
















# ANNIE's Gazetteer Lists

- Set of lists compiled into Finite State Machines
- 60k entries in 80 types, inc.:
  `organization; artifact; location; amount_unit; manufacturer; transport_means; company_designator; currency_unit; date; government_designator; ...`
- Each list has attributes MajorType and MinorType and Language):
  `city.lst: location: city: english`
  `currency_prefix.lst: currency_unit: pre_amount`
  `currency_unit.lst: currency_unit: post_amount`
- List entries may be entities or parts of entities, or they may contain contextual information (e.g. job titles often indicate people)





# ANNIE's NE grammars

- The named entity tagger consists of a set of rule-based JAPE grammars run sequentially
- Previous lecture introduced basic JAPE rule writing
- More complex rules can also be created





# NE Rule in JAPE

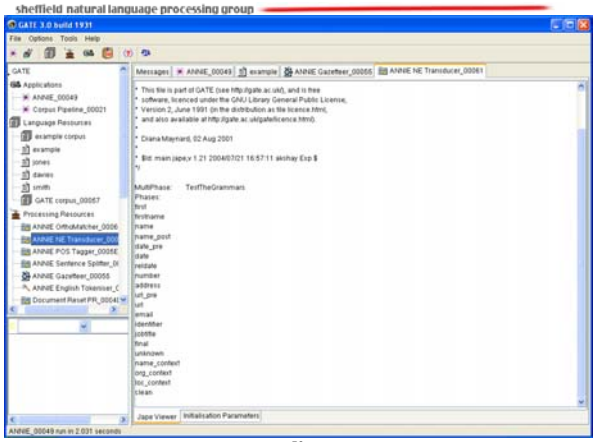
```
Rule: Company1
 Priority: 25
  (
    ( {Token.orthography == upperInitial} )+ //from tokeniser
    {Lookup.kind == companyDesignator} //from gazetteer lists
  ):match
 -->
    :match.NamedEntity = { kind=company, rule="Company1" }
```

# Using phases

- Grammars usually consist of several phases, run sequentially
- Only one rule within a single phase can fire
- Temporary annotations may be created in early phases and used as input for later phases
- Annotations from earlier phases may need to be combined or modified
- A definition phase (conventionally called main.jape) lists the phases to be used, in order
- Only the definition phase needs to be loaded











# Advanced JAPE

- Coming later in the lecture





# JAPE Hints and Tricks

- JAPE is quite limited in some respects as to what can be done
  - There is no negative operator
  - It can be slow if it is badly written, e.g. ({Token})*
  - Context is consumed, which can make rule-writing awkward
  - Priority can be difficult to set correctly
- But fear not, there is generally a sneaky way around it…..





# How to prevent a pattern from matching

Rule: disablePattern

Priority: 1000

(<pattern>)

→

{}

- Instead of having a negative operator, we can simply put a high priority rule which does nothing when fired.
- This will be preferred to a lower priority rule which performs the action intended, i.e. only in the case when the former pattern doesn't apply.





# How to play with input annotations

Input: Person Organisation VerbWork Split

…

Rule: RelationWorkIn

({Person} {VerbWork} {Organisation}):relation

→ :relation.WorksFor={}

- Use existing annotations to find relations
- We ignore Tokens to enable more flexibility, i.e. there could be additional words between the annotations specified
- Split ensures we don't cross sentence boundaries

## How to deal with overlapping annotations

- Because matched annotations are consumed, when two annotations overlap (e.g. in gazetteer lists), the second one will never be matched.
- E.g. for the string “hALCAM” with Lookups hAL, ALCAM, and CAM, ALCAM will never be matched
- Solution is to delete the annotations once matched (needs use of Java on RHS – example later), and then rerun the same grammar phase over the text
- The process may need to be repeated several times (determine by trial and error)





## More examples

- In the GATE User Guide under the section “Useful tricks with JAPE”
- Look in the ANNIE grammars and in the foreign language grammars – there are many examples of little tricks
- Check the GATE mailing list archives





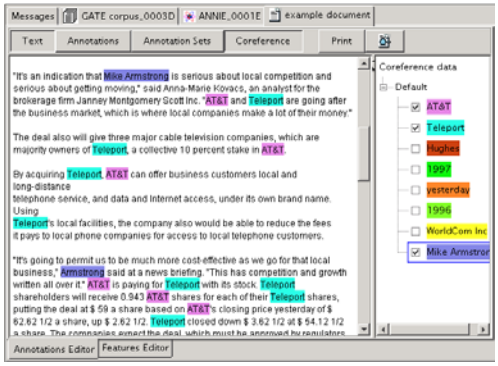
## Using co-reference

- Orthographic co-reference module matches proper names in a document
- Improves results by assigning entity type to previously unclassified names, based on relations with classified entities
- May not reclassify already classified entities
- Classification of unknown entities very useful for surnames which match a full name, or abbreviations, e.g. `[Bonfield]` will match `[Sir Peter Bonfield]`; `[International Business Machines Ltd.]` will match `[IBM]`





## Named Entity Coreference








## Hands-On Exercise (6)

- Load all ANNIE PRs you think are useful
- Order them into a corpus pipeline
- Run it on the populated corpus of newspaper docs (docs-to-populate-corpus directory)
- See if you get results you want
- File/Load ANNIE with defaults
- Compare your IE pipeline against default one, did you get them right?





## Structure of today’s lecture

- ✓ What is Information Extraction (IE)?
- ✓ Developing and evaluating IE systems
- ✓ ANNIE: GATE’s IE Components
- ANNIC: Finding patterns in corpora
- GATE Programming API
- Advanced JAPE Rule writing

# Identifying patterns in corpora

- ANNIC – ANNotations In Context
- Provides a keyword-in-context-like interface for identifying annotation patterns in corpora
- Uses JAPE LHS syntax, except that + and * need to be quantified
- e.g. *{Person}{Token}*3{Organisation}* – find all Person and Organisation annotations within up to 3 tokens of each other
- To use, pre-process the corpus with ANNIE or your own components, then query it via the GUI





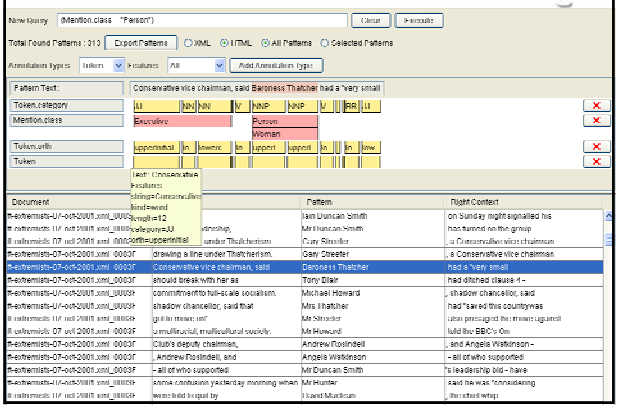
# ANNIC – ANNotations In Context

- Inspired from KWIC (Key-word In Context)
- Useful as a corpus analysis tool
- More powerful than standard IR
- Based on GATE + Lucene
- Can be used for authoring of IE patterns for rules





# ANNIC Demo





# ANNIC Demo

- Formulating queries
- Finding matches in the corpus
- Analysing the contexts
- Refining the queries
- Demo: http://gate.ac.uk/demos/movies.html#annic





# Hands-On Exercise (7)





# Structure of today's lecture

- ✓ What is Information Extraction (IE)?
- ✓ Developing and evaluating IE systems
- ✓ ANNIE: GATE's IE Components
- ✓ ANNIC: Finding patterns in corpora
- GATE Document and Annotation API
- Advanced JAPE Rule writing

| Content Manipulation | |
|---|---|
| Method | Purpose |
| DocumentContent getContent() | Gets the Document content. |
| void edit(Long start, Long end, DocumentContent replacement) | Modifies the Document content. |
| void setContent(DocumentContent newContent) | Replaces the entire content. |
| Annotations Manipulation | |
| Method | Purpose |
| public AnnotationSet getAnnotations() | Returns the default annotation set. |
| public AnnotationSet getAnnotations(String name) | Returns a *named* annotation set. |
| public Map getNamedAnnotationSets() | Returns *all* the named annotation sets. |
| void removeAnnotationSet(String name) | Removes a named annotation set. |
| Input Output | |
| String toXml() | Serialises the Document in XML format. |
| String toXml(Set aSourceAnnotationSet, boolean includeFeatures) | Generates XML from a set of annotations only, trying to preserve the original format of the file used to create the document. |

Document API



# Annotation Sets

They are Java objects, represented using the Set paradigm as defined by the Java collections library and they are hence named annotation sets

*How to iterate from left to right over all annotations of a given type?*

```
AnnotationSet annSet = ...;
String type = "Email";
//Get all email annotations
AnnotationSet emailSet = annSet.get(type);
//Sort the annotations
List emailList = new ArrayList(emailSet);
Collections.sort(emailList,
  new gate.util.OffsetComparator());
//Iterate
Iterator emailIter = emailList.iterator();
while(emailIter.hasNext()){
...
}
```





# AnnotationSet methods

| Annotations Manipulation | |
|---|---|
| Method | Purpose |
| Integer add(Long start, Long end, String type, FeatureMap features) | Creates a new annotation between two offsets, adds it to this set and returns its id. |
| Integer add(Node start, Node end, String type, FeatureMap features) | Creates a new annotation between two nodes, adds it to this set and returns its id. |
| boolean remove(Object o) | Removes an annotation from this set. |
| Nodes | |
| Method | Purpose |
| Node firstNode() | Gets the node with the smallest offset. |
| Node lastNode() | Gets the node with the largest offset. |
| Node nextNode(Node node) | Get the first node that is relevant for this annotation set and which has the offset larger than the one of the node provided. |
| Set implementation | |
| Iterator iterator() | |
| int size() | |



# AnnotationSet Methods (2)

| Searching | |
|---|---|
| AnnotationSet get(Long offset) | Select annotations by offset. This returns the set of annotations whose start node is the least such that it is less than or equal to offset. If a positional index doesn't exist it is created. If there are no nodes at or beyond the offset parameter then it will return null. |
| AnnotationSet get(Long startOffset, Long endOffset) | Select annotations by offset. This returns the set of annotations that overlap totally or partially with the interval defined by the two provided offsets. The result will include all the annotations that either:<br>• start before the start offset and end strictly after it<br>• start at a position between the start and the end offsets |
| AnnotationSet get(String type) | Returns all annotations of the specified type. |
| AnnotationSet get(Set types) | Returns all annotations of the specified types. |
| AnnotationSet get(String type, FeatureMap constraints) | Selects annotations by type and features. |
| Set getAllTypes() | Gets a set of java.lang.String objects representing all the annotation types present in this annotation set. |



# Gate.Annotation

- Always created via the appropriate annotation set

```
public interface SimpleAnnotation
extends FeatureBearer, IdBearer, Comparable, Serializable {

public String getType(); /** The type of the annotation */

public Node getStartNode(); /** The start node. */

public Node getEndNode();  /** The end node. */

  /** Ordering */
  public int compareTo(Object o) throws ClassCastException;

} // interface SimpleAnnotation
```





# Corpora

- `gate.corpora.CorpusImpl` used for transient corpora.
- `gate.corpora.SerialCorpusImpl` used for persistent corpora that are stored in a serial datastore
- Methods to list document names

```
Corpus corpus=Factory.newCorpus("A Corpus");
File directory = ...;
URL url = directory.toURL();
java.io.FileFilter filter = new
  gate.util.ExtensionFileFilter();
filter.addExtension("xml");
corpus.populate(url, filter, null, false);
```

Encoding

| Method | Purpose |
|---|---|
| void setParameterValue(String paramaterName, Object parameterValue) | Sets the value for a specified parameter. method inherited from gate.Resource |
| void setParameterValues(FeatureMap parameters) | Sets the values for more parameters in one step. method inherited from gate.Resource |
| Object getParameterValue(String paramaterName) | Gets the value of a named parameter of this resource. method inherited from gate.Resource |
| Resource init() | Initialise this resource, and return it. method inherited from gate.Resource |
| void reInit() | Reinitialises the processing resource. After calling this method the resource should be in the state it is after calling init. If the resource depends on external resources (such as rules files) then the resource will re-read those resources. If the data used to create the resource has changed since the resource has been created then the resource will change too after calling reInit(). |
| void execute() | Starts the execution of this Processing Resource. |
| void interrupt() | Notifies this PR that it should stop its execution as soon as possible. |
| boolean isInterrupted() | Checks whether this PR has been interrupted since the last time its Executable.execute() method was called. |



## Structure of today's lecture

- ✓ What is Information Extraction (IE)?
- ✓ Developing and evaluating IE systems
- ✓ ANNIE: GATE's IE Components
- ✓ ANNIC: Finding patterns in corpora
- ✓ GATE Programming API
- Advanced JAPE Rule writing





## Overview

- You can write any Java code on JAPE RHS
- Learn well the annotations and their features
  - By inspecting them in the GUI
  - Looking in the documentation
- Learn well the annotation API
- The better you know the API, the more you can do with JAPE





## RHSAction.java

Interface that is implemented by all RHS JAPE rules

```
public void doit(Document doc, Map bindings, AnnotationSet annotations,
        AnnotationSet inputAS, AnnotationSet outputAS,
        Ontology ontology)
    throws JapeException;
```





## Back to the example

```
Rule: Entity
    (
     {Gpe}|
     {Organization}|
     {Person}|
     {Location}|
     {Facility}
):entity
    -->
    {
    gate.AnnotationSet entityAS =
            (gate.AnnotationSet)bindings.get("entity");
    gate.Annotation entityAnn = (gate.Annotation)entityAS.iterator().next();
    gate.FeatureMap features = Factory.newFeatureMap();
    features.put("type", entityAnn.getType());
    outputAS.add(entityAnn.getStartNode(), entityAnn.getEndNode(),
            "Entity", features);
    inputAS.removeAll(entityAS);
}
```





## Accessing the document

```
Rule: Email
Priority: 150
(
   {message}
) -->
 {
   doc.getFeatures().put("genre", "email");
 }
```

# Transforming markup in corpora

**Example document:**

The <I-ORG>European Commission</I-ORG> said on Thursday it disagreed with <I-LOC>German</I-LOC> advice to consumers to shun <I-LOC>British</I-LOC> lamb …





```
Phase: foo
Input: I-PER I-ORG I-LOC
Options: control = first

Rule: One
({I-PER}):iperx
-->
:iperx.Person={}

Rule: Two
({I-ORG}):iorgx
-->
:iorgx.Organization={}

Rule: Three
({I-LOC}):ilocx
-->
:ilocx.Location={}
```





# More on the API

- Using GATE as a library
- Stand-alone use of gate, integration into web services, Protégé plugins, etc.
- To be covered in “Advanced GATE” lecture





# Next Lecture

- Advanced GATE, incl. batch use, multilingual tools, etc.
- Thanks!
- Any more questions?





# Lifecycle of a CREOLE Resources

- A resource lives in a .jar file and has a creole.xml metadata file with it. These files are loaded via URLs.
- When GATE initialises (or if you ask to load a new plugin) it reads the creole.xml file, including the name of the .jar.
- `gate.CreoleRegister` holds all the data about resources, in ResourceData objects.
- Resources are beans: so no params in constructors; instead use properties. These are specified in creole.xml.
  - For PRs some are "runtime" - not specified at creation time.
- To create a resource (instantiate the class) we use `Factory.createResource`, which applies the parameters, calls the default constructor and records the presence of the instance in the register.
- To make a resource available to be garbage-collected `Factory.deleteResource()`.





# Sample creole.xml

```
<!-- creole.xml for the Unicode tokeniser -->
<RESOURCE>
    <NAME>GATE Unicode Tokeniser</NAME>
    <CLASS>gate.creole.tokeniser.SimpleTokeniser</CLASS>
    <PARAMETER NAME="document“ RUNTIME="true"> gate.Document
    </PARAMETER>
    <PARAMETER NAME="annotationSetName" RUNTIME="true“
        OPTIONAL="true"> java.lang.String
    </PARAMETER>
    <PARAMETER
        DEFAULT="resources/tokeniser/DefaultTokeniser.rules“
        SUFFIXES="rules“ NAME="rulesURL">
        java.net.URL
    </PARAMETER>
    <PARAMETER DEFAULT="UTF-8“ NAME="encoding">
        java.lang.String
    </PARAMETER>
    <ICON>tokeniser</ICON>
</RESOURCE>
```

## GATE's Factory Class

- A programmer using GATE should never call the constructor of a resource:
  - always use the Factory class, also to delete resource!
- Creating a resource involves providing the following information:
  - fully qualified class name for the resource. This is the only required value. For all the rest, defaults will be used if actual values are not provided.
  - values for the initialisation time parameters.
  - initial values for resource features
  - a name for the new resource;





## Creating documents

```
URL u = new
  URL("http://gate.ac.uk/");
FeatureMap params =
  Factory.newFeatureMap();
params.put("sourceUrl", u);
FeatureMap features =
  Factory.newFeatureMap();
Document doc = (Document)
Factory.createResource("gate.corpora.
  DocumentImpl", params,
 features, "GATE Homepage");
```





## Other Factory Methods

- `newFeatureMap()` creates a new Feature Map (as used in the example above).
- `newDocument(String content)` creates a new GATE Document starting from a String value that will be used to generate the document content.
- `newDocument(URL sourceUrl)` creates a new GATE Document using the text pointed by an URL to generate the document content.
- `newDocument(URL sourceUrl, String encoding)` same as above but allows the specification of an encoding to be used while downloading the document content.
- `newCorpus(String name)` creates a new GATE Corpus with a specified name.





## Applications

```
// create a serial analyser controller to run tokeniser with
SerialAnalyserController theController =
   (SerialAnalyserController) Factory.createResource(
   "gate.creole.SerialAnalyserController",
    Factory.newFeatureMap(), Factory.newFeatureMap(), "TokApp");
// load tokeniser using default parameters
FeatureMap params = Factory.newFeatureMap();
ProcessingResource pr = (ProcessingResource)
   Factory.createResource("gate.creole.tokeniser.DefaultTokeniser", params);
// add the PR to the pipeline controller
theController.add(pr);
// Tell the controller about the corpus you want to run on
Corpus corpus = ...;
theController.setCorpus(corpus);
// Run it
theController.execute();
```





## Structure of today's lecture

- √ GATE After 10 years – Hamish Cunningham
- √ GATE Architecture
- √ GATE GUI
- √ JAPE: GATE's rule writing engine
- √ Performance evaluation
- Corpus processing
- GATE Programming API





## Datastores

- Creation
- Saving documents and corpora
- Running applications on persistent corpora

# Hands-On Exercise (3)

- Datastores
  - Create a datastore
  - Save the corpus there
  - Close all documents
  - Set the corpus as parameter to application
  - Re-run the application again, to see GATE automatically loading and unloading the documents