The Javadoc NLP Corpus Generation Doclet

- GATE Components
- Corpora
- Software Engineering

toc_collapse=0; Table of Contents

1. Overview

This page describes the process of generating a corpus from source code and source code comments using Javadoc. The SSLDoclet is a custom doclet that is passed as a parameter to Javadoc in order to create an Abstract Syntax Tree (AST) that can be used as a corpus within NLP frameworks such as GATE.

```
package org.argouml.model;

import java.beans.PropertyChangeListener;

/**
 * Abstract class that implements the convenience methods of the
 * {@link ModelEventPump} interface.
 *
 * @author Linus Tolke
 */
public abstract class AbstractModelEventPump implements ModelEventPump {
    /*
     * @see org.argouml.model.ModelEventPump#addModelEventListener(
     *          java.beans.PropertyChangeListener, java.lang.Object,
     *          java.lang.String)
     */
    public void addModelEventListener(PropertyChangeListener listener, Object modelelement, String eventName) {
        addModelEventListener(listener, modelelement, new String[] {eventName});
    }
}
```

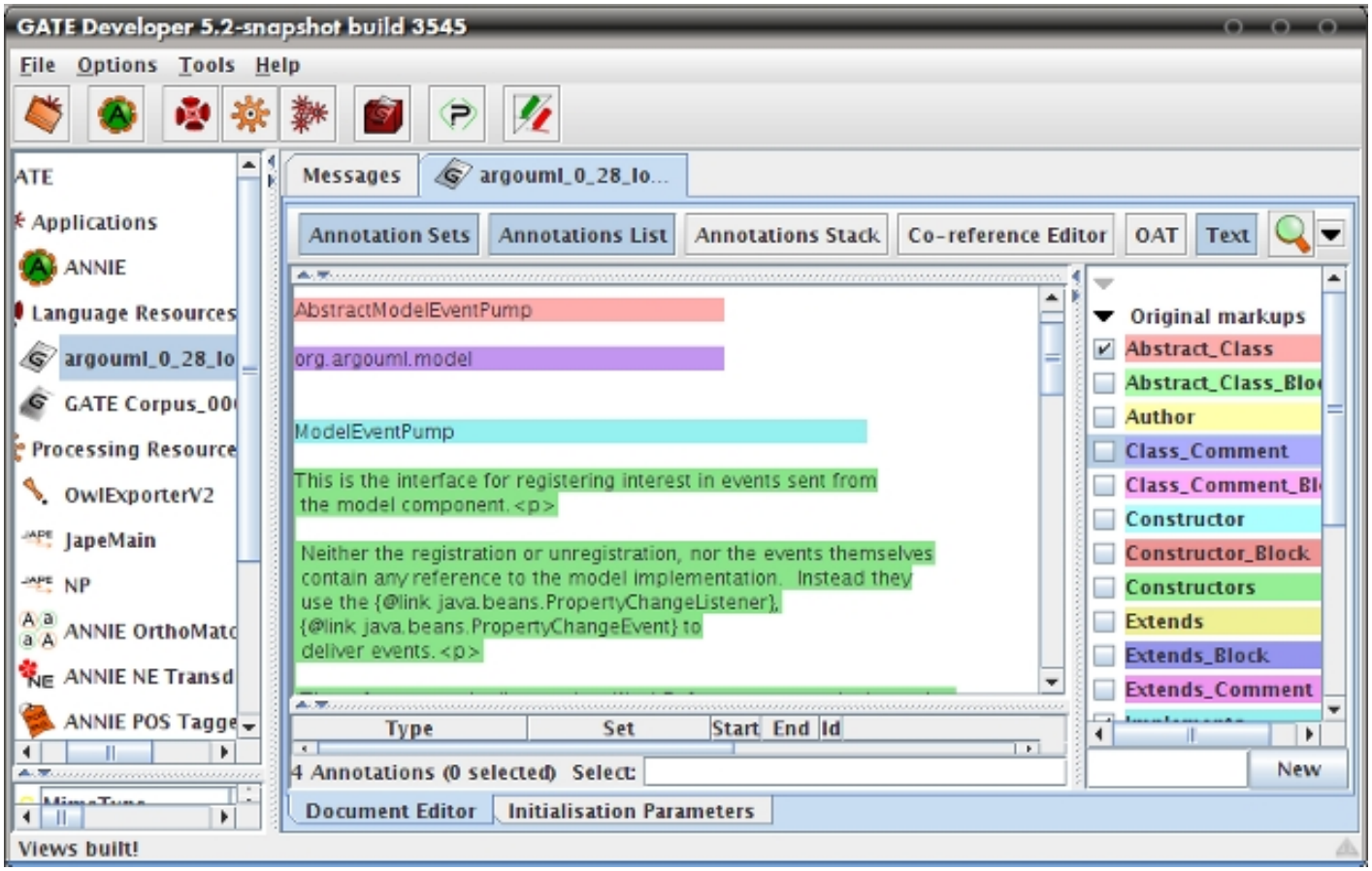
A Sample of Java code and comments taken from ArgoUML

The semantics found in source code and source code comments are modelled using an AST schema. The SSLDoclet generates an AST that is designed to be used as a corpus. The schema of the AST uses a structure that enables the tags, attributes and elements of the generated XML to be translated as annotations, features and entities of a corpus when loaded within an NLP framework.

```
- <Abstract_Class_Block>
    <Abstract_Class> AbstractModelEventPump </Abstract_Class>
    <Package> org.argouml.model </Package>
  - <Implements_Block>
      <Implements qualifiedType="org.argouml.model.ModelEventPump" type="Interface"> ModelEventPump
      </Implements>
    - <Implements_Comment>
        This is the interface for registering interest in events sent from the model component.<p> Neither the registration or
        unregistration, nor the events themselves contain any reference to the model implementation. Instead they use the {@link
        java.beans.PropertyChangeListener}, {@link java.beans.PropertyChangeEvent} to deliver events.<p> The reference to the
        listener is a WeakReference so you don't need to call removeWHATEVERListener, you can just forget about your listener
        and it is eventually finalized and removed. This also means that you will have to keep a reference to your listener while it is
        active. Since the garbage collecting mechanism is not really deterministic a forgotten about listener might still receive
        events. Unless it can handle them in a harmless way, this approach should not be used. TODO: (Is this still true or does it
        refer to the NSUML implementation? - tfm 20051109) (This is part of the contract that is established between the Model
        subsystem and its users. If that is not fulfilled by the current implementation, then the current implementation is incorrect.
        Linus 20060411).<p> TODO: What event names? The event names generated are {@link String}s and their values and
        meanings are not really well documented. In general they are the name of an an association end or attribute in the UML
        metamodel.<p> Here are some highlights:<ul> <li>"remove" - event sent when the element is removed. </ul>
      </Implements_Comment>
    </Implements_Block>
  - <Extends_Block>
      <Extends qualifiedType="java.lang.Object" type="Class"> Object </Extends>
      <Extends_Comment/>
    </Extends_Block>
  - <Class_Comment_Block>
    - <Class_Comment>
        Abstract class that implements the convenience methods of the {@link ModelEventPump} interface.
      </Class_Comment>
      <Author> Linus Tolke </Author>
    </Class_Comment_Block>
  - <Constructors>
    - <Constructor_Block>
        <Constructor modifier="public" visibility="public" signature="()"> AbstractModelEventPump </Constructor>
      </Constructor_Block>
    </Constructors>
  - <Methods>
    - <Method_Block>
        <Method abstract="true" modifier="public abstract" visibility="public" signature="
        (java.beans.PropertyChangeListener, java.lang.Object, java.lang.String[])"> addModelEventListener </Method>
      - <Parameter_Block>
          <Parameter fulltype="java.beans.PropertyChangeListener" type="PropertyChangeListener"> listener </Parameter>
        </Parameter_Block>
      - <Parameter_Block>
          <Parameter fulltype="java.lang.Object" type="Object"> modelelement </Parameter>
        </Parameter_Block>
      - <Parameter_Block>
          <Parameter fulltype="java.lang.String[]" type="String"> eventNames </Parameter>
        </Parameter_Block>
      </Method_Block>
```

Screenshot of a Corpus Generated using the SSLDoclet

The SSLDoclet is designed to generate a corpus using the eXtensible Markup Language (XML) that uses a schema that is easily processed by an NLP Framework. The XML nodes are interpreted as annotations, the attributes are interpreted as features of the annotations, and finally the elements are interpreted as entities of the annotations.



The Generated Corpus Loaded within GATE

2. Documentation

Our doclet is implemented using the Javadoc doclet API library, and is passed as a parameter to Javadoc when generating a corpus using a source directory. The figure below shows the "docs" task that is part of the ant build included with the SSLDoclet source code.

```
-<target name="docs" depends="clean, jar">
    <mkdir dir="${doc.dir}"/>
    <javadoc doclet="${doclet}" docletpath="${jar.dir}/${doclet.jar}"
     sourcepath="${src.dir}" packagenames="info.semanticsoftware.doclet"
     destdir="${doc.dir}" additionalparam="-J-Xmx256m" private="yes">
    </javadoc>
  </target>
</project>
```

Docs Task

Included in the ANT Build

The parameters needed to generate a corpus from source code are:

- doclet: The name of the doclet

- docletpath: The directory containing the doclet
- sourcepath: The directory containing the source code that needs to be processed
- packagenames: The name of the packages that need to be processed.
- destdir: The directory where the doclet will place the generated XML documents.

To run the SSLDoclet, both ANT and Java must be part of your system's environment variable, and:

- Setup the doc task with the valid parameter values.
- Issue the following command using the command line
  " ant docs "

3. Download

- The SSL Javaoc Doclet Version 1.2
- Our research paper about the Semantic Software Lab Javadoc Doclet (SSL Javadoc Doclet)
- The GNU GPL license under which you can use this tool.

If you use our component, please cite our paper: Khamis, N., R. Witte, and J. Rilling, "Generating an NLP Corpus from Java Source Code: The SSL Javadoc Doclet", *New Challenges for NLP Frameworks*, Valletta, Malta : ELRA, pp. 41–45, May 22, 2010.

4. Feedback

For questions, comments, etc., please use the Forum.

5. Version history

- 1.2: 13.04.2010. Added line number information.
- 1.1: 01.03.2010. Initial public release.