
GIAnT Documentation

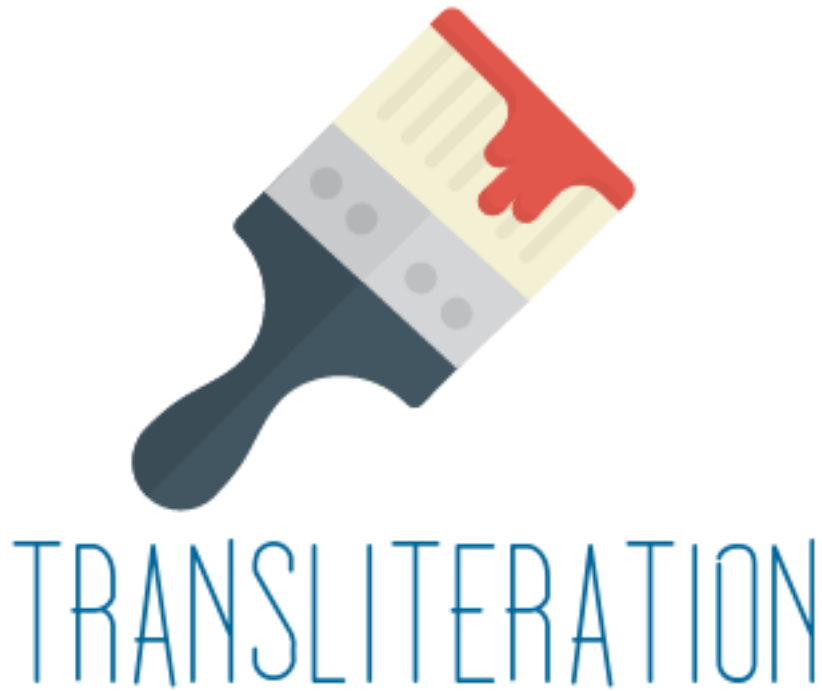
Daniel Pollithy

May 01, 2019

1	Video	3
2	GIANt resources	5
3	Guides	7
3.1	Guides	7
3.1.1	Feature overview	7
3.1.2	Installation	11
3.1.3	Login to neo4j	11
3.1.4	Getting started	12
3.1.4.1	Uploading an image	12
3.1.4.2	Creating a fragment	13
3.1.4.3	Transfer to neo4j	14
3.1.5	Using the editor	14
3.1.5.1	Overview	14
3.1.5.2	Layers	14
3.1.5.3	Nodes	15
3.1.5.4	Relations	16
3.1.5.5	Properties	17
3.1.6	Data scheme in neo4j	18
3.1.6.1	Images	19
3.1.6.2	Fragments	19
3.1.6.3	Tokens	21
3.1.6.4	Singular Tokens	21
3.1.6.5	Groups	22
3.1.6.6	Edges	23
3.1.6.7	Using Cypher	24
3.1.7	Heatmap tool	24
3.1.7.1	Normalization techniques	24
3.1.7.2	Normalization 1: Position in image	24
3.1.7.3	Normalization 2: Position in scritte (bounding box)	25
3.1.7.4	Normalization 3: Bounding box centered	26
3.1.7.5	Performance	27
3.1.8	Data constraints	28
3.1.8.1	The workflow	28
3.1.8.2	Design of the constraints	29
3.1.8.3	Writing constraints	29

3.1.8.4	Example for count constraint	29
3.1.8.5	Example for a free constraint	29
3.1.8.6	Security	30
3.1.9	Exporting your data	30
3.1.10	Transferring your data	31
3.1.10.1	1. Move static files	31
3.1.10.2	2. Relocate the Neo4j database	31
3.1.10.3	2/a Dump and load Neo4j (UNIX)	31
3.1.10.4	2/b Second variant for neo4j (WIN)	31
3.1.11	Installing an update	32
3.1.11.1	Compatibility	32
3.1.11.2	How to	32
3.1.11.3	Keeping your data	32
3.1.12	Accessing your data on multiple devices	34
4	Articles	35
4.1	Articles	35
4.1.1	Analyzing 3000 graffiti	35
4.1.2	Cypher vs. SQL	38
4.1.2.1	Example scenario	38
4.1.2.2	Storing the graph	41
4.1.2.3	Retrieving some data	42
4.1.2.4	Extending the example	42
4.1.2.5	Querying just a little more	43
4.1.2.6	Result	43
4.1.3	Possible use cases	43
4.1.4	Continuous integration: Testing and deploying code	44
4.1.4.1	How to write a test	44
4.1.4.2	Contributing	45
4.1.4.3	Documentation	45
4.1.5	Details on the CI	45
4.1.5.1	Trigger an automatic build	45
4.1.5.2	How to Deploy manually	45
4.1.5.3	Deployment on Travis-CI	46
4.1.5.4	What is Travis-CI	46
4.1.5.5	Setup Travis-CI	46
4.1.5.6	What does Travis do?	46
4.1.5.7	Detailed description of the Configuration	46
4.1.5.8	Deployment on AppVeyor	47
4.1.6	Developing a Codec: Test driven development	48
4.1.6.1	The GraphML format	49
4.1.6.2	How can we verify that the codec works as wanted?	50
4.1.6.3	What is the biggest XML file it can transform	50
4.1.6.4	Batch-add (+checksum)	50
4.1.7	Electron for cross platform applications	50
4.1.7.1	Overview of cross platform frameworks	50
5	Technical Documentation	51
5.1	Technical guides	51
5.1.1	Constraints	51
5.1.2	Editor settings	52
5.1.3	Autocomplete functions	52
5.1.4	Exif data extraction	53
5.1.5	Image dimensions	53

5.1.6	Where are which files	53
5.1.7	Making a backup	53
5.1.8	Corrupted database	53
5.1.9	Express.js and pug	53
5.1.10	Middleware	54
5.1.11	Logging	54
5.1.12	Electron application	54
5.1.13	Database	54
6	FAQs	57
6.1	Keyboard shortcuts	57
6.2	Why only jpeg	57
6.3	The time is behind	57
6.4	Batch add does not create elements in the database	57
7	CHANGELOg	59
7.1	v1.0	59
7.1.1	v1.1.0	59
7.1.2	v1.0.1	59
7.1.2.1	v1.1.1	59
8	Indices and tables	61



GIAnt is an open source cross platform desktop application designed to aid scientists with graphical image annotation in the process of creating a corpus.

If you have a set of images, want to make annotations or even analyze how utters interact with each other this application might be useful for your project.

We built it for analyzing grafitis in Rome. Never the less the application is designed to be adapted for other studies.

CHAPTER 1

Video

This video shows v1.0.0 of the application GIANt in action.

<https://www.youtube.com/embed/4NRxlxq0TEY>

CHAPTER 2

GIAnt resources

What	Where
Download GIAnt	https://github.com/DanielPollithy/GIAnt/releases
Report a problem	https://github.com/DanielPollithy/GIAnt/issues
Get the code	https://github.com/DanielPollithy/GIAnt
Check the API	https://github.com/DanielPollithy/GIAnt/
Automatic win32 tests	https://ci.appveyor.com/project/DanielPollithy/giant
Automatic unix tests	https://travis-ci.org/DanielPollithy/GIAnt
Code coverage report	https://coveralls.io/github/DanielPollithy/TransliterationApplication
Licenses	https://github.com/DanielPollithy/GIAnt/blob/master/LICENSES.txt

This documentation is divided into three sections:

1. **Guides:** An overview of the features for non-users and explanations to get you started
2. **Articles** explaining the main features and advantages
3. **Technical Documentations:** A Technical documentation which enables programmers to extend this application

3.1 Guides

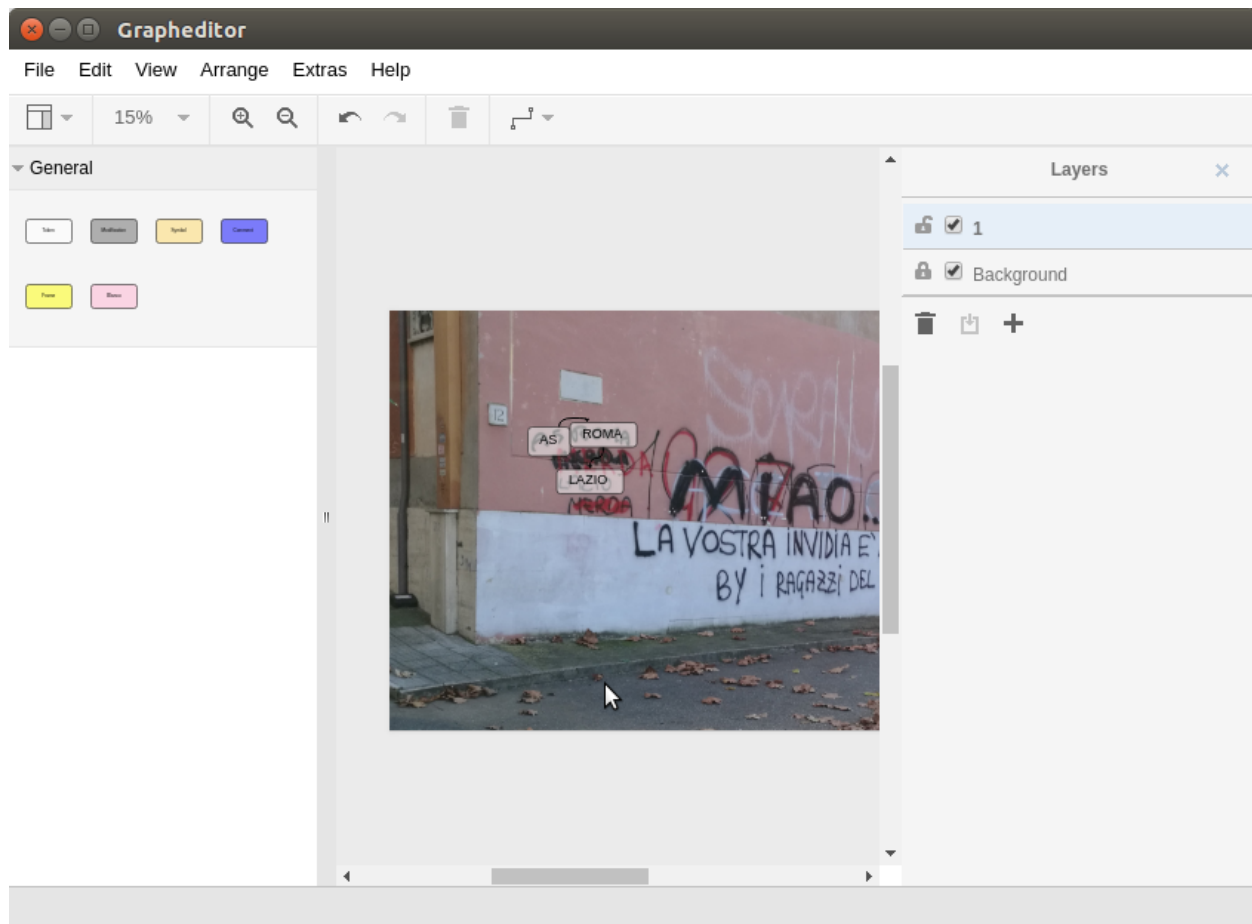
3.1.1 Feature overview

GIANt is designed to work on **win32**, **osx** and **linux**.

It uses the **neo4j database** <https://neo4j.com> as graph storage which is **free to use** in the community edition.

Modelling your annotation use case happens in a graph environment: **Think about your data as nodes and edges!**

GIANt comes with an **Editor** which speeds up the annotation process of your images.



It features:

- different node types (with positional information and without)
- layers which can be interpreted as chronology
- properties of nodes with auto completion
- copy'n'paste and standard image editor features
- meta information (like EXIF data)

You store all of your images in the so called **Index**.

Hauptübersicht

Hier sind alle Bilder aufgelistet

Show 10 entries

Id	Pfad	Erstellungsdatum	Fragmente	Löschen
44356	jakob-owens-224352.jpg	8/15/2017, 7:16:29 PM	Fragmente (0/0 fertig)	Löschen
44388	IMG_20161215_122844.jpg	12/15/2016, 1:28:45 PM	Fragmente (0/1 fertig)	Löschen

Showing 1 to 2 of 2 entries

Every image has **fragments**: They are based on the same image but store another subgraph.

Fragmente für Bild 44388

Hier sind alle Fragmente für ein Bild aufgelistet

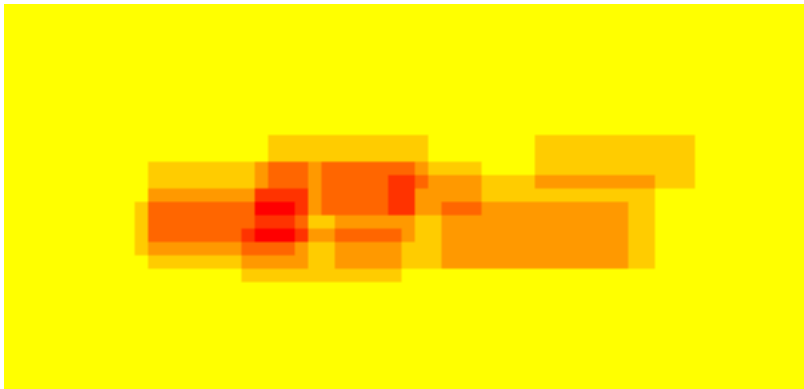
Show entries

Image-Id	Image-Pfad	Fragment-Id	Fragment-Name	Kommentar	Erstellungsdatum	Fertig	Editor	Neo4j Übertrag	Löschen
44388	IMG_20161215_122844.jpg	44364	first	Kommentar	8/15/2017, 7:04:20 PM	In Bearbeitung	Editor	Übertragen	Löschen

Showing 1 to 1 of 1 entries

You can add **comments** to fragments and mark them as **done**.

The **Heatmap Tool** can show you the distribution of the nodes on your annotated images. You can use the **CYPHER** graph query language to select the input nodes for the heatmap algorithm.



Exporting your data and images is possible. The data can be exported to **CSV** and **SQL**.

Export

The export provides four different tables/sheets: Nodes, Relations, NodeProperties and RelationProperties

Download SQL

or

Download CSV (zipped)

The **editor's behaviour can be configured** to a certain extend.

Settings

These are general editor settings

Default font size (pt)

Should the edges be curved



Default stroke width of relations (1-200)

Setting javascript

```

1  TOKEN_CONFIG = {
2    "tokens": {
3      "symbol": {
4        "properties": {
5          "them_mak": ""
6        }
7      },
8      "token": {
9        "properties": {
10         "ground": "",
11         "tool": "",
12         "lang": "",

```

Custom behaviour can be implemented via **javascript**.

```

82  CUSTOM_PROPERTY_CHANGE_HANDLERS = {
83    'relation_type': function (cell, cell_style, cell_content, graph, property, property_value, token_type) {
84      if (TOKEN_CONFIG.relations.hasOwnProperty(property_value)) {
85        graph.setCellStyles('strokeColor', TOKEN_CONFIG.relations[property_value].color, [cell]);
86      }
87    }
88  };
89
90  ALLOW_LOOPS = false;
91  ALLOW_DANGLING_EDGES = false;

```

Data integrity can be assured by using **data constraints**.

Constraints

These functions/queries become executed every single time a fragment is added to Neo4j.

The ID of the fragment is available as {fragment_id} in cypher

Boolean constraints

These constraints have to result in one row containing true.

Example

```
MATCH(f:Fragment)-[]-(t:Token {value: 'Token'}) WHERE ID(f) = {fragment_id} RETURN COUNT(t) > 0;
```

Bool constraint #1498747292635 delete

```
1 MATCH(f:Fragment)-[]-(t:Token {value: 'Token'}) WHERE ID(f) = {fragment_id} RETURN COUNT(t) > 0;
```

Create bool constraint

Count constraints

These constraints return rows. The constraint fails if the number of rows is not in between the boundaries

The interval is defined as follows: [min; max[(if you leave them empty they are ignored)

Example

```
MATCH(f:Fragment)-[]-(t:Token {value: 'Token'}) WHERE ID(f) = {fragment_id} RETURN t;
```

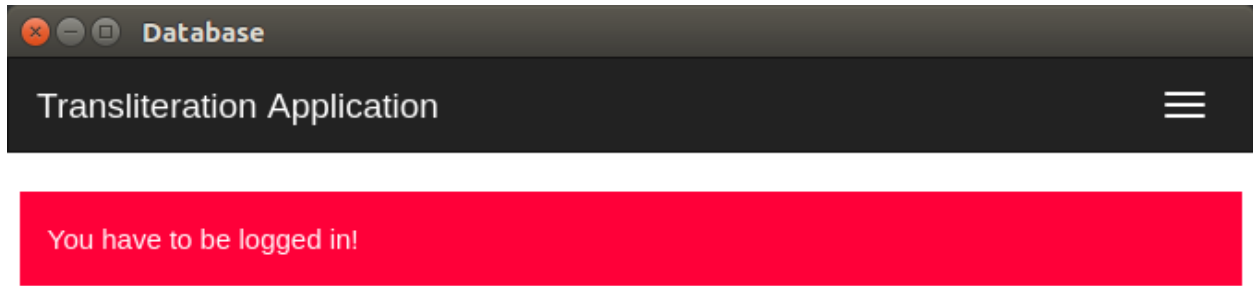
3.1.2 Installation

GIANt needs a database connection to neo4j. This can be local (on your machine) or over the internet.

1. Install Neo4j: <https://neo4j.com/docs/operations-manual/current/installation/>
2. Download GIANt for your operating system: <https://github.com/DanielPollithy/GIANt/releases>
3. Unzip the downloaded archive and start the **GIANt**

3.1.3 Login to neo4j

Once you have started the GIANt you are asked to login to your neo4j database:



You have to be logged in!

Database settings

The application works only with a connection to a Neo4j-database. It can be local or remote.

You have to be logged in!

bolt://localhost:7687	neo4j	1234	login
-----------------------	-------	------	-------

The first box asks for the end point: `bolt://localhost:7687`

- `bolt://` stands for the protocol which is used for the connection
- `localhost` stands for the local machine (you could replace this by an IP-address in order to connect to a server)
- `:7687` is the database port on which the neo4j application is listening on

The second box asks for the username: *default is neo4j*

The third box asks for the password: *default is neo4j* **The password is not explicitly stored in the application. Therefore you have to enter it on every login.**

3.1.4 Getting started

Now that you are logged in you are able to use the application. This guide shows you how to use the application. Don't enter any serious data until you came up with a scheme for your special purpose!

3.1.4.1 Uploading an image

Clicking on the index button gets you to a table containing all of your uploaded images.

Transliteration Application

Index



At the bottom of the page there is an upload button.

Attention: You can only upload jpeg files

Upload

Choose File

No file chosen

upload

Once uploaded a line appears in the table. Before that the creation date of the picture and the **GPS location** are being extracted and stored in the database.

Id	Pfad	Erstellungsdatum	Fragmente	Löschen
44388	IMG_20161215_122844.jpg	12/15/2016, 1:28:45 PM	Fragmente (0/0 fertig)	Löschen

- The **ID** is a unique identifier
- The **Pfad** contains the name of the uploaded image
- **Erstellungsdatum** is the EXIF creation date which was stored in the JPEG image
- **Fragmente** is a link to the fragments that belong to the image
- **Löschen** signifies *deletion*

You can click on the table headers in order to sort the table by specific columns.

3.1.4.2 Creating a fragment

The same procedure can be repeated for the creation of a fragment.

Hinzufügen

Neues Fragment

New Fragment

erstellen

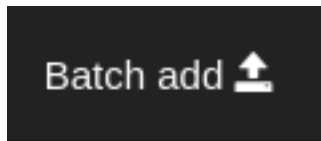
A fragment has a name for human identification, you can add a comment, mark the fragment *ready* open the **Editor** for the fragment and transfer the specific fragment to the neo4j database.

Image-Id	Image-Pfad	Fragment-Id	Fragment-Name	Kommentar	Erstellungsdatum	Fertig	Editor	Neo4j Übertrag	Löschen
44388	IMG_20161215_122844.jpg	44367	New Fragment	Kommentar	8/15/2017, 7:44:16 PM	In Bearbeitung	Editor	Übertragen	Löschen

3.1.4.3 Transfer to neo4j

In order to transfer your data created within the editor you can

- use the ‘Übertragen’ link in the fragment’s line
- or click on the ‘Batch add’ button in the menu bar



The batch add makes use of hash codes. That means: Only fragments that have been changed or are not in the database right now are transferred to neo4j.

3.1.5 Using the editor

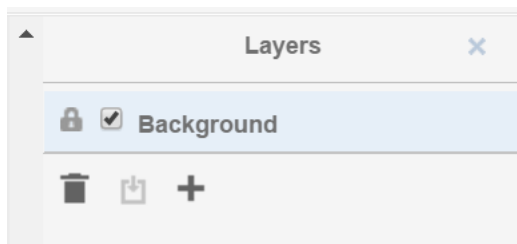
3.1.5.1 Overview

The editor is where your work happens. We implemented some features to improve your productivity. But first comes the basics.

3.1.5.2 Layers

You start off only with the Background layer. It is not possible to attach any data to this layer (that is why you can’t change the status of the lock icon). The only function it has is the checkbox which switches the visibility of the layer on and off.

Imagine you already have a quite populated layers on top of the image. Sometimes it can be better (less distraction) to hide the background image for a while.



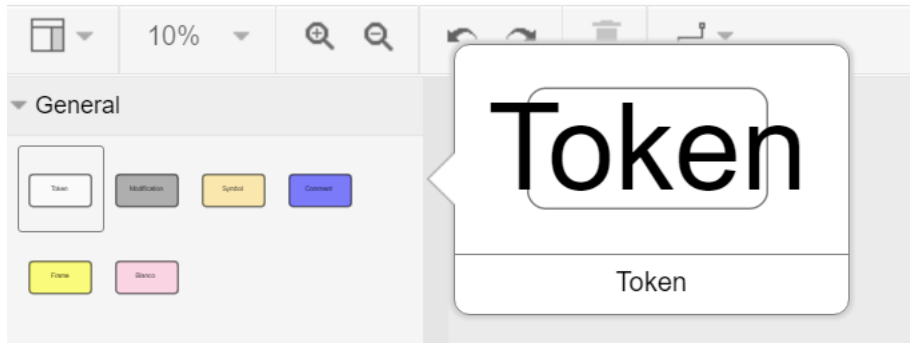
To start editing you have to click on the ‘+’-Button in order to add a new layer. The name of the layer is no changable on purpose. The lock will be open on this layer because you are still editing it. If you want to make sure that no errors occur meanwhile you are editing another layer, feel free to lock it.

The trash icon of course stands for the deletion of the layer. The only sideeffect that can happen here is that gaps in the enumeration appear. Say we created 3 layers. Deleted the second one. Now only layer “1” and “3” remain. The chronology of course is still there.

If you closed the layers panel by clicking the “x” button in the upper right corner, you can get the panel by clicking “View” -> “Outline”.

3.1.5.3 Nodes

We call every box which is drawn on a layer a node. The different types of boxes can be configured through the settings. In general every different entity should have its own node type (box type).



By double clicking on one box you can edit the content of it. This might not be necessary for your use-case but is auxiliary if you want to make your annotations easy readable.

The given box types are divided into to categories:

- Singular Tokens (positional)
- Group Tokens

The default setup interprets them as follows:

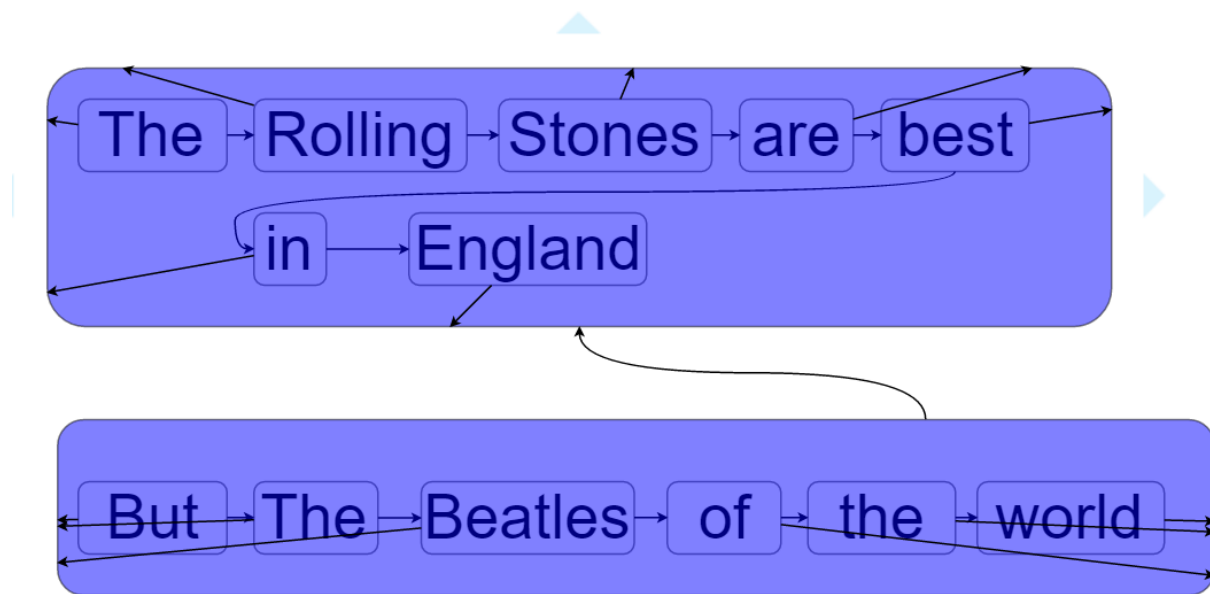
The singular tokens contain text or at least a symbole.

- “Token” is a real textual component (maybe a word)
- “Modification” is always overlapping a “Token” (maybe a strike-through or an overwriting)
- “Symbol” is straight-forward a symbole

The Group Tokens shall not carry positional information. They are called “Groups”.

- “Comment” is a group of singular tokens that relates over one single entity to another group (Imagine to grafitis on neighbouring walls relating to each others)
- “Frame” shall carry the psychological framing. The “Frame” Group is special because it connects to MetaGroups. (see next paragraph)
- “Blanco” is a general purpose group

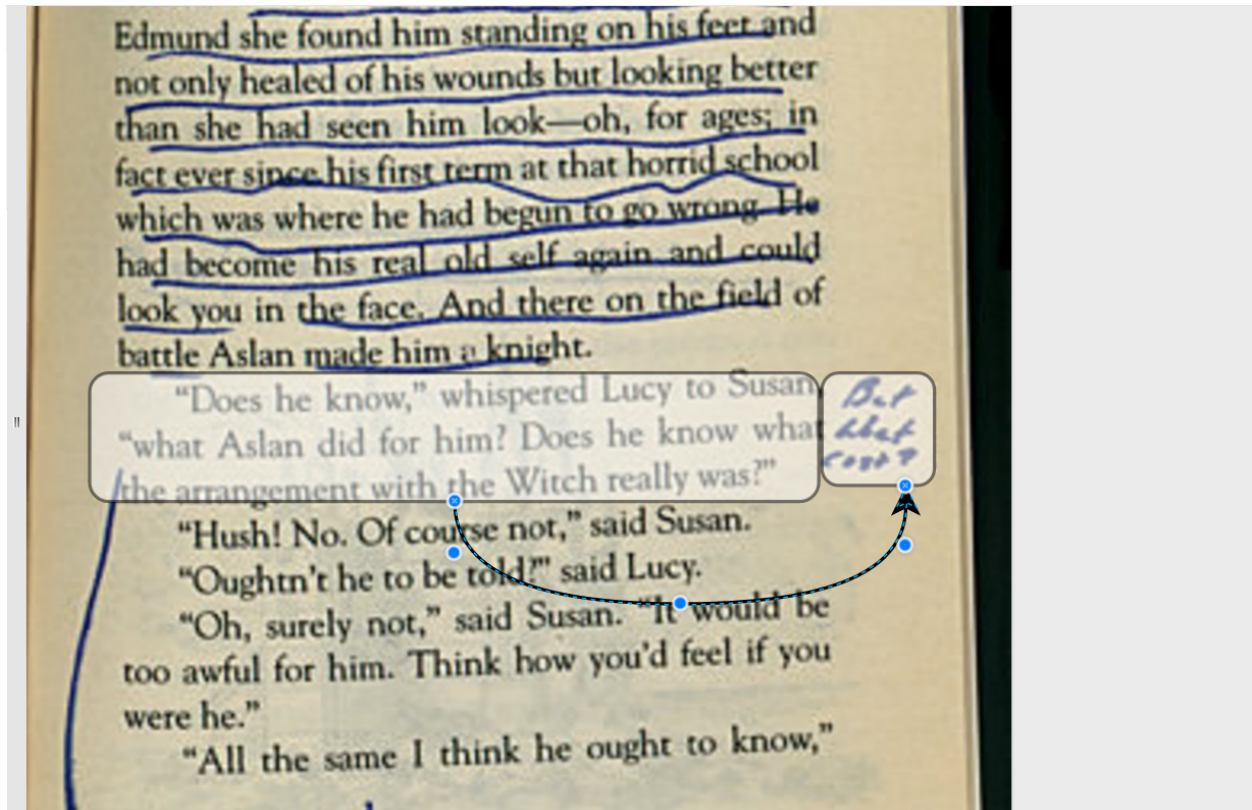
Fictional example for comment groups:



Properties can be attached to tokens. See section “properties”.

3.1.5.4 Relations

The entities of your image annotations are expressed by tokens. GIANt becomes handy when these tokens interact with each others or the important information lays in “between” them. (This is also the case where graph databases can serve with their graph query languages.)



Different kinds of exemplary relations can be examined:

- "part of" relations: a word is part of a sentence etc.
- "follows" relation: a word follows another word
- "negates" relation: an expression negates the related expression
- "opens frame" relation: a word or symbol opens a psychological frame
- ... (lots of possibilities)

Properties can be attached to relations. See section "properties".

3.1.5.5 Properties

Properties are attached to relations and tokens. The stored information is transferred directly into the graph database so it can be used for querying your data.

color: black

dialect: bavarian

ground: paper

lang: german

tool: pencil

Enter Property Name Add... Apply

Cool Features: Every property you have transferred into the neo4j graph database is used to enhance your experience by providing autocompletion: There is autocompletion on:

- property names and
- property values

Selecting a new property from the autocompletion.

Enter Property Name Add... Apply

color

dialect

lang

tool

ground

Selecting a value for the given property from the autocompletion.

tool: printer

typewriter

spraycan

Add... Apply

The autocompletion is token type, property and relation type sensitive.

Different tokens need distinct properties. You can configure this with the settings and even give default values and javascript validations. There are a lot of possibilities to model your use-case with this tools.

3.1.6 Data scheme in neo4j

The GIANt can be seen as a Graphical Image Annotation Tool that stores your data in Neo4j. So you get all of the advantages graph databases have.

Access to your neo4j database is usually at this local url: <http://127.0.0.1:7474>

The boxes and edges you drawn in the editor are reflected by the following scheme in Neo4j.

3.1.6.1 Images

Every uploaded image is represented by a node. Neo4J label: `:Image`

The following Cypher query retrieves it for you:

```
$ MATCH (n:Image) RETURN n LIMIT 1
```



Image properties

Property	Name
A unique ID	<code>id</code>
File path	<code>file_path</code>
The width in pixels	<code>width</code>
The height in pixels	<code>height</code>
EXIF date or upload date	<code>upload_date</code>

Image `<id>: 16` **file_path:** Wallace-Annotated-CS-Lewis.jpg **width:** 647 **height:** 550 **upload_date:** 1505992150952

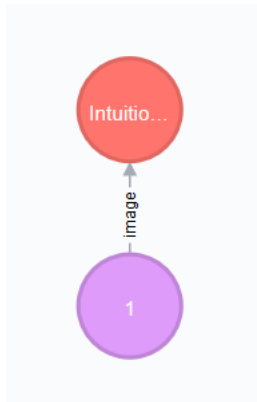
By expanding the child relations (lower circle segment button)...



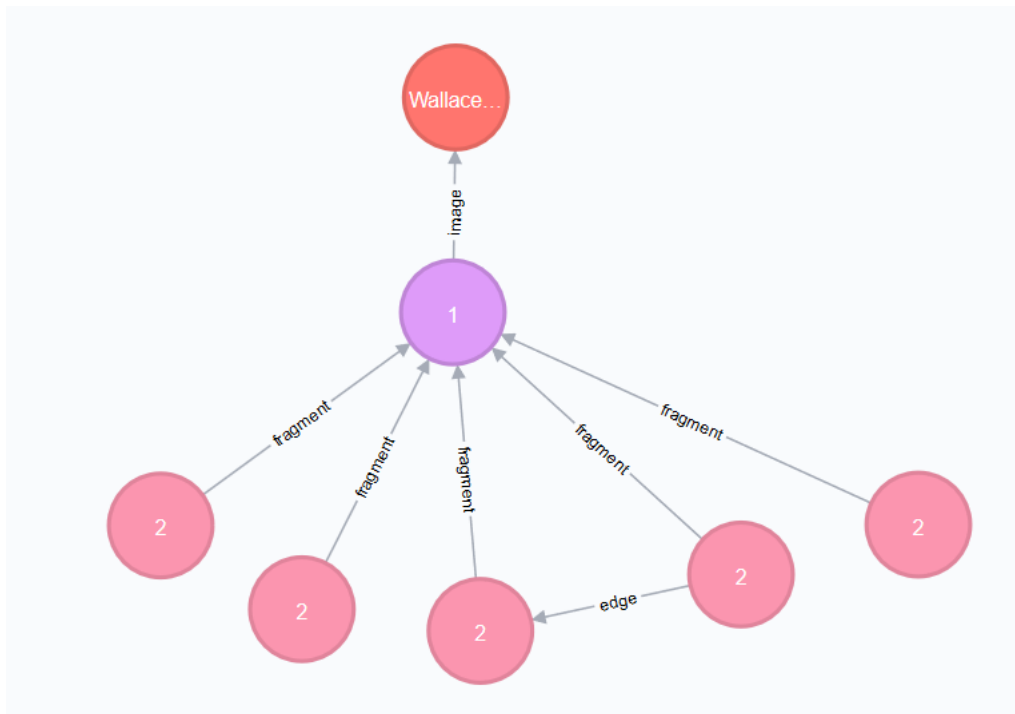
3.1.6.2 Fragments

You see that images are connected to fragments. Neo4J label: `:Fragment` Fragments are interpretations or multiple areas of one image. Explicit: One image relates to many fragments but one fragment only relates to one image. We call this 1-n relationship.

The Neo4J Label of the relation between Image and Fragment is called `:image`.



By expanding the Fragment's relations we see that the boxes we drew in the Editor are nodes on this hierarchy level.



The Neo4J Label of the relation between Fragment and Token is called :`fragment`. Every fragment is connected to many tokens (1-n relationship).

Properties of Fragments

Property	Name
A unique ID	<code>id</code>
Fragment name	<code>fragment_name</code>
Use with batch-add?	<code>completed</code>
Helps to detect changes	<code>checksum</code>
Creation date	<code>upload_date</code>

Fragment `<id>: 49` `fragment_name: 1` `comment:` `completed: false` `hash: 462cc19bc8b40ecba0d0ab656257b599ff1d56f0` `upload_date: 1505992155412`

3.1.6.3 Tokens

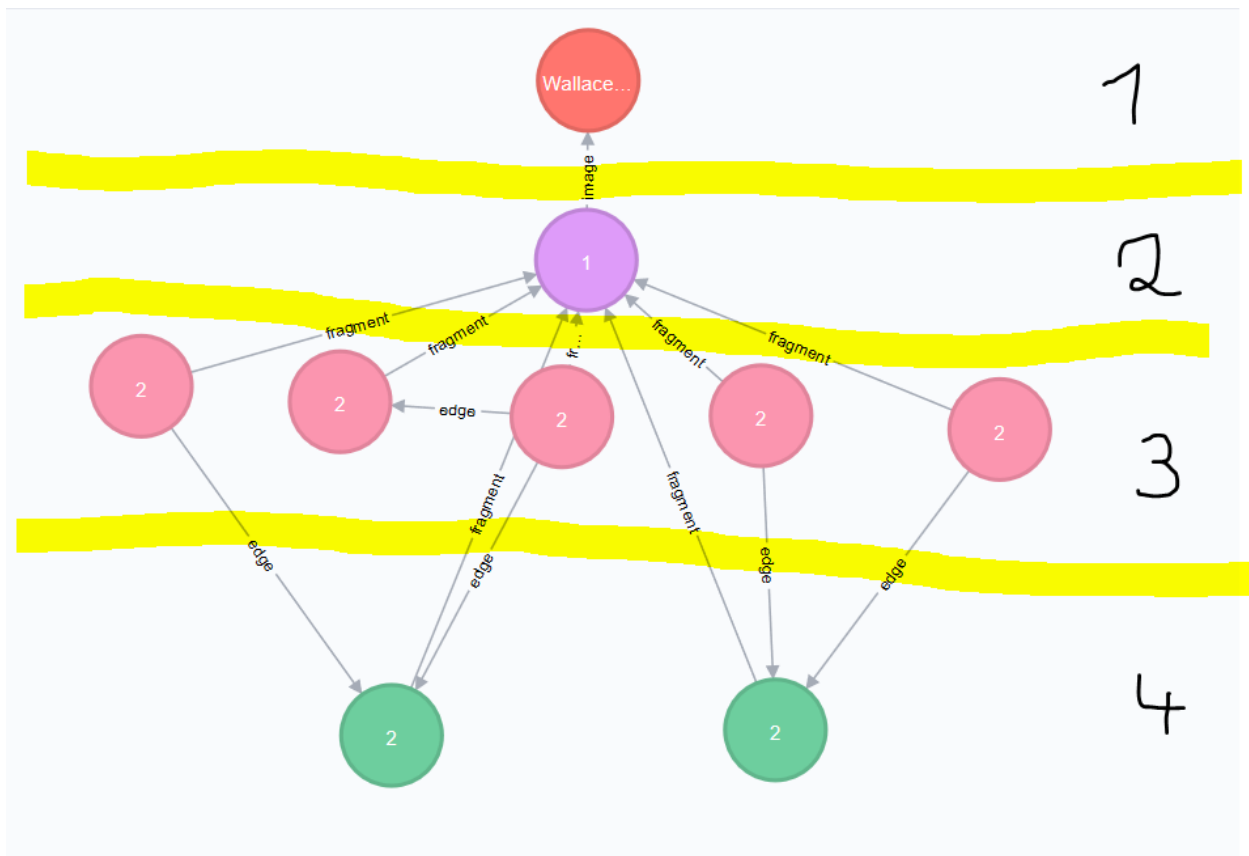
Tokens are what we also called boxes in the context of the Editor. There are two groups:

- Singular Tokens: they carry positional information
- Group tokens: they group together and stand as an entity for multiple tokens that need to have relations between other groups

The Neo4j label for singular Tokens is `:Token`.

The Neo4j label for Group Tokens is `:Group`.

The following image illustrates how the Groups and Singular Tokens can be seen as distinct hierarchy layers.



1. Image
2. Fragment
3. Singular Token (Symbole, Modification, Text)
4. Group Token (Comment, Frame, Blanco)

3.1.6.4 Singular Tokens

Interesting Properties of **Singular Tokens**

Property	Name
A unique ID	id
width [pixels]	width
height [pixels]	height
position [pixels]	x, y
The type of the Token*	:tokenType
The box's content**	value
The number of the layer	hand
All custom properties	e.g. color, tool, ...

(*) Possible default tokenTypes are: token, symbol, modification

(**) The content of the box is what you enter when you double click into the box

```
Token <id>: 35 parent: 2 dialect: whiteSpace: wrap color: vertex: 1 tool: fillColor: #FFFFFF as: geometry width: 180 x: 2510 rounded: 1 y: 1030 fontSize: 60
ground: html: 0 id: 3 enumerator: 3 lang: opacity: 50 tokenType: token value: Burn height: 80 hand: 1
```

3.1.6.5 Groups

Interesting Properties of **Group Tokens**

Property	Name
A unique ID	id
The type of the Group*	:groupType
The box's content**	value
The number of the layer	hand
All custom properties	e.g. frame_type...

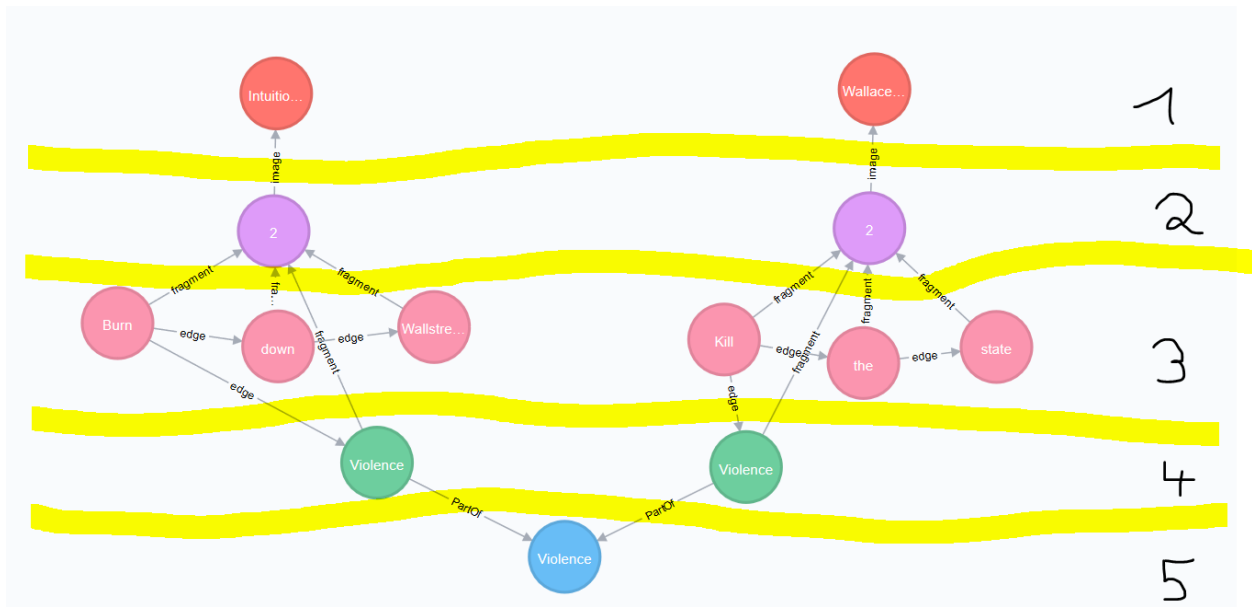
(*) Possible default groupTypes are: comment, frame, blanco

(**) The content of the box is what you enter when you double click into the box

```
Group <id>: 38 parent: 2 groupType: frame whiteSpace: wrap vertex: 1 frame_type: fillColor: #ffff00 as: geometry width: 250 x: 2480 rounded: 1 y: 1230 fontSize: 60
html: 0 id: 8 enumerator: 8 opacity: 50 value: Violence height: 110 hand: 1
```

Attention: The Group Token “Frame” introduces another Graph Database Node called **MetaFrame**. The Neo4j label for this is :MetaGroup. This node is an interconnection of all “Frames” with the same name in order to ease graph exploration.

Example: The Token with text “Kill” in one fragment is connected to a Frame called “Violence”. In another image’s fragment there is also a Token connected to Frame called “Violence”. Both Token Groups “Frame” are connected to the MetaGroup “Violence” which is created automatically.



Interesting Properties of **MetaGroups**

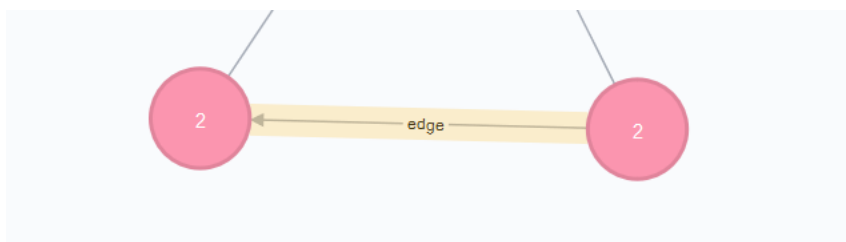
Property	Name
A unique ID	id
The type of the Group*	:groupType
value	value

(*) Only MetaFrame is possible so far!

3.1.6.6 Edges

The tokens are interconnected with multiple edges. This is not a 1-n relationship but a many-to-many relationship (called m-n). The Neo4J Label of the relation between Token and Token is called :edge.

Do not confuse this with the relations between Images and Fragments nor Fragments and Tokens!



Interesting Properties of **Edges**

Property	Name
A unique ID	id
Type of relation	:relation_type
All custom properties	e.g. frame_type...

```
edge <id>: 55 parent: 2 curved: 1 orthogonalLoop: 1 source: 4 relation_type: follows edgeStyle: orthogonalEdgeStyle target: 5 as: geometry edge: 1 rounded: 0 html: 1
id: 7 strokeColor: black exitY: 0.5 value: exitX: 1 jettySize: auto relative: 1 hand: 1
```

3.1.6.7 Using Cypher

Now that you know how your data is structured in the graph database you might already have ideas on what kind of information you want to retrieve from your 'corpus'.

Using the GIAnt but refusing to use Cypher is a waste of time! Cypher is the SQL oriented query language for neo4j graph databases. <https://neo4j.com/developer/cypher-query-language/>

It can really help you to find quickly what you are looking for but you have to get into it a little bit and design your corpus accordingly.

3.1.7 Heatmap tool

The heatmap tool can be used to analyze the positions of tokens. It might be interesting to see the density of tokens in a region or the outline they form. The input for this tool has to be a Cypher query. It will then only work with the tokens, therefore it is recommended to build your query to only return tokens.

Example query: `MATCH (s:Token) RETURN s;`

The query is not analyzed or guarded. This means that any code can be executed. As a consequence must this feature kept on a local system and not exposed through a webserver!

The color scheme of the heatmap reaches from 0 (yellow) to 1 (red).

3.1.7.1 Normalization techniques

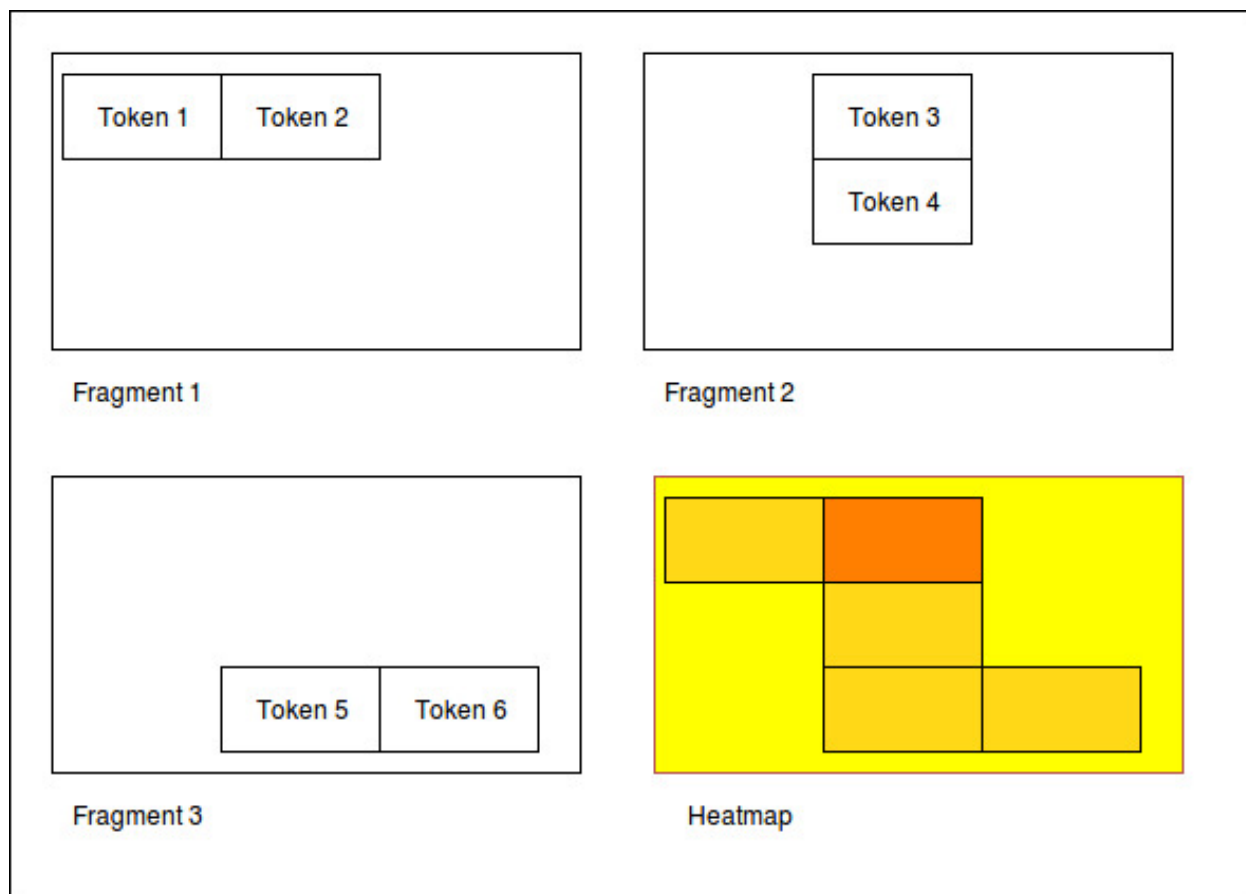
In this context the term normalization refers to an algorithm that makes the positions of tokens in different images comparable.

There are three types of normalizations present:

3.1.7.2 Normalization 1: Position in image

All images are normalized to the output size. The tokens are scaled accordingly.

The final result is the distribution of tokens over the images.

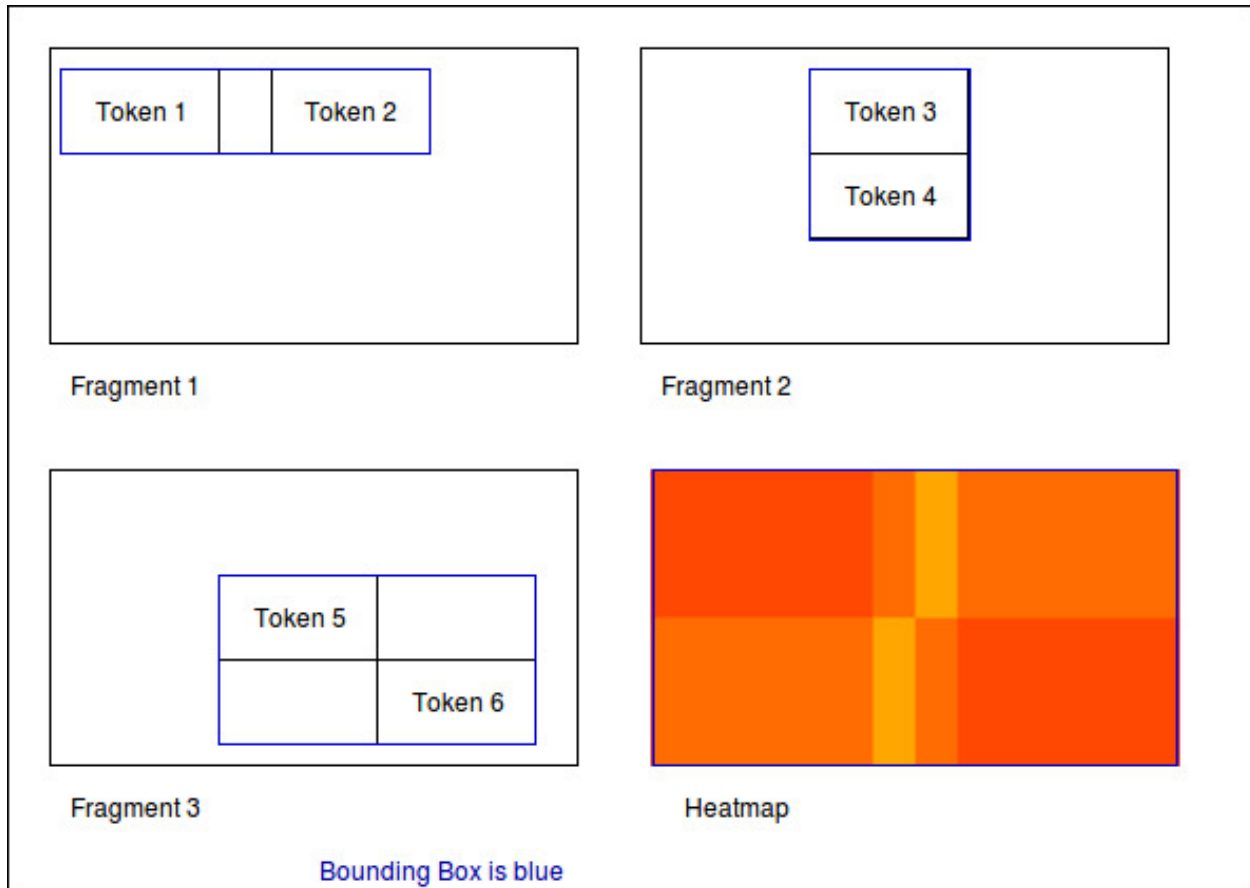


3.1.7.3 Normalization 2: Position in scritte (bounding box)

This method fetches the Bounding Box(see below) of every image and scales the tokens according to it.

The bounding box is the rectangle spanned by the lowest coordinate to the highest one.

The result shows the distribution of tokens within the bounding box.

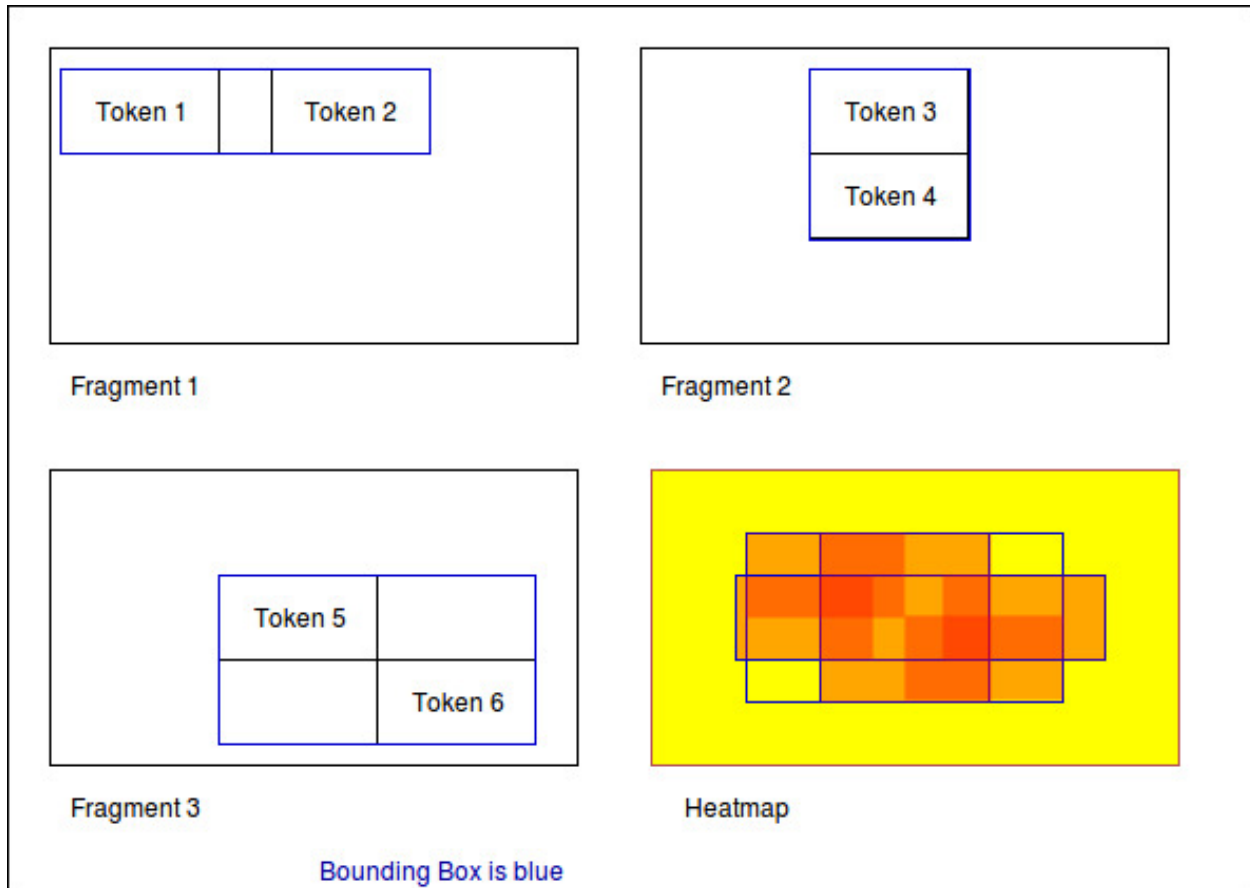


3.1.7.4 Normalization 3: Bounding box centered

Here the bounding box is placed into the normalised image. But the position is changed: The box's center is placed over the normalisation target center.

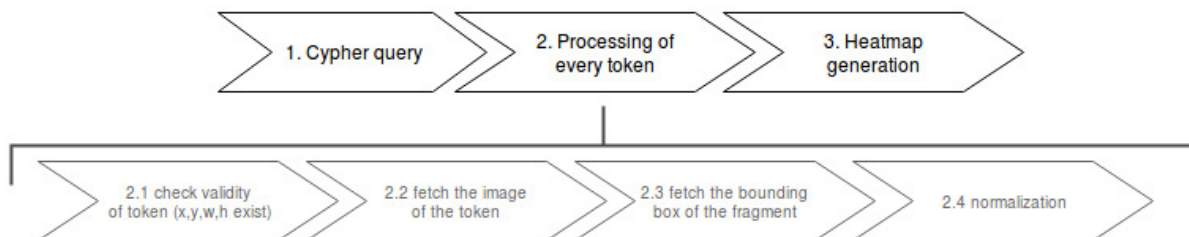
As a result, the bounding boxes and by that way the outlines of all scritte are comparable.

This method could be used to extract the outline of fragments. Example: Do they have a horizontal orientation or are they grouped like a triangle.



3.1.7.5 Performance

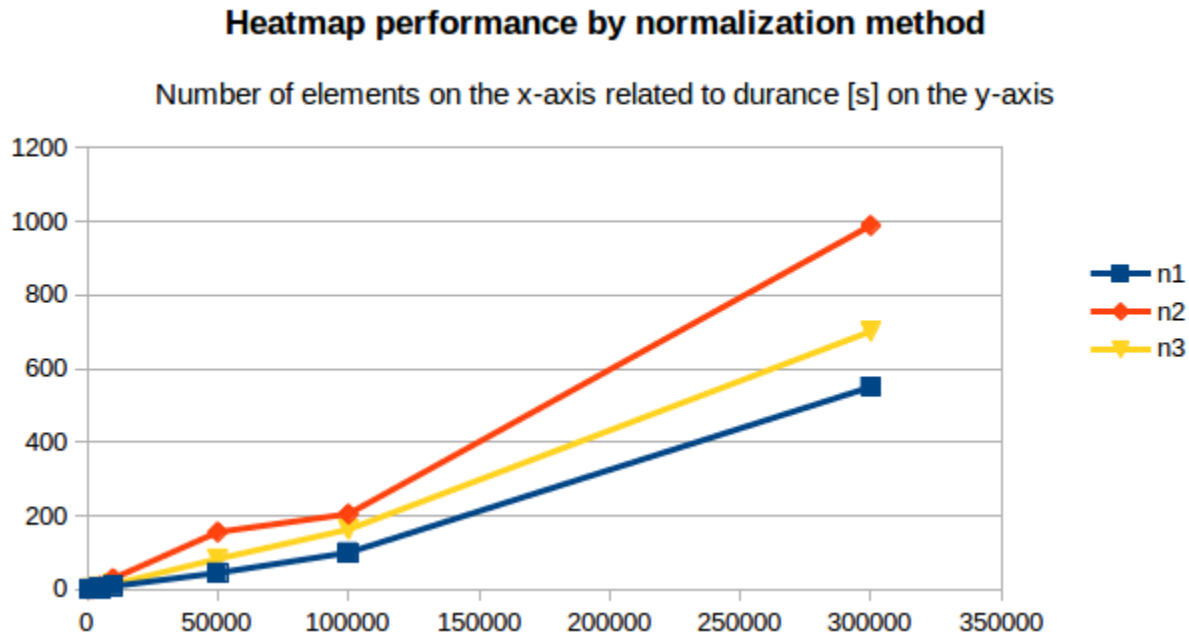
The heatmap creation process happens as a stream.



Processing every single token means some computation effort. Especially fetching images/bounding boxes from the database costs resources, in particularity time. The heatmap tool was never designed to be a big data application but rather a medium data application that shall work with up to 3000 images containing 1 fragment containing 100 elements.

Based on this requirement a ram cache was introduced which is not persistent between heatmap generations but could be implemented easily. The cache prevents unnecessary database request.

The performance evaluation has shown that it is possible to generate heatmaps in reasonable time.



The most computation intense normalization took 15 minutes with 300.000 elements. The data points of this chart are:

Total elements	Normalization #1 [s]	Norm. #2 [s]	Norm. #3 [s]
1.000	1	1	1
2.000	1	1	2
5.000	4	9	7
10.000	9	30	14
50.000	45	156	83
100.000	100	205	163
300.000	550	989	701

The performance test took place on regular Ubuntu 16.04 with 64 bit on a laptop with Intel® Core™ i5-2520M CPU @ 2.50GHz × 4, and 3,7 GiB RAM and a HDD.

3.1.8 Data constraints

In database context ‘check constraints’ are a mean to assure data integrity.

This application could have varying use cases. From case to case the constraints for the graph scheme differ.

One use case might restrict the number of edges between nodes. Another one the total amount of nodes and so on.

As a result this application does not contain a set of ‘hard coded’ constraints but a configuration file that contains Cypher queries which will be executed every time a graph was inserted into Neo4j.

3.1.8.1 The workflow

GraphEditor -> codec.js -> Neo4j -> constraints.js

Opposite to usual RDMS Neo4j only comes with a limited set of data integrity constraints. Usually these constraints are checked before inserting data into the database.

This workflow does not do so because we want the user to be able to write Cypher query code. In a future version it could be possible that a failing constraint triggers the transaction to be rolled back. So far this doesn't happen.

3.1.8.2 Design of the constraints

In the end the constraints have to validate so their output is boolean.

If all constraints are true -> then the constraint checking succeeded and there is no error

Else: We hand the error to the user.

3.1.8.3 Writing constraints

There is an entry in the menu which is called 'Constraints'. In that view you can create three types of constraints:

- bool constraints: Your query has to return 'true' to succeed
- count constraints: You write a query and provide a minimum and/or maximum of accepted results to your query
- free constraints: you write javascript code (in detail a Promise: see below for an example)

3.1.8.4 Example for count constraint

You provide a query like `MATCH (a:Token)-[]-(i:Image) RETURN DISTINCT a;` and the boundaries (lower is contained, upper excluded): `[0, 200[`

3.1.8.5 Example for a free constraint

You can supply any javascript code returning a promise. If it resolves your constraint succeeds.

Variables handed into the scope

session is a neo4j session

`session.run(cypher_string)` returns a Promise. This will be the entry point for most of the free constraints.

The following example checks whether tokens with the value 'Token' exist.

```

1  new Promise(
2    function(resolve, reject){
3      var variables = {"fragment_id": fragment_id};
4      session.run("MATCH (f:Fragment)-[]-(t:Token {value: 'Token'}) " +
5        "WHERE ID(f) = {fragment_id} RETURN t.value as value;", variables)
6        .then(function(result){
7          var value;
8          result.records.forEach(function(res){
9            value = res.get('value');
10           if (value === "Token2") {
11             reject("There was a token called Token.");
12           }
13         });
14         resolve();
15       }).catch(
16         function(err){
17           reject(err);
18         }
19       );

```

3.1.8.6 Security

The cypher queries are checked to not contain `CREATE`, `MERGE`, `SET` or any other operation that could change the data while performing the check. If that happens only a message is prompted to the user.

These operations could be in the query willingly so they will still get executed in order to enhance the power of the user on the data.

3.1.9 Exporting your data

It is very likely that you don't only want to analyze your data but also take it with you to another application.

There are three possibilities to export your data:

- SQL export
- CSV export
- Copy the Neo4J database (see section Migration)

The SQL and CSV export are structured in order to give the ability to reassemble the graph.

Both of them are structured into four tables:

- Nodes table
- Properties of nodes table
- Relations table
- Properties of relations table

With the following schema (see section data schema for reference):

Nodes table

Node_ID	value	tokenType	groupType
1	NULL	NULL	NULL
2	NULL	NULL	NULL

Node properties table

Node_ID	key	value
1	file_path	1234.jpg
2	comment	fragment#1

Relations table

Relation_ID	relationType	SourceNode_ID	TargetNode_ID
1	image	1	2

Relation properties table

Relation_ID	key	value
1	label	image

3.1.10 Transferring your data

If you want to take your data to another computer follow these steps.

3.1.10.1 1. Move static files

The folder containing the application contains a folder called 'media'. Compress this folder (e.g. zip it), transport the archive to the new computer and uncompress it into the new 'media' folder.

This folder contains settings files, the graph editor's xmls and the images.

3.1.10.2 2. Relocate the Neo4j database

We have to move the Neo4j database to the new computer. There are two options:

- 2/a: has not worked in the tests but is recommended by Neo4j -> dump'n'load (UNIX)
- 2/b: has worked but is a little dangerous -> copy'n'paste (WINDOWS)

3.1.10.3 2/a Dump and load Neo4j (UNIX)

According to <https://neo4j.com/docs/operations-manual/current/tools/dump-load/>

- `stop neo4j`
- `neo4j-admin dump --database=<database> --to=<destination-path>`
- `neo4j-admin load --from=<archive-path> --database=<database> [--force]`
- `start neo4j`

Example for unix:

On machine 1

- `sudo service neo4j stop`
- `neo4j-admin dump --to=dump.db`
- `sudo service neo4j start`

On machine 2

- `sudo service neo4j stop`
- `neo4j-admin load --from=dump.db --force`
- `sudo service neo4j start`

Use the 'force' option to overwrite existing data

3.1.10.4 2/b Second variant for neo4j (WIN)

If this doesn't work it is possible to copy the whole database directory to the new computer. By default the database is called graph.db and is situated under `/var/lib/neo4j/data/database/graph.db/`. Move the whole folder to your new computer and `service neo4j start`

Example workflow for UNIX:

- `sudo service neo4j stop`

- `zip -r ~/Desktop/dump.zip /var/lib/neo4j/data/databases/graph.db/`
- `sudo unzip dump.zip -d /`
- `service neo4j start`

3.1.11 Installing an update

If a new version of GIAnT is released you might want to use it depending on the fixes or new tools.

3.1.11.1 Compatibility

It is easy for you to decide whether there will be a conflict installing the new version or not.

GIAnT uses semantic versioning. This means that the applications version number “x.y.z” is not only a counter for the changes made to the code but also an indicator for what has changed.

`<major>.<minor>.<patch>` is the system you might know.

- **If `patch` changes then there was a small security fix or a bug closed. You can install the new version** without breaking anything.
- **If `minor` was increased then there were big changes made but they are still backwards compatible.** You can install the new version without breaking your database but for example a tool of the software might have been removed.
- **If `major` was increased then the new version contains NOT backwards compatible changes.** You will have to migrate your data to a new schema.

If you really want to find out what changed you have to look into the *CHANGELIST*.

3.1.11.2 How to

Updating is easy. Check your compatibility with the new version. And then download the new version and copy your data(next section).

3.1.11.3 Keeping your data


















There are two data stores for GIAnT.

- The Neo4J database
- Your filesystem















If you updated GIAnT you don’t have to touch the database.

What you have to do is copy your files to the new application.

When you navigate to the folder where GIAnT is located in you will see a directory structure like this:

Name
 locales
 resources
 blink_image_resources_200_percent.pak
 content_resources_200_percent.pak
 content_shell.pak
 GIAnt
 icudtl.dat
 libffmpeg.so
 libnode.so
 LICENSE
 LICENSES.chromium.html
 natives_blob.bin
 pdf_viewer_resources.pak
 snapshot_blob.bin
 ui_resources_200_percent.pak
 version
 views_resources_200_percent.pak

Please navigate to 'resources' -> 'app'

Name
 docs
 editor
 media
 node_modules
 src
 test
 appveyor.yml
 install_neo4j.sh
 keyboard short codes.txt
 LICENSE
 LICENSES.txt
 package.json
 README.md
 yuidoc.json

The ‘media’ folder contains all of your stored files and settings.

You have to replace the new ‘media’ folder by the old ‘media’ in order to keep your data.

3.1.12 Accessing your data on multiple devices

It is possible to synchronize the application’s data on multiple devices so you can use GIAnt on - let’s say - your laptop and your PC.

Use one of the many sync-services out there like Google Drive or Dropbox to sync:

- Your Neo4J database folder
- Your installation’s ‘/media/’ folder which is located under ‘GIAnt/resources/app/media’

But please don’t use them in parallel. Better safe than sorry. I also recommend to switch off the syncing while working on your data.

4.1 Articles

4.1.1 Analyzing 3000 graffiti

The GIANt application (GrapiCal Image Annotation Application) was written initially written for Sebastian Lasch who was writing his PhD thesis at the Ludwig Maximilians Universität (LMU) Munich, Germany.

In the hope that others find it interesting I am going to describe the process how and why we designed this application to serve its first use case.

Sebastian Lasch's dissertation is about the **unique communication aspects of the graffiti** in rome, also called 'scritte murali'.



Looking at a scritte murali one can see that different authors were writing at this wall. They were overwriting and changing the sense of each other's work.

The interaction between the different ‘hands’ (that is what we called the authors) are a key aspect of this communication method.

Before Sebastian and I met, the proposed solution for the analysis was demanding. The corpus will contain at least 3000 pictures and initially should have been build as follows:

1. Identify every single token on the image and use a 5x5 raster as a rudimentary position
2. Write the token (word) with its attributes line by line to a SQL database
3. Add all relevant meta data by hand to the SQL database
4. Store the interactions in a text format (see below)



ID	Token	Position	Mano
Roma_15	STALIN	1	1
Roma_15	C'É	3	1
Roma_15	simb1	4	1
Roma_15	simb1	5	1
Roma_15	NON	2	2
Roma_15	simb2	4	2
Roma_15	simb2	5	2
Roma_15	PIÙ	6	2
Roma_15	simb3	7	2
Roma_15	simb	8	2
Roma_15	{xxx}	2	3
Roma_15	simb?	4	3
Roma_15	simb?	5	3
Roma_15	{xxx}	6	3
Roma_15	simb1	7	3
Roma_15	simb1	8	3
Roma_15	[---]	1	4
Roma_15	simb4	4	4
Roma_15	{xxx}	7	4
Roma_15	{xxx}	8	4

- ➔ Interaktionsform (Überschreiben, Ausstreichen, usw.) lässt sich über ‚doppelt belegte‘ Positionen extrahieren
- ➔ Symbol-Bedeutungen besser integriert
- ➔ ‚Original-Texte‘ (Hand 1) lassen sich sehr einfach auslesen (wichtig bei großer Token Anzahl)

30

Text format The last step would have been to write a so called transliteration and modification form. Which would have looked as follows for the scritte murali you have seen previously:

```
AS ROMA {[MERDA] [~~~]} [MERDA]} LAZIO {[MERDA] [~~~]}
{[[[1900]] [[198.]]] [1927]} FIGHE ACCATTONA MIAO ... \simb_emo_2\ ! CIAO CIAO
LA VOSTRA INVIDIA E` LA NOSTRA FORZA !!! BY I RAGAZZI DEL KENNEDI !!!
{[[[//]] [xxx]]} {[[//]] [xxx]} {[[//]] [xxx]} NO {[[//]] [xxx]} [[.CALE]]
```

The purpose for this long system was that you want to store the interaction between the tokens in some form. Here [~~~] stands for a strike-through or [xxx] for an erasure of the word on the left side.

Such a system would have been inspired by an archaeological system called [Leidener Klammersystem](#).

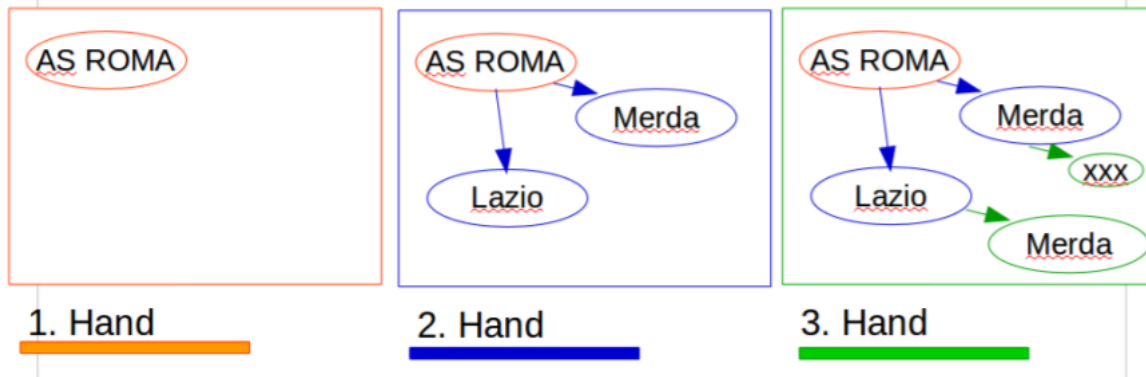
Of course this system has many downsides:

- No automatic information processing is possible with this kind of ‘format’
- The process of creating such a string is long and exhausting

- As this is a left-bound parenthesis system every element in such a string can only have one parent and one descendant. This means that building more complex systems is close to impossible. Imaging a word from the beginning referencing the last word. You would not only have to reconstruct your whole query but it is also possible that there might be no solution to this problem.
- The attributes of an element are separated from its representation in the query
- Once you have managed to transfer this system into your SQL database you are confronted with writing many JOIN queries in order to get your data together.

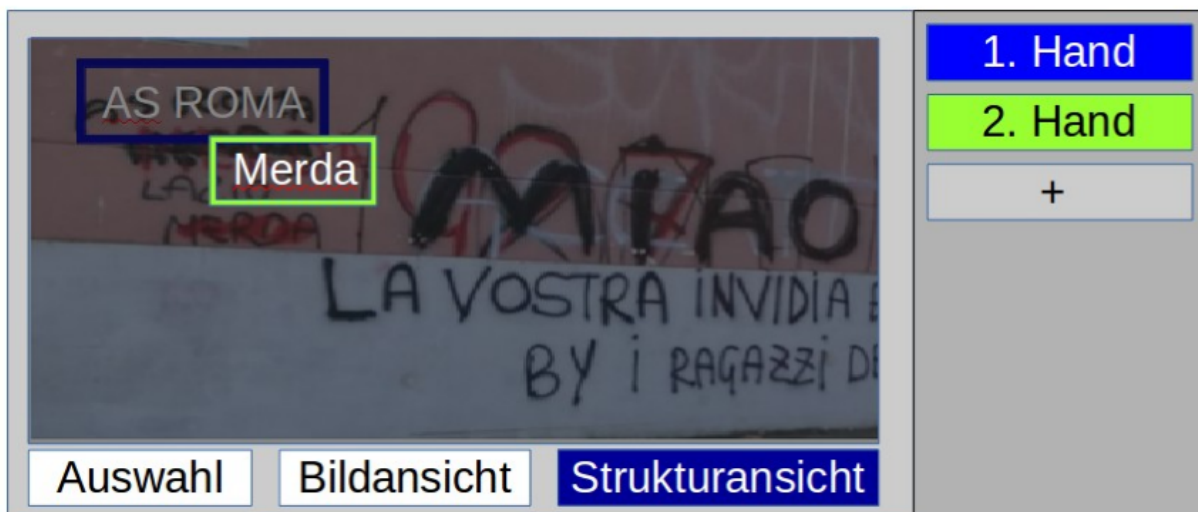
As you can see there were some **good reasons** to spend time on this project and propose another system.

Beispielhaft könnte das wie folgt aussehen:



The big differences of the first change were:

- Think about the tokens and all the entities as nodes in a graph.
- Have as many interactions possible between them as you need (they are now the edges in the graph).
- Attach the chronological information (which hand wrote this) as one of the many attributes to nodes and relations.



And of course a Graphical User Interface would help a lot! Imagine the difference between writing the structured data of a graph in a text editor to the ease of dragging boxes around and storing the graphical editor's content automatically in the correct format.

Now that we have gone so many steps the last was to replace the SQL backend by a graph database in which relations

are ‘first citizens’. These databases are designed to store graphs efficiently and - more important - provide **query languages** to help in the step of information retrieval on graphs. The community version of the Neo4J graph database was chosen.

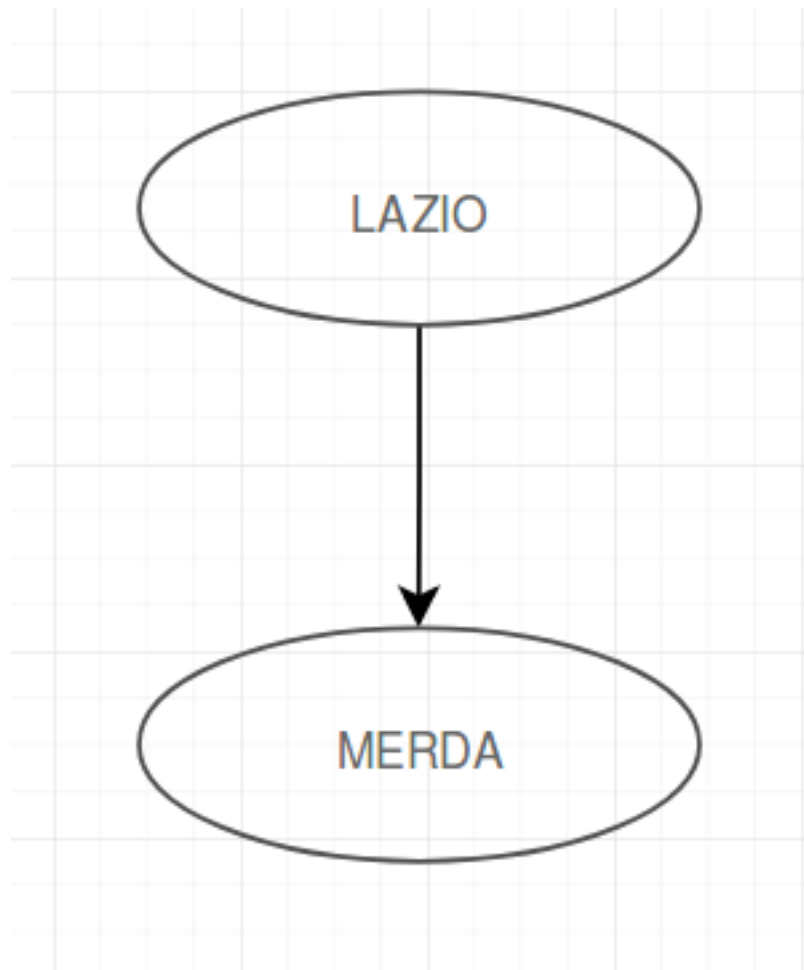
These were the steps we went from manually working on tables and *Leidener Klammersystem* to using an extended graphical image annotation tool that works with graphs.

4.1.2 Cypher vs. SQL

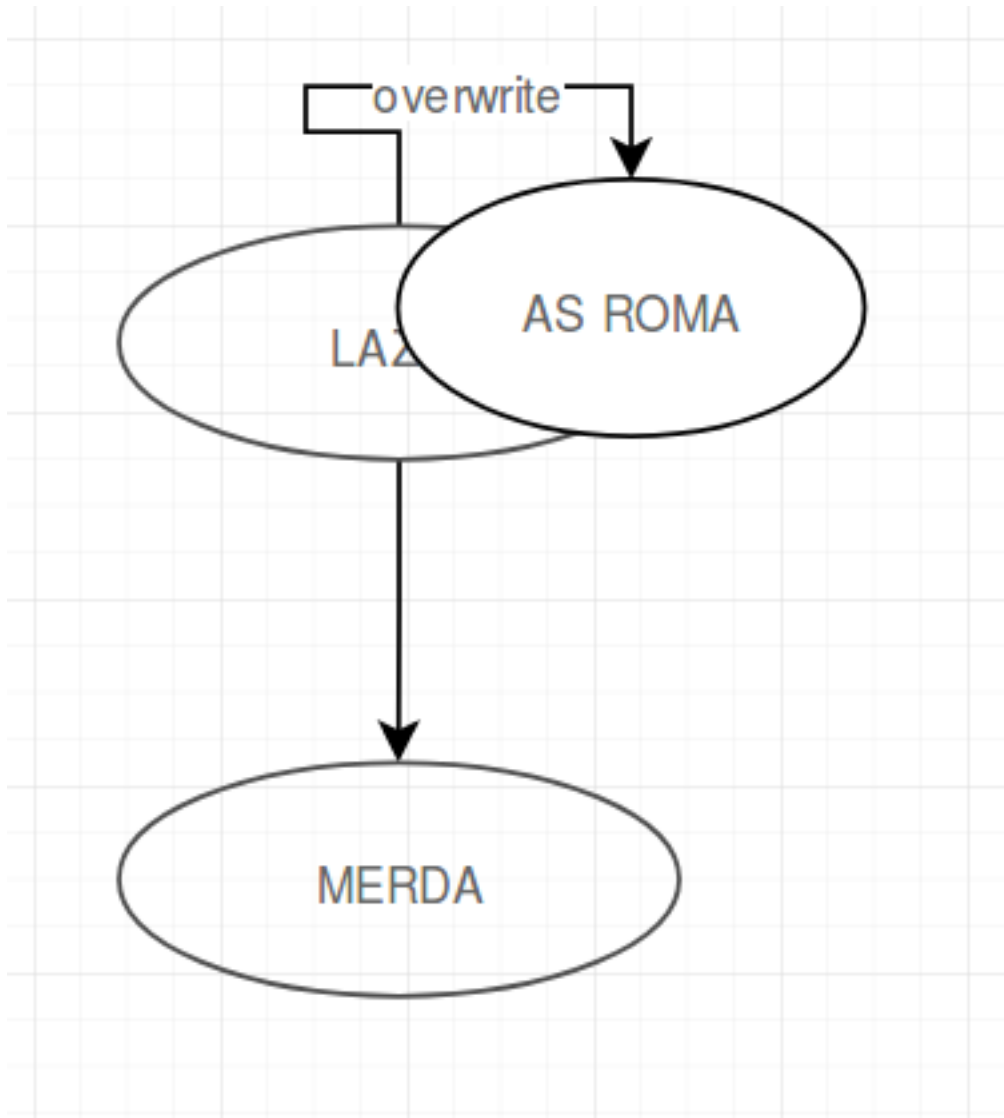
This article shows why Cypher is superior to plain SQL for information extraction on graphs. We are going to exercise a complete example from storing a graph to retrieving data and calculating some metrics.

4.1.2.1 Example scenario

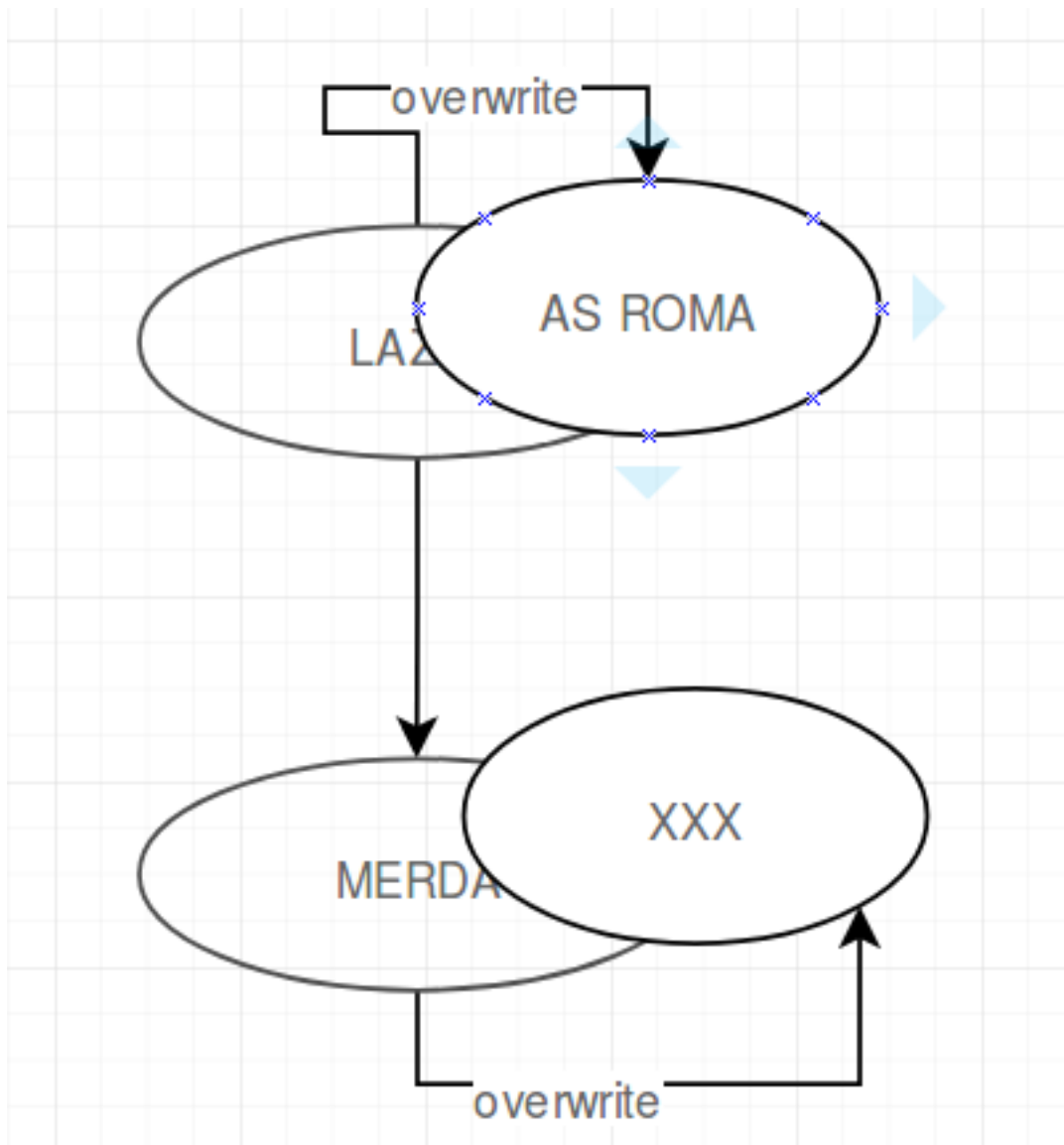
Imagine someone wrote to the wall ‘Lazio merda’.



Now another person came and erased Lazio and wrote above it ‘AS ROMA’



And in the last step the 'merda' was erased.



One could suppose that these three steps were the beginning of the fragment on the left side of the following image, which is part of Sebastian Lasch's corpus:



4.1.2.2 Storing the graph

Obviously storing a graph in a graph database does not need a lot of setups.

The basics of neo4j can be found here: <https://neo4j.com/blog/data-modeling-basics/>

Example for adding two connected nodes:

Cypher `CREATE (a:Token {text:"Lazio"})-[:follows]->(b:Token {name:"Merda"})`

Under the hood storing a graph in a relational database is not different to the graph database way to do it. But the main difference is that every node stores its own relationships. So no other look up is required to get from Node 1 to neighbouring Node 2.

Most of the graph databases also give the ability document oriented databases have that you don't have to stick to a given schema. So you don't have to define exactly at the beginning what attributes a node or an edge should have. (Of course reading the schema dynamically is performance downside)

SQL In SQL you are going to need a basic setup. At least a table with nodes and one with edges.

```
CREATE TABLE Nodes (
  id INT AUTO_INCREMENT,
  type VARCHAR(20),
  text VARCHAR(100),
  PRIMARY KEY(id)
)

CREATE TABLE Edges (
  fromNode INT,
  toNode INT,
  type VARCHAR(20)
)
```

Now insert the two nodes connected by one edge.

```
INSERT INTO Nodes (type, text) VALUES ("Token", "Lazio")
INSERT INTO Nodes (type, text) VALUES ("Token", "Merda")
INSERT INTO Edges (type) VALUES ("follows")
```

We stored the same graph in both databases. Of course there are more differences between the two database systems but for our example this shows enough of a difference.

4.1.2.3 Retrieving some data

In this section comes the big advantage of the graph database. Although relational databases work well on lots of data rows the graph databases are well equipped when it comes to retrieving and analyzing graphs.

Imagine the simple case you just want to get the words that directly follow after “Lazio”:

Cypher `MATCH (a:Token {text:"Lazio"})-[:follows]->(b:Token) RETURN b.text`

This will return a list of tokens that followed the nodes called “Lazio”.

SQL

Imagine a first try for this query. It may take us 2 minutes to write it.

```
SELECT text
FROM Nodes
WHERE id IN (
  SELECT toNode
  FROM Edges
  WHERE fromNode
  IN (
    SELECT id
    FROM Nodes
    WHERE text = "Lazio" AND type = "Token"
  )
);
```

This is already a quite long statement for such an easy case. We try to make it shorter and maybe more readable.

After 3 minutes we come up with this query:

```
SELECT n1.text
FROM Nodes AS n1
  INNER JOIN Edges ON n1.id = fromNode
  INNER JOIN Nodes AS n2 ON toNode = n2.id
WHERE n1.type = "Token" AND n1.text = "Lazio";
```

We were able to shorten it but it is still far away from being a one-liner.

4.1.2.4 Extending the example

Now imagine we would like to model the second step from the example scenario. Lazio was overwritten by “AS ROMA”.

Let’s add the new node and a ‘overwritten_by’ relation.

First in **Cypher**:

```
MATCH (a:Token {text:"Lazio"}) WITH a
CREATE (a)-[:overwritten_by]->(:Token {name:"AS ROMA"})`
```

Now in **SQL** (notice how it is not really possible to catch the last insertion ID):


```

INSERT INTO Nodes (type, text) VALUES ("Token", "AS ROMA");
INSERT INTO Edges (fromNode, toNode, type) VALUES
  (
    (SELECT id FROM Nodes WHERE type = "Token" AND text = "Lazio"),
    (SELECT id FROM Nodes WHERE type = "Token" AND text = "AS ROMA"),
    "overwritten_by"
  );

```

4.1.2.5 Querying just a little more

Now we would like to see what kind of words are followed **visibly** by merda. In our example this means that Lazio is not longer the correct answer but AS ROMA is. (Attention: overwriting over overwriting is possible but just the highest text shall be returned.)

After 10 minutes we come up with this.

Cypher

```

MATCH (a:Token {value:"Merda"})-[:edge {relation_type:"follows"}]-(b:Token)
OPTIONAL MATCH r=(b)-[:edge*{relation_type:"grak"}]-(c)
WITH LAST(relationships(r)) as relatio, b
WITH FILTER(rel in [b, relatio] WHERE rel IS NOT null) as relations
WITH LAST(relations) as relation
WITH relation ORDER BY relation.hand DESC LIMIT 1
MATCH ()-[relation]->(t:Token)
RETURN t;`

```

This query is not too simple but the problem is neither. Actually to solve such a problem recursive functions are necessary. Take a look at the second row: the asterisk stands for a variable number of hops. This is a powerful feature of neo4j. You can easily match recursive relations or others by adding only an asterisk.

SQL

In SQL we really have to program SQL now. We could write a CTE (common table expression) [https://technet.microsoft.com/en-us/library/ms186243\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/ms186243(v=sql.105).aspx) that provides recursion for that one special query we need.

Or we have a database that supports CONNECT BY. https://en.wikipedia.org/wiki/Hierarchical_and_recursive_queries_in_SQL

We would really have to get a programmer here to get the analysis going.

4.1.2.6 Result

We modeled a small graph example in SQL and neo4j. We tried to get some data out of it.

RDMS (relational database management systems) and graph databases are concurring in this field. RDMS are much broader, proven to work and really established.

But as we have seen there are cases in which it makes sense to switch to a GraphDB.

4.1.3 Possible use cases

This software was built for the special use case described in “Analyzing 3000 Graffiti” but it should be able to serve more general purposes.

Main Question: Who could use this software? The answer can only be given when someone decides to use GIAN T but we can speculate to whom it might be useful.

Maybe in order to analyze book annotations: .. image:: sources/images/book_ann.png

Or to analyze movements: .. image:: sources/images/provinzen.jpg

Maybe in order to show developments of city centres: .. image:: sources/images/map.jpg

If you need think about using GIAnt but don't know how to model your data please feel free to get in touch.

4.1.4 Continuous integration: Testing and deploying code

Continuous Integration is the idea to perform integration tests on a regularly base in order to avoid conflicts in the development process of software.

It involves automatic testing, building and deploying.

GIAnt has a good test coverage with close to 90%. The tests are written with the use of mocha.js and chai.js. The test coverage is calculated by instanbul.js. To recalculate the coverage locally run the command: `npm run cover`

Every JS-file in the test folder is recognized as a test and run automatically. You can run the test suite locally with the command: `npm run test`

Every commit to the projects triggers the automatic tests of GIAnt on CI platforms: * Appveyor for windows * Travis-CI for GNU/Linux and MacOS

If the tests were run successfully and the commit was tagged with a label, then GIAnt gets automatically built and uploaded to <https://github.com/DanielPollithy/GIAnt/releases> To build the application locally you can run the command: `npm run release`

4.1.4.1 How to write a test

You have to import the test suites, connect to the database and activate a chai syntax.

```
var mocha = require('mocha');
var chai = require('chai');

var database = require('../src/database');
database.login('bolt://localhost:7687', 'neo4j', '1234');

// Activate should-syntax (http://chaijs.com/guide/styles/#should)
chai.should();
```

Now you can write a synchronous test like this: .. code-block:: javascript

```
describe('your test area name', function() {
    it("your test case name", function(){
        var awaited_hash = '9ebc1323b5ebc175d79ef27d7b4363ea41fa7ac4';
        var hash = utils.hash_of_file_content('../test/misc/hash_example.txt')
        hash.should.equal(awaited_hash);
    })
})
```

And an **asynchronous** test like that: .. code-block:: javascript

```
describe('your other test area name', function() {
    it("chain promise zero length", function(done){
        utils.chain_promises([]).then(function(){done()}).catch(function(err){done()})
    })
})
```

4.1.4.2 Contributing

If you want to contribute to this project please remember that contributing also means writings tests!

4.1.4.3 Documentation

Sphinx with readthedocs is used to provide this documentation.

4.1.5 Details on the CI

Since this project needs to be build on multiple platforms it is a good idea to not just run tests automated but also use **TRAVIS-CI** and **AppVeyor** to build and deploy the application. The configuration files can be found in the root directory of the repository.

Travis is used for linux 32bit and 64bit builds and AppVeyor for both architectures for windows (called win32). See Deployment-Travis or Deployment-AppVeyor for configuration and documentation.

4.1.5.1 Trigger an automatic build

A build shall only be triggered on tag push. This is how to do this:

- Tag the commit in Git by `git tag -a v0.0.0 -m "Release 0.0.0"`
- Push the tag `git push origin v0.0.0`

Now the build triggers automatically the deployment to github releases. After some minutes there is a new release on the GitHub-Page with the zipped applications attached.

4.1.5.2 How to Deploy manually

If you made a change to the code and want to ship this change in an electron app follow this procedure

- Tag the commit in Git by `git tag -a v0.0.0 -m "Release 0.0.0"`
- Push the tag `git push origin v0.0.0`
- Package the electron app for your platform with `electron-packager`
- If you haven't done so, install `electron-packager` globally: `npm install -g electron-packager`
- If you are on the OS that you want to target with the build execute `electron-packager` (consultate the documentation of `electron-packager`)
- Compress the created folder. Example for linux: `zip -r [archive-name].zip [name of the folder]`
- Navigate to the releases tab of the github repository
- Edit your release: Add the compressed archive.
- Add the release to the download section of the documentation
- Describe the changes made in the CHANGELIST of the documentation

4.1.5.3 Deployment on Travis-CI

This is an explanation of the configuration and how the workflow works. If you know Travis-CI this will be boring.

4.1.5.4 What is Travis-CI

It is a CI (continuous integration) provider that is free to use for public git repositories. It provides virtual machines that can automatize jobs for you, e.g. run your unit tests or test your code with various code versions and a lot more.

4.1.5.5 Setup Travis-CI

It is easily setup if you have a github account. You can use the github account as a single sign on for Travis-CI.org. There you activate the wanted repository in the overview.

After you set that up, every time you push to the master branch Travis will start to work for you.

4.1.5.6 What does Travis do?

He (look at the logo) will pull our code and look for a configuration file called `.travis.yml`. This file contains all the things we want Travis to do for us.

In our case this is:

- downloading and installing the latest neo4j-community server
- install all other dependencies
- run the tests
- build the electron application and package it
- deploy the package to github releases (if the release doesn't exist so far, he will create one)

4.1.5.7 Detailed description of the Configuration

In Line 22 there is the installation script for neo4j triggered. This will only work on unix systems.

In Line 40 you see there is a BASH variable. If you want to use this deploy configuration you have to go to github.com, login into your account, from there create an access token with the option “repo” ticked and copy paste the generated token into the environment variables in the TRAVIS web frontend. **Don't forget to activate encryption for this variable** or else everyone can access your with ease.

```

1  osx_image: xcode7.3
2  sudo: required
3  dist: trusty
4  language: c
5  matrix:
6    include:
7      - os: osx
8      - os: linux
9      env: CC=clang CXX=clang++ npm_config_clang=1
10     compiler: clang
11  cache:
12    directories:
13      - node_modules
14      - "$HOME/.electron"
```

(continues on next page)

(continued from previous page)

```

15   - "$HOME/.cache"
16 addons:
17   apt:
18     packages:
19       - libgnome-keyring-dev
20       - icnsutils
21 before_install:
22   - ./install_neo4j.sh
23   - mkdir -p /tmp/git-lfs && curl -L https://github.com/github/git-lfs/releases/
24     ↪download/v1.2.1/git-lfs-$(
25     "$TRAVIS_OS_NAME" == "linux" ] && echo "linux" || echo "darwin")-amd64-1.2.1.tar.gz
26     | tar -xz -C /tmp/git-lfs --strip-components 1 && /tmp/git-lfs/git-lfs pull
27   - if [[ "$TRAVIS_OS_NAME" == "linux" ]]; then sudo apt-get install --no-install-
28     ↪recommends -y icnsutils graphicsmagick xz-utils; fi
29 install:
30   - nvm install 6
31   - npm install electron-builder@next
32   - npm install
33   - npm prune
34 script:
35   - npm run release
36 branches:
37   except:
38     - "/^v\\d+\\.\\d+\\.\\d+$/ "
39 deploy:
40   provider: releases
41   api_key: "$GH_TOKEN"
42   file_glob: true
43   file: "*.zip"
44   skip_cleanup: true
45   on:
46     tags: false

```

4.1.5.8 Deployment on AppVeyor

AppVeyor provides a similar service to Travis-CI but is focused on windows. So they provide a **Power shell** on a win32 host system you can configure.

You have to put a `appveyor.yml` file into the base dir of the repo which might look like this.

This configuration excludes the test because we already ran the tests on travis and at the moment you are going to have a hard time installing neo4j into their machines.

```

1 version: 0.1.{build}
2
3 platform:
4   - x86
5   - x64
6
7 cache:
8   - node_modules
9   - app\node_modules
10  - '%APPDATA%\npm-cache'
11  - '%USERPROFILE%\electron'
12

```

(continues on next page)

(continued from previous page)

```

13 init:
14   - git config --global core.autocrlf input
15
16 install:
17   - ps: Install-Product node 6 x64
18   - git reset --hard HEAD
19   - npm install npm -g
20   - npm install electron-builder@next # force install next version to test electron-
    ↪ builder
21   - npm install
22   - npm prune
23
24 build_script:
25   - node --version
26   - npm --version
27   - npm run release
28
29 test: off
30
31 deploy:
32   release: GIAnt-v$(appveyor_build_version)
33   description: 'GIAnt'
34   provider: GitHub
35   auth_token:
36     secure: QBn6bw8znM2WsrG32eTzA55Iu0iE6oymujVBos6XFUldN/biNahd6Csr6d9Y4u+E
37   artifact: '**\*.zip' # upload all NuGet packages to release assets
38   draft: true
39   prerelease: true
40   on:
41     branch: master # release from master branch only

```

4.1.6 Developing a Codec: Test driven development

The heard of this program is the codec which can create a graph in neo4j from a mxGraph input file.

An example for a mxGraph:

```

<mxGraphModel dx="811" dy="514" grid="1" gridSize="10" guides="1" tooltips="1"
↪ connect="1" arrows="1" fold="1" page="1" pageScale="1" pageWidth="806" pageHeight=
↪ "566" background="#ffffff">
  <root>
    <mxCell id="0"/>
    <mxCell id="1" style="locked=1;" parent="0"/>
    <mxCell id="2" value="Hand 1" parent="0"/>
    <object label="Token" color="ros" dialect="no" lang="ital" pos="NPR" them_mak="ult
↪ " tool="spr" id="4">
      <mxCell style="tokenType=token;rounded=1;whiteSpace=wrap;fillColor=#FFFFFF;
↪ opacity=50;html=1;" parent="2" vertex="1">
        <mxGeometry x="180" y="10" width="120" height="60" as="geometry"/>
      </mxCell>
    </object>
  </root>
</mxGraphModel>

```

4.1.6.1 The GraphML format

The coded can also transform the arbitrary mxGraph format to the open GraphML standard, which looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<graphml>
  <key id="label" for="node" attr.name="label" attr.type="string"/>
  <key id="color" for="node" attr.name="color" attr.type="string"/>
  <key id="dialect" for="node" attr.name="dialect" attr.type="string"/>
  <key id="lang" for="node" attr.name="lang" attr.type="string"/>
  <key id="pos" for="node" attr.name="pos" attr.type="string"/>
  <key id="them_mak" for="node" attr.name="them_mak" attr.type="string"/>
  <key id="tool" for="node" attr.name="tool" attr.type="string"/>
  <key id="id" for="node" attr.name="id" attr.type="string"/>
  <key id="parent" for="node" attr.name="parent" attr.type="string"/>
  <key id="vertex" for="node" attr.name="vertex" attr.type="string"/>
  <key id="hand" for="node" attr.name="hand" attr.type="string"/>
  <key id="x" for="node" attr.name="x" attr.type="string"/>
  <key id="y" for="node" attr.name="y" attr.type="string"/>
  <key id="width" for="node" attr.name="width" attr.type="string"/>
  <key id="height" for="node" attr.name="height" attr.type="string"/>
  <key id="as" for="node" attr.name="as" attr.type="string"/>
  <key id="tokenType" for="node" attr.name="tokenType" attr.type="string"/>
  <key id="rounded" for="node" attr.name="rounded" attr.type="string"/>
  <key id="whiteSpace" for="node" attr.name="whiteSpace" attr.type="string"/>
  <key id="fillColor" for="node" attr.name="fillColor" attr.type="string"/>
  <key id="opacity" for="node" attr.name="opacity" attr.type="string"/>
  <key id="html" for="node" attr.name="html" attr.type="string"/>
  <graph id="G" edgedefault="directed">
    <node id="4">
      <data key="label">Token</data>
      <data key="color">ros</data>
      <data key="dialect">no</data>
      <data key="lang">ital</data>
      <data key="pos">NPR</data>
      <data key="them_mak">ult</data>
      <data key="tool">spr</data>
      <data key="id">4</data>
      <data key="parent">2</data>
      <data key="vertex">1</data>
      <data key="hand">Hand 1</data>
      <data key="x">180</data>
      <data key="y">10</data>
      <data key="width">120</data>
      <data key="height">60</data>
      <data key="as">geometry</data>
      <data key="tokenType">token</data>
      <data key="rounded">1</data>
      <data key="whiteSpace">wrap</data>
      <data key="fillColor">#FFFFFF</data>
      <data key="opacity">50</data>
      <data key="html">1</data>
    </node>
  </graph>
</graphml>
```

4.1.6.2 How can we verify that the codec works as wanted?

Verification is always difficult. Because the codec is a input-output program without sideeffects testing can be seen as an appropriate way.

In order to make sure that the tests make sense and the coverage is high Test Driven Development (TDD) was employed.

4.1.6.3 What is the biggest XML file it can transform

The codec is not optimized to work on big files but a few thousand elements with even more attributes are possible. Increasing the performance of the codec can be achieved easily.

4.1.6.4 Batch-add (+checksum)

By clicking on the “Batch-add”-Button all fragments are checked whether they have changed or not. Only if they have changed they are updated in the database. A SHA1 hash from the file original mxGraph XML file is stored every time one of these files is transferred to the database.

4.1.7 Electron for cross platform applications

Operating systems differ. Especially when it comes to writing graphical user interfaces. That is the reason why “cross platform frameworks” exist. They try to provide general functions that work on every major operating system.

If you want to build a desktop application you first have to decide whether you want to use a “cross platform framework” and if so, which one?

4.1.7.1 Overview of cross platform frameworks

Qt and wx have been the best known frameworks. But since the beginning of the 10s the electron framework is there. Developed by the company behind Github and driven by their own need as they were writing the code editor Atom.

The basics of their new framework are not new. They package a light-weight browser (in particular a version of Google Chrome) with the runtime environment for Node.JS execution.

As a consequence you can write a web application and ship it as a desktop application. This procedure has the advantages of the web applications and and also that it could easily be provided as one in the future.

5.1 Technical guides

5.1.1 Constraints

If you want to ensure the integrity of your data it is a good idea to use constraints.

You can add the in the GUI or to /media/settings/settings.json.

The constraints are evaluated from the GUI. They are currently not automatically called because they were not used in the first project.

```
"count_constraints": [
  {
    "id": 1498744279660,
    "min": "0",
    "max": "3",
    "query": "MATCH(f:Fragment)-[]-(t:Token) WHERE ID(f) = {fragment_id} RETURN t;
↪"
  }
],
"free_constraints": [
  {
    "id": 1498746658301,
    "query": "// session = a neo4j session\r\n// session.run(cypher_string)
↪returns a promise (see the docs)\r\nnew Promise(function(resolve, reject){\r\n
↪var variables = {\r\n\"fragment_id\": fragment_id;\r\n    session.run(\r\n
↪\"MATCH(f:Fragment)-[]-(t:Token {value: 'Token'}) WHERE ID(f) = {fragment_id} RETURN
↪t.value as value;\r\n\", variables)\r\n    .then(function(result){ \r\n
↪var value;\r\n        result.records.forEach(function(res){\r\n
↪value = res.get('value');\r\n        if (value === \"Token2\") {\r\n
↪    reject(\"There was a token called Token.\");\r\n        }\r\n
↪    });\r\n        resolve();\r\n    }).catch(function(err){\r\n
↪reject(err);\r\n    });\r\n});"
```

(continues on next page)

(continued from previous page)

```
    }
  ],
  "bool_constraints": [
    {
      "id": 1498747292635,
      "query": "MATCH (f:Fragment)-[]-(t:Token {value: 'Token'}) WHERE ID(f) =
↪ {fragment_id} RETURN COUNT(t) > 0;"
    }
  ]
]
```

Every constraint has a unique identifier which is automatically the unixtime if the constraint was created by GIAnt.

5.1.2 Editor settings

The file `js_editor_settings.js` is hooked into the GUI in the settings tab. You can edit it there or in `media/settings/` directly. Be careful with this file.

It is injected into the `mxEditor` on startup. You could even make changes to the editor from this file.

The `TOKEN_CONFIG` object file is primarily used for the default properties an edge or a node can have.

The `CUSTOM_PROPERTY_CHANGE_HANDLERS` is called every time the user applies changes made to the properties.

The following code shows how a relation's color can be changed depending on a property.

5.1.3 Autocomplete functions

The autocomplete functions are provided by a http json endpoint of the application's server.

The following urls are called to retrieve the keys and values:

- `/autocomplete/:token_type/values`
- `/autocomplete/:token_type/keys`

Where `:token_type` has to be one of the hard coded strings:

- `'modification'`,
- `'token'`,
- `'symbol'`,
- `'comment'`,
- `'frame'`,
- `'blanco'`

If you are using this application and the hard coded terminology is bothering you feel free to contact me.

The query to get all property keys looks like this:

```
MATCH (p:Label {...}) WITH DISTINCT keys(p) AS keys
UNWIND keys AS keyslisting WITH DISTINCT keyslisting AS allfields
WHERE allfields CONTAINS {search_string}
RETURN allfields;
```

5.1.4 Exif data extraction

When an image is uploaded in the JPG format it is automatically processed by the “exif” package. The exif date format is parsed by the “exif-date” package.

If there was no exif meta data the current date is used.

5.1.5 Image dimensions

The npm package “image-size” is used to get the dimensions of an uploaded image.

5.1.6 Where are which files

Every file used by the application which belongs to the user is stored in the media folder. Its subfolders are:

- export
- settings
- uploaded_images
- uploaded_xmles

In the export folder all zipped exports from the export tab are stored.

In the settings folder the editor settings and constraints are stored.

uploaded_xmles contains all xmles with the fragment’s primary key as name.

5.1.7 Making a backup

See the section in the general documentation!

5.1.8 Corrupted database

In the case your database is corrupted you can rebuild with your data from the filesystem.

- Step 1) Export your data as CSV. Backup the <Giant-App>/resources/app/media/uploaded_xmles/ folder.
- Step 2) Stop GIANt, stop Neo4j, create a new database for neo4j, start it and restart GIANt
- Step 3) Your data is located here: <Giant-App>/resources/app/media/

Make sure uploaded_xmles/ contains your xmles, uploaded_images/ your images and settings/ your settings

If that’s not the case copy them to these locations.

- Step 4) Upload the dumped CSV files (Relations: *csv-relations-.csv*, Node properties: *csv-nodeprops-.csv*)

ATTENTION: This function will overwrite the current active neo4j database.

5.1.9 Express.js and pug

Express.js server is configured and the views are written in the file server.js!

The rendering engine pug is used. The view’s templates are located under the folder src/views.

5.1.10 Middleware

The login view template is 'db_settings.pug'. There is a middleware registered in the server which checks whether a user is logged in. If not, the mentioned login view is displayed.

5.1.11 Logging

The server logs to the following default locations via the package electron-log:

- on Linux: ~/.config/<app name>/log.log
- on OS X: ~/Library/Logs/<app name>/log.log
- on Windows: %USERPROFILE%\AppData\Roaming\<app name>\log.log

5.1.12 Electron application

The desktop application is started from within the atom.js file. It creates the window instance and opens the first URL.

Currently the same process also hosts the server which can be seen as a practice which is improvable.

5.1.13 Database

The database with its methods are well described with the YUIDocs API description.

```
/**
 * Get the fragment of an image by name
 *
 * @method get_fragment
 * @param image_file_path {String} The unique id for the image
 * @param fragment_name The identifier for the fragments
 * @return {Promise}
 */
Database.get_fragment = function(image_file_path, fragment_name) {
  var session = this._get_session();
  var prom = session.run("MATCH (a:Fragment {fragment_name: {fragment_name}}
    \"RETURN ID(a) as ident, ID(i) as image_id, a.fragment_name AS fragment
    {fragment_name: fragment_name, file_path: image_file_path})
    .then(function (result) {
      session.close();
      var records = [];
```

```
// get the database singleton
var database = require('../src/database');

// Login and handle the promise
database.login('bolt://localhost:7687', 'neo4j', '1234').then(...).catch(...);

// Add image or anything else provided by the database API
database.add_image(...)
```

(continues on next page)

(continued from previous page)

```
// Run custom cypher query
database._get_session().run("...");

// logout
database.logout();
```

Good to know The `database._hygiene()` is called automatically on login. It removes elements that are not connected to others. E.g. images without fragments.

6.1 Keyboard shortcuts

Where can I find the list of keyboard shortcuts?

From within the editor go to the menu entry Help > Help.

6.2 Why only jpeg

I want to upload png! Why is only jpeg allowed?

That is because of the exif data which can only be read from the jpeg files.

6.3 The time is behind

It is noticeable that the timestamps are shifted some hours. Why is that so?

The times are in Universal Time (UT former GMT).

6.4 Batch add does not create elements in the database

When I look into the database I see no elements were created by the batch add?

The batch add only works with fragments which are marked as “ready”.

7.1 v1.0

7.1.1 v1.1.0

Added the “restore from filesystem” feature the database tab of the application.

7.1.2 v1.0.1

Improved the SQL export (restructured the FOREIGN KEY part of the queries).

7.1.2.1 v1.1.1

Chained batch-add and chained heatmap queries in order to make them deterministic.

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`