

# Advanced GATE Embedded

## Track II, Module 8

Second GATE Training Course  
May 2010

# Outline

- 1** GATE and UIMA
  - Introduction to UIMA
  - UIMA and GATE compared
  - Integrating GATE and UIMA
- 2** GATE in Web Applications
  - Introduction
  - Multi-threading and GATE
  - Servlet Example
  - The Spring Framework
- 3** GATE and Groovy
  - Introduction to Groovy
  - Scripting GATE Developer
  - The Groovy Script PR
  - Writing GATE Resource Classes in Groovy

# Outline

- 1** GATE and UIMA
  - Introduction to UIMA
  - UIMA and GATE compared
  - Integrating GATE and UIMA
- 2** GATE in Web Applications
  - Introduction
  - Multi-threading and GATE
  - Servlet Example
  - The Spring Framework
- 3** GATE and Groovy
  - Introduction to Groovy
  - Scripting GATE Developer
  - The Groovy Script PR
  - Writing GATE Resource Classes in Groovy

## What is UIMA?

- Language processing framework originally developed by IBM
- Similar document processing pipeline architecture to GATE
- Concentrates on performance and scalability
- Supports components written in different programming languages (currently Java and C++)
- Native support for distributed processing via web services

## UIMA Terminology

- Processing tasks in UIMA are encapsulated in *Analysis Engines* (AEs)
- In UIMA, AEs can be *primitive* (~ a single PR in GATE terms), or *aggregate* (~ a GATE controller).
  - Aggregate AE can include other primitive or aggregate AEs
- GATE includes interoperability layer to run
  - GATE controller as a (primitive) AE in UIMA
  - UIMA AE (primitive or aggregate) as a GATE PR

# UIMA and GATE

- In GATE, unit of processing is the *Document*
  - Text, plus features, plus annotations
  - Annotations can have arbitrary features, with any Java object as value
- In UIMA, unit of processing is *CAS* (common analysis structure)
  - Text, plus *Feature Structures*
  - Annotations are just a special kind of FS, which includes start and end offset features

## Key Differences

- In GATE, annotations can have any features, with any values
- In UIMA, feature structures are *strongly typed*
  - Must declare what types of annotations are supported by each analysis engine
  - Must specify what features each annotation type supports
  - Must specify what *type* feature values may take
    - Primitive types - string, integer, float
    - Reference types - reference to another FS in the CAS
    - Arrays of the above
  - All defined in XML descriptor for the AE

# Integrating GATE and UIMA

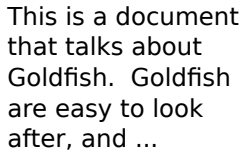
- So the problem is to map between the loosely-typed GATE world and the strongly-typed UIMA world
- Best explained by example. . .

## Example 1

- Simple UIMA annotator that annotates each instance of the word “Goldfish” in a document.
- Does not need any input annotations
- Produces output annotations of type `gate.example.Goldfish`

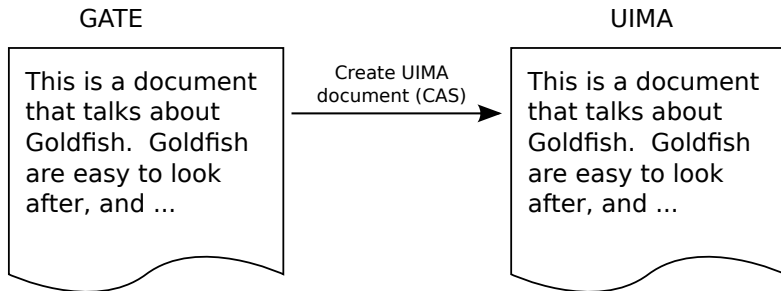
## Example 1

### GATE



This is a document  
that talks about  
Goldfish. Goldfish  
are easy to look  
after, and ...

## Example 1



## Example 1

### GATE

This is a document  
that talks about  
Goldfish. Goldfish  
are easy to look  
after, and ...

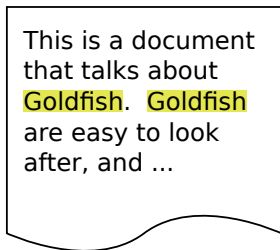
### UIMA

This is a document  
that talks about  
**Goldfish**. **Goldfish**  
are easy to look  
after, and ...

UIMA AE runs, creating  
gate.example.Goldfish  
annotations

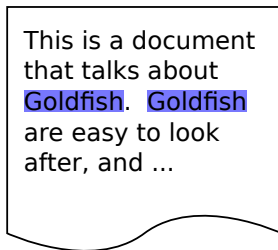
## Example 1

GATE



Create GATE annotations of type Goldfish at the corresponding places

UIMA



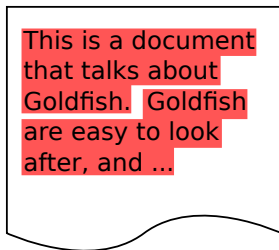
← Copy annotations back

## Example 2

- We may want to copy annotations, as well as text, from the original GATE document.
- Consider a UIMA annotator that
  - takes `gate.example.Sentence` annotations as input
  - annotates “Goldfish” as before
  - also adds a feature `GoldfishCount` to each `Sentence` giving the number of goldfish annotations in that sentence

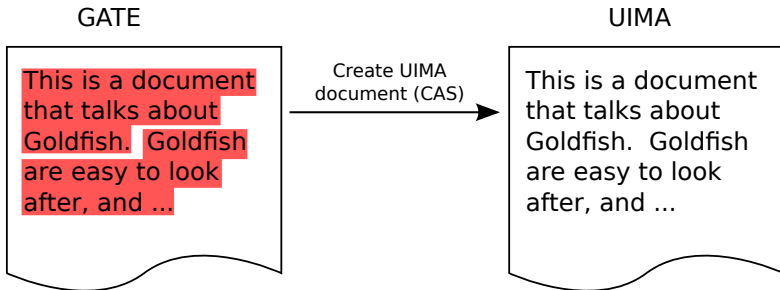
## Example 2

### GATE

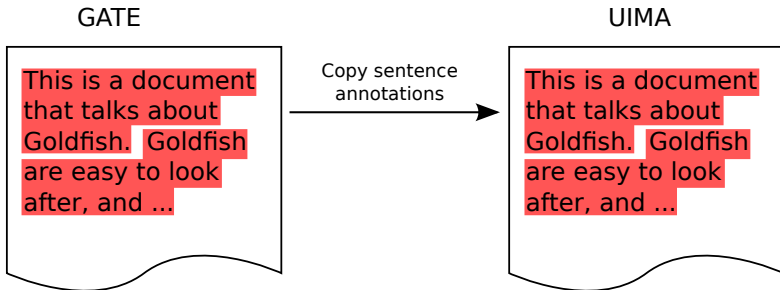


GATE document containing  
Sentence annotations

## Example 2

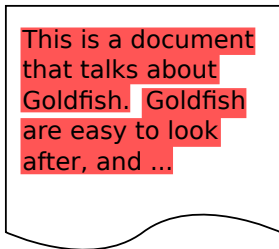


## Example 2

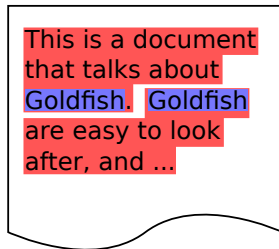


## Example 2

### GATE



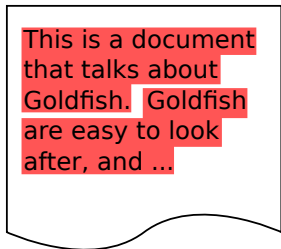
### UIMA



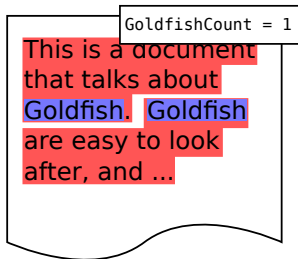
UIMA AE runs, creating  
gate.example.Goldfish  
annotations

## Example 2

### GATE

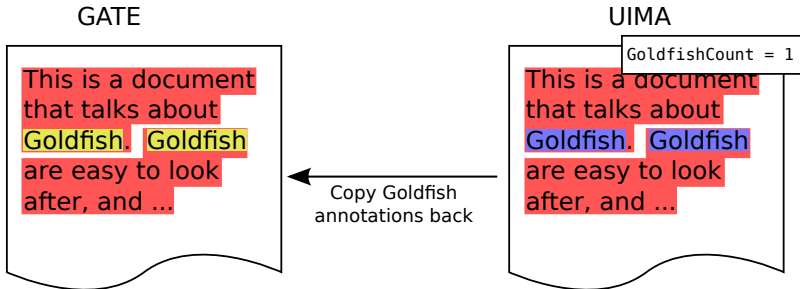


### UIMA

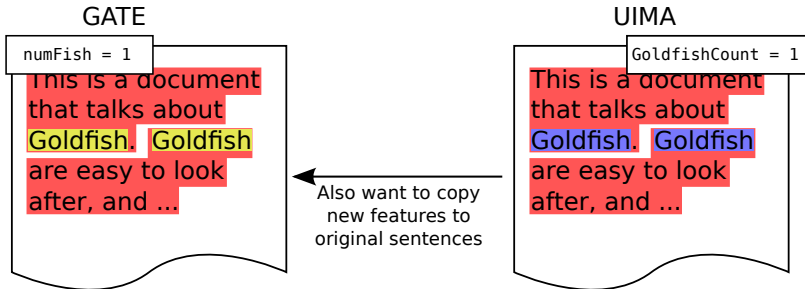


and adding a feature to each sentence

## Example 2

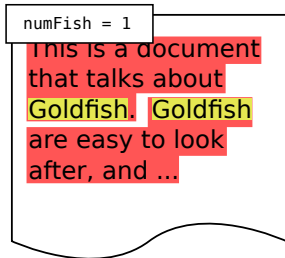


## Example 2

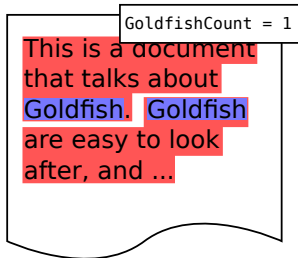


## Example 2

GATE



UIMA



We need an index linking the UIMA annotations to the GATE annotations they came from

## Defining the Mapping

The mapping is defined by the user in an XML file:

```
<uimaGateMapping>
  <inputs>
    <uimaAnnotation type="gate.example.Sentence"
                    gateType="Sentence"
                    indexed="true" />
  </inputs>
```

## Defining the Mapping

The mapping is defined by the user in an XML file:

```
<uimaGateMapping>
  <inputs>
    <uimaAnnotation type="gate.example.Sentence"
      gateType="Sentence"
      indexed="true" />
  </inputs>
```

For each GATE annotation of type Sentence ...

## Defining the Mapping

The mapping is defined by the user in an XML file:

```
<uimaGateMapping>
  <inputs>
    <uimaAnnotation type="gate.example.Sentence"
      gateType="Sentence"
      indexed="true" />
  </inputs>
```

...create a UIMA annotation of type `gate.example.Sentence` at the same place ...

## Defining the Mapping

The mapping is defined by the user in an XML file:

```
<uimaGateMapping>  
  <inputs>  
    <uimaAnnotation type="gate.example.Sentence"  
                    gateType="Sentence"  
                    indexed="true" />  
  </inputs>
```

...and remember this mapping.

## Defining the Mapping

```
<outputs>  
  <added>  
    <gateAnnotation type="Goldfish"  
      uimaType="gate.example.Goldfish" />  
  </added>
```

For each UIMA annotation of this type ...

## Defining the Mapping

```
<outputs>  
  <added>  
    <gateAnnotation type="Goldfish"  
      uimaType="gate.example.Goldfish" />  
  </added>
```

...add a GATE annotation at the same place.

## Defining the Mapping

```
<updated>
  <gateAnnotation type="Sentence"
    uimaType="gate.example.Sentence">
    <feature name="numFish">
      <uimaFSFeatureValue
        name="gate.example.Sentence:GoldfishCount"
        kind="int" />
      </feature>
    </gateAnnotation>
  </updated>
</outputs>
</uimaGateMapping>
```

For each UIMA annotation of this type ...

## Defining the Mapping

```
<updated>
  <gateAnnotation type="Sentence"
    uimaType="gate.example.Sentence">
    <feature name="numFish">
      <uimaFSFeatureValue
        name="gate.example.Sentence:GoldfishCount"
        kind="int" />
      </feature>
    </gateAnnotation>
  </updated>
</outputs>
</uimaGateMapping>
```

...find the GATE annotation it came from ...

## Defining the Mapping

```
<updated>
  <gateAnnotation type="Sentence"
    uimaType="gate.example.Sentence">
    <feature name="numFish">
      <uimaFSFeatureValue
        name="gate.example.Sentence:GoldfishCount"
        kind="int" />
      </feature>
    </gateAnnotation>
  </updated>
</outputs>
</uimaGateMapping>
```

...and set this annotation's `numFish` feature ...

## Defining the Mapping

```
<updated>
  <gateAnnotation type="Sentence"
    uimaType="gate.example.Sentence">
    <feature name="numFish">
      <uimaFSFeatureValue
        name="gate.example.Sentence:GoldfishCount"
        kind="int" />
      </feature>
    </gateAnnotation>
  </updated>
</outputs>
</uimaGateMapping>
```

...to the value of the `GoldfishCount` feature from the UIMA annotation.

## Embedding UIMA in GATE

- Write the mapping descriptor
  - Must ensure that all the annotations and features declared as input capabilities by the UIMA AE are supplied by the mapping.
  - Must not attempt to map to a UIMA FS type that is not declared in the AE's type system.
- For a Java AE, need to get UIMA AE implementation class onto the GATE ClassLoader: define a plugin with just the relevant <JAR> entries:

```
1 <CREOLE-DIRECTORY>
2   <JAR>myUimaAE.jar</JAR>
3   <JAR>some-dependency.jar</JAR>
4 </CREOLE-DIRECTORY>
```

- Load this plugin (in addition to the UIMA plugin)

## Embedding UIMA in GATE

- For C++ AEs, put the implementation library somewhere Java can find it.
- For remote service AEs no additional config is required.
- Create an instance of `gate.uima.AnalysisEnginePR` (“UIMA Analysis Engine” in GATE Developer)
- Init parameters are URLs to the UIMA AE descriptor XML and the mapping descriptor.
- Runtime parameter is the `annotationSetName` containing the annotations to map.
  - If you need to map annotations from several sets, use annotation set transfer or JAPE.

## Embedding GATE in UIMA

- Embedding a GATE `CorpusController` as a UIMA AE is the mirror-image of this process.
- Controller must be saved as an `.xgapp` with all PR runtime parameter values (except document and corpus) pre-configured correctly.
- Mapping descriptor format is the same (but `<gateAnnotation>` in the input section and `<uimaAnnotation>` in the output section)
- Each `<gateAnnotation>` or `<uimaAnnotation>` element can specify an `annotationSet` attribute, to support mapping to/from several GATE annotation sets.
  - on input – create the GATE annotation in this set
  - on output – look for the GATE annotation in this set

## Embedding GATE in UIMA

- Include `gate.jar`, the appropriate JARs from GATE's `lib`, and `uima-gate.jar` from the UIMA plugin on classpath.
- GATE provides a skeleton AE descriptor which needs to be customized
  - type system and capabilities to match the GATE mapping
  - external resource bindings to point to the saved `.xgapp` and the mapping descriptor.
- The AE will initialize GATE if necessary – UIMA application doesn't need to know it's embedding GATE.
- For more details, see the user guide (<http://gate.ac.uk/userguide/chap:uima>) and the test directory under `plugins/UIMA`.

## Exercise 1: Embedding UIMA in GATE

Run some of the example UIMA-in-GATE code provided with GATE

- Load the UIMA plugin
- Load `plugins/UIMA/examples` as a plugin (you'll need to "Add a CREOLE repository")
  - This loads the implementation classes for the example UIMA AEs.
- Load a default ANNIE application
- Create a UIMA Analysis Engine PR with these parameters (relative to `plugins/UIMA/examples/conf`) and add it to the end of the ANNIE application
  - `analysisEngineDescriptor:`  
`uima_descriptors/CountLowercaseAnnotator.xml`
  - `mappingDescriptor:`  
`mapping/TokenHandlerMapping.xml`

## Exercise 1: Embedding UIMA in GATE

- Run the application over a document of your choice - Token annotations have a `numLower` feature giving the number of lowercase letters in the token.
- Code is in `plugins/UIMA/examples/src`, have a look at the code and the mapping descriptor, see how the mapping is configured.
- Try changing the mapping to map the `LowerCaseLetters` feature from UIMA to a different name in GATE.
- Other AE descriptors and their associated mappings if you want to experiment further.

## Exercise 2: Embedding GATE in UIMA

- The `plugins/UIMA/test` directory contains an example UIMA AE descriptor that wraps a GATE application.
- `conf/TokenizerAndPOSTagger.xml` is an aggregate AE that runs
  - A native UIMA token and sentence annotator
  - The GATE POS tagger to add POS tags to the tokens
- UIMA provides a basic UI to run an AE and inspect the results, which you can run with

```
../..bin/ant documentanalyser in
plugins/UIMA (backslashes on Windows).
```

  - This starts up the tool with a classpath that includes the relevant JARs to run the GATE application AE.

## Exercise 2: Embedding GATE in UIMA

- Start the document analyser tool.
- Create an empty directory, and set the “Output directory” option to point to it.
- Set the “Location of Analysis Engine XML Descriptor” to point to the aggregate descriptor  
(`test/conf/TokenizerAndPOSTagger.xml`).
- Click the “Interactive” button
- Type (or paste) some text and click “Analyze”.
- If you're a confident UIMA user, try modifying the mapping to change the POS feature name (you will need to edit the type system to match).

# Outline

- 1 GATE and UIMA
  - Introduction to UIMA
  - UIMA and GATE compared
  - Integrating GATE and UIMA
- 2 GATE in Web Applications
  - Introduction
  - Multi-threading and GATE
  - Servlet Example
  - The Spring Framework
- 3 GATE and Groovy
  - Introduction to Groovy
  - Scripting GATE Developer
  - The Groovy Script PR
  - Writing GATE Resource Classes in Groovy

# Introduction

- Scenario:
  - Implementing a web application that uses GATE Embedded to process requests.
  - Want to support multiple concurrent requests
  - Long running process - need to be careful to avoid memory leaks, etc.
- Example used is a plain HttpServlet
  - Principles apply to other frameworks (struts, Spring MVC, Metro/CXF, Grails. . .)

## Setting up

- GATE libraries in `WEB-INF/lib`
  - `gate.jar` + JARs from lib
- Usual GATE Embedded requirements:
  - A directory to be "gate.home"
  - Site and user config files
  - Plugins directory

## GATE in a Multi-threaded Environment

- GATE initialization needs to happen once (and only once) before any other GATE APIs are used.
- The `Factory` is synchronized internally, so safe for use in multiple threads.
- Individual PRs/controllers are *not* safe – must not use the same PR instance concurrently in different threads
  - this is due to the design of runtime parameters as Java Beans properties.
- Individual LRs (documents, ontologies, etc.) are only thread-safe when accessed read-only by *all* threads.
  - if you need to share an LR between threads, be sure to synchronize (e.g. using `ReentrantReadWriteLock`)

## Initializing GATE using a ServletContextListener

ServletContextListener called by container at startup and shutdown (only startup method shown).

```
1 public void contextInitialized(ServletContextEvent e) {
2     ServletContext ctx = e.getServletContext();
3     File gateHome = new File(
4         ctx.getRealPath("/WEB-INF"));
5     Gate.setGateHome(gateHome);
6     File userConfig = new File(
7         ctx.getRealPath("/WEB-INF/user.xml"));
8     Gate.setUserConfigFile(userConfig);
9     // default site config is gateHome/gate.xml
10    // default plugins dir is gateHome/plugins
11    Gate.init();
12 }
```

## Initializing GATE using a ServletContextListener

You must register the listener in `web.xml`

```
1 <listener>
2   <listener-class>
3     gate.web.example.GateInitListener
4   </listener-class>
5 </listener>
```

## Handling Concurrent Requests

Naïve approach – new PRs for every request

```
1 public void doPost(request, response) {
2     ProcessingResource pr = Factory.createResource(...);
3     try {
4         Document doc = Factory.newDocument(
5             getTextFromRequest(request));
6         try {
7             // do some stuff
8         }
9         finally {
10            Factory.deleteResource(doc);
11        }
12    }
13    finally {
14        Factory.deleteResource(pr);
15    }
16 }
```

## Handling Concurrent Requests

Naïve approach – new PRs for every request

```
1 public void doPost(request, response) {  
2     ProcessingResource pr = Factory.createResource(...);  
3     try {  
4         Document doc = Factory.newDocument(  
5             getTextFromRequest(request));  
6         try {  
7             // do some stuff  
8         }  
9         finally {  
10            Factory.deleteResource(doc);  
11        }  
12    }  
13    finally {  
14        Factory.deleteResource(pr);  
15    }  
16 }
```

Many levels of try/finally  
– make sure you clean up  
even when errors occur

## Problems with Naïve Approach

- Guarantees no interference between threads
- But inefficient, particularly with complex PRs (large gazetteers, etc.)
- Hidden problem with JAPE:
  - Parsing a JAPE grammar creates and compiles Java classes
  - Once created, classes are never unloaded
  - Even with simple grammars, eventually `OutOfMemoryError` (PermGen space)

## Take Two: using ThreadLocal

Store the PR/Controller in a thread-local variable

```
1 private ThreadLocal<CorpusController> controller =
2     new ThreadLocal<CorpusController>() {
3
4     protected CorpusController initialValue() {
5         return loadController();
6     }
7 };
8
9 private CorpusController loadController() { ... }
10
11 public void doPost(request, response) {
12     CorpusController c = controller.get();
13     // do stuff with the controller
14 }
```

## An Improvement. . .

- Only initialise resources once per thread
- Interacts nicely with typical web server thread pooling
- But if a thread dies (e.g. with an exception), no way to clean up its controller

## One Solution: Object Pooling

- Manage your own pool of Controller instances
- Take a controller from the pool at the start of a request, return it (in a finally!) at the end
- Number of instances in the pool determines maximum concurrency level

## Simple Example of Pooling

Setting up and cleaning up:

```
1 private BlockingQueue<CorpusController> pool;  
2  
3 public void init() {  
4     pool = new LinkedBlockingQueue<CorpusController>();  
5     for(int i = 0; i < POOL_SIZE; i++) {  
6         pool.add(loadController());  
7     }  
8 }  
9  
10 public void destroy() {  
11     for(CorpusController c : pool) {  
12         Factory.deleteResource(c);  
13     }  
14 }
```

## Simple Example of Pooling

Processing requests:

```
15 public void doPost(request, response) {  
16     CorpusController c = pool.take();  
17     try {  
18         // do stuff  
19     }  
20     finally {  
21         pool.add(c);  
22     }  
23 }
```

## Simple Example of Pooling

Processing requests:

```
15 public void doPost(request, response) {  
16     CorpusController c = pool.take();  
17     try {  
18         // do stuff  
19     }  
20     finally {  
21         pool.add(c);  
22     }  
23 }
```

←  
This blocks when the  
pool is empty. Use `poll`  
for non-blocking check.

## Creating the pool

- Typically to create the pool you would use `PersistenceManager` to load a saved application several times.
- But this is not always optimal, e.g. large gazetteers consume lots of memory.
- GATE provides API to *duplicate* an existing instance of a resource: `Factory.duplicate(existingResource)`.
- By default, this simply calls `Factory.createResource` with the same class name, parameters, features and name.
- But individual Resource classes can override this if they know better by implementing the `CustomDuplication` interface.
  - e.g. `DefaultGazetteer` uses a `SharedDefaultGazetteer` — same behaviour, but shares the in-memory representation of the lists.

## Other Caveats

- With most PRs it is safe to create lots of identical instances
- But *not all!*
  - e.g. training a machine learning model with the batch learning PR (in the `Learning` plugin)
  - but it is safe to have several instances *applying* an existing model.
- When using `Factory.duplicate`, be careful not to duplicate a PR that is being used by another thread
  - i.e. either create all your duplicates up-front or else keep the original prototype “pristine”.

## Exporting the Grunt Work: Spring

- <http://www.springsource.org/>
- “Inversion of Control”
- Configure your business objects and connections between them using XML or Java annotations
- Handles application startup and shutdown
- GATE provides helpers to initialise GATE, load saved applications, etc.
- Built-in support for object pooling
- Web application framework (Spring MVC)
- Used by other frameworks (Grails, CXF, ...)

## Using Spring in Web Applications

- Spring provides a `ServletContextListener` to create a single *application context* at startup.
- Takes configuration by default from `WEB-INF/applicationContext.xml`
- Context made available through the `ServletContext`
- For our running example we use Spring's `HttpRequestHandler` interface which abstracts from servlet API
- Configure an `HttpRequestHandler` implementation as a Spring bean, make it available as a servlet.
  - allows us to configure dependencies and pooling using Spring

## Initializing GATE via Spring

applicationContext.xml:

```
1 <beans
2   xmlns="http://www.springframework.org/schema/beans"
3   xmlns:gate="http://gate.ac.uk/ns/spring">
4   <gate:init gate-home="/WEB-INF"
5     plugins-home="/WEB-INF/plugins"
6     site-config-file="/WEB-INF/gate.xml"
7     user-config-file="/WEB-INF/user-gate.xml">
8     <gate:preload-plugins>
9       <value>/WEB-INF/plugins/ANNIE</value>
10    </gate:preload-plugins>
11  </gate:init>
12 </beans>
```

## Loading a Saved Application

To load an application state saved from GATE Developer:

```
1 <gate:saved-application  
2     id="myApp"  
3     location="/WEB-INF/application.xgapp"  
4     scope="prototype" />
```

- `scope="prototype"` means create a new instance each time we ask for it
- Default scope is “singleton” — one instance is created at startup and shared.

## Duplicating an Application

- Alternatively, load the application once and then duplicate it

```
1 <gate:duplicate id="myApp" return-template="true">  
2   <gate:saved-application location="..." />  
3 </gate:duplicate>
```

- `<gate:duplicate>` creates a new duplicate each time we ask for the bean.
- `return-template` means the original controller (from the `saved-application`) will be returned the first time, then duplicates thereafter.
- Without this the original is kept pristine and only used as a source for duplicates.

## Spring Servlet Example

Write the `HttpRequestHandler` assuming single-threaded access, we will let Spring deal with the pooling for us.

```
1 public class MyHandler
2     implements HttpRequestHandler {
3     // controller reference will be injected by Spring
4     public void setApplication(
5         CorpusController app) { ... }
6
7     // good manners to clean it up ourselves though this isn't
8     // necessary when using <gate:duplicate>
9     public void destroy() throws Exception {
10         Factory.deleteResource(app);
11     }
```

## Spring Servlet Example

```
13 public void handleRequest(request, response) {  
14     Document doc = Factory.newDocument(  
15         getTextFromRequest(request));  
16     try {  
17         // do some stuff with the app  
18     }  
19     finally {  
20         Factory.deleteResource(doc);  
21     }  
22 }  
23 }
```

## Tying it together

In `applicationContext.xml`

```
1 <gate:init ... />
2 <gate:duplicate id="myApp" return-template="true">
3   <gate:saved-application
4     location="/WEB-INF/application.xgapp" />
5 </gate:duplicate>
6
7 <!-- Define the handler bean, inject the controller -->
8 <bean id="mainHandler"
9     class="my.pkg.MyHandler"
10    destroy-method="destroy">
11   <property name="application" ref="myApp" />
12   <gate:pooled-proxy max-size="3"
13     initial-size="3" />
14 </bean>
```

## Tying it together: Spring Pooling

```
12 <gate:pooled-proxy max-size="3"  
13     initial-size="3" />
```

- A *bean definition decorator* that tells Spring that instead of a singleton `mainHandler` bean, we want
  - a pool of 3 instances of `MyHandler`
  - exposed as a single *proxy* object implementing the same interfaces
- *Each method call* on the proxy is dispatched to one of the objects in the pool.
- Each target bean is guaranteed to be accessed by no more than one thread at a time.
- When the pool is empty (i.e. more than 3 concurrent requests) further requests will block.

## Tying it together: Spring Pooling

- Many more options to control the pool, e.g. for a pool that grows as required and shuts down instances that have been idle for too long, and where excess requests fail rather than blocking:

```
1 <gate:pooled-proxy
2   max-size="10"
3   max-idle="3"
4   time-between-eviction-runs-millis="180000"
5   min-evictable-idle-time-millis="90000"
6   when-exhausted-action-name="WHEN_EXHAUSTED_FAIL"
7 />
```

- Under the covers, `<gate:pooled-proxy>` creates a Spring `CommonsPoolTargetSource`, attributes correspond to properties of this class.
- See the Spring documentation for full details.

## Tying it together: web.xml

To set up the Spring context:

```
1 <listener>
2   <listener-class>
3     org.springframework.web.context.
4       ContextLoaderListener
5   </listener-class>
6 </listener>
```

## Tying it together: web.xml

To make the `HttpRequestHandler` available as a servlet, create a `servlet` entry in `web.xml` with the same name as the (pooled) handler bean:

```
7 <servlet>
8   <servlet-name>mainHandler</servlet-name>
9   <servlet-class>
10      org.springframework.web.context.support.
11         HttpRequestHandlerServlet
12   </servlet-class>
13 </servlet>
```

## Exercise: A simple web application

- In `hands-on/webapps` you have an implementation of the `HttpRequestHandler` example.
- `hands-on/webapps/gate` is a simple web application which provides
  - an HTML form where you can enter text to be processed by GATE
  - an `HttpRequestHandler` that processes the form submission using a GATE application and displays the document's features in an HTML table
  - the application and pooling of the handlers is configured using Spring.
- Embedded Jetty server to run the app.
- To keep the download small, most of the required JARs are not in the `module-8.zip` file – you already have them in GATE.

## Exercise: A simple web application

- To run the example you need ant (use the one in GATE's `bin` directory if you don't have a standalone copy).
- Edit `webapps/gate/WEB-INF/build.xml` and set the `gate.home` property correctly.
- In `webapps/gate/WEB-INF`, run ant.
  - this copies the remaining dependencies from GATE and compiles the `HttpRequestHandler` Java code from `WEB-INF/src`.
- `WEB-INF/gate-files` contains the site and user configuration files.
- This is also where the webapp expects to find the `.xgapp`.
- No `.xgapp` provided by default – you need to provide one.

## Exercise: A simple web application

- Use the statistics application you wrote yesterday.
- In GATE Developer, create a “corpus pipeline” application containing a tokeniser and your statistics PR.
- Right-click on the application and “Export for Teamware”.
  - This will save the application state along with all the plugins it depends on in a single zip file.
  - Just accept the defaults in the dialog asking for input and output annotation sets – this is necessary for Teamware but not for us.
- Unpack the zip file under `WEB-INF/gate-files`
  - don't create any extra directories – you need `application.xgapp` to end up in `gate-files`.

## Exercise: A simple web application

- You can now run the server – in hands-on/webapps run `ant -emacs`
- Browse to `http://localhost:8080/gate/`, enter some text and submit
- Watch the log messages...
- Notice the result page includes “GATE handler *N*” – each handler in the pool has a unique ID.
- Multiple submissions go to different handler instances in the pool.
- `http://localhost:8080/stop` to shut down the server gracefully
- Try editing `gate/WEB-INF/applicationContext.xml` and change the pooling configuration.
- Try opening several browser windows and using a longer “delay” to test concurrent requests.

## Not Just for Webapps

- Spring isn't just for web applications
- You can use the same tricks in other embedded apps
- GATE provides a `DocumentProcessor` interface suitable for use with Spring pooling

```
1 // load an application context from definitions in a file
2 ApplicationContext ctx =
3     new FileSystemXmlApplicationContext("beans.xml");
4
5 DocumentProcessor proc = ctx.getBean(
6     "documentProcessor", DocumentProcessor.class);
7
8 // in worker threads...
9 proc.processDocument(myDocument);
```

## Not Just for Webapps

The beans.xml file:

```
1 <gate:init ... />
2 <gate:duplicate id="myApp">
3   <gate:saved-application
4     location="resources/application.xgapp" />
5 </gate:duplicate>
6
7 <!-- Define the processor bean to be pooled -->
8 <bean id="documentProcessor"
9     class="gate.util.
10        LanguageAnalyserDocumentProcessor"
11     destroy-method="cleanup">
12   <property name="analyser" ref="myApp" />
13   <gate:pooled-proxy max-size="3" />
14 </bean>
```

## Conclusions

Two golden rules:

- Only use a GATE Resource in one thread at a time
- Always clean up after yourself, even if things go wrong (`deleteResource` in a finally block).

# Outline

- 1 GATE and UIMA
  - Introduction to UIMA
  - UIMA and GATE compared
  - Integrating GATE and UIMA
- 2 GATE in Web Applications
  - Introduction
  - Multi-threading and GATE
  - Servlet Example
  - The Spring Framework
- 3 GATE and Groovy
  - Introduction to Groovy
  - Scripting GATE Developer
  - The Groovy Script PR
  - Writing GATE Resource Classes in Groovy

# Groovy

- Dynamic language for the JVM
- Groovy scripts and classes compile to Java bytecode – fully interoperable with Java.
- Syntax very close to regular Java
- Explicit types optional, semicolons optional
- Dynamic dispatch – method calls dispatched based on runtime type rather than compile-time.
- Can add new methods to existing classes at runtime using *metaclass* mechanism
- Groovy adds useful extra methods to many standard classes in `java.io`, `java.lang`, etc.

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- **def** keyword declares an untyped variable

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- **def** keyword declares an untyped variable
- but dynamic dispatch ensures the `get` call goes to the right class (`AnnotationSet`).

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- **def** keyword declares an untyped variable
- but dynamic dispatch ensures the `get` call goes to the right class (`AnnotationSet`).
- **findAll** and **collect** are methods added to `Collection` by Groovy

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- **def** keyword declares an untyped variable
- but dynamic dispatch ensures the `get` call goes to the right class (`AnnotationSet`).
- **findAll** and **collect** are methods added to `Collection` by Groovy
  - <http://groovy.codehaus.org/groovy-jdk> has the details.

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- **def** keyword declares an untyped variable
- but dynamic dispatch ensures the `get` call goes to the right class (`AnnotationSet`).
- **findAll** and **collect** are methods added to `Collection` by Groovy
  - <http://groovy.codehaus.org/groovy-jdk> has the details.
- `?.` is the *safe navigation* operator – if the left hand operand is **null** it returns **null** rather than throwing an exception

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3     anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

■ =~ for regular expression matching

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- =~ for regular expression matching
- unified access to JavaBean properties – `it.startNode` shorthand for `it.getStartNode()`

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- `=~` for regular expression matching
- unified access to JavaBean properties – `it.startNode` shorthand for `it.getStartNode()`
- and Map entries – `anchor.features.href` shorthand for `anchor.getFeatures().get("href")`

## Groovy example

Find the start offset of each absolute link in the document.

```
1 def om = document.getAnnotations("Original markups")
2 om.get('a').findAll { anchor ->
3   anchor.features?.href =~ /^http:/
4 }.collect { it.startNode.offset }
```

- `=~` for regular expression matching
- unified access to JavaBean properties – `it.startNode` shorthand for `it.getStartNode()`
- and Map entries – `anchor.features.href` shorthand for `anchor.getFeatures().get("href")`
- Map entries can also be accessed like arrays, e.g. `features["href"]`

# Closures

Parameter to `collect`, `findAll`, etc. is a *closure*

- like an anonymous function (JavaScript), a block of code that can be assigned to a variable and called repeatedly.
- Can declare parameters (typed or untyped) between the opening brace and the `->`
- If no explicit parameters, closure has an implicit parameter called `it`.
- Closures have access to the variables in their containing scope (unlike Java inner classes these do not have to be `final`).
- The return value of a closure is the value of its last expression (or an explicit `return`).
- Closures are used all over the place in Groovy

## More Groovy Syntax

- Shorthand for lists: `["item1", "item2"]` declares an `ArrayList`
- Shorthand for maps: `[foo: "bar"]` creates a `HashMap` mapping the key `"foo"` to the value `"bar"`.
- Interpolation in *double-quoted* strings (like Perl):  
`"There are ${anns.size()} annotations of type ${annType}"`
- Parentheses for method calls are optional (where this is unambiguous): `myList.add 0, "someString"`
  - When you use parentheses, if the last parameter is a closure it can go outside them: this is a method call with two parameters  
`someList.inject(0) { last, cur -> last + cur }`
- “slashy string” syntax where backslashes don’t need to be doubled: `/C:\Program Files\Gate/` equivalent to  
`'C:\\Program Files\\Gate'`

## Operator Overloading

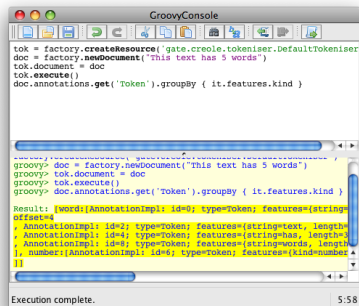
- Groovy supports operator overloading cleanly
- Every operator translates to a method call
  - `x == y` becomes `x.equals(y)` (for reference equality, use `x.is(y)`)
  - `x + y` becomes `x.plus(y)`
  - `x << y` becomes `x.leftShift(y)`
  - full list at <http://groovy.codehaus.org>
- To overload an operator for your own class, just implement the method.
- e.g. List implements `leftShift` to append items to the list:  
`['a', 'b'] << 'c' == ['a', 'b', 'c']`

# Groovy in GATE

- Groovy support in GATE is provided by the `Groovy` plugin.
- Loading the plugin
  - enables the Groovy scripting console in GATE Developer
  - adds utility methods to various GATE classes and interfaces for use from Groovy code
  - provides a PR to run a Groovy script.

## Scripting GATE Developer

- Groovy provides a Swing-based *console* to test out small snippets of code.
- The console is available in the GATE Developer GUI via the Tools menu. To enable, load the `Groovy` plugin.



```
tok = factory.createResource('gate.creole.tokeniser.DefaultTokeniser')
doc = factory.newDocument("This text has 5 words")
tok.document = doc
tok.execute()
doc.annotations.get('Token').groupBy { it.features.kind }
```

```
groovy> doc = factory.newDocument("This text has 5 words")
groovy> tok.document = doc
groovy> tok.execute()
groovy> doc.annotations.get('Token').groupBy { it.features.kind }
```

```
Result: [word:{AnnotationImpl: id=0; type=Token; features={string=
offset=4
}, AnnotationImpl: id=2; type=Token; features={string=text, length=
}, AnnotationImpl: id=4; type=Token; features={string=has, length=3
}, AnnotationImpl: id=8; type=Token; features={string=words, length
}, number; AnnotationImpl: id=6; type=Token; features={kind=number
}]
```

Execution complete. 5:58

## Imports and Predefined Variables

The GATE Groovy console imports the same packages as JAPE RHS actions:

- `gate`, `gate.annotation`, `gate.util`, `gate.jape` and `gate.creole.ontology`

The following variables are implicitly defined:

- corpora** a list of loaded corpora LRs (`Corpus`)
- docs** a list of all loaded document LRs (`DocumentImpl`)
- prs** a list of all loaded PRs
- apps** a list of all loaded Applications (`AbstractController`)

## Exercise 1: The Groovy Console

- Start the GATE Developer GUI
- Load the `Groovy` plugin
- Select Tools → Groovy Tools → Groovy Console
- Experiment with the console
- For example to tokenise a document and find how many “number” tokens it contains:

```
1 doc = Factory.newDocument(new URL('http://gate.ac.uk'))
2 tokeniser = Factory.createResource('gate.creole.tokeniser.
   DefaultTokeniser')
3 tokeniser.document = doc
4 tokeniser.execute()
5 tokens = doc.annotations.get('Token')
6 tokens.findAll { it.features.kind == 'number' }.size()
```

## Exercise 1: The Groovy Console

- Variables you assign in the console (without a `def` or a type declaration) remain available to future scripts in the same console.
- So you can run the previous example, then try more things with the `doc` and `tokens` variables.
- Some things to try:
  - Find the names and sizes of all the annotation sets on the document (there will probably only be one named set).
  - List all the different `kinds` of token
  - Find the longest word in the document

## Exercise 1: Solution

Some possible solutions (there are many...)

```
1 // Find the annotation set names and sizes
2 doc.namedAnnotationSets.each { name, set ->
3     println "${name} has size ${set.size()}"
4 }
5
6 // List the different kinds of token
7 tokens.collect { it.features.kind }.unique()
8
9 // Find the longest word
10 tokens.findAll {
11     it.features.kind == 'word'
12 }.max { it.features.length.toInteger() }
```

## Groovy Categories

- In Groovy, a class declaring static methods can be used as a *category* to inject methods into existing types (including interfaces)
- A static method in the category class whose first parameter is a Document:

```
public static SomeType foo(Document d, String arg)
```

- ...becomes an instance method of the Document class:

```
public SomeType foo(String arg)
```

- The `use` keyword activates a category for a single block
- To enable the category globally:

```
TargetClass.mixin(CategoryClass)
```

## Utility Methods

- The `gate.Utils` class (mentioned in the JAPE module) contains utility methods for documents, annotations, etc.
- Loading the `Groovy` plugin treats this class as a category and installs it as a global mixin.
- Enables syntax like:

```
1 tokens.findAll {  
2   it.features.kind == 'number'  
3 }.each {  
4   println "${it.type}: length = ${it.length()}, "  
5   println "  string = ${doc.stringFor(it)}"  
6 }
```

## Utility Methods

- The Groovy plugin also mixes in the `GateGroovyMethods` class.
- This extends common Groovy idioms to GATE classes
  - e.g. implements `each`, `eachWithIndex` and `collect` for `Corpus` to do the right thing when the corpus is stored in a datastore
  - defines a `withResource` method on `Resource`, to call a closure with a given resource as a parameter, and ensure the resource is deleted when the closure returns:

```
1 Factory.newDocument(someURL).withResource { doc ->  
2   // do something with the document  
3 }
```

## Utility Methods

- Also overloads the subscript operator `[]` to allow:
  - `annSet["Token"]` and `annSet["Person", "Location"]`
  - `annSet[15..20]` to get annotations within given span
  - `doc.content[15..20]` to get the `DocumentContent` within a given span
- See `src/gate/groovy/GateGroovyMethods.java` in the Groovy plugin for details.

## Exercise 2: Using a category

In the console, try using some of these new methods:

```
1 tokens = doc.annotations["Token"]
2 tokens.findAll {
3     it.features.kind == 'number'
4 }.each {
5     println "${it.type}: length = ${it.length()}, "
6     println "    string = ${doc.stringFor(it)}"
7 }
```

## The Groovy Script PR

- The `Groovy` plugin provides a PR to execute a Groovy script.
- Useful for quick prototyping, or tasks that can't be done by JAPE but don't warrant writing a custom PR.
- PR takes the following parameters:

**scriptURL** (init-time) The path to a valid Groovy script

**inputASName** an optional annotation set intended to be used as input by the PR

**outputASName** an optional annotation set intended to be used as output by the PR

**scriptParams** optional parameters for the script as a `FeatureMap`

## Script Variables

The script has the following implicit variables available when it is run

**doc** the current document

**content** the string content of the current document

**inputAS** the annotation set specified by `inputASName` in the PRs runtime parameters

**outputAS** the annotation set specified by `outputASName` in the PRs runtime parameters

**scriptParams** the parameters `FeatureMap` passed as a runtime parameter

and the same implicit imports as the console.

## Exercise 3: Using the Script PR

- Write the Goldfish annotator from the UIMA section as a Groovy script
  - Annotate all occurrences of the word “goldfish” (case-insensitive) in the input document as the annotation type “Goldfish”.
  - Add a “numFish” feature to each Sentence annotation giving the number of Goldfish annotations that the sentence contains.
- Put your script in the file  
`hands-on/groovy/goldfish.groovy`
- To test, load `hands-on/groovy/goldfish-app.xgapp` into GATE Developer (this application contains tokeniser, sentence splitter and goldfish script PR).
- You need to re-initialize the Groovy Script PR after each edit to `goldfish.groovy`

## Exercise 3: Solution

One of many possible solutions:

```
1 def m = (content =~ /(?!i)goldfish/)
2 while(m.find()) {
3     outputAS.add(m.start(), m.end(),
4         'Goldfish', [:].toFeatureMap())
5 }
6
7 def allGoldfish = outputAS["Goldfish"]
8 inputAS["Sentence"].each { sent ->
9     sent.features.numFish =
10         allGoldfish[sent.start()..sent.end()].size()
11 }
```

## Writing Resources in Groovy

- Groovy is more than a scripting language – you can write classes (including GATE resources) in Groovy and compile them to Java bytecode.
- Compiler available via `<groovyc>` Ant task in `groovy-all` JAR.
- In order to use GATE resources written in Groovy, `groovy-all` JAR file must go into `gate/lib`.

## Groovy Beans

- Recall unified Java Bean property access in Groovy
  - `x = it.someProp` means `x = it.getSomeProp()`
  - `it.someProp = x` means `it.setSomeProp(x)`
- Declarations have a similar shorthand: a field declaration with no **public**, **protected** or **private** modifier becomes a private field plus an auto-generated public getter/setter pair.
- But you can provide explicit setter or getter, which will be used instead of the automatic one.
  - Need to do this if you need to annotate the setter (e.g. as a `CreoleParameter`).
  - Declare the setter **private** to get a read-only property (but not if it's a creole parameter).

## Example: a Groovy Regex PR

```
1 package gate.groovy.example
2
3 import gate.*
4 import gate.creole.*
5
6 public class RegexPR extends AbstractLanguageAnalyser {
7     String regex
8     String annType
9     String annotationSetName
10
11    public void execute() {
12        def aSet = document.getAnnotations(annotationSetName)
13        def matcher = (document.content.toString() =~ regex)
14        while(matcher.find()) {
15            aSet.add(matcher.start(), matcher.end(),
16                annType, [:].toFeatureMap())
17        }
18    }
19 }
```

## Further Reading

- **UIMA:** `http://incubator.apache.org/uima`
  - `http://gate.ac.uk/userguide/chap:uima` for the GATE integration layer.
- **Spring:** `http://www.springsource.org`
- **Groovy:** `http://groovy.codehaus.org`
  - `http://gate.ac.uk/userguide/sec:api:groovy` for GATE details.
  - Also worth a look: **Grails:** `http://grails.org`. A Groovy- and Spring-based rapid development framework for web applications (we use Grails for GATE Wiki and Mimir).