# KAT: an Annotation Tool for STEM Documents

Deyan Ginev, Sourabh Lal, Michael Kohlhase, Tom Wiesing

Jacobs University Bremen

**Abstract.** Contemporary natural language processing (NLP) systems are based on corpora of annotated documents for training and evaluation. To extend NLP to documents from Science, Technology, Engineering, and Mathematics (STEM) we need annotation systems that can deal with structured elements like mathematical formulae, tables, and possibly even diagrams. Current linguistic annotation systems treat documents as word sequences and disregard the structure of complex document elements, and are therefore unsuited for STEM annotation as this very structure carries important syntactic and semantic information.
We present the KAT system, a browser-based annotation tool for linguistic/semantic annotations in structured (XHTML5, i.e. HTML + MathML + SVG in XML serialization) documents. As KAT is parametric in the annotation ontology and represents annotations as RDF, it can easily be integrated into RDF-based corpus management systems; we present an integration into the CorTeX system.

## 1 Introduction

Natural language processing systems need a "gold standard" – a corpus of annotated documents – for training and evaluation. Gold standards are created by manually annotating a corpus (often a subcorpus of an existing larger one). Annotations are metadata about a document, and do not actually alter the content of the document. They can be thought of as a layer on top of a document which contains information about the text in the document. Managing the annotations in a document or a corpus requires an annotation tool. There are many other use cases for annotating documents (see [San] for a collection), we will concentrate on the maths linguistic use case here. Conventional linguistic annotations might consist of "part of speech (POS) tags" (syntactic annotations) or crossrefs from words into wikipedia (for semantic dismbiguation/grounding). Corpora are usually annotated my multiple annotators and are evaluated for inter-annotator agreement.

### 1.1 Document Annotation Tools

Currently there are several annotations tools available to use for online text annotations including Hypothes.is [HYP], brat [BR], Yawas [YW]

and Annotatie [AN]. Annotation tools can be categorized according to the type of annotations they make. Generally state of the art annotation tools create one of the following types of annotations:

1. *Dynamic Annotations* - Annotations that are anchored to the text of the document.
2. *Static Annotations* - Annotations that are anchored to a particular position in the page of the document.

For the propose of linguistic annotations of STEM documents only dynamic annotations are useful, so we will disregard static annotators like Annotatie [AN]. For dynamic annotation tools we can distinguish two classes:

1. *web highlighters* (also called social bookmarking systems) that aim to "add a new layer to the web", in which users can share and discuss comment on document fragments, and
2. *structured annotation tools*, which classify text fragments and relate them to each other according to a fixed or flexible ontology.

We will now briefly present one paradigmatic system from each category:

*Hypothes.is* Hypothes.is [HYP] is a is is an online, dynamic annotation tool that can highlight and annotate pdfs as well as web pages. Annotations are free-form comments that can be stored locally or published/shared. If hypothes.is is enabled on a PDF or web page, private and shared annotations can be viewed. Hypothes.is provides annotation management features such as the ability to make an annotation public or private, and discussing annotations.

*brat* The "brat rapid annotation tool" [BR] is a web based structured annotation tool for text documents. It is designed to create annotations that have a fixed form that can be automatically processed and interpreted. brat can be used to create both regular text span annotations, as well as relational annotations that connect two regular annotations. brat provides several annotation management functionalities, e.g.

1. an Advanced annotation searching tool.
2. An annotation export interface that can convert the internal storage format to PDF or HTML.
3. Unique address to access each annotation.

While they have many important functionalities also needed for our application, both classes of tools are insufficient for creating linguistic gold standards for STEM documents. Web highlighters do not support structured annotations that are needed for syntactic/semantic analysis, and structured annotation systems are restricted to un-structured text

documents – the adjective "structured" only applies to the structure of annotations.

### 1.2 Modalities of STEM Documents

For understanding the syntactic/semantic structure of STEM documents, we need to be able to make use of the complex document structure. We will point out some of the phenomena a STEM-linguistic annotation tool (SLAT) needs to annotate to act as a set of requirements.

*Formulae* STEM documents contain structured formulae whose parts carry meaning. Examples include mathematical and chemical formulae, e.g.
1) "For each $\epsilon > 0$"
2) "..., then $\epsilon^2/7 > \delta$..."
3) "The OH group in  (ethanol)..."
We see that the formulae have a role in the syntactic structure of the sentences, and even though the formulae in 1) and 2) are very similar syntactically, the first introduces an identifier ($\epsilon$), whereas the second one is a proposition. In 3) the formula "OH" contributes to the definite description that acts as a subject to the sentences and references a part of the formula displayed later. A SLAT needs to be able to annotate formula fragments, e.g. the $\epsilon$ as a "declarandum" in the declaration $\epsilon > 0$ in 1). A necessary precondition for that is that the format for representing annotations (the annotation format) can reference formula fragments.

*Tables, & Diagrams* act as visual short forms of or supplementary elements for explanatory text. Just as in formulae, they have grammatical roles, and their parts are referenced in the text or elaborated on. For instance, Figure 1 introduces a KannSpec, which is later defined and explained in Section 2.2. Assuming that tables and diagrams are given as structured representations – e.g. as group-structured vector graphics or in row/column markup, we need a fragment representation format that can deal with the structural components of tables and diagrams.

*Images, Quotes & Listings* are visual presentations of external objects or artefacts employed to help readers better understand document content. Even though they are usually not themselves structured objects, they contain regions that form syntactic and semantic structures to the reader and need to be annotated. For these, a SLAT may have to delegate annotation to a static annotation tool.

### 1.3 Requirements for Linguistic Annotations for Structured Documents

A SLAT needs to be able to cope with all the modalities introduced above. Some of the structures to be annotated will already be explicitly representable in the document format – and maybe even represented in the document (an instance of the document format), while some will have to be identified in the annotation process itself. Both need to be available as arguments for subsequent annotations.

An ontology is a specification of a set of concepts and their properties/relations, an annotation ontology is one whose concepts concern meaning-carrying structures in documents or the structure of the knowledge conveyed in them. Suitable Ontologies range from a part-of-speech ontology [OLiA] that provides basic syntactic classes to the OMDoc ontology that concentrates on semantic structures [Lan11].

As research on the syntactic/semantic structures in STEM documents are still in a relatively early state, it is too early to fix ontologies, therefore a SLAT should be parametric in the annotation ontology to be an effective research tool.

A SLAT should natively support STEM document formats that 1. can encode all the modalities above in a structured form and 2. that support fragment identifiers for the structured and unstructured components. XML-based formats are well-suited, since they have standardized fragment identification languages like XPath [XPa10] and XPointer [Gro+03]. As these can be embedded into URIs, we can directly use RDF [SR14] for the representation of annotations. RDF has the additional advantage that the triple-of-URIs representation can directly make use of the concepts and relation of the annotation ontology, if that is given in RDFS or OWL format.

For STEM documents, the XHTML5 [Hic+14] format seems like a good basis: HTML5 augments the HTML document model (which covers tables, images, quotes, and listings) with MathML [Aus+10] for mathematical formulae and SVG [Dah+11] for diagrams, and XHTML5 is the XML serialization. As (X)HTML5 is supported by all major browsers – MathML coverage varies though – we can implement an annotation tool in the style of web highlighters: as a JavaScript library that can be embedded into web pages.

In the next section we will present the KAT (KWARC Annotation Tool) system that is based on these ideas. Section 3 concludes the paper.

## 2 The KAT System Architecture & Implementation

KAT is a browser plugin for XHTML5 (i.e. HTML + MathML + SVG in XML serialization) documents and generates a set of RDF triples that represent the annotations provided by the user as output. The system is implemented in the form of a JavaScript library that expects tokenized[1] (sentences and words are marked up in `h:span` elements for structure) XHTML5. It can easily be embedded into an existing web-page. As KAT is based on XHTML5, we can employ the XML tool chain and rely on standard libraries for the implementation.

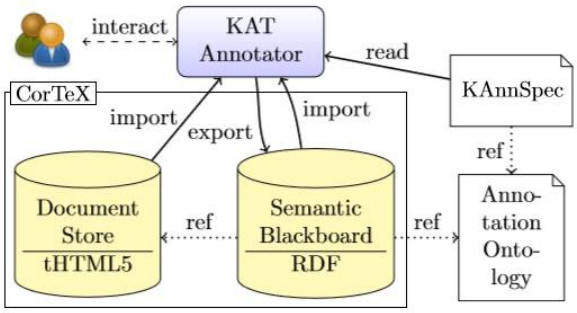


**Fig. 1.** The KAT System Architecture

The KAT system architecture can be seen in Figure 1. Even though KAT can work as a standalone library that can be added to any STEM document in HTML5, it is best used as a component of a corpus management system, such as the CorTEX system [CT]. In this situation the user requests an annotation task from CorTEX, which serves the TEI-tokenized document set from its document store and instruments it with the KAT library. The user can then annotate the document. As the annotation interface is parametric in the annotation ontology KAT needs an UI specification to specialize the user interaction. This is provided by the *KAnnSpec*s that specifiy a UI aspect and reference an annotation ontology. Once the annotation process is complete, KAT can export annotations in RDF/XMLto the semantic blackboard – an RDF triple store – maintained by CorTEX as an annotation server for storage, management, linguistic analysis and content harvesting.

[1] While KAT could work with non-tokenized documents it is much easier to work with documents that are tokenized up to the word level. In this case it is sufficient for KAT to store fragments of the documents that are annotated as a range of XHTML elements instead of having to go down to the character level (see Section 2.1). Furthermore, re-implementing an existing TEI-tagger in JavaScript would have provided unnecessary effort.

### 2.1 Representing Annotations

Each annotation in the system is bound to a fragment of the XML document – a contiguous range. This range can contain any type of content – text, formulae, tables, diagrams or images for example. Since we are working with word-tokenized XML documents and words are the smallest meaning-carrying document fragments, we can use XPaths [XPa10] specify such ranges instead of XPointer, which also offers ranges in text nodes, but is much less implemented. Since the range we specify is contiguous, it is sufficient to give XPaths to the first and last elements to locate it. Because the KAT system eventually has to find all elements contained in the selection we additionally give an XPath to the least common ancestor of both elements. We then combine all these paths with the document URL to generate a single URI of the form `doc#cse(con,start,end)` – where `doc` is the document url, `con` the XPath to the container, `start` and `end` the pointers to the first and last element of the selection respectively.

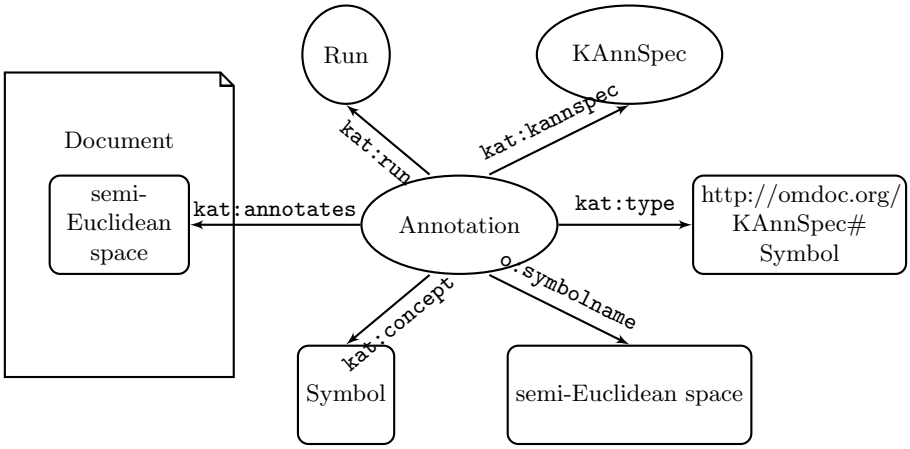


**Fig. 2.** An Annotation Graph

An annotation connects a document fragment with the annotation content. In Figure 2 we give an example of this. We can see that the text *semi-Eucledian space* has an annotation linked to it. In the KAT system every concept in an ontology has a list of fields that act as properties. The ontology is specified in the KAnnSpec and the one we are using here will be explained later in the context of Listing 1.1. The concept *Symbol* used here has a single field, given by the *o:symbolname* relation. The

name of the annotation is *semi-Eucledian space*. In general, fields can be instantiated multiple times per annotation and some can have additional constraints.

### 2.2 KAnnSpecs As Ontology Descriptions

KAT is not tied to a particular annotation ontology (or ontology format). At startup, the system is reads a set of *KAnnSpecs* – custom XML files that describe the annotation interface, the constraints between the components of an annotation frame, and the RDF to be produced. The Concept of OMDoc symbols used in Figure 3 specifies *i*) the fields of the annotation form, their values and validation constraints, *ii*) their display, and *iii*) RDF attributes required to create RDF triples. Note that this KAnnSpec classifies the annotated word as an OMDoc symbol (via the rdf:type predicate) and relates it to its name via the o:symbolname relation from the OMDoc ontology.

In Listing 1.1 we show part of such a KAnnSpec and define the OMDoc symbol. First we give this concept a name and a basic description. In addition, we give it an RDF type. This will be used in RDF export later. Next, we declare that this concept has a text field *name*, representing the name of the symbol to be declared. We give this a basic description and an RDF predicate *o:symbolname* which comes from the annotation graph (Figure 2). Futhermore we give the field a default value and state that its value is valid if it contains any kind of text. Finally we need to declare that the field occurs exactly once for each concept.

**Listing 1.1.** A KAnnSpec concept declaration for an OMDoc Symbol

```
<concept name="Symbol" rdftype="o:Symbol">
  <documentation>An OpenMath/OMDoc Symbol</documentation>
  <field name="name" type="text" rdfpred="o:symbolname">
    <documentation>
       The name of the symbol defines it in a theory.
    </documentation>
    <value>Name</value>
    <default>Symbol</default>
    <validation>.*</validation>
    <number atleast="1" atmost="1"/>
  </field>
  <display>
    <template><b>Symbol:</b> <br/> {name}</template>
  </display>
```

```
15 </concept>
```

### 2.3 Exporting Annotations As RDF

We use the Resource Description Framework [SR14] to export and import annotations. In particular we use RDF triples – subject/predicate/object triples – to represent annotations. The subjects are the location of the annotation, the predicates are defined in the KAnnSpec and the objects are the field values – either text or the URI to an annotation.

In Listing 1.2 we show a sample of how an annotation is exported to RDF. Each annotation consists out of a single node (lines 6-12). Additionally, each annotation has meta-data, such as the used KAnnSpec, which is omitted here. To generate this node, we use the properties of the annotation graph shown above in Figure 2.

We first give the range of the document that is being annotated. We make use of the *kat:annotates* relation (line 5). We use the XPath format as described above – in this case the URI `https://kwarc.github.io/KAT/content/sample1.html#cse(//*[@id='sentence.11'],//*[@id='word.202'],//*[@id='word.203'])`[2] with ids *word.202* and *word.203*. Next we use the concept of a *kat:run* (line 6). This is intended to provide meta-information such as when and how this annotation was generated, this has been omitted from the listing.

Continuing, we give the concept that the annotation uses. We first reference a KAnnSpec (line 7) and then a concept (line 8). We further specify the type of the annotation with the *kat:type* relation (line 9). This attribute is specified in the KAnnSpec.

Finally, we provide all the fields and their values. In this case, we just give the concept the name *semi-Euclidean space* (line 10).

**Listing 1.2.** Exported RDF generated for a single annotation of an OMDOC Symbol

```
1 <rdf:RDF xmlns:o="http://omdoc.org/KAnnSpec#" xmlns:rdf="
    http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:kat="
    https://github.com/KWARC/KAT/">
2   <!-- omitted a lot of meta-information here -->
3
4   <rdf:Description rdf:nodeID="KAT_1433087821332_4477">
```

[2] Actually, the XPath specification would allow bare name paths – e.g. `#word202` here, but external library we currently use does not support this form.

```
5       <kat:annotates rdf:resource="https://kwarc.github.io/KAT
            /content/sample1.html#cse(%2F%2F*%5B%40id%3D'sentence
            .11'%5D%2C%2F%2F*%5B%40id%3D'word.202'%5D%2C%2F%2F*%5
            B%40id%3D'word.203'%5D)" />
6       <kat:run rdf:nodeID="kat_run"/>
7       <kat:kannspec rdf:nodeID="KAT_1433087757661_OMDoc"/>
8       <kat:concept>Symbol</kat:concept>
9       <kat:type rdf:resource="http://omdoc.org/KAnnSpec#Symbol
            " />
10      <o:symbolname>semi-Euclidean space</o:symbolname>
11    </rdf:Description>
12
13  </rdf:RDF>
```

### 2.4 Making Annotations Inside The Browser

Because KAT runs inside the browser, the annotation workflow itself is form-based (see Figure 3). All forms are displayed inside a sidebar that can be hidden in order to focus on the content. The user interface has two modes: An annotation mode, in which annotations can be created, edited and deleted, and a reading mode, which can be used to view existing annotations. The modes can be switched either via the context menu or a button in the sidebar.

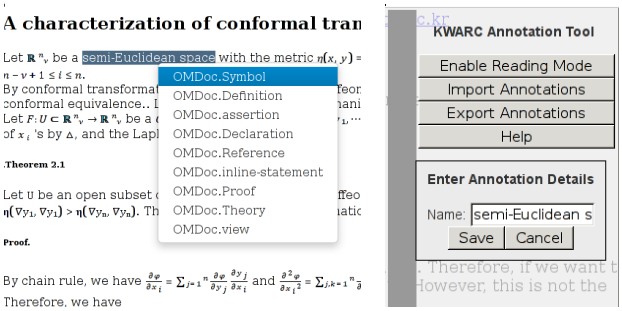


**Fig. 3.** Annotating in KAT: Selection and Form-Filling

The annotation process itself is shown in Figure 3. The annotator selects a text range and then uses the context menu to select an annotation type to create and then fills out a form inside the sidebar with classifica-

tions and relations as given by the KAnnSpec. Here we specifically show the *Symbol* that has been exported in Listing 1.2.

Once an annotation has been created, it is highlighted in the document. If in annotation mode, it can be edited or deleted via the context menu. This menu can be seen in Figure 4.

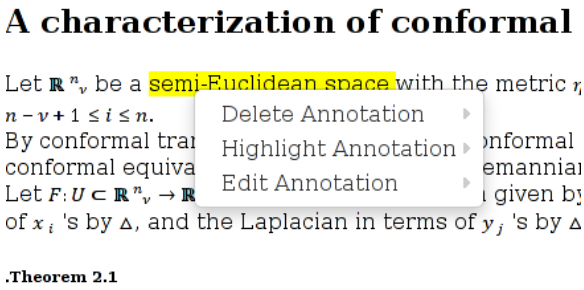


**Fig. 4.** Context Menu Of An Existing Annotation in Annotation Mode

Importing from and exporting to RDF can be achieved via the buttons in the context menu or the sidebar.

## 3 Future Work & Conclusion

We have presented the KAT system, an open, parametric, and browser-based annotation system for STEM documents encoded in XHTML5. The code base is released under the Gnu Public License and is available at [KG].

The KAT system has been used for annotating mathematical documents in a computational linguistics course at Jacobs University. We used the OMDoc ontology for direct semantic annotations and an experimental ontology for declarations ("for all $\epsilon > 0, \ldots$" or "$f$ be a smooth real function") as a test case. For these cases the system was practically usable.

While KAT has a fully functional user-interface, there are still some aspects that can be further improved upon. These range from both new features to minor tweaks to enhance the user interface. While some of these ideas can be developed immediately, others require further back-end development before implementation. The three most pressing ones are:

1. *Display the information about an annotation* – Once an annotation is created, currently there is no way to see what details the user has specified for it. The KAnnSpec has a `<template>...</template>` that will be used for this in the future. A tooltip will be provided when an annotation is hovered upon.
2. Visualization of relational annotations can also be improved. An ideal solution would be to display an arrow connecting the two annotations when either annotation is hovered upon. As it is difficult to draw (and compute positions of) arrows on XHTML pages, a simpler solution would be to integrate this into the tooltip.
3. *Distinction between annotations* - Currently, all annotations are marked in the same way - making it difficult to distinguish two annotations if they are overlapping. In order to solve this problem, annotations could be color coded depending on the concept they represent.

We want to integrate KAT with the CorTeX system [CT] as described above. We want KAT to mainly serve two purposes in this sense:

1. *to annotate individual documents* within a larger corpus and provide users with an interface to do so and
2. *to review annotations* which were either performed automatically or by another user.

For both purposes we need to be able to distribute work between multiple instances of KAT and CorTeX. The main task in this is to implement a JavaScript client for the Gearman Job Server [Gea] underlying CorTeX; once we have completed this, distributed processing and corpus support in KAT are automatic.

The more immediate next steps will be to refine our annotation ontologies and integrate more linguistic ontologies to get more coverage. The annotation ontologies might benefit from being compatible with the open annotation data model [SCS13]. Note that the KAT system is compatible with this, since it is parametric in the annotation ontology.

## References


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

[Aus+10] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 3.0*. W3C Recommendation. World Wide Web Consortium (W3C), 2010. URL: `http://www.w3.org/TR/MathML3`.

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

[CT] *CorTEX Framework*. URL: http://cortex.mathweb.org (visited on 02/14/2014).

[Dah+11] *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), Apr. 2011. URL: http://www.w3.org/TR/SVG11.

[Gea] *Gearman Job Server*. 2015. URL: http://gearman.org/ (visited on 06/19/2015).

[Gro+03] Paul Grosso et al. *W3C XPointer Framework*. W3C Recommendation. World Wide Web Consortium (W3C), Mar. 25, 2003. URL: http://www.w3.org/TR/2003/REC-xptr-framework-20030325/.

[Hic+14] Ian Hickson et al. *HTML5. A Vocabulary and Associated APIs for HTML and XHTML*. W3C Recommentation. World Wide Web Consortium (W3C), Oct. 28, 2014. URL: http://www.w3.org/TR/html5/.

[HYP] *Hypothes.is*. URL: http://hypothes.is (visited on 05/30/2015).

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

[Lan11] Christoph Lange. *The OMDoc Ontology*. Jan. 8, 2011. URL: http://kwarc.info/projects/docOnto/omdoc.html (visited on 02/03/2012).

[OLiA] *OLiA Ontologies*. URL: http://nachhalt.sfb632.uni-potsdam.de/owl/ (visited on 11/09/2013).

[San] Robert Sanderson, ed. *Annotation Use Cases*. URL: http://w3c.github.io/dpub-annotation/ (visited on 05/30/2015).

[SCS13] Robert Sanderson, Paolo Ciccarese, and Herbert Van de Sompel, eds. *Open Annotation Data Model*. 2013. URL: http://www.openannotation.org/spec/core/.

[SR14] Guus Schreiber and Yves Raimond. *RDF 1.1 Primer*. W3C Working Group Note. World Wide Web Consortium (W3C), 2014. URL: http://www.w3.org/TR/rdf-primer.

[XPa10] *XPath Reference*. 2010. URL: http://www.w3.org/TR/xpath/ (visited on 06/05/2010).

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