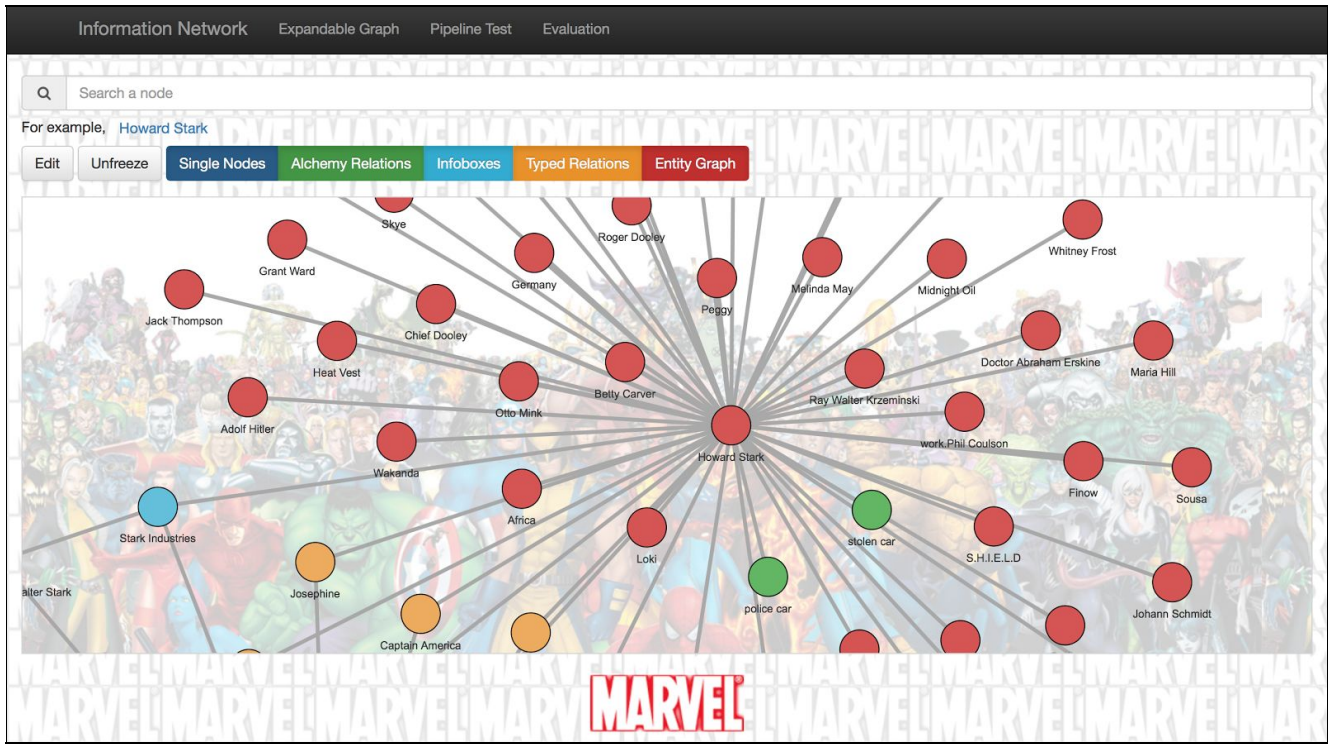
Information Network

Question Answering Technologies behind and with IBM Watson
Documentation - SS2016
Simon Schimmels, Christoph Schott, Simon Dif, Kai Steinert

TECHNISCHE UNIVERSITÄT DARMSTADT



Contents

1. Introduction
2. Data
3. Services

3.1. Alchemy, especially the Alchemy Language

   a. Entity Extraction
   b. Typed Relation Extraction
   c. Relations Extraction
   d. Keyword Extraction

3.2. IBM Graph Database

   a. Schemata
   b. Gremlin Language

4. Pipeline

4.1 Simple Pipeline - on the fly evaluation of an file

4.2 “Big” Pipeline - for all extracted Informations

5. Web Application - what can you do

5.1 Expandable Graph

5.2 Simple Pipeline

5.3 Evaluation

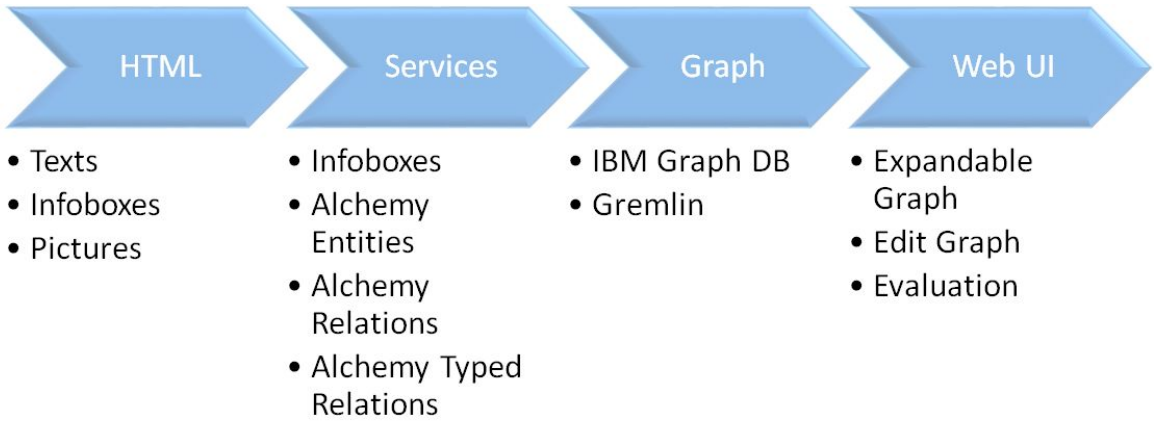
6. Evaluation
7. Discussion
8. Future Work
9. Who did what

1. Introduction

The problem with many series, film productions or book collections is that their content/story is often strongly dependent on previous episodes, films or books which spread over many years. Therefore it often happens that the viewer/reader gets lost.
With our project we want to constitute an easier overview over these complex network structures and contents.
An example of use would be, the recently published Harry Potter book.
The last book was published in 2007 and before that, there were 6 more books, of which the first was revealed in 1997. So we think that maybe you wouldn't remember more than the coarse story.
So what would you do ? Don't you care? Would you read the seven books again? Watch the movies? Now with this project we want to offer an approach with which one can easily work up forgotten contents and contexts again while reading or watching. We build an Information network.

In the following figure we can see the roughly project components and their order. We have divided the task with respect to the front-end and back-end.
The back-end consists of the first 3 components: HTML, Services and Graph.
We used the crawled data from the Marvel Wikia as a data lake. Our main focus is on IBM Watson services. Watson is a question answering system that IBM built. It works with several technologies like natural language processing, information retrieval, machine learning etc. and is used for the field of open domain question answering. Besides IBM services, we also used static information from the infoboxes.
We have built two different pipelines. The first one uses the services and static computations on the fly for one file. The second one is intended for a large amount of data. First the services are used to analyze the wiki articles content and extract entities and relationships that are then stored. The stored data are combined as a result of the pipeline in the IBM Graph Database. In addition to the standard Rest API, we have made it possible to use the graph query language Gremlin.

The front-end consists of the last component, the Web UI. We built a web application that displays the graphs and further information, allows to interact with the graph and the different services and permits the evaluation of the results.

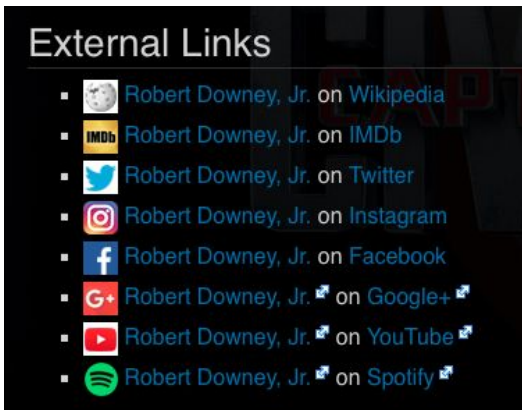


2. Data

The crawled data from the Marvel wikia (http://marvel.wikia.com/wiki/Marvel_Database), which is provided to us by another project group, is the source for all the data that we analyze.
We kept our entire application as general as possible such that any other wikia can be used instead of our Marvel Cinematic Universe Wikia.

In total 7365 files are the result of crawling this wikia, each file correspond to a webpage.
For each file, we extract the important informations for our use case (title, url, text content, etc.).

At the beginning we would only use the url that is stored in each file, because of the IBM Watson's Alchemy Service can work with them directly and extract the text from the corresponding page.
Due to the variety of elements and special features, this unfortunately delivered sometimes poor results, mostly based on transforming metadata into text.
Then we used the crawled html files directly. The result was significantly better, but unfortunately there were still much meta content included, which doesn't contribute to real article content.



A problem was, for example, the external links section. For example, it is right to recognize, that there are entities like Wikipedia, Instagram, YouTube etc., but they are not important for the understanding of the content and its combined relationships. So the Wikipedia article may still make sense, but Spotify, Youtube, etc. are rather uninteresting. All of these entities would later be considered very important because of their frequent occurrence.

Hereafter we decided to look more into the details of the crawled html files. There, we discovered that the textual information which is important for understanding the content is always in paragraph marks (<p>).

```
<section>
<h2>Captain America</h2>
<p>TALK PAGE</p>
<p>Captain Steven "Steve" Grant Rogers is a Super Soldier
world's first superhero. After a top secret Super-Soldier
Steve Rogers into the powerful and heroic Captain America,
exploits made him a living legend. Rogers attacked multipl
Howling Commandos, to the dismay of the Red Skull. Rogers
but crashed into the Arctic during his final mission. Awak
learned that he had spent 67 years trapped in the glacial
<p>Steven Rogers found himself alone in a modern world tha
Nick Fury, director of the international peacekeeping agen
Rogers to help save the world again, he quickly suited up
iconic shield and bringing his strength, leadership, and n
the Avengers in the Chitauri Invasion.</p>
<p>After fighting alongside the Avengers, Rogers became a
completed many operations with fellow agent Black Widow, w
Avengers. Along with Maria Hill and Falcon, they destroyed
HYDRA Uprising, he went off on his own to search for his f
help of Sam Wilson. In the midst of his quest, Rogers reas
working to bring down what was left of HYDRA, and made use
again in the battle against the psychotic artificial intel
members of the original Avengers, Rogers remained as a mem
incarnation of the team after the Ultron Offensive.</p>
<p>In response to the massive loss of life caused by the A
ordered by Thaddeus Ross to sign the Sokovia Accords which
have to operate under the rule of a governing body. Rogers
disagreed on the issue and when the Winter Soldier became
disobeyed his orders to protect his friend. This action be
which Rogers and a small team fought against Iron Man's te
eventually learning that Helmut Zemo was behind the devast
the Winter Soldier had killed his parents, Rogers was forc
with Black Panther and go on the run.</p>
</section>
```

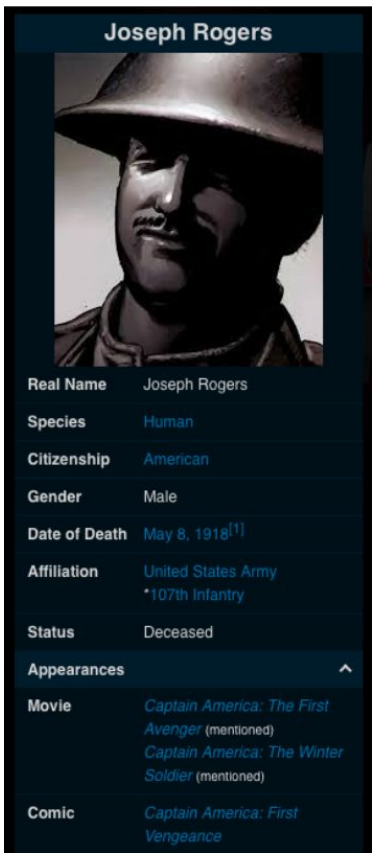
On the left, we can see an example excerpt from the overview section regarding the Captain America article.
So we have decided that we combine all paragraphs of a WebPage to one continuous text.
We have done this for all files and for each file we saved this text, title, url and original filename in a Json object.
The schema is: ***filename|link|text|title***
For example:

```
{
    "filename":"Adolf Hitler-3439",
    "link":"http://marvelcinematicuniverse.wikia.com/wiki/Adolf_Hitler",
    "text":" Adolf Hitler was the Führer und Reichskanzler of Nazi Germany
    His reign of terror also gave rise to another great war criminal, Joha
    Special Weapons Division. There, he met Johann Schmidt, the German phy
    That purge later became known as the Night of the Long Knives. Hitler
    However, Schmidt saw his new post more as an exile than the reward. Hi
    As stated by Schmidt, Hitler has sent several expeditions to search fo
    "title":"Adolf Hitler"
}
```

In a later status meeting we decided to use the infobox of each file, if it exists. In contrast to the previously extracted text information on which we want to work with the Alchemy Services, these infobox informations can be directly extracted to relationships.

The extraction of entities (nodes) and relationships (edges) works as follows: Node 1 is always the Person, Place, Organisation and so on, that the web page is about. The Edge and Node 2 are directly extracted from the Infobox. We have added the type, so that we are able to distinguish later in the application from where the relationship comes and we can filter between the different types.

For example, we extracted the relationships of the following web page: http://marvelcinematicuniverse.wikia.com/wiki/Joseph_Rogers




```
<h2>Infobox Joseph Rogers:</h2>
    <table>
        <tr><td>Real Name</td><td>Joseph Rogers</td></tr>
        <tr><td>Species</td><td>Human</td></tr>
        <tr><td>Citizenship</td><td>American</td></tr>
        <tr><td>Gender</td><td>Male</td></tr>
        <tr><td>Date of Death</td><td>May 8, 1918[1]</td></tr>
        <tr><td>Affiliation</td><td>United States Army, *107th Infantry</td></tr>
        <tr><td>Status</td><td>Deceased</td></tr>
        <tr><td>Movie</td><td>Captain America: The First Avenger (mentioned), Captain Ame
        <tr><td>Comic</td><td>Captain America: First Vengeance</td></tr>
    </table>
</section>
```

```
[
    {"edge":"Real Name","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Joseph Rogers"},
    {"edge":"Species","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Human"},
    {"edge":"Citizenship","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"American"},
    {"edge":"Gender","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Male"},
    {"edge":"Date of Death","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"May 8, 1918"},
    {"edge":"Affiliation","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"United States Army"},
    {"edge":"Affiliation","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"107th Infantry"},
    {"edge":"Status","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Deceased"},
    {"edge":"Movie","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Captain America: The First Avenger (mentioned)"},
    {"edge":"Movie","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Captain America: The Winter Soldier (mentioned)"},
    {"edge":"Comic","nodeOne":"Joseph Rogers","type":"infobox","nodeTwo":"Captain America: First Vengeance"}
]
```

Problems with the infoboxes:

1. A comma was used to separate the entries. This is not a good separator since it is also used elsewhere, e.g. for a date, therefore the date must be handled differently,
2. Variety of the Infoboxes. Infoboxes come with multiple different formats and this can be very tedious to adapt to them.

For example: http://marvelcinematicuniverse.wikia.com/wiki/Cold_War

On the left, we can see an excerpt of the infobox, and on the bottom, we can see what the crawler creates.
So if you currently crawl this infobox, the crawler cannot handle them, and produces a wrong parsing, which has no real meaning, especially for our graph/relationship creator.

```
<tr><td>United States        NATO        S.H.I.E.L.D.</td>
<td>        HYDRA,        Soviet Union        Warsaw Pact        Cuba</td></tr>
```

At the end, we used the url from each crawled html file to load the uncleaned html code and extract the link to the main picture, that is placed in the infobox and downloaded it. So we could later use it as additional information.

Note:

- Not all crawled web pages have an infobox e.g. http://marvelcinematicuniverse.wikia.com/wiki/0-8-4_(Code)
- Not all crawled web pages have a figure placed in the infobox, but most of them have one,
- There are web pages that are removed e.g. http://marvelcinematicuniverse.wikia.com/wiki/Ali_Hillis

3. Services

All other services that we used come from the services provided by IBM Watson. On the one hand, we use the Alchemy Language Services to extract information from the text. Each service may be performed independently of the other. As we will see later, these will be partly set in relationships. On the other hand we use the IBM Graph to store the extracted relations information and combine them into a whole graph.

3.1 Alchemy, especially the Alchemy Language

AlchemyAPI offers a range of services to help to build applications for the analysis and interpretation of the content and context of texts on web pages, in news articles etc.

The following functions are available:

- Entities
- Keywords
- Concepts
- Taxonomy
- Document emotion
- Targeted emotion
- Document sentiment
- Targeted sentiment
- Typed Relations
- Relations
- Title
- Authors
- Publication Date
- Language
- Text Extraction
- Feeds

Alchemy Language is a set of text analysis APIs that use natural language processing to derive semantic information from your content.

Your Input can be an HTML-file, an url, or text.

If you want to test the different functions quickly, the following demo is recommended:
https://alchemy-language-demo.mybluemix.net/

The API references can be found under the following link:
http://www.ibm.com/watson/developercloud/alchemy-language/api/v1/

Note: If you use the services you must note that they allow different input sizes and these sizes may vary after the input type. For example:

Maximum size of source text after page cleaning: 50 KB
Maximum size of requested or posted HTML documents: 600 KB
Source text in Typed Relations calls is truncated to 5 KB

In our code we have implemented an equal-frequency approach. So that we can ensure that the text is not only divided in parts, but also that each call approximately contains the same amount of data, since it is so that many services make use of the surrounding information of the text (e.g. the entity recognition)
For example: when the maximum size of text, a service can work with, is 50KB and you have one file which is 52KB big, you divide this file not into a 50KB Part and a 2KB Part, but in two 26KB parts, such that each part contains the same amount of information.

Note:

1. The student license includes one thousand service calls per day. Some services can make another intern call, for example, for entity extraction, if it is enabled, like the Alchemy Relation Extraction. So it can be that with one call, you use 2 or 3 of these calls.
2. To short text's are a problem, because Alchemy is often not sure about the language and throws an error. This is often the case when the excerpt is short and the names occurring in it, are not clearly english.

```
i=3479 : ./extractedInformationNEW/Jerzy Skolimowski-4020.json
Sep 03, 2016 9:17:37 AM com.ibm.watson.developer_cloud.service.AlchemyService execute
SCHWERWIEGEND: {"error":"unsupported-text-language","code":400}
```


```
{
    "filename":"Jerzy Skolimowski-4020",
    "link":"http://marvelcinematicuniverse.wikia.com/wiki/Jerzy_Skolimowski",
    "text":" Jerzy Skolimowski portrayed Georgi Luchkov in The Avengers.",
    "title":"Jerzy Skolimowski"
}
```


Some other examples:

- Akua Santewa portrayed Protestor #1 in Luke Cage.
- Sahar Bibiyan portrayed Gulmira Mom in Iron Man.
- Vanessa Bednar portrayed Viking Villager in Thor.
- Daniel Eghan portrayed Hospital Visitor in Doctor Strange.

The AlchemyAPI uses a predefined domain model for all its services. This can be individually learned for a special domain. This is very time consuming and has high costs. So we used a standard model for English websites and news content.
To learn a model for a new domain, you must use the Watson Knowledge Studio.

First entrance points may be:

- https://www.ibm.com/marketplace/cloud/supervised-machine-learning/us/en-us
- http://www.ibm.com/watson/developercloud/doc/alchemylanguage/customizing.shtml

3.1.a Alchemy Entity Extraction

This Service extracts the entities from a plain text, web page or HTML. Because we use the standard model, we could have 42 primary types and 976 sub types for the recognized entities.

To get a better understanding about the types, the primary types are listed here:
Anatomy, Anniversary, Automobile, City, Company, Continent, Country, Crime, Degree, Drug, EntertainmentAward, Factility, FieldTerminology, FinancialMarketIndex, GeographicFeature, HealthCondition, Holiday, JobTitle, Movie, MusicGroup, NaturalDisaster, OperatingSystem, Organization, Person, PrintMedia, Product, ProfessionalDegree, RAdioProgram, RadioStation, Region, Sport, SportingEvent, StateOrCountry, Technology, TelevisionShow, TelevisionStation, EmailAddress, TwitterHandle, Hashtag IPAdress, Quantity, Money

On the right, we can see the extracted Entities from the Joseph Rogers articel (http://marvelcinematicuniverse.wikia.com/wiki/Joseph_Rogers)


```
{
  "count":7,
  "text":"Joseph Rogers",
  "type":"Person",
  "relevance":0.904197
},
{
  "count":4,
  "text":"Steve Rogers",
  "type":"Person",
  "relevance":0.581804
},
{
  "count":3,
  "text":"Sarah Rogers",
  "type":"Person",
  "relevance":0.369227
},
{
  "count":1,
  "text":"107th Infantry Regiment",
  "type":"Organization",
  "relevance":0.243262
},
{
  "count":2,
  "text":"World War",
  "type":"FieldTerminology",
  "relevance":0.213352
},
{
  "count":1,
  "text":"United States Armed Forces",
  "type":"Organization",
  "relevance":0.211032
},
{
  "count":1,
  "text":"United States Army",
  "type":"Organization",
  "relevance":0.176365
},
{
  "count":1,
  "text":"Purple Heart Medal",
  "type":"Organization",
  "relevance":0.154761
},
```

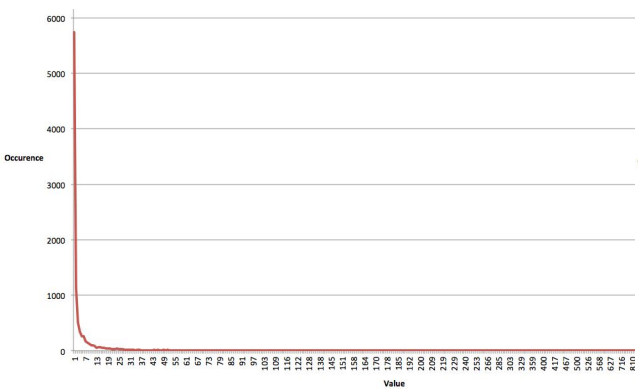

The text entry is the detected Entity in the text.
The other both values are defined as:

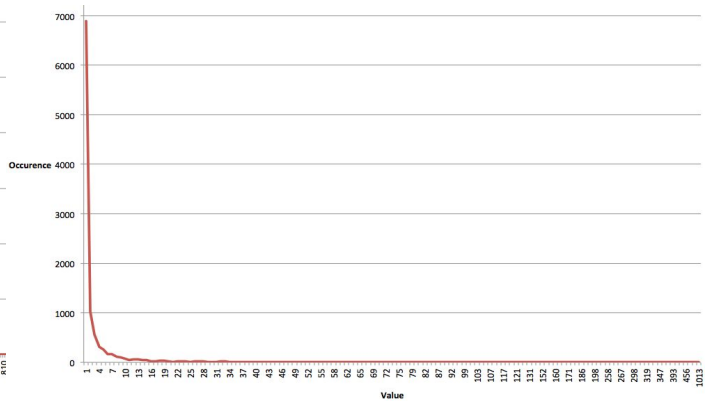
- Count: how often an entity is noticed
  - For example:
    - Joseph Rogers -> count = 1
    - Detect that e.g “joseph” or “he” in the context stand for Joseph Rogers -> count = 2
- Relevance: depicts the significance of each unique term
  - The higher the relevance score, the more important Is that term for the central meaning of the document

Some failures we have recognized using this service:

- Mixing different entities, e.g. Chan.Â Raina
- Extract the wrong type, e.g. take a look at the last entity “Purple Heart Medal” on the right site

The following two diagrams serve only to get a rough overview of the entity frequency.
On the x-axis we can see the Value, for example an entity occurs one time. On the y-axis we can see the Occurrences, for example, there are 6894 entities that occur only once.
On the left we see the distribution for the Entities, if we count them only once per File and this for all files and on the right, we count the multiple occurrences in a file for all files.





For both cases, the most common 15 elements are listed below.
As you can see, the most 15 differ slightly in both cases. We think that this is related to the fact that in the left table, there are the entities that are more relevant to the entire Marvel Universe. And that the right table also gives us good references to entities that are important only in a particular area.

Most popular 15 Entities - Counted only once Per File	
Entity	**Occurrences**
Captain America	1013
Phil Coulson	641
S.H.I.E.L.D.	567
HYDRA	456
Tony Stark	432
Skye	408
New York City	393
Jessica Jones	381
Grant Ward	368
Melinda May	347
Leo Fitz	326
Ultron	321
United States	319
Civil War	303
Jemma Simmons	299

Most popular 15 Entities - Each occurrence in a file is counted	
Entity	**Occurrences**
Phil Coulson	6930
Skye	4593
Peggy Carter	4200
Tony Stark	3730
Grant Ward	3697
Howard Stark	3598
Wilson Fisk	3130
Matt Murdock	2901
Jemma Simmons	2683
Leo Fitz	2473
HYDRA	2467
S.H.I.E.L.D.	2438
Captain America	2427
Thor	2307
Jessica Jones	2306

3.1.b Alchemy Relations Extraction

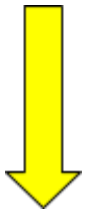
This Service extracts the SOA Relations from a plain text, web page or HTML.
SOA stands for:

- S - Subject
- O - Object
- A - Action

The good thing at this service is, that you can restrict the service, that the response contains only elements that contain at least one entity. This has removed innately a lot of the garbage in the response. The remaining subjects and objects are filtered again through the extracted entities.
In the following we can see an example from the Joseph Rogers article again.


```
{
    "sentence":" After Joseph died, Sarah had to raise Steve by herself, lamenting that
                 Joseph left them too soon, and noting that Steve was very much like his
                 father, wanting to follow Joseph's steps as a soldier.",
    "subject":{"text":"Sarah"},
    "action":{"lemmatized":"have to raise","verb":{"text":"raise","tense":"past"},
              "text":"had to raise"},
    "object":{"text":"Steve"}
}
```


Generate Relation



Simple string matching for the subject for Graph and object with the alchemy entities that were found in the same file.


```
{
    "subject":  "Sarah Rogers",
    "action":   "had to raise",
    "object":   "Steve Rogers"
}
```


What is important to note is that the service does not always reflect the complete triple. So it could be that you only get values for two of the three. In this case we have discarded the answer.

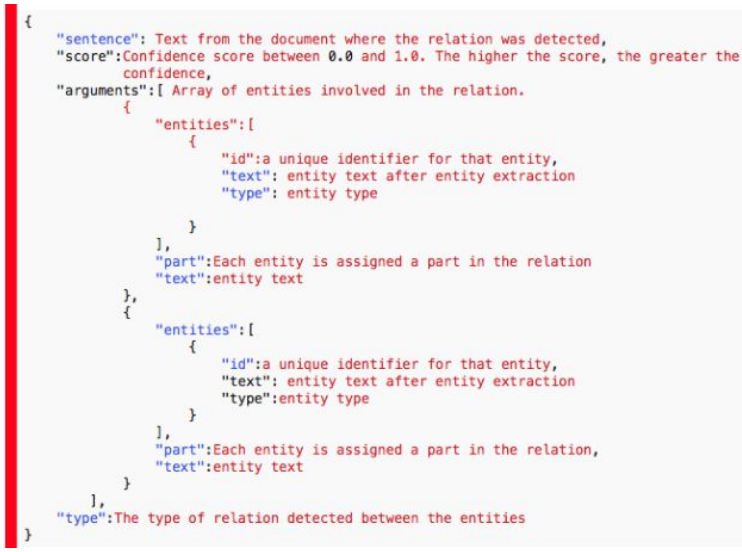
3.1.c Alchemy Typed Relations Extraction

This service extracts the Typed Relations from a plain text, web page or HTML. It uses the predefined model to identify the Typed Relation between two detected entities.
For the confidence you get a score between 0.0 and 1.0. So the higher the score, the greater is the confidence. The service tries to recognize the entities using the entity service. This doesn't work for each entity, so we later filter the relations with not so good entities with the help of our extracted entities out. The service also defines a type for its extracted entities as well as for the relation.
The possible types are defined by the previously created Watson Knowledge Studio model. So here we note only the relationship defined in the standard english model.

Some examples for the entity type: person, organization, geopoliticalEntity
Some examples for the relations type: parentOf, partOfMany, employedBy, agentOf, loctedAt

Until about 2 months ago this service was still in beta status and not integrated into the AlchemyAPI. At this time, the entities were not be very good either.

In the following we can see the response of the service for one typed relation on the left and an explanation of the fields on the right.



Out of this, the following is extracted for our graph:

entityOne*|*typeEntityOne*|*entityTwo*|*typeEntityTwo*|*relaiton*|*score*|*sentence*|*url

For the above example we would get:

```
entityOne: I. Joseph Rogers
typeEntityOne: Person
entityTwo: Steve Rogers
typeEntityTwo: Person
relation: parentOf
score: 0.824973
sentence: Joseph Rogers was the father of Steve Rogers, and a soldier with the 107th
          Infantry Regiment of the United States Army
url:http://marvelcinematicuniverse.wikia.com/wiki/Joseph_Rogers
```

We extract here only the relations, in which the entities make sense and have not been filtered out.

3.1.d Alchemy Keyword Extraction

This service returns all the keywords for a given text.
At the beginning, we would use this service to present further information for the individual articles. After we have tested this service, we found out that we didn't really get an improvement of knowledge compared to the entities we already have in the graph. So we skipped it and used for further information the url's to the relevant websites and text excerpts, from which the vertex or node originate. The service is still available and can be integrated as additional information for the graph.

```
[
    Rogers=0.983888,
    Steve Rogers=0.721584,
    Captain America=0.711578,
    Peggy Carter=0.711301,
    frail Steve Rogers=0.664395,
    Steven Grant Rogers=0.657844,
    War II. =0.656162,
    Steven Rogers=0.651402,
    Joseph Rogers=0.65137,
    enhanced physicality Rogers=0.647241,
]
```

3.2 IBM Graph Database

We use the service IBM Graph to store and create our graph, which is a graph database, based on Apache TinkerPop. In our project we use IBM graph to store our graph: nodes, edges and deeper information generated by the services will be saved and later queried by the Web Application. This means that the graph database acts like a middleware between backend and frontend of our application.

The graph database can be accessed via a RESTful interface. It is possible to add, edit and delete graphs, nodes, edges and schemata.

3.2.a Schema

Like any other database, IBM Graph needs a schema to be defined. In the following section, the used schema and its components are described. Once set, the schema defines what information we save in the database, considering project requirements and future uses of the database.

IBM graph distinguishes between two kinds of data elements: nodes and edges. These types are equivalent to the nodes and edges from the graph theory: The graph basically consists of a set of each type.

The graph database uses the schema displayed in the figure on page 14, which will be explained in the following: A schema in IBM graph contains five keys: *“propertyKeys”, “vertexLabels”, “edgeLabels”, “vertexIndexes”* and *“edgeIndexes”.*

The *vertexLabels* and *edgeLabels* contain arrays of only Strings, which will be used to label either a node or an edge. A node or an edge can only have one label that can’t be changed later, and that has to be one of the predefined labels. The labels we chose for the edges, are the most used types found by the Alchemy Typed Relations Extraction (see also section 3.1.c), whereas the labels we chose for the vertices are the most used types used by the Alchemy Entity Extraction (see also section 3.1.a). So basically the vertex labels and edge labels from our graph database represent the types of the entities and relations found by the said services. If a service found a type which is not in the list of our predefined labels, then the label *“otherV”* (for vertices) or “*otherE*” (for edges) will be used. In that case the property *otherVertexLabel* or *otherEdgeLabel* will be set with the found type.

The first key of the schema *propertyKeys* contains a list of properties which can be added to a vertex or an edge. Each property key of this list has a name and a datatype, both defining a property. If added to a data element, a property has to be filled with a value of the predefined datatype. In order to use a property for an edge or a vertex, the property has to be associated in the index fields of the schema.

The last two keys *vertexIndexes* and *edgeIndexes* contain the different indexes. Each index has a name, a composite flag and an associated property key. It is only possible to query the database with the property keys, that are associated with the indexes. This means, if a property key is not contained in an index, it is not possible to query data based on that property. The composite flag indicates if the index is a composite or a mixed index. Composite indexes are faster and simpler, but allow only exact match querying. With mixed indexing it is possible to query the database and find results that don’t match exactly. For example with the command

```
textContains('name', 'New')
```

all elements that have a name that contains the String ‘New’ will be returned. In our project we found out that the matching behind that is token based. So with the example above, names like *New York* or *New Years Eve* will be found, but *Newsreporter* not.

If a data element(node or edge) is added, but the database already contains one element with the same label and name/title, then these elements will be merged: The properties of the new element will be merged by different rules with the properties of the old element, and after that the old element will be altered in the database (no new element is added).

After applying the schema to our database, a node can have the following properties:

- relevance: The relevance score, retrieved from the Alchemy Entity Extraction. When merged, the maximum of the relevance scores will be saved.
- sourceURL: Contains the URL where the node was found. When merged, the URLs will be concatenated.
- sourceTitle: Contains the title of the page where the node was found. When merged, the titles will be concatenated.
- name: Contains the name of the node, as retrieved from the Alchemy Entity Extraction.
- count: The vertex count, retrieved from the Alchemy Entity Extraction. When merged the counts will be summed up.
- dummy: This property is always ‘dummy’ and used for querying all nodes.
- otherVertexLabel: If a label is different to the predefined labels, this property will be set with the found label.
- sentences: Contains the sentence where the node was found. When merged, the sentences will be concatenated.

Furthermore, an edge can have one or more of the properties:

- relationshipScore: Contains the relationship score retrieved from the Alchemy Typed Relations Extraction. When merged, the maximum value will be saved.
- title: Contains the title of the page where the Edge was found. When merged the URLs will be concatenated.
- link: Contains the URL where the edge was found. When merged the URLs will be concatenated.
- otherEdgeLabel: If a label is different to the predefined labels, this property will be set with the found label.
- dummyE: This property is always ‘dummyE’ and used for querying all edges.
- sentence: Contains the sentence where the node was found. When merged, the sentences will be concatenated.
- relationsMode: Either 1 (Alchemy Typed Relations), 2 (Infobox relations), 4 (Alchemy Relations) or 8 (Entity Graph), according to the different sources for the edge (see section 3.2.c).
- alchemyEdgeCount: Contains the number of edges found with the same vertices, labels and titles. When merged, this value will be increased by 1.

The properties *sentence, sentences, link, title, sourceURL* and *sourceTitle* contain a list of the explained elements, separated by a String separator.

```
{
    "propertyKeys": [
        {"name": "alchemyEdgeCount",    "dataType": "Integer"},
        {"name": "relevance",           "dataType": "Float"},
        {"name": "sourceURL",           "dataType": "String"},
        {"name": "sourceTitle",         "dataType": "String"},
        {"name": "name",                "dataType": "String"},
        {"name": "count",               "dataType": "Integer"},
        {"name": "dummy",               "dataType": "String"},
        {"name": "otherVertexLabel",    "dataType": "String"},
        {"name": "sentences",           "dataType": "String"},
        {"name": "relationShipScore",   "dataType": "Float"},
        {"name": "title",               "dataType": "String" },
        {"name": "link",                "dataType": "String" },
        {"name": "otherEdgeLabel",      "dataType": "String"},
        {"name": "dummyE",              "dataType": "String"},
        {"name": "sentence",            "dataType": "String"},
        {"name": "relationsMode",       "dataType": "Integer"}
    ],
    "vertexLabels": [
        {"name": "Person"},
        {"name": "Facility"},
        {"name": "Organization"},
        {"name": "City"},
        {"name": "Location"},
        {"name": "GPE"},
        {"name": "otherV"}
    ],
    "edgeLabels": [
        {"name": "employedBy"},
        {"name": "affectedBy"},
        {"name": "partOfMany"},
        {"name": "locatedAt"},
        {"name": "agentOf"},
        {"name": "otherE"}
    ],
    "vertexIndexes": [
        {"name": "vByRelevance",        "composite": false, "propertyKeys": ["relevance"]},
        {"name": "vBysourceURL",        "composite": false, "propertyKeys": ["sourceURL"]},
        {"name": "vBysourceTitle",      "composite": false, "propertyKeys": ["sourceTitle"]},
        {"name": "vByName",             "composite": false, "propertyKeys": ["name"]},
        {"name": "vByCount",            "composite": false, "propertyKeys": ["count"]},
        {"name": "vByDummy",            "composite": true,  "propertyKeys": ["dummy"]},
        {"name": "vByOtherVLabel",      "composite": true,  "propertyKeys": ["otherVertexLabel"]},
        {"name": "vBySentence",         "composite": true,  "propertyKeys": ["sentences"]}
    ],
    "edgeIndexes": [
        {"name": "eByrelScore",         "composite": false, "propertyKeys": ["relationShipScore"]},
        {"name": "eByTitle",            "composite": false, "propertyKeys": ["title"]},
        {"name": "eByLink",             "composite": true,  "propertyKeys": ["link"]},
        {"name": "eOtherEdgeLabel",     "composite": true,  "propertyKeys": ["otherEdgeLabel"]},
        {"name": "eByDummy",            "composite": true,  "propertyKeys": ["dummyE"]},
        {"name": "eBySentence",         "composite": true,  "propertyKeys": ["sentence"]},
        {"name": "eByRelationsMode",    "composite": true,  "propertyKeys": ["relationsMode"]},
        {"name": "eByAlchemyEdgeCount", "composite": false, "propertyKeys": ["alchemyEdgeCount"]}
    ]
}
```

Schema used for our graph database

3.2.b Gremlin

Gremlin is a graph traversal language provided by IBM graph. With Gremlin simple operations like adding or deleting nodes and edges can be performed. Also, one can traverse over a graph or modify the properties of a data element. In our project we mainly use Gremlin to query data.
An example of a complex query is given in the figure below.

```
def g = graph.traversal();
g.V().has('name', textContains('Robert'))
             .bothE()
             .has('relationsMode', within(0, 1, 2, 4))
             .has('relationShipScore', gt(0.88))
```

Gremlin example query

In the second line this method searches for all nodes that have a property *name* that contains the String "*Robert*". From those nodes we traverse over all neighboring edges: `.bothE()` gets the incoming and outgoing edges of a node. In the next step of the query, those edges will be filtered according to the property 'relationsMode'. At last all edges with a relationShipScore <= 0.88 are dropped. Finally the whole gremlin query returns all edges, that have a relationShipScore > 0.88, one of the relationsModes 0, 1, 2 or 4, and are direct neighbors to nodes that contain the name "*Robert*". It should be noted, that this query only returns edges. If one wants to get the nodes, the term `.bothV()` has to be appended to the query.

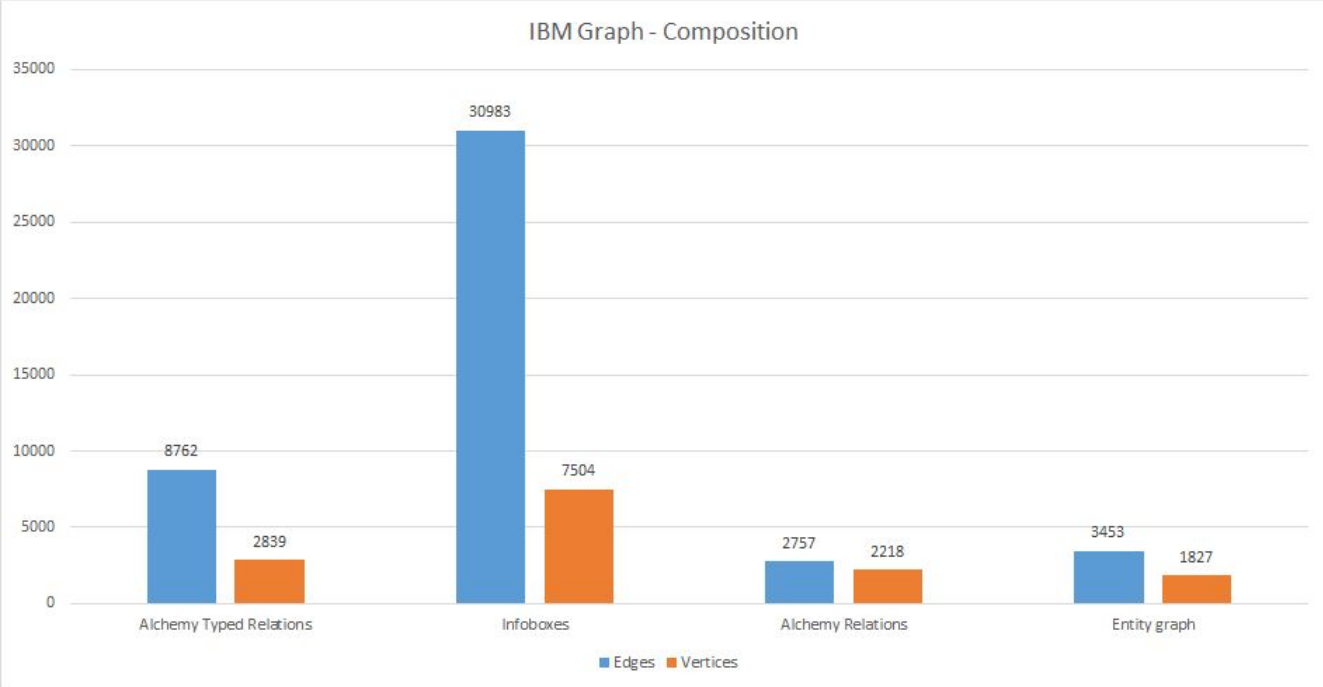
3.2.c Composition

In the last step of our pipeline, the given data by the services will be pushed to the graph database, before the frontend can retrieve them from there.

In our backend we create edges from four different sources, but create the nodes only from the Alchemy Entity Extraction. The different source for the edges can be labeled as four different graphs, although the graph contains paths that go through multiple graph sources. The four graphs are named:

- Alchemy Typed Relations: For this graph the relations retrieved from the Alchemy Typed Relations Extraction are used. The relations are filtered such that the nodes of each relation has to be in the already extracted entities from the Alchemy Entity Extraction.
- Infoboxes: The relations of the structured infoboxes are simply parsed row by row of each infobox. Only relations where the nodes match with entities of the Alchemy Entity Extraction are considered. Because of this straight forward approach this graph type contains the most nodes and edges.
- Alchemy Relations: This graph works almost exactly like the Alchemy Typed Relations, except that the Alchemy Relations Service is used for generating edges.
- Entity graph: When there are two entities in one file (from the Alchemy Entity Extraction), then an edge between these entities is generated and saved along with the two nodes. Because there can be too many nodes, we only consider the 50 entities that occur most in one file.

The whole graph database composition is displayed in the following figure. It is to be noted that one node can have multiple edges from different sources. The orange bars only count the number of nodes which are used by the created edges.



Composition of the graph database of our project

4. Pipeline

In this chapter our pipeline will be presented. The descriptions will show how we get from our raw data to the finished graph in the graph database (back-end) and how the Web-UI (front-end) can interact with the database. The functionalities will be explained based on our so called Simple Pipeline, that generates a graph only for one file. After that, the “Big” Pipeline, which handles all the data, will be introduced.

If you want to run the project on your own system, you need to place the following credential files in the according folders ($HOME is the root folder of the git project, after cloning):

Filename	Folder	Description
graphDBCredentials	$HOME/InfNet/nbproject/private	IBM Graph (backend)
graphDBCredentials	$HOME/front-end/src/main/resources	IBM Graph (frontend)
smalltestGraphDBCredentials	$HOME/InfNet/nbproject/private	IBM Graph - Simple Pipeline (backend)
smalltestGraphDBCredentials	$HOME/front-end/src/main/resources	IBM Graph - Simple Pipeline (frontend)
alchemyAPICredentials.json	$HOME/InfNet	Alchemy (backend)

Credential files and where to place them.

4.1 Simple Pipeline - on the fly evaluation of a file

The Simple Pipeline was developed with the purpose to show what the graph looks like at the current development status. Our whole pipeline can be traversed with only looking at one single file.

The pipeline can be executed by simply running the main method of the class ibm.pipeline.SimplePipeline in the InfNet-project (backend). To select which html file should be analyzed, the field FILENAME has to be set to the path of the html file. After that, the pipeline will run through all the steps:

1. **HTML → Data:** In the first step of the pipeline the data will be extracted from the HTML files (for details see chapter 2).
2. **Data → Services:** The data extracted from the html files is now put into the described services: Alchemy Entity Extraction, Alchemy Relations Extraction and Alchemy Typed Relations Extraction (see section 3.1).
3. **Services → Graph database:** With the services data it is now possible to generate the relations which are pushed to the database. As mentioned before, there are four graphs to be distinguished, which are described in section 3.2.c.
4. **Graph database → Web UI:** When the graph is pushed into the database, the Web UI (frontend) can query the graph and display parts of it.

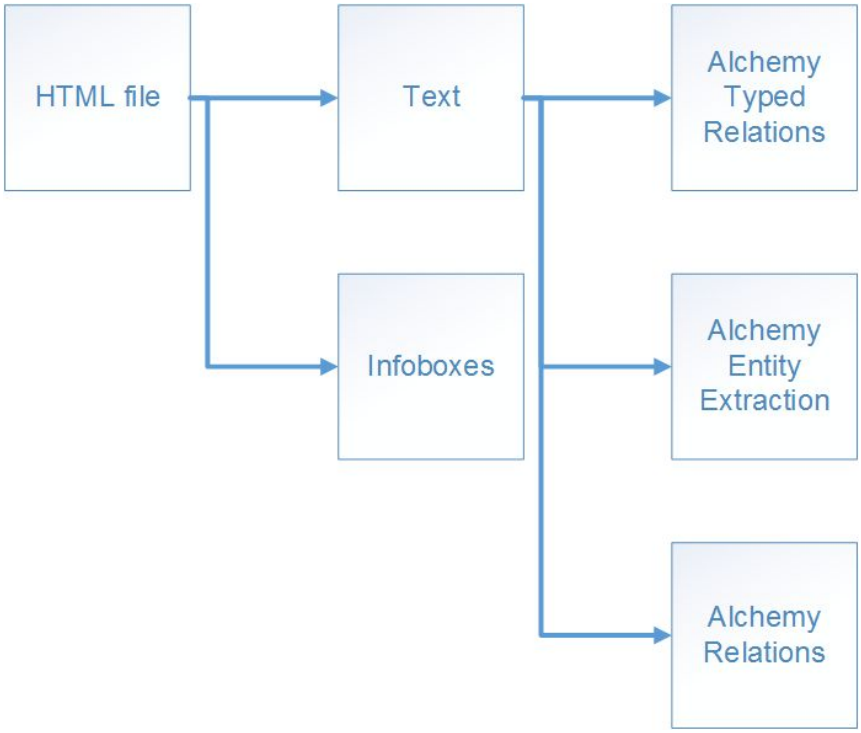
The graph will be saved in an own instance of IBM Graph, which has the same schema as described in section 3.2.a. The Web UI has a own servlet called *Simple Pipeline* which displays all nodes and edges in this separate graph database. In order to display the right graph, the field *GRAPH_ID* in the class servlets.SmallestTestServlet has to be modified. When executing the SimplePipeline from the backend a new generated graph id will be outputted on the console. This graph id has to be set in the field GRAPH_ID of the SmallestTestServlet.

4.1 “Big” Pipeline - for all extracted Informations

The “Big” Pipeline in our project is theoretically the same process as the Simple Pipeline, except that it will be executed for all given HTML files. We quickly encountered the problem that the Alchemy service has a per day limit of 1000 API calls and that services can call each other, limiting the API calls even further. So it is impossible to just click and execute our big pipeline and have a big graph at the end in our database. This alone would take several hours or even days. Instead we process the pipeline step by step and save the results to json files (to be processed by the next step).

At the start of our pipeline each HTML file has to be in the same folder. The first step extracts data (primarily texts and infoboxes) from each HTML file and saves the result in a separate json file in a specified folder.

In the next steps, each of the used Alchemy services is used to generate the described service data. For each of the service one file will be saved with the results of this service. The following figure visualizes these steps of our pipeline. Each square displayed in the figure represents one file which contains the respective results: For each input HTML file, five more json files are saved, each in a different folder to better distinguish them.



Created files and file dependencies of the "Big" Pipeline

The next step of the pipeline is to push data to the graph database. For each input file, we now can get the service results from the other result files. So we can generate the relations from each file and upload them. For the different graphs there are the following main classes available in the source code, which did exactly this process:

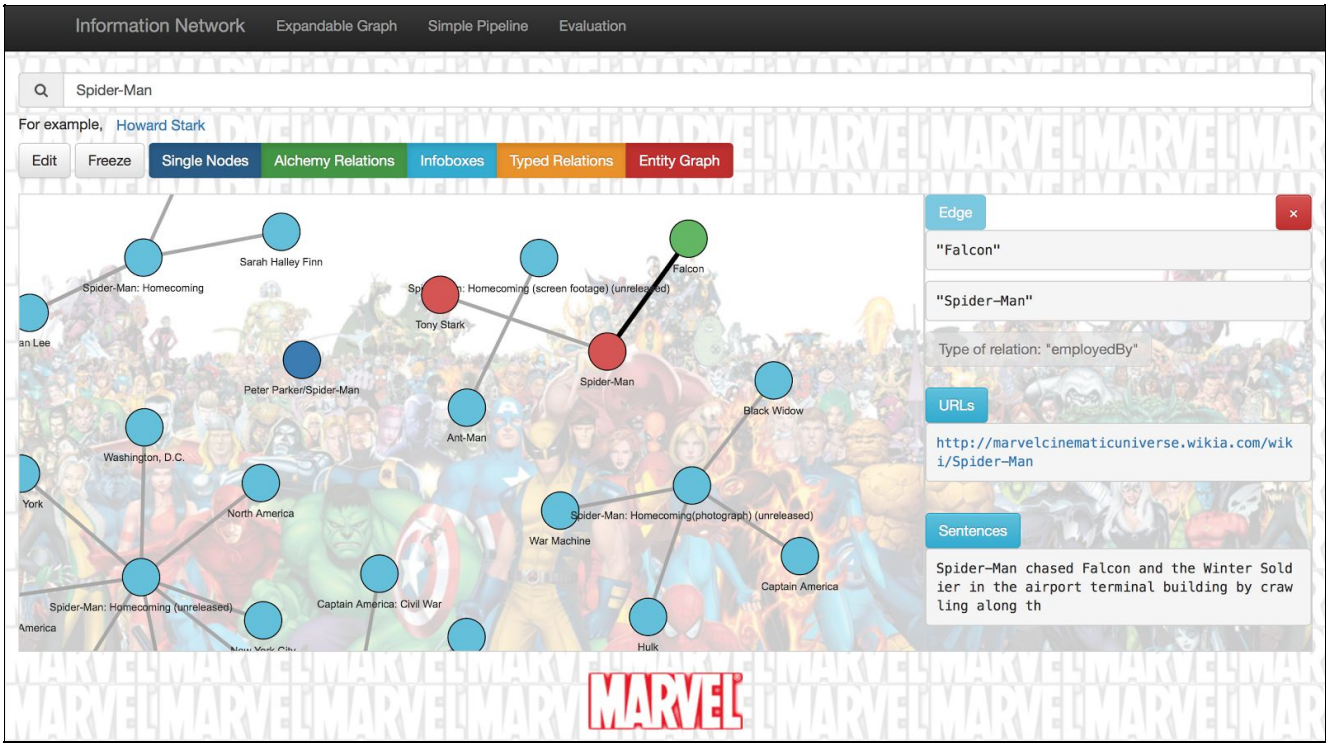
- Alchemy Typed Relations: *FillGraphDB*
- Infoboxes: *FillGraphDBWithInfoboxRelations*
- Alchemy Relations: *FillGraphDBWithSOARelations*
- Entity graph: *FillGraphDBWithEntityGraph*

Finally, the graph can be queried from inside the frontend, which will be described in the next chapter.

5. Web application

In this part we describe the web application we developed, allowing the user to visualize and interact with our Information Network. It is divided in three different parts:

- the so-called Expandable Graph: this is the main result of our work, enabling the exploration of the full Information Network,
- the Simple Pipeline: we decided to add this second graph page to make it possible to parse directly a file and extract the latent Information Network on the fly,
- The evaluation page has been used to facilitate the evaluation for the different services that we used.



View of the Expandable Graph after searching for "Spider-Man" and clicking on the edge "Spider-Man" - "Falcon"

5.1 Expandable graph

As cited above, the expandable graph is the main outcome of our work, this page enables the visualization of the Information Network for the Marvel universe. At the beginning there is no graph displayed: to start the user either searches for a node label via the search bar on top of the page or uses one of the preconfigured examples.
This acts as a seed for the network exploration: the nodes(s) and its direct neighbours are displayed. This network can then be expanded: when double-clicking on a node, the neighbours of this node are added to the current network. Doing this, the user can progressively explore the whole Marvel universe.

Double-clicking on a node displays its neighbourhood. Additionally a single click opens a box displaying information to the user. Nodes and edges can be clicked. The displayed information contains the sources for the selected node/edge: the urls to the articles of the Wiki as well as sentences directly extracted from the articles. This way, the user is able to get more details about a relationship or a node and understand why these elements are present in the Information Network.

The network is displayed using vis.js[1], a visualization library. Vis involves a gravity model for the nodes, moving them in order to display them as nicely as possible. Nevertheless, in some cases the simulation doesn't stabilize after a few seconds and the nodes keep moving for a long time. To enable the user to interact with the network in such circumstances, a "freeze" button was implemented. As its name says, it allows to stop the node's movement. This can be reactivated by clicking the same button again ("unfreeze").

Another feature that we added is the possibility to edit the network content. Although we think the present Information Network is globally of good quality, it surely contains erroneous elements: nodes, relationships, labels etc. If a user notices such an error, she can correct it directly. To do this, we

[1] http://visjs.org

implemented a series of edit features, that are accessible via the button “edit”. The user can either add, modify or delete a node/edge.

As explained in the previous section, we used several services to obtain our Information Network. We decided to keep all the the information we gathered available in our web application. However, we added the possibility to filter the network to display only nodes coming from specific services. The different nodes colors show from which service a graph element originates. The user can then switch on/off each service to update the network.

5.2 Simple Pipeline

The second graph we have included in our web application represents the Information Network that is extracted from one file on the fly. The methodology is detailed in Section 4.1. The features available in this visualization are globally the same as for the Expandable Graph, only that the graph cannot be expanded. Instead, we display the full network directly. We decided to do so because the network from only one file tends to be quite small: using an expandable graph in this case often results in a game in which the user tries to double-click on each node to find the ones that can be expanded - most of them don’t have other neighbours.

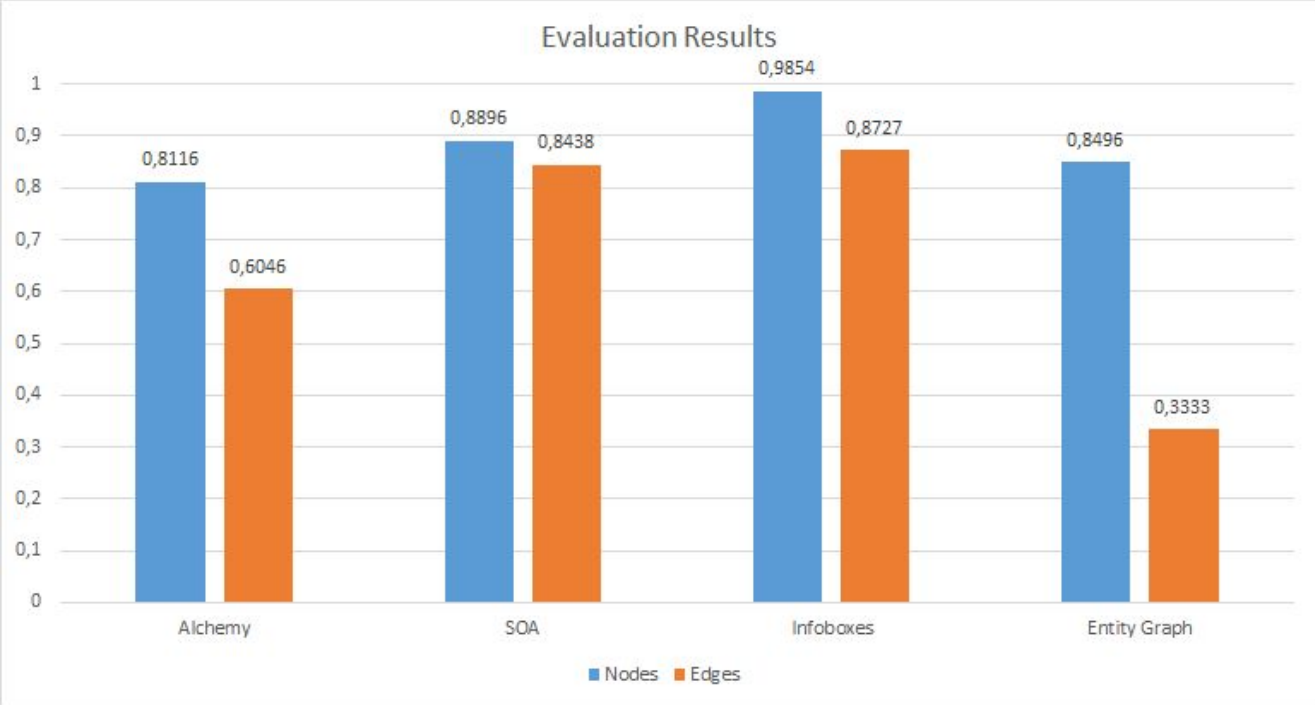
5.3 Evaluation

The last part of the web application is used for evaluation purposes. The user is proposed a series of nodes and edges and has to decide whether these elements make sense to her or not. The process is the following: an edge is selected randomly among all edges from a given service, first, the two nodes are proposed for evaluation to the user. If the user decides that both nodes are correct, she gets a new question about the edge between these two nodes. Indeed, evaluating an edge between nodes that are not correct in the first place does not make sense, that is why we evaluate first the nodes individually. More details about our methodology for the evaluation are presented in the next section.

6 Evaluation

The evaluation focuses on assigning a quality value to the extracted nodes and edges. A quantitative evaluation of the graph could be the following process. A human reads through the HTML input files and writes down all the entities and their respective relationships that she finds. Then, the entities and their relationships are being compared with the nodes and edges in the information graph. Entities and relationships that have no corresponding representation in the graph are considered a missing error. Also, nodes and edges that are unreasonable given the HTML sites and have no counterpart in entities or relationships are considered a mistake. These two kinds of errors can be treated equally or weighted according to which one is considered more important for the evaluation. Ideally, this process is done repeatedly by different humans. Then, either their results are averaged or the knowledge base of entities and relationships is created via majority voting.
While this process leads to a quantitative evaluation of the resulting graph given the web site text inputs it is very time consuming, tedious to do and beyond the scope of this project. Instead, we

chose to do a qualitative evaluation of the information graph. The qualitative evaluation process is as follows. An edge is selected randomly. Then, the user is presented with one of the two corresponding nodes of that edge. She then has to decide whether that node is sensible or not. After the first node the second node is presented, again with a request to decide its right to exist. Finally, the edge between the two nodes is presented and the last demand to decide its value. If for one of the two nodes the user's response is negative not only the node itself is dismissed but also the edge. Note, that this treatment is highly subjective and out of context. A user has to be familiar with the Marvel Universe to be able to assess validity of a graph item. In comparison with the quantitative evaluation process the qualitative evaluation does not distinguish between error types and therefore is less informative regarding possible entry points for improvement.
For this project we realized the qualitative evaluation process as described above. Therefore, 150 Edges and their 300 respective nodes were selected at random and judged. This was carried out for each of the four sources Alchemy, SOA, Infoboxes and Entity Graph and repeated four times. The aggregated results of this evaluation can be found in the figure below.



In the figure above the four different sources are listed on the x-axes and the percentages of positively answered questions is on the y-axes. As can be seen all services produce a performance of at least 80% of correct nodes. Infoboxes having the highest performance with over 98%. Also for the edges Infoboxes perform best. This result is somewhat expected as the Infoboxes are structured content that is supposed to capture some kind of semantic relationship between entities. The SOA service comes second in both nodes and edges. This result is explainable with the structured type of relationship which is subject, object and action. The Entity graph ranks last especially for the quality of the edges. Again, this result is expected due to the creation process of the entity graph. Edges are simply created based on concurrence in the input documents without a semantic consideration.
Even without quantitative results the qualitative evaluation gives an performance overview of the different services used. This evaluation can be used as a guideline to decide which service is worth further development.

7. Discussion

This chapter discusses the positive and negative aspects of the project. It especially covers features and their possible improvements.

First, the entire pipeline can process any given textual input in english. It is possible to create any graph from any source. The only thing that may need to be modified is the parser of the HTML-files, so that all the text and infoboxes can be processed correctly. In addition to that, it might take several days to complete the graph, even if there were no limits in API calls, due to the size of the input data.
Another plus point of our application, is that the expandable graph allows the user to explore the whole (Marvel) universe by him- or herself. Instead of being overwhelmed with too much information at once, the user can expand only the nodes that are interesting for him/her.
Another nice feature is, that the user can compare how different services work from IBM by only displaying nodes from selected sources.
Also, the user can use the automatically generated graph to generate a ground truth with the help of the edit function.

Regarding the disadvantages of our project, at this moment, when expanding a node, there are too many neighboring nodes displayed. An improvement would be to add some sort of filtering technique, which will be discussed in the next chapter.

8. Future Work

In order to achieve better end results, we were able to optimize every step of our pipeline.
First, we suggest to improve the parser, because the parser can be used for all wikia's. Here you can start with e.g better processing of infoboxes, focus on their special features and use a better separator etc.. In addition, most articles cover a separate season. During parsing these sections can be marked and later identified to enable filtering according to seasons. This prevents the user from being spoilered.

As a second step we would suggest creating a domain dependent model with the IBM Knowledge Studio for the marvel universe or for wikia articles, so you can use it for other wikia's too.
Next we would suggest to spend more work on the string matching, which is a larger current research topic. The string matching is used in several places: When filtering the responses of the services, matching the nodes when they are inserted into the graph, or if you use the search field in the WebUI. We have not treated ambiguities in the scope of this project, this direction could be explored, since there are often several nodes which reflect a same entity e.g. Spider-Man.

As one of the most important steps, we see the choice of the right scoring and counting parameters. We have got a feeling for it, but it would have to be evaluated again to see how the quality of the results varies under different count values based on the file size and the number of found entities. A first step would be to implement the appropriate gremlin queries to filter the data out of the WebUI, to get a feeling for the different values. The functionality that allows this is already ready.
As a last point we would call the extension of the WebUI e.g. allow filters for the scores and seasonal graphs as previously mentioned. Do not select the simple pipeline file, graph credentials etc. hardcoded any longer, use instead a user interaction dialog for choosing the file. Choose via gui which service/source should be evaluated and automatically choose the corresponding nodes and edges to compute the correctness value.

9. Who did what

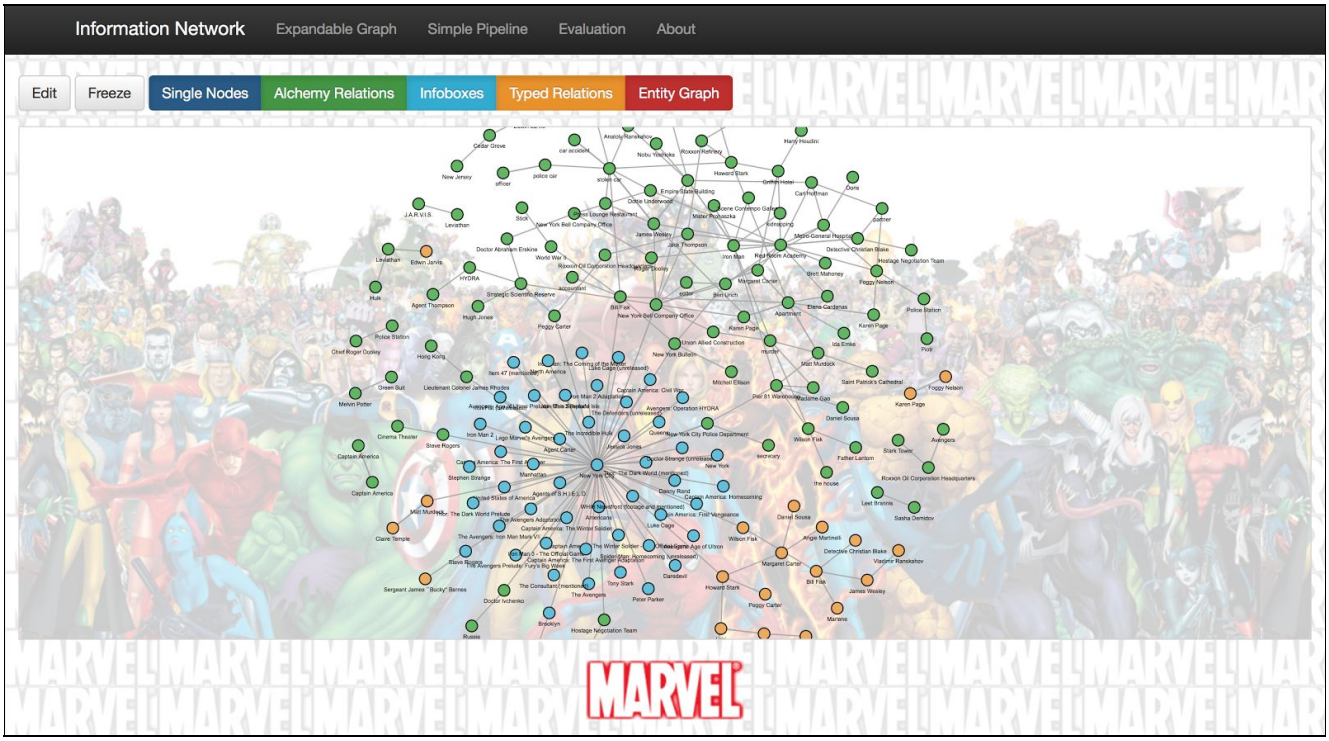
Simon S.: Backend focus on the GraphDB, filling the GraphDB with the Pipeline, Gremlin Language and interface to the front-end, Documentation: Chapter 3.2-4,7

Christoph: Backend with focus on the data, data processing, using of the different services and handling and parsing of the responses, Documentation: Chapter 1-3.1,8

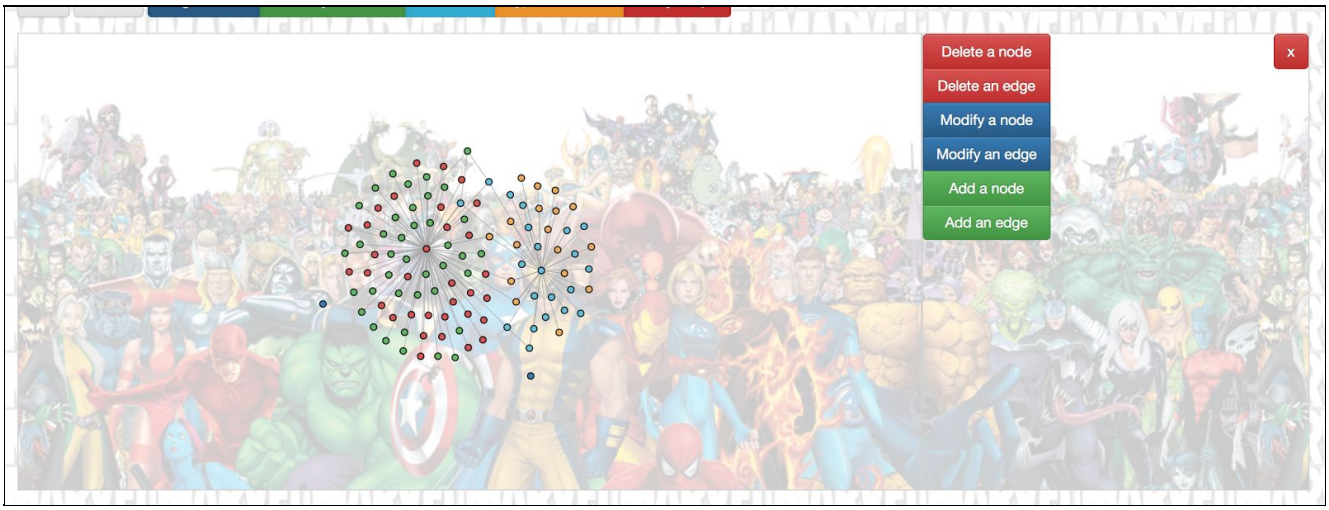
Simon D.: Web interface with its several features, interface with gremlin queries for IBM graph, Documentation chapter 5 + correcting other chapters

Kai: Evaluation, css styling and layout, documentation chapter 6 + correcting other chapters

APPENDIX



View of the Information Network obtained with the Simple Pipeline for the article New York in the Wikia



View of the Edit "box"

- Delete a node: click on "Delete a node" and then click on the node to delete in the network
- Delete an edge: click on "Delete an edge" and then click on the edge to delete in the network
- Modify a node: click on "Modify a node", select the node to modify in the network and fill in the new labels for this node
- Modify an edge: click on "Modify an edge", select the edge to modify in the network and fill in the new labels for this edge
- Add a node: click on "Add a node" and fill in the labels for this node
- Add an edge: click on "Add an edge" and select the two nodes to connect in the network