KAT: An Annotation Tool for STEM Documents

Alex Dumitru, Deyan Ginev, Michael Kohlhase
Vlad Merticariu, Stefan Mirea, Tom Wiesing

Computer Science, Jacobs University Bremen

February 15, 2014

Abstract

This report presents the KAT system, a generic annotation tool for linguistic/semantic annotations in structured (XHTML5) documents. [1]

EdN:1

---

[1]EDNOTE: MK: continue

Contents

1   Introduction

Text annotation is the practice of adding a note to a text, which may include highlights or underlining, comments, footnotes or links. In most of the cases,annotations can be thought of as text-metadata because they are usually added post hoc and provide information about the text without fundamentally altering it.

A web annotation is an online annotation associated with a web resource. The annotation of web-based documents by user communities is a widely used method of augmenting and adding value to these resources and there are numerous use cases where the process can disambiguate contexts or improve the overall readability.

In this report we present the KAT annotation tool for (X)HTML documents, built with the following objectives:

1. *Usability* – The tool should be convenient and easy to use by regular internet users allowing them varying degrees of complexity in the user interface.

2. *Ease of integration* – The tool should be easy to integrate into existing projects, and should have minimal to none dependencies on the server side.

3. *Semantic Richness* – The annotations provided by our users should contain the necessary level of information for NLP tools to use.

KAT is open source, it is licensed under the Gnu Public License [Fre91] and hosted on GitHub [KG].

Development was started in Spring 2011 by Alex Mircea Dumitru and Vlad Merticariu as a Computational Semantics Project at Jacobs University under the supervision of Deyan Ginev and Michael Kohlhase. Stefan Mirea contributed the firefox plugin and the reviewing functionality, and Tom Wiesing added JOBAD integration and general interface polish.

In the next section we review the state of the art in linguistic/semantic annotation systems, [2] Section **??** concludes this report.   EdN:2

2   State of the art

In this section we will present a few of the current annotation tools available on the web, extracting the most relevant features that were taken into account when building KAT.

Different types of web-based projects will require different approaches to annotations, allowing us to distinguish between 2 main categories:

1. **Dynamic annotations**: implemented by systems which allow the annotation of the text itself. In such a system, the anchor of an annotation is a piece of digitized text.

---

[2]EDNOTE: MK: continue

2. **Static annotations**: implemented by systems which allow the annotation of a specific region of content. In this case, the anchor of an annotation is the fixed, in-page, position of the annotated region.

Digitized, mathematical text lays in the center of our research direction during this project and, for this reason, we are going to focus mainly on the first category. However, in order to have a complete overview of the available features, the last subsection of this part presents one example from the second category.

An important aspect of our desired system is that, even though it is very close to a dynamic annotation system, we are looking for a more stable and semantically rich format, such as OMDoc, to anchor the annotations.

2.1   the brat Annotation Tool

brat [BR] Is a web based tool for text annotation. It is designed in particular for structured annotation, where the notes are not freeform text but have a fixed form that can be automatically processed and interpreted by a computer and implicitly belongs to the category of dynamic annotation tools. The most important features that we identified were:
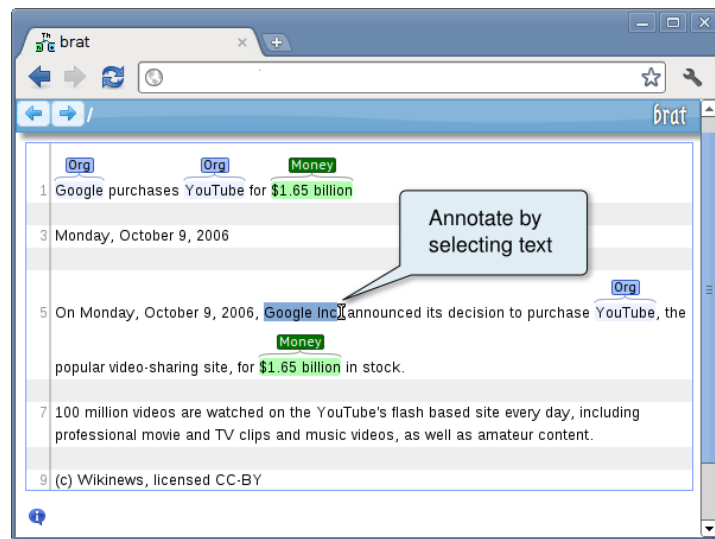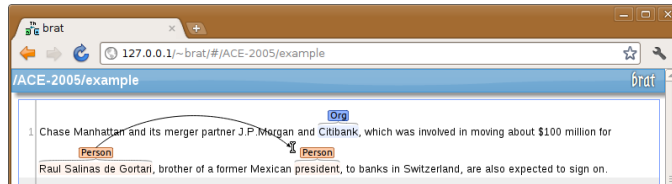

Figure 1: Annotation with brat

1. The text must be preprocessed into a particular, fixed format before being annotated.

2. There are 2 types of annotation supported:

   - text span annotations: simple annotation of a piece of text:

4

- relation annotations: two separated pieces of text are annotated and a connection between them is created:

3. Extensive search functionality for annotations, see Figure 2

4. Export interface: annotations can be exported in an internal format, which can then be converted in other formats such as PDF or HTML. Furthermore, visualizations can be exported as well, into SVG and bitmap formats (PNG).

5. Each annotation is accessible by URL: every brat annotation can be uniquely addressed within the brat server. Together with the URL of the server, this form of addressing provides a globally unique address for every brat annotation.

6. Brat provides a validation system for its annotations: the admin can define validation grammar rules, forcing the user to adhere to a specific annotation format.

7. Annotations are anchored based on a word-counter: this makes the annotated text unchangeable and constitutes one of our main concerns regarding the system.

2.2 Yawas Annotation Tool

Yawas [YW] is an annotation system designed as an extension for Firefox and Google Chrome; see Figure 3 It is different than the other systems that we analyzed in that the annotation doesn't belong to the text but to the user. After installing the extension, the use can navigate to any page and can highlight any piece of text. The annotation is saved in the user's google account and is displayed every time the user accesses the page.

Feature-wise the system is much weaker than the previously analyzed brat (and much older), however it contains a potentially useful idea: user-specific annotations which can later be made accessible only to specific users or groups of users.

Figure 2: Search in brat

2.3   Annotatie Systeem Annotation Tool

Annotatie [AN] is an annotation tool for printed documents and belongs to the static annotation category.

The anchoring of the annotations is done by page positioning: this seems inflexible at a first glance, however, it constitutes a good alternative when the text in the page is not digitized. The feature that caught our attention is the extensive comment sectioned derived from the annotations:

- Each annotation represents a new comment thread.

- In each comment thread other users can further discuss on both the contents of the annotated text and the annotation itself.

- All the annotations in a page are displayed in the right side of the page, as collapsed threads.

We believe that allowing discussion based on one user's annotation is an important feature in such a system, as the context introduced by annotations is one which emphases user collaboration.

3   KAT System & Information Architecture

3.1   KAT Annotation Data Model

The annotations that the users will provide must be well structured in order for our third requirement to be accomplished. This assures us that the work of annotators can best be
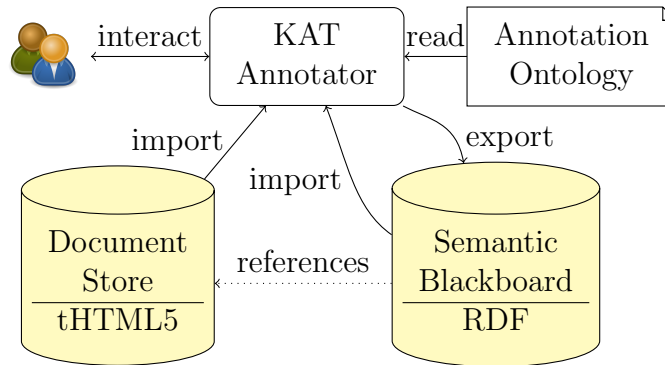
Figure 3: Annotation in Yawas

Figure 4: The KAT System Architecture

put to use by automatic processors or computer interpreters.

A good example of a use case is a self-learning software that translates text fragments from one language to another. For it to function properly it needs to know the grammatical structure of the languages and here is where annotation tools can help. Users can annotate predefined sentences or paragraphs and identify specific parts of speech (e.g. a noun) that the translation tool software developers can then use as a training material for their machine-learning product.

Our system is flexible enough for administrators to define how annotations should be structured and what connections can be made between different annotations. Furthermore this flexibility is also be applicable in the visual interfaces with which users interact. In a typical system you will have users coming from different backgrounds with various skills and interests. We believe that this variety should not be suppressed but rather encouraged by allowing for multiple annotation interfaces to be presented to the user so that he can choose the best suited one for his contributions. This feature helps projects where crowdsourcing is detrimental by making the users comfortable with the UI and allowing them to become proficient without a steep learning curve.

3.2 Concept Architecture

At the center of our system is the **annotation concept**, several of which being described by a user-supplied ontology. The format in which the ontology should be provided is the standard OWL model. Each annotation concept should correspond to an OWL instance that is a member of the annotation class to which the following elements should be added:

- fields - this section describes how a user can input the necessary information for the annotation to be valid according to its concept. The section should contain for each field describing the annotation concept an entry that describes how a user can populate this information in a form.

  Each child of the entry should be of the following form:

  - filed - the field wrapper, all further options are children of this element. It has *name* and *type* as attributes. The *type* can be one of: text, select, reference, radios or text area.
  - documentation - further information about the field, to be displayed to the user.
  - value - a default value for the field.
  - validation - a regular expression that the user input must match.
  - number - a field having 2 attributes: *atleast* and *atmost*, indicating how many inputs of this type can the user make in the same annotation.
  - options [only available for field type = "select"] - the options from which the user should choose from. Each new option is an individual element having children of type *documentation* and *value*.
  - referencedType [only available for field type = "reference"] - indicates the concept name that should be referenced by this input.

- display - this section describes how an annotation is displayed. It consist of the following fields:

  - template - an HTML string containing the desired display format of the annotation fields. Each annotation field that is mentioned between curly braces will be replaced by the actual value of the field.
  - style - a list of CSS valid rules to be applied to the annotation display.

In Appendix A2 you can find an example of an annotation ontology. It defines 2 concepts which allow the user to annotate text as Symbols or definitions of Symbols. As you can see in the example, the model gives us the necessary flexibility at the administrator level while at the same time providing for a clear customizable method of displaying the annotation form and representation for the user.

4 Annotation Workflows

4.1 Adding an annotation

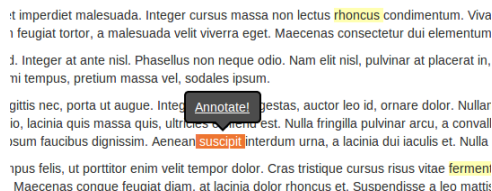The following steps must be followed for adding a KAT annotation:
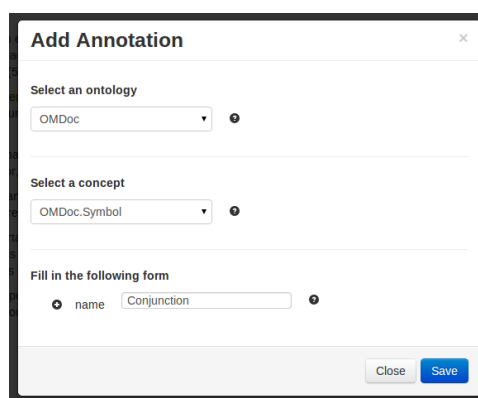

Figure 5: Selecting the word "suscipit" for annotation

Figure 6: Annotating "suscipit" as an OMDoc.Symbol with the name "Conjunction"

1. The target text of the annotation is highlighted (by simple selection), see Figure 5

2. The first step triggers an annotation-menu.

3. The annotation-menu is populated with the available annotation types, given by the available ontology.

4. The user clicks on the desired annotation concept.

5. A modal box containing the annotation form opens.

6. The annotation form is populated with the fields indicated in the ontology, see Figure 6

7. After completing the form fields, the user saves the annotation.

This approach ensures 2 things:

- The navigation flow is never interrupted: adding an annotation is done via pop-up and modal boxes and windows, so after completing the steps, the user finds himself in the same place in the document.

- Annotation type checking is implicit: the annotation form is populated with the fields indicated in the ontology so the user doesnt have to check the required types himself, see Figure 7

Figure 7: Annotating a definition for "suscipit". In the ontology, the OMDoc.Definition concept is defined to be for annotations of type Symbol. The "for" field is automatically populated with all the annotations of this type.

4.2 Displaying an annotation

Annotated text is marked with special CSS style so it becomes easily identifiable. The type of the annotation is also displayed above the annotation itself so one can have a clear overview of the whole document.

At click, a pop-up window containing the annotation display opens.

The annotation display consists of the following elements:

1. The type of the annotation.

2. The formatted annotation input according to the rules described in the ontology.

3. For the annotations of type reference, an arrow indicated the referenced element.

4.3 Reviewing annotation versions

Reviewing is done in a side-by-side view which allows the user to see two versions of the same document in order to see discrepancies and correct them; see Figure 9. The viewer
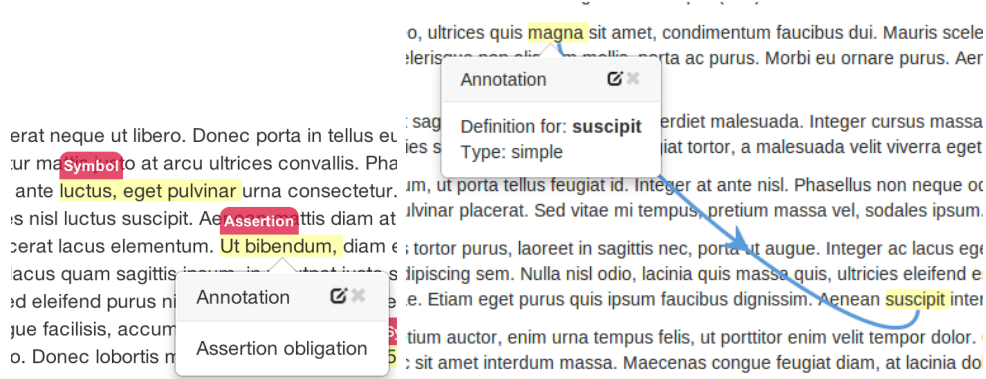
Figure 8: Display for simple text and reference annotations

allows to toggle view one ontology at a time which can be toggled individually on each panel by clicking on the.

While reviewing, the left panel (also called the main panel) is the current document with the local annotations while the right panel (referred to as the mirror panel) is the the different version of the document being reviewed against.

KAT is only enabled in the main panel so changes/edits/annotations can still be done while reviewing, while the mirror panel is read-only. Scrolling the main panel will cause the mirror panel to automatically follow suite.
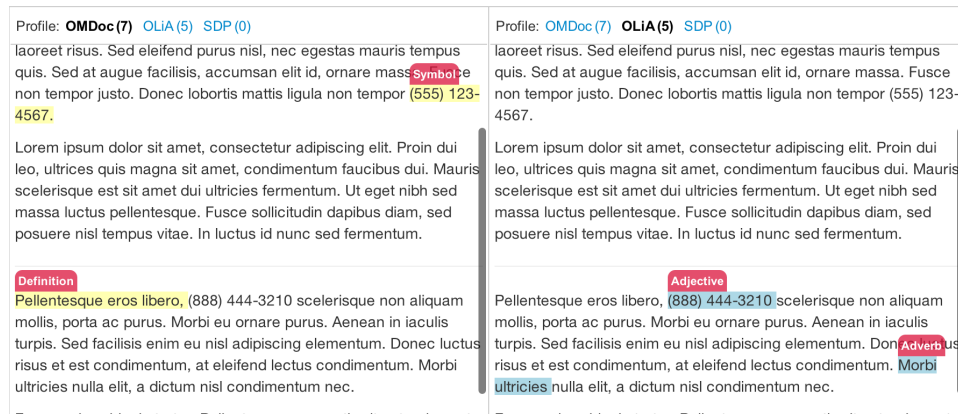
Figure 9: Reviewing two ontologies of the same document side-by-side

5   Storage

The annotation tool is storage agnostic per-se. As no back-end platform is provided alongside the application, administrators are free to develop their own custom workflows of storing an annotation. By default annotations are stored in the user's browser database (or localStorage if the browser is older) and can be automatically transferred to a given

URI through a POST request. Annotations can also be loaded through GET requests from a configurable URI or from the user's local storage.

Although the system is flexible in this regard, we recommend the storage of annotations in a triple store as it provides a nicely continuity with the existing ontology based model. The annotations can be easily represented as RDF structures and the application itself stores it as so internally. Furthermore we believe that graph database are best suited for storing and querying this kind of metadata.

References

[AN]   *Annotation tool.* URL: Http://www.annotatiesysteem.nl (visited on 02/15/2014).

[BR]   *brat rapid annotation tool.* URL: htp://brat.nlplab.org (visited on 02/15/2014).

[CG]   GitHub repository. URL: https://github.com/dginev/CorTeX/.

[CT]   *CorTEXFramework.* URL: http://cortex.mathweb.org (visited on 02/15/2014).

[Fre91] Free Software Foundation. *GNU General Public License.* 1991. URL: http://www.gnu.org/copyleft/gpl.html (visited on 05/22/2013).

[KG]   GitHub repository. URL: https://github.com/KWARC/KAT/.

[YW]   *Yawas - The Original Web Highlighter.* URL: http://www.keeness.net/yawas/ (visited on 02/15/2014).

A Installation Guide

In order to deploy KAT, please follow the next steps:

1. In Makefile.in, change INSTALLDIR to the desired installation directory (e.g. /srv/http/kat or /var/www/kat).

2. Rename Makefile.in into Makefile. Run make all && make install.

3. You will find a proof of concept demo at `http://localhost/katInstallDir/test`.

4. In order to add annotations, you need to add an annotation ontology first. You can do that in the KAT Control Panel (link in the bottom-right of the page). For starting you can copy the example you find in test/annotations.xml.

5. The default storage environment is the browser's local storage. An API to connect to different storage environment can be found in the classes from the remote/ directory.

B An Annotation Specification

[3]   EdN:3

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- A KAT annotations specification for OMDoc annotations
        copyright 2013 Michael Kohlhase released under the GPL -->
<annotation xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:o="http://omdoc.org/ontology#" name="OMDoc">
  <documentation>
    The OMDoc ontology is a flexiformal data model for matheamtical (and
    general STEM) documents
  </documentation>
    <concepts>
      <concept name="Symbol">
        <documentation>An OpenMath/OMDoc Symbol</documentation>
          <fields>
            <field name="name" type="text">
              <documentation>
                The name of the symbol defines it in a theory
              </documentation>
              <value>Name</value>
              <default>Symbol</default>
              <validation>[A-Z][a-z]*</validation>
              <number atleast="1" atmost="2"></number>
            </field>
          </fields>
          <display>
            <template>
```

EdNote: MK: explain this and the features it corresponds to

```
            <b>Symbol:</b>
            <br/>
            {name}
          </template>
        </display>
        <rdf:RDF>
          <rdf:Description>
          <rdf:type rdf:resource="http://omdoc.org/ontology#Symbol"></rdf:type>
            <o:symbolname>{name}</o:symbolname>
          </rdf:Description>
        </rdf:RDF>
</concept>

<concept name="Definition">
  <fields>
    <field name="for" type="reference">
      <value>For</value>
      <referencedType>OMDoc.Symbol</referencedType>
      <number atleast="1"></number>
    </field>
    <field name="type" type="select">
      <options>
        <option default="true">
          <value>simple</value>
          <documentation>
            Definiens is does not contain the definiendum
          </documentation>
        </option>
        <option>
          <documentation>
            definiendum applied to formal variables
          </documentation>
          <value>pattern</value>
        </option>
      </options>
    </field>
  </fields>
  <display>
    <template>
      <b>Definition</b>
      <br/>
      {for}
      <br/>
      {type}
    </template>
  </display>
  <rdf:RDF>
    <rdf:Description>
      <rdf:type rdf:resource="http://omdoc.org/ontology#GeneralDefinition">
      </rdf:type>
      <o:defines rdf:resource="{for}"></o:defines>
```

```
        </rdf:Description>
      </rdf:RDF>
    </concept>
  </concepts>
</annotation>
```

C   Developer Guide

KAT is written using FlancheJS: a simple library that provides javascript classical inheritance. The classes are structured java-like, having:

- constructors: provided by the method *init*.

- properties: the library automatically generates getters and setters.

- methods: publicly available functions.

- internals: properties and methods available only by underscore (_) prefixing.

The table below describes the KAT classes (available under **src/js/**) and their behavior:

Class Name	Class Description
main namespace:	
kat.main.KATService	The main entry point of the service. The KAT Service requires a CSS3/XPATH selector to identify the container on which annotations can be made, and optionally a CorTeX service URL [CG; CT] and a document identifier for the annotated document.
annotation namespace:	
kat.annotation.Annotation	Describes an annotation that was collected from a user and can be saved and transported over network.
kat.annotation.AnnotationRegistry	Describes an annotation registry that keeps track of all the annotations for the current document
kat.annotation.Concept	Class to describe an annotation concept. Annotation concepts describe the annotation model (i.e. the fields contained by the annotation) and the behavior of the annotation (i.e. user interaction and display).
kat.annotation.ConceptRegistry	A registry to keep track of all available concepts for this document.

kat.annotation.Ontology	Class to describe an annotation ontology. Annotation ontologies describe the annotation concepts.
kat.annotation.OntologyRegistry	A registry to keep track of all available ontologies for this document.
kat.annotation.AnnotationForm	Class for handling the form displayed when an annotation is added.
display namespace:	
kat.display.AnnotationEditForm	Class for handling the form displayed when an annotation is edited.
kat.display.ControlPanel	This class provides a tool for displaying and handling the KAT Control Panel.
kat.display.AnnotationRenderer	Class for handling the display of a singe annotation.
kat.display.ArrowConnector	Creates an svg arrow that can be used to connect two DOM elements, for example a reference field annotation to the referenced item.
kat.display.Display	Creates and controls the annotation displays.
Class Name	**Class Description**
input namespace:	
kat.input.Form	The Form class decides which fields to be displayed in the current form.
kat.input.FormParser	A form parser can be used to parse the fields and documentation from a given concept object.
kat.input.FieldParserRegistry	The Field Parser Registry contains all the field parsers that are available to parse for a concept.
input.fieldparser namespace:	
kat.input.fieldparser.FieldParser	A field parser parses an annotation:field into an html string. This trait serves only as an interface that the extending classes follow.
kat.input.fieldparser.Checkboxes	Parses a field of type checkboxes outputting html.
kat.input.fieldparser.Reference	Parses a field of type reference outputting html.
kat.input.fieldparser.Select	Parses a field of type select outputting html.
kat.input.fieldparser.TextArea	Parses a field of type text area outputting html.
kat.input.fieldparser.TextField	Parses a field of type text outputting html.

input.fieldparser namespace:	
kat.input.view.Form	Class that renders an annotation form containing the fields described in the concept.
kat.input.view.ConceptSelector	Class to describe an input element in the form container that is used to select a concept to be used in the annotation form.
kat.input.view.FormContainer	Describes a class that acts as a container for an annotation form and a concept selector.
preprocessor namespace:	
kat.preprocessor.TextPreprocessor	Used to add counters around text but this functionality has been deprecated. Now it only adds selection listeners in the text, for adding annotations.
remote namespace:	
kat.remote.CoreTexAnnotationInserter	Sends the annotations being created on this document to the CoreTeX system.
kat.remote.CoreTeXAnnotationReceiver	Retrieves the document and the annotations from the CoreTeX service and populates the internal registry.