Fall | 2017

Technical Report

Project: Knowledge Graph of SOSE Research Community

Team-5
Imre: +1 650-495-6101, imre.nagi@sv.cmu.edu
Sheersha: +1 650-282-0101, sheersha.kandwal@sv.cmu.edu
Shreyansi: +1 925-408-8103, shreyank@andrew.cmu.edu
Siva: +1 408-597-2127, skuntamu@andrew.cmu.edu

Instructor: Dr. Jia Zhang
Enclosed in this document is the technical report for 18656 Data Intensive Workflow for Software Engineer

Table of Contents

Introduction

a. **Background**: The innovations of SOC also offer many interesting avenues of research for scientific and industrial communities. Recent advances in service oriented computing and cloud computing, including computational power, storage, and networking, and infrastructure innovations, are providing exciting opportunities to make significant progress in understanding and addressing complex real-world challenges. In this project, we aimed to design and develop a software system focusing on collecting, aggregating, classifying, analyzing, reporting, and visualizing data related to Service Oriented Software Engineering (SOSE) research community.

b. **Architecture:**

- Neo4j: It is a graph database management system developed by Neo4j, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing. Considering the vastness of the relationship amongst the various type of data we had, we decided to use graphs for visual presentation. It ensures scalability,performance and is also easy to interpret the relationship using basic cypher queries.
- Play: Play is targeted as a lightweight, stateless, web friendly architecture. Built on Akka, Play provides predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications. Since it makes it easy to build web applications with Java and Scala, we used it to connect the front-end framework and back-end services.
- AngularJS: AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop. Due to the flexibility and the extensibility of the framework we used it for the frontend for developing cross-platform mobile apps.
- Visualization techniques: We utilize D3 and GoogleMap API to help us visualize data. D3 emphasises on web standards to give us the full capabilities to combine powerful visualization components and a data-driven approach to DOM manipulation.

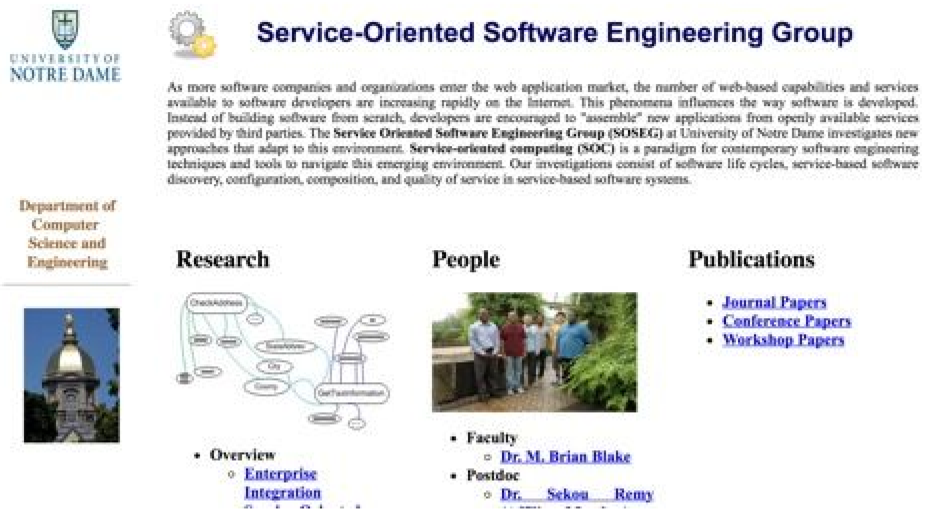
c. **Other tools:** We used Github for version control,so each of the team members are able to work independently on their branches to develop new features. We complied with the conventional Software Engineering Methodology. We used trello a Kanban-style project management tool that is a visual way to see all the tasks that you or your team need to accomplish, which makes it easier to see what everyone on the team is doing at a given time.It helped us in organising the

d. **Functionality**: Our platform currently supports the following functionalities.

- Paper related: We generate a paper-paper network to show relationships among all papers. Each paper has its publication information, similar topics can link them together. Papers are categorized by topics, geolocation, and publication channel.
- Author related: We generated a person-person network, researchers have coauthor relationship, and they are associated to each other in similar research area. We demonstrate that relationships among these researchers conforms to Six degrees of separation theory.
- Other: We also provide simple recommendation functionality for search query. Each time, a user search for something, we provide similar recommendations based on search history of others. We also provide follower-followee functionality for better community support.

Motivation

Dblp provides information on major computer science journals and proceedings, but the information is scattered and has ambiguous relationships. We looked into if there were similar other projects to bring all the information and relationships together. We found one by *University of Notre Dame,* at http://www3.nd.edu/~soseg/soseg/ .



Service-Oriented Software Engineering Group

As more software companies and organizations enter the web application market, the number of web-based capabilities and services available to software developers are increasing rapidly on the Internet. This phenomena influences the way software is developed. Instead of building software from scratch, developers are encouraged to "assemble" new applications from openly available services provided by third parties. The **Service Oriented Software Engineering Group (SOSEG)** at University of Notre Dame investigates new approaches that adapt to this environment. **Service-oriented computing (SOC)** is a paradigm for contemporary software engineering techniques and tools to navigate this emerging environment. Our investigations consist of software life cycles, service-based software discovery, configuration, composition, and quality of service in service-based software systems.

Research

- Overview
  - Enterprise Integration

People

- Faculty
  - Dr. M. Brian Blake
- Postdoc
  - Dr. Sekou Remy

Publications

- Journal Papers
- Conference Papers
- Workshop Papers

However, we could identify potential problems there including :

i. Outdated dataset
ii. Implicit relationship
iii. Ambiguous Classification
iv. Lack of interactivity among users

Hence, with our SOSE research community, we worked through previous issues and established accomplishing the following **goals** -

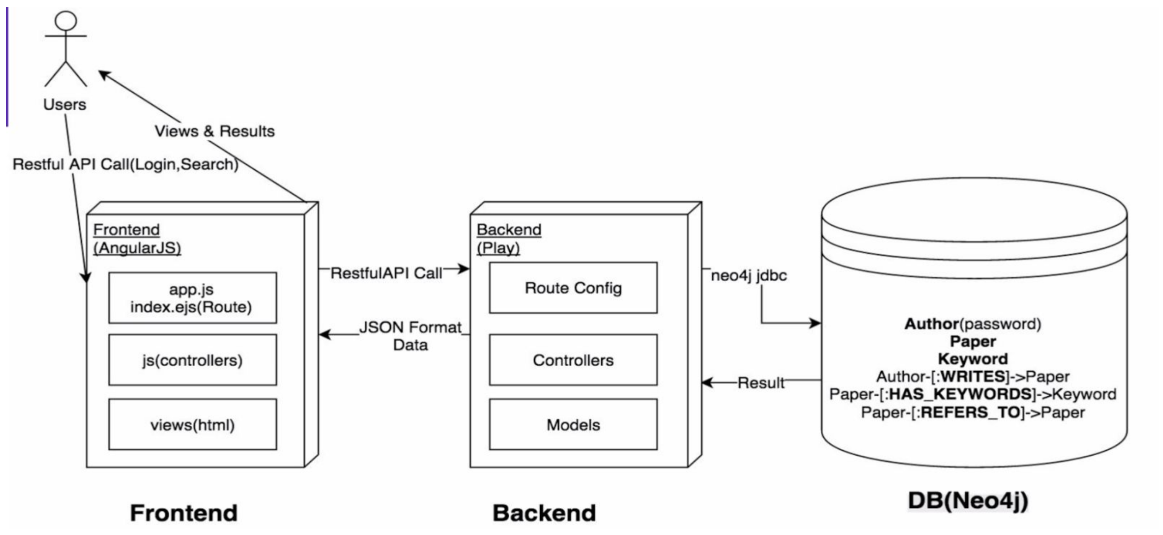
i. Create one stop source for information about authors and their work.
ii. Connect Subject Experts to fellow Authors.
iii. Stay updated about papers published by researchers.
iv. Increase collaborative research amongst Authors.

Related work

We utilised the existing knowledge graph in the technical research community as the sources of our information -
Google Scholars
Research Gate

We noticed, that while these websites have good organisation of information on the authors, they do not have any features to show the graph network of paper/authors collaborations. We also crawled the web to get information on Authors/Person if the data for them was missing.

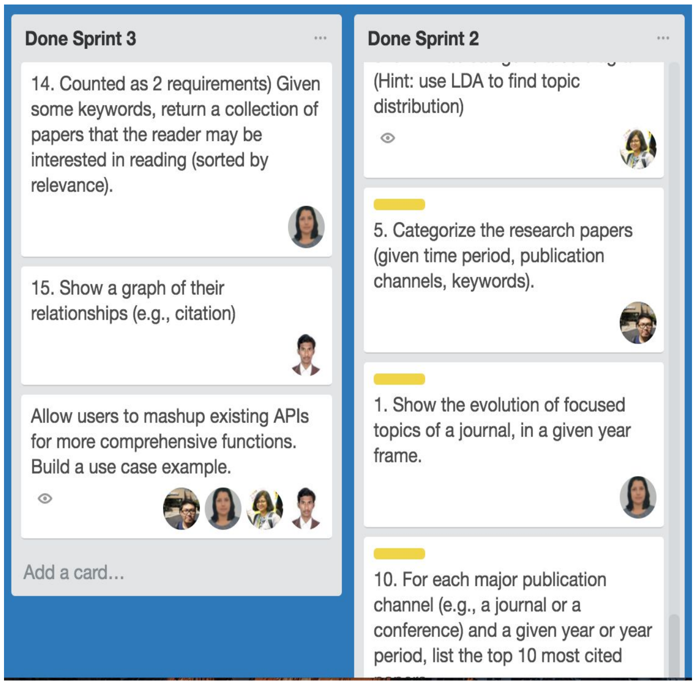
System design



- About our system design and architecture, we chose a service oriented architecture (SOA).
- In our server side (service provider), we design some APIs to complete the needed service.
- For client side (service user), each business process calls several corresponding APIs to provide a complete use case for users.
- With this kind of SOA, the interaction between server side and client are all about web services.

Project Development Routine

- Instead of just focusing on the product, we pay more attention to the process.
- We start with each iteration with estimation and design. We used Trello board to track our work.



Done Sprint 3

14. Counted as 2 requirements) Given some keywords, return a collection of papers that the reader may be interested in reading (sorted by relevance).

15. Show a graph of their relationships (e.g., citation)

Allow users to mashup existing APIs for more comprehensive functions. Build a use case example.

Add a card...

Done Sprint 2

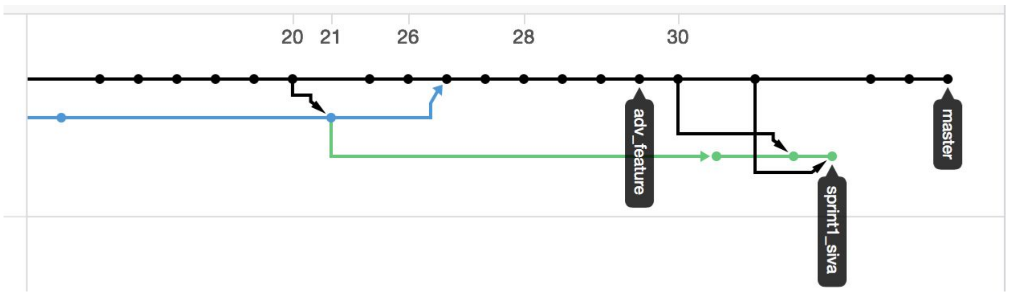
(Hint: use LDA to find topic distribution)

5. Categorize the research papers (given time period, publication channels, keywords).

1. Show the evolution of focused topics of a journal, in a given year frame.

10. For each major publication channel (e.g., a journal or a conference) and a given year or year period, list the top 10 most cited

- We split requirements into tasks and pick tasks under a work in process limit and start our development. We used Github for version control and code reviews.



- We are implementing many SWE practices mentioned in class. What really works well is we always do refactoring before implementing new features.
- We learn from each other and improve our software quality by pair programming. And we do reflection to get feedback from previous work and try to improve.
- And then we do a comprehensive retrospective to discuss the technical debt, improve the workflow.

Software Framework & Libraries

1. Front Tier: HTML, CSS, Angular JS
2. Back Tier: Play Framework, Python
3. Database: Neo4j

Front End

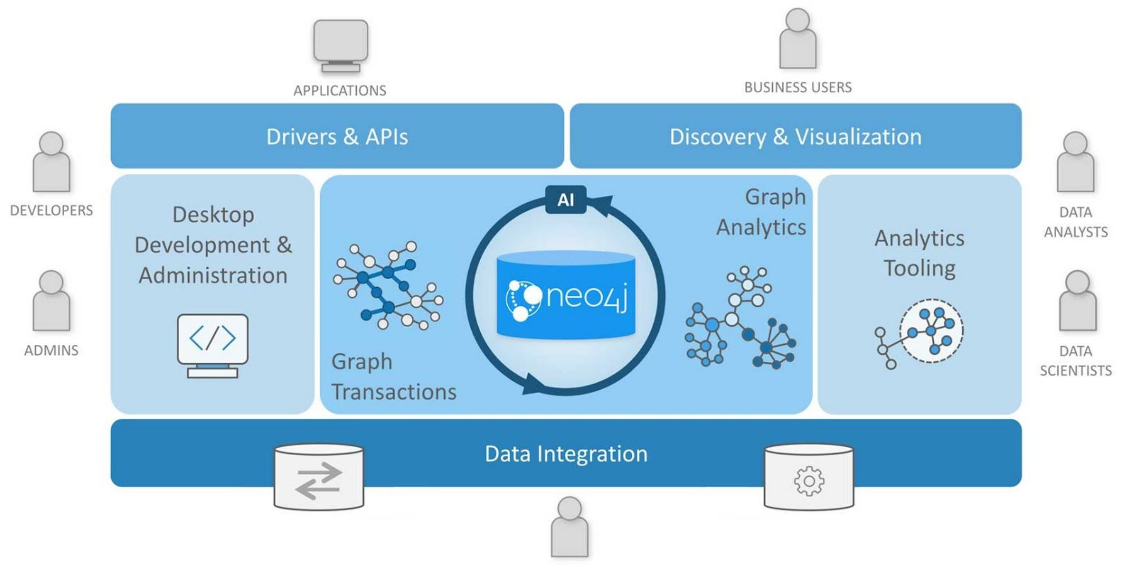
- Twitter Bootstrap [http://getbootstrap.com/] - Using Twitter Bootstrap, we can easily create functional and standard websites that the average consumer can easily navigate.
- AngularJS [https://angular.io] - Using AngularJS, we can easily extend the HTML, building a responsive and fast website.

Back End

- Python [https://www.python.org/] - Useful for data crawling and loading data to the Neo4j database.
- Play[https://www.playframework.com/] - Play is based on a lightweight, stateless, web-friendly architecture. Built on Akka, Play provides predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications.
- Tableau[https://www.tableau.com] - Tableau produces interactive data visualization products focused on business intelligence.

Database

- Neo4j[https://neo4j.com] - The graph platform takes a connections-first approach to data.



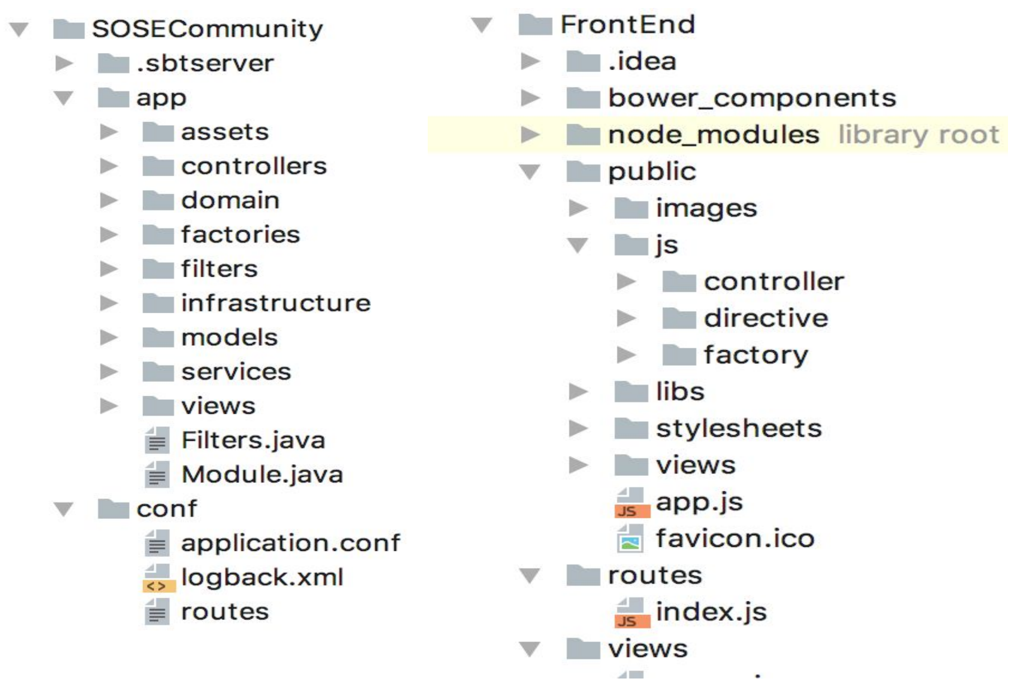
Code Structure

The most important and basic step in our implementation was separate front and backend Play servers.

The code was structured into a backend Play with REST API, with a frontend in Angularjs that called that API.

Our implementation consists of a backend server which serves a similar purpose to that of the legacy code.

We maintained the Angular controllers on those pages to control the behavior, to call the backend API and maintain the page models, but used local storage as opposed to the Angular rootscope to maintain session information.



```
▼ SOSECommunity
  ▶ .sbtserver
  ▼ app
    ▶ assets
    ▶ controllers
    ▶ domain
    ▶ factories
    ▶ filters
    ▶ infrastructure
    ▶ models
    ▶ services
    ▶ views
      Filters.java
      Module.java
  ▼ conf
      application.conf
      logback.xml
      routes
```

```
▼ FrontEnd
  ▶ .idea
  ▶ bower_components
  ▶ node_modules  library root
  ▼ public
    ▶ images
    ▼ js
      ▶ controller
      ▶ directive
      ▶ factory
    ▶ libs
    ▶ stylesheets
    ▶ views
      app.js
      favicon.ico
  ▼ routes
      index.js
  ▼ views
```

System Implementation

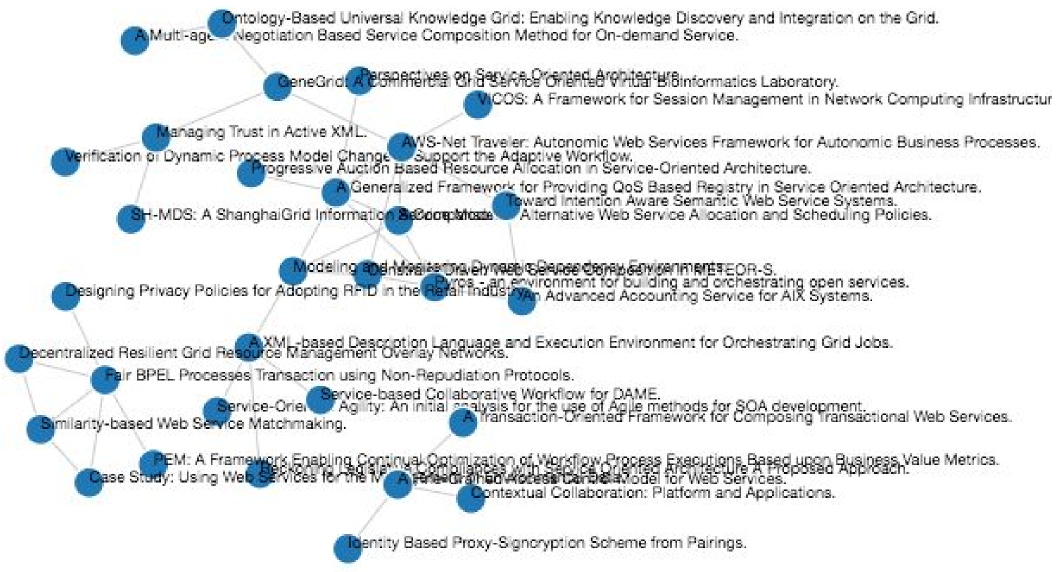
Basic Requirements

We built on top of the legacy codebase. For the basic features requirement, we added/modified/fixed the following basic features. Below are few of the key basic features we worked on.

1. *Generate Paper-Paper Network*

a. Implementation - For the Paper- Paper network, we were missing the paper citation data which relates a paper to another paper. To generate the below network we used a python parser to crawl the DBLP citation dataset to get a paper data csv. We then saved the csv data in neo4j. In addition to the csv, we also saved the data relationship i.e. in this case the citation relationship between the papers to Neo4j .

b. Visualization - We used angular-nvd3(third party library which embed the d3 into angular environment) for visualizing the below network.

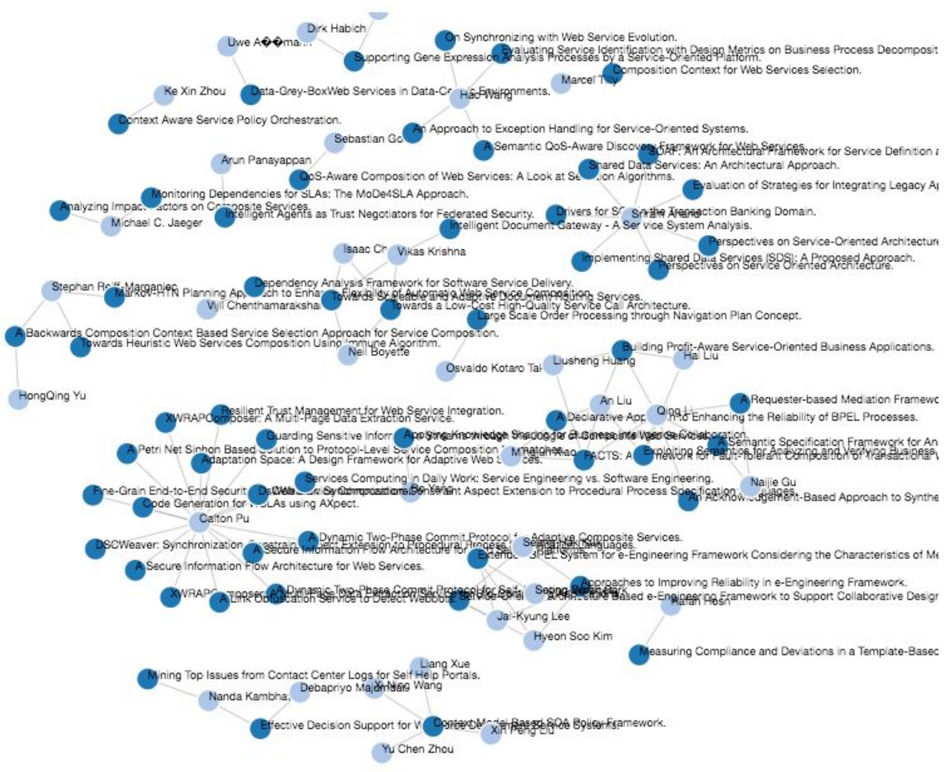


Ontology-Based Universal Knowledge Grid: Enabling Knowledge Discovery and Integration on the Grid.
Negotiation Based Service Composition Method for On-demand Service.
GeneGrid: A Commercial Grid Service Oriented Virtual Bioinformatics Laboratory.
Perspectives on Service Oriented Architecture.
VICOS: A Framework for Session Management in Network Computing Infrastructur
Managing Trust in Active XML.
AWS-Net Traveler: Autonomic Web Services Framework for Autonomic Business Processes.
Support the Adaptive Workflow.
Progressive Auction Based Resource Allocation in Service-Oriented Architecture.
A Generalized Framework for Providing QoS Based Registry in Service Oriented Architecture.
Toward Intention Aware Semantic Web Service Systems.
Alternative Web Service Allocation and Scheduling Policies.
SH-MDS: A ShanghaiGrid Information
Designing Privacy Policies for Adopting RFID in the Retail Industry.
Pyros - an environment for building and orchestrating open services.
An Advanced Accounting Service for AIX Systems.
Decentralized Resilient Grid Resource Management Overlay Networks.
A XML-based Description Language and Execution Environment for Orchestrating Grid Jobs.
Fair BPEL Processes Transaction using Non-Repudiation Protocols.
Service-based Collaborative Workflow for DAME.
Agility: An initial analysis for the use of Agile methods for SOA development.
A Transaction-Oriented Framework for Composing Transactional Web Services.
Similarity-based Web Service Matchmaking.
PEM: A Framework Enabling Continual Optimization of Workflow Process Executions Based upon Business Value Metrics.
Case Study: Using Web Services for the
Contextual Collaboration: Platform and Applications.
Identity Based Proxy-Signcryption Scheme from Pairings.

2. *Generate Paper-Person Network*

a. Implementation - We generate paper-paerson network by using information about the authors of a paper. a

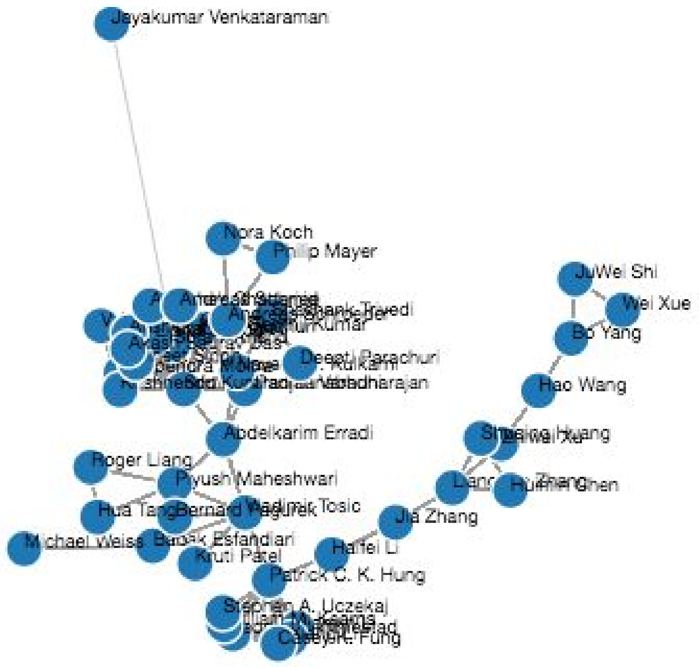
b. Visualization - We used angular-nvd3(third party library which embed the d3 into angular environment) for visualizing the below network.



3. Generate Author-Author Network

a. Implementation - To get authors relationships, we use the coauthor information of a paper. Each author who writes a paper will have coauthor relationship with the other authors for that paper. By utilizing this relationship, we can generate the author-author relationship.

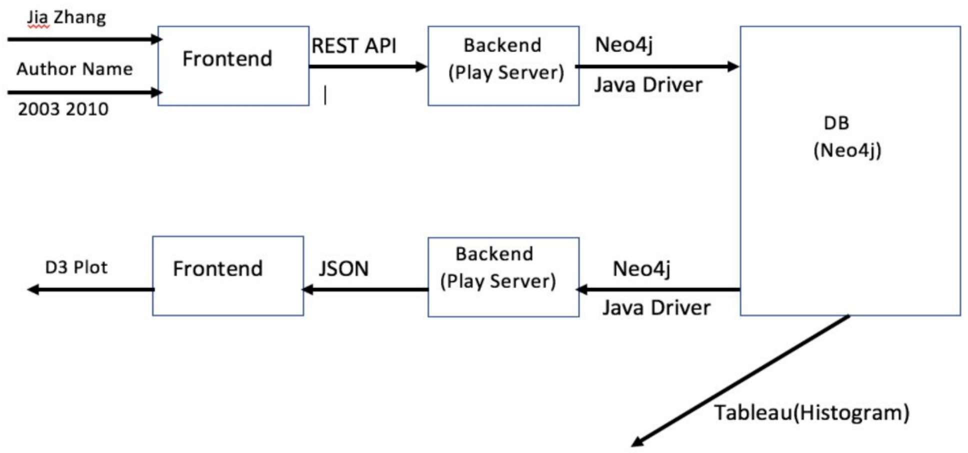
b. Visualization - We used angular-nvd3(third party library which embed the d3 into angular environment) for visualizing the below network.

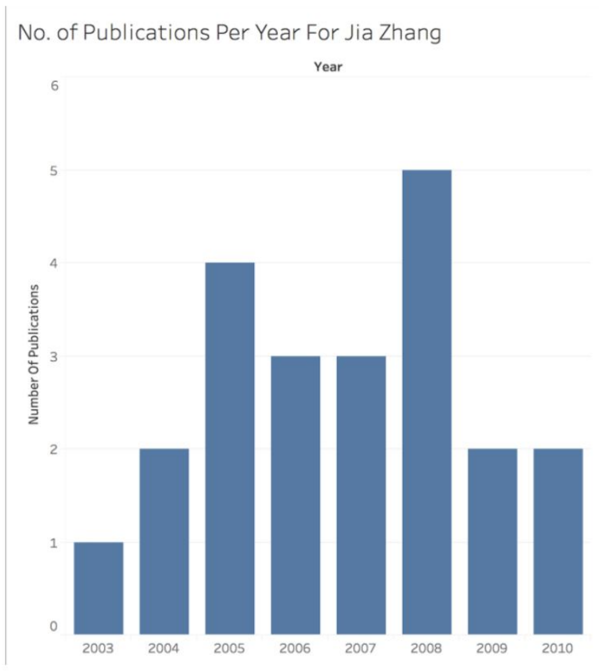


4. Given a time period of years for an author - generate a histogram for publications per year

a. Implementation - For the number of yearly publication for a given author in the specified time frame requirement, we crawled the dblp db to get the author and publication information as a csv. We then saved the csv data in neo4j. In addition to the csv, we also saved the data relationship i.e. in this case the author writes paper relationship to Neo4j .

b. Visualization - We used angular-nvd3 on the front end(third party library which embed the d3 into angular environment) for visualizing the yearly publication distribution. We also used to tableau to generate a histogram for visualizing yearly publication distribution.





Advance Requirements

For the advance features requirement, we implemented the features in Vistrails workflow. Below is the snapshot of some of the key features which we implemented.

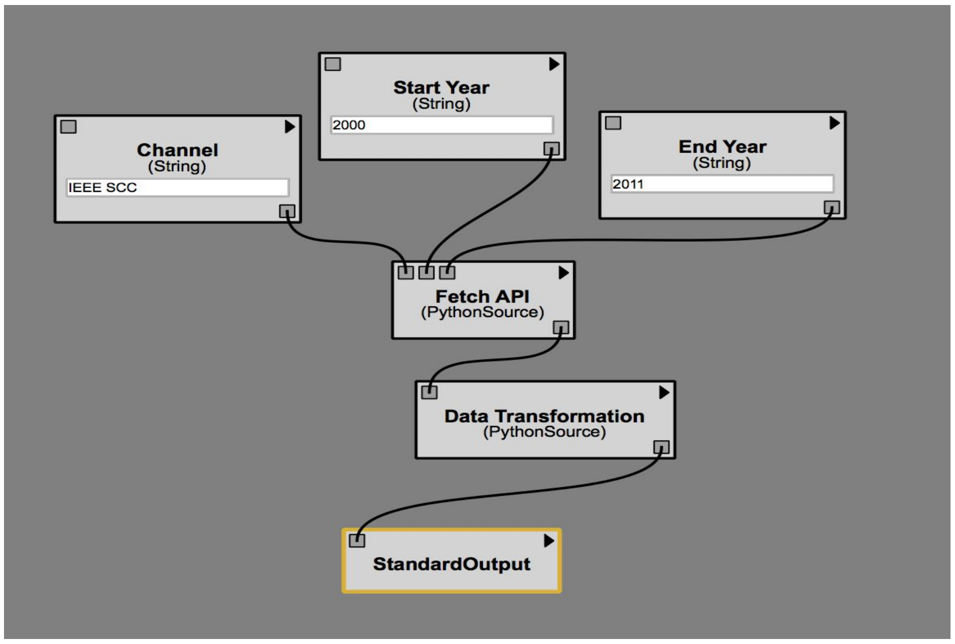
1. ***Evolution of topics of a journal - in a given time frame***

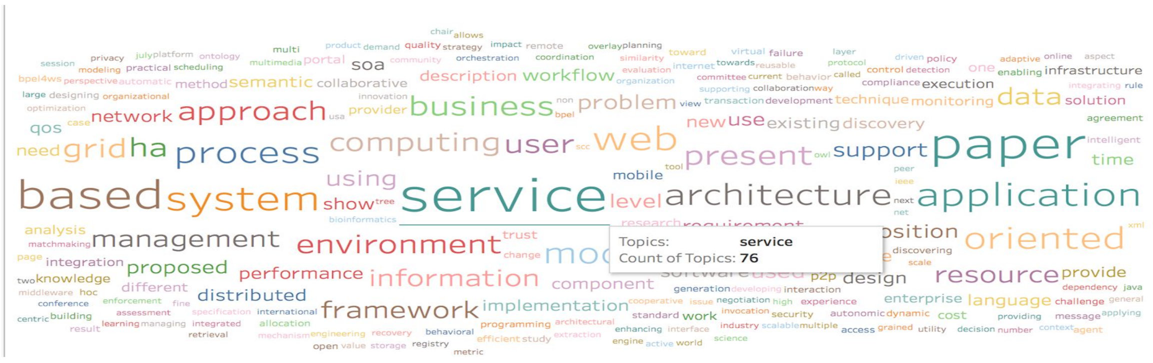
a. Implementation- We used Vistrails workflow tool to implement the advance requirement. Below is a snapshot of our workflow for the evolution of topics feature. We developed the feature in the workflow in the following steps:

1) Give inputs as per the requirement
2) Call the respective REST API using python
3) Transform data format from neo4j output to json

4) Display the final output in json format

b. Visualization - We used the cloud word feature of tableau to visualize the json data, below is a snapshot of the same.



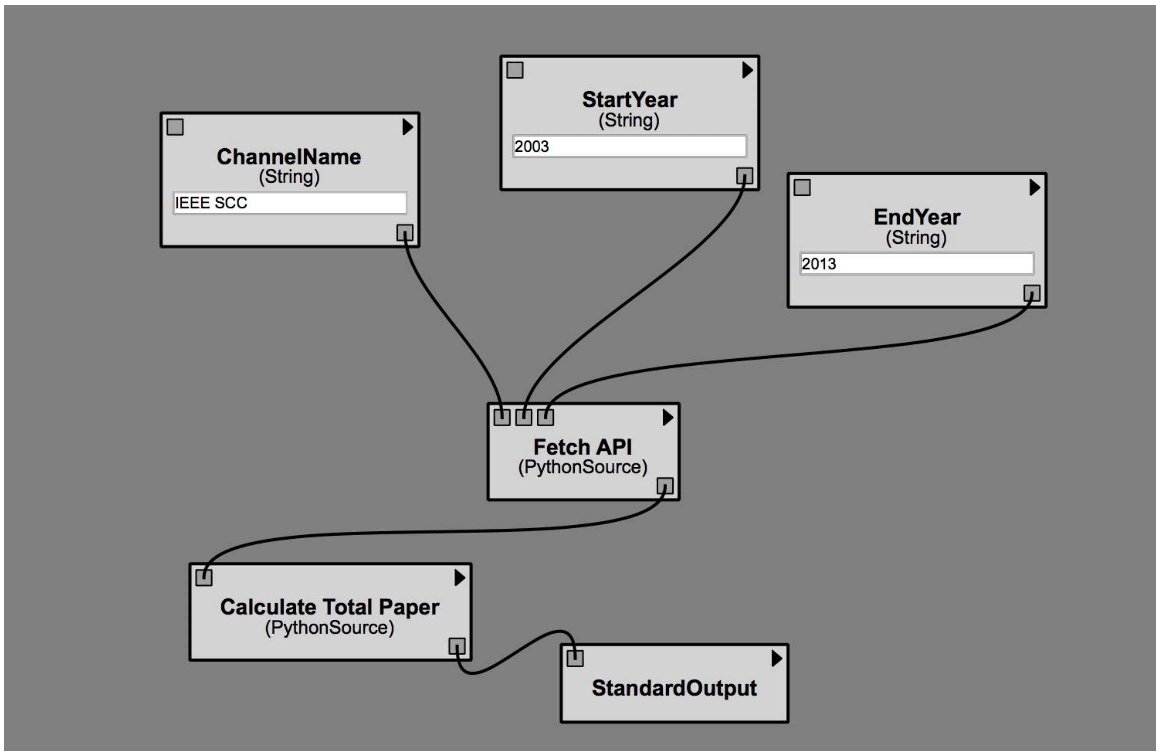


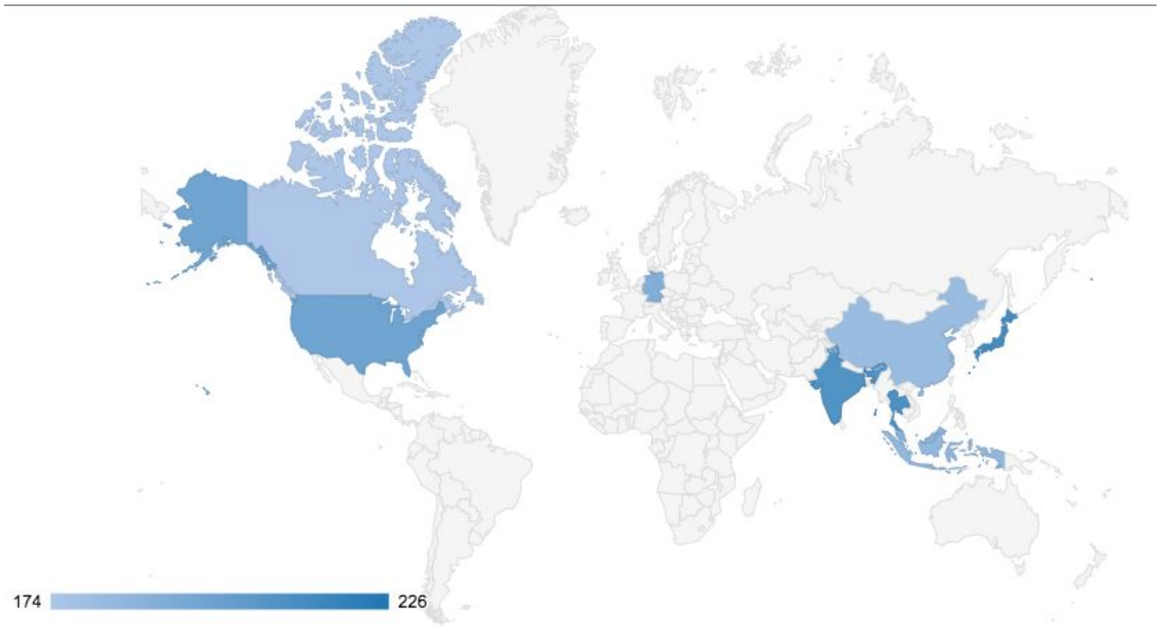
2. *Given some geographical area (e.g., country) and some keywords, generate a graph on Google map the publications on the topic, whose authors come from the geographical area (at least one or more authors).*

a. Implementation- We used Vistrails workflow tool to implement the advance requirement. Below is a snapshot of our workflow for the evolution of topics feature. We developed the feature in the workflow in the following steps:

5) Give inputs as per the requirement
6) Call the respective REST API using python
7) Transform data format from neo4j output to json
8) Display the final output in json format

b. Visualization - We used the GeoChart feature of google maps to visualize the json data, below is a snapshot of the same.





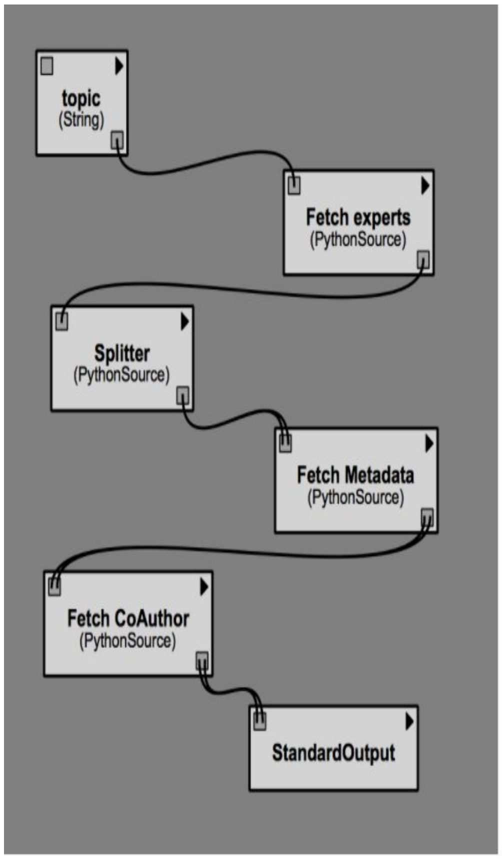
Mashup API Service

In mashup API, we decided to create API with this following requirement:

Given a topic keywords, retrieve all informations about the experts of that topics and their co-authors.

To implement this requirement, we are combining these three following requirements:

1. Given some keywords, search for researchers who are experts in the field (i.e., published significantly in the fields)
2. Given an author name, show a knowledge card with his/her metadata
3. Given the name of researcher, generate a graph* showing the direct collaboration network of the author



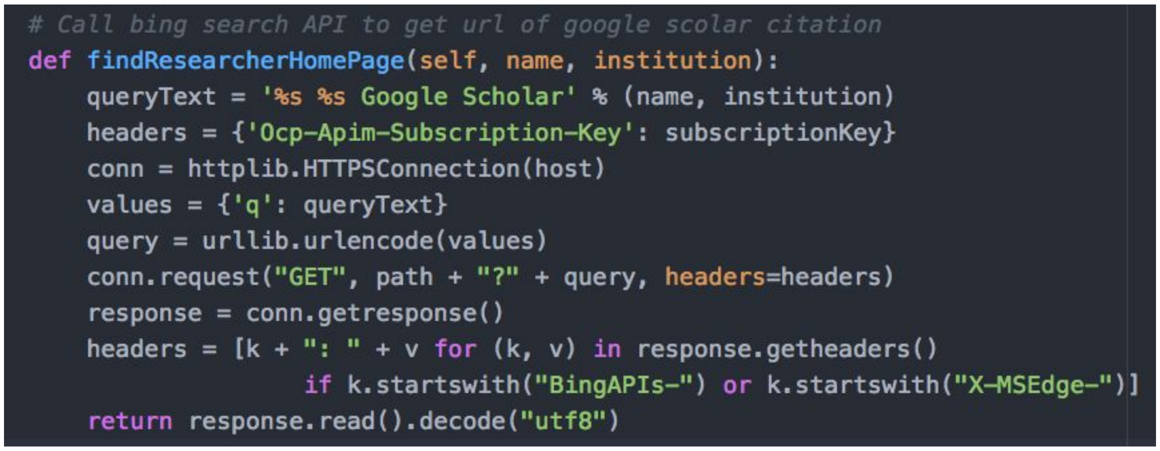
Experiment and Analysis

Google Scholar Crawler

Previously, the pre-processed data that we had has no informations about the author metadata such as author's homepage, research interests, image url, and his/her affiliation. DLPB dataset also doesn't has any information about these metadata. So, in order to get this information, we need to crawl data from google scholars page.

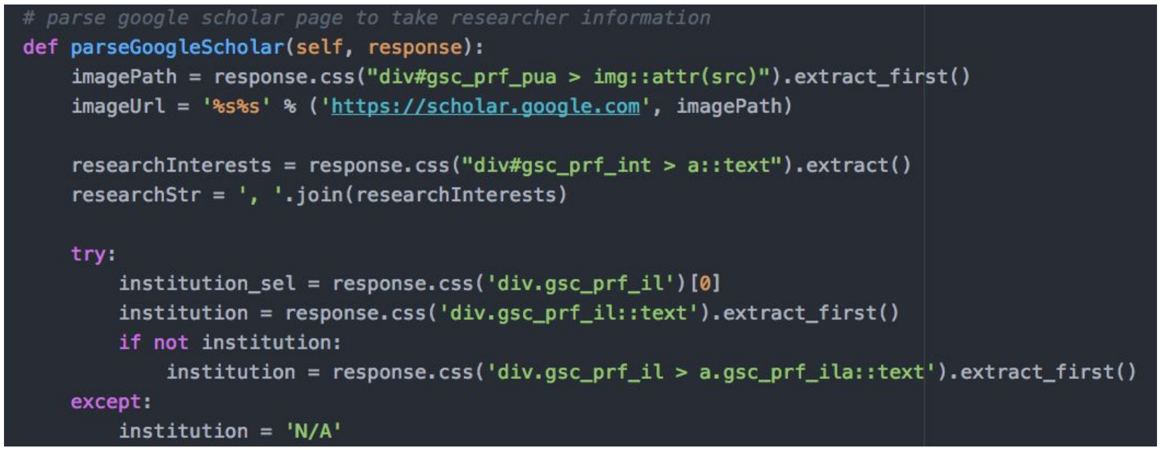
In this experiment, we are using crawler framework, called Scrapy, written in Python. Scrapy is an open source and collaborative framework for extracting the data we need from websites in a fast, simple, yet extensible way. However, before we use Scrapy to crawl the google scholar page, we need to get the URL of the author we were looking for in Google Scholar website.

We utilized Microsoft Bing search API to get the author's URL. We use author's name appended with "Google Scholar" text as the search query to help us getting the google scholar search result as the top 10 result of our query.



```python
# Call bing search API to get url of google scolar citation
def findResearcherHomePage(self, name, institution):
    queryText = '%s %s Google Scholar' % (name, institution)
    headers = {'Ocp-Apim-Subscription-Key': subscriptionKey}
    conn = httplib.HTTPSConnection(host)
    values = {'q': queryText}
    query = urllib.urlencode(values)
    conn.request("GET", path + "?" + query, headers=headers)
    response = conn.getresponse()
    headers = [k + ": " + v for (k, v) in response.getheaders()
                  if k.startswith("BingAPIs-") or k.startswith("X-MSEdge-")]
    return response.read().decode("utf8")
```

In addition, we also use the code snippet below to parse the data from google scholar page.



```python
# parse google scholar page to take researcher information
def parseGoogleScholar(self, response):
    imagePath = response.css("div#gsc_prf_pua > img::attr(src)").extract_first()
    imageUrl = '%s%s' % ('https://scholar.google.com', imagePath)

    researchInterests = response.css("div#gsc_prf_int > a::text").extract()
    researchStr = ', '.join(researchInterests)

    try:
        institution_sel = response.css('div.gsc_prf_il')[0]
        institution = response.css('div.gsc_prf_il::text').extract_first()
        if not institution:
            institution = response.css('div.gsc_prf_il > a.gsc_prf_ila::text').extract_first()
    except:
        institution = 'N/A'
```

As result, we were able to pull more or less 130 Author's metadata to our database. Actually we should be able to get more results from google scholar. However, because Google detected our application as robot, Google redirect all of our request to a robot verification page which make us couldn't continue the crawling process.

Known Issues in previous project

1. Previous project uses inconsistent format of API response. It uses JSON and plain text as the API response. Obviously, different format of API response will emerge new problem in frontend application. This is also bad from RESTful API design best practice.
2. There is no standard for returning the graph data in JSON format it used. There are several API implementations which produce graph data and each of them

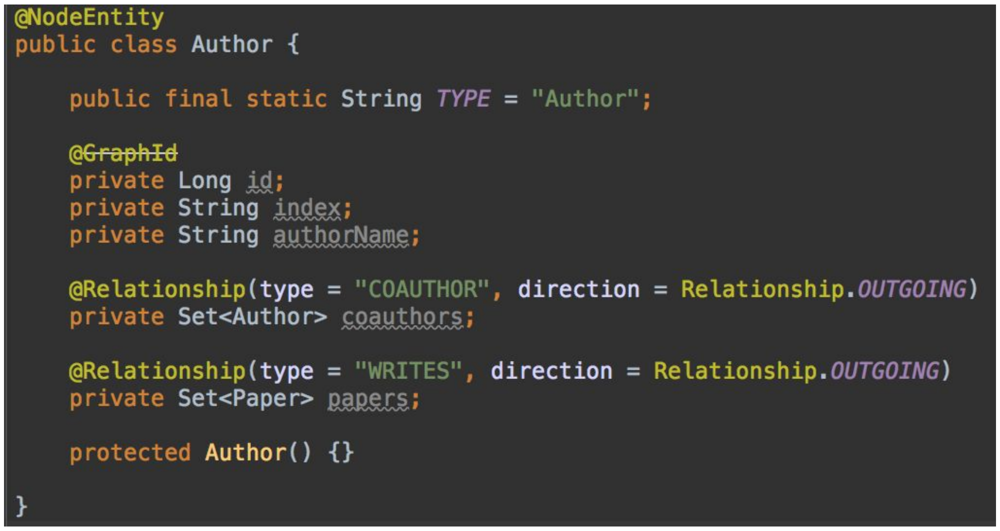
returned different JSON format to present the result to frontend app.

3. There is no chosen codestyle in project development. This problem is reflected in any code on this project. Each file written by developer will have different codestyle with another file written by different developer.
4. We also noted that there are some UI and logical bugs that we have fixed during our project iterations.

So, to fix some of those issues, we introduced several improvements on the projects.

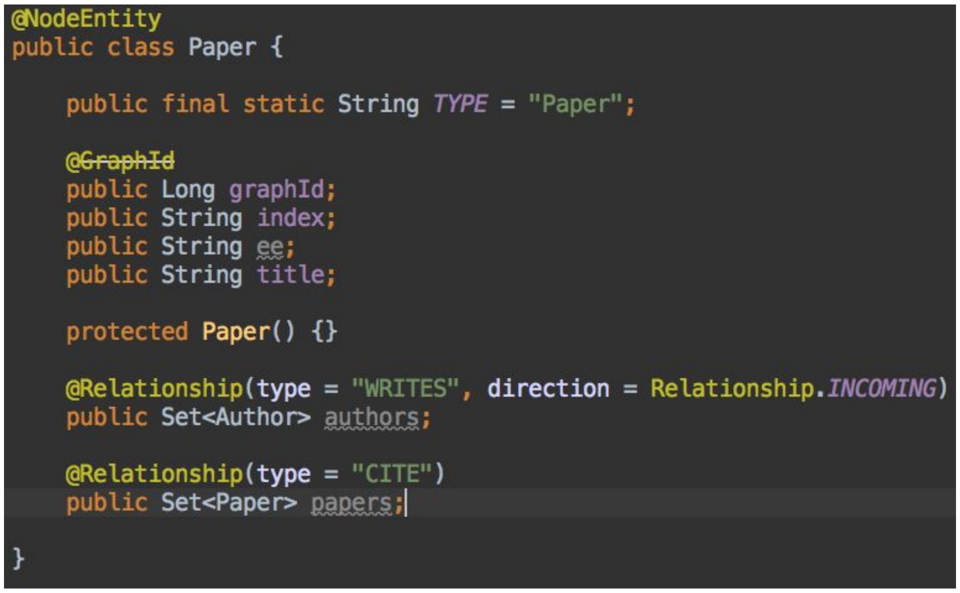
Neo4j OGM Driver

Due to the bad modeling that this project has, we are trying to improve object modeling by using Neo4J OGM driver. Neo4J OGM driver basically enables us to introduce the domain model and concept of the application to the code and to improve code maintainability of the code.



```
@NodeEntity
public class Author {

    public final static String TYPE = "Author";

    ~~@GraphId~~
    private Long id;
    private String index;
    private String authorName;

    @Relationship(type = "COAUTHOR", direction = Relationship.OUTGOING)
    private Set<Author> coauthors;

    @Relationship(type = "WRITES", direction = Relationship.OUTGOING)
    private Set<Paper> papers;

    protected Author() {}

}
```

In this example, we created Author model which has several attributes and relationships. By using OGM driver, we can easily define the relationship of Author entity to any other entities in the graph network. For example, by using @Relationship annotation, we can defined that author can have a set of co-authors which are related by COAUTHOR relationship. It is also the same as author relationship with a set of paper entity.

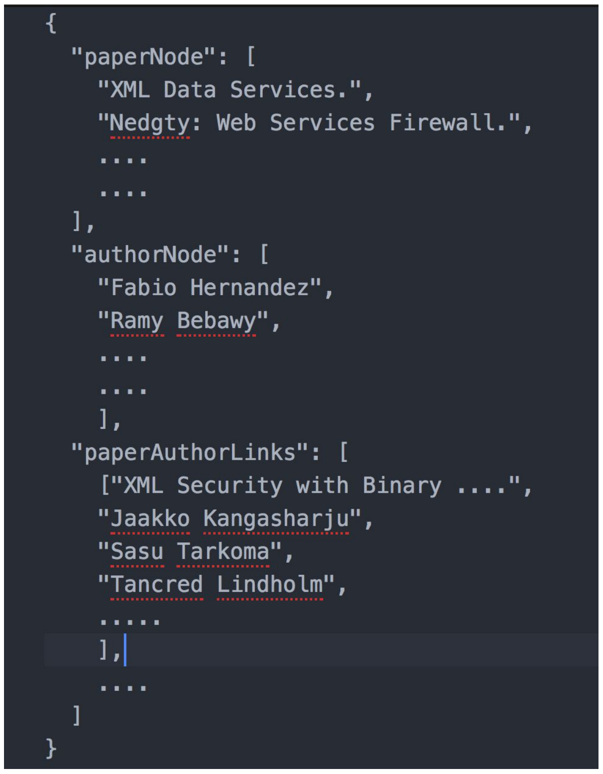


```
@NodeEntity
public class Paper {

    public final static String TYPE = "Paper";

    @GraphId
    public Long graphId;
    public String index;
    public String ee;
    public String title;

    protected Paper() {}

    @Relationship(type = "WRITES", direction = Relationship.INCOMING)
    public Set<Author> authors;

    @Relationship(type = "CITE")
    public Set<Paper> papers;

}
```

In this example, we can also use @Relationship annotation to define the relationship between Paper entity with a set of Author entities via WRITE relationship. Thus, by using OGM driver, we can easily see the relationships are explicitly defined on the code.

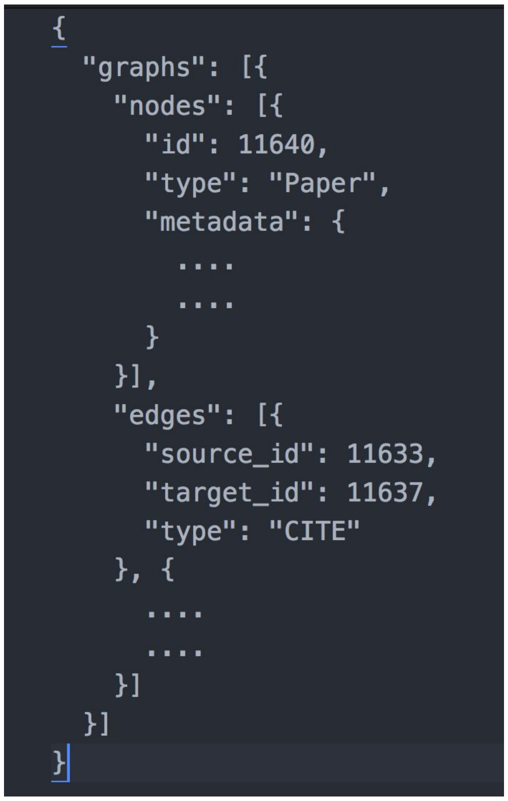
JSON Representation for graph data

We also try to fix JSON representation in graph data returned by backend.



```
{
  "paperNode": [
    "XML Data Services.",
    "Nedgty: Web Services Firewall.",
    ....
    ....
  ],
  "authorNode": [
    "Fabio Hernandez",
    "Ramy Bebawy",
    ....
    ....
    ],
  "paperAuthorLinks": [
    ["XML Security with Binary ....",
    "Jaakko Kangasharju",
    "Sasu Tarkoma",
    "Tancred Lindholm",
    .....
    ],
    ....
  ]
}
```

As you can see on image above, the old project use bad technique to represent the graph information. Using paperNode, authorNode, and paperAuthorLinks as the field name is really a bad idea. Imagine what will be happened if we have other entities that we want to represent in our graph network. Let's take Journal as another entity. Should we use journalNode as the JSON key? And what if the journal entity has relationship with author and paper? Should we then define journalAuthorLinks and journalPaperLinks? This design is not scalable at all.



```
{
  "graphs": [{
    "nodes": [{
      "id": 11640,
      "type": "Paper",
      "metadata": {
        ....
        ....
      }
    }],
    "edges": [{
      "source_id": 11633,
      "target_id": 11637,
      "type": "CITE"
    }, {
      ....
      ....
    }]
  }]
}
```

In our team project, we introduce better way to represent graph information in JSON format. We start by using graphs as the parent key for the JSON object. Then, we will have nodes and edges, which are JSON array. The nodes array will contains all nodes that we have in our network. Each node object will also have it's id, node type, and metadata which can contain a lot of information about that node. Then, to represent the relationship of the nodes, we have edges array which will have all edges for all nodes in our network. Each edge node will have source_id and target_id and also the type of relationship.

By using this design, we can create more reusable and generic JSON format which can be used by any types of nodes. If we want to add new node, let's say Journal node, we don't have to change the JSON structure at all. We just need to add the journal as a node along with it's metadata and then we can easily add the relationship of that node with the other nodes just by adding new edge object.

The other benefits of using this design is, we can have reusable JSON adapter in the frontend application. Thus, anytime we add new nodes to our JSON, frontend code doesn't has to change the interface implementation at all.

Tutorial & Project Setup

All project installation and setup is explained in detail in README file in github repository. Here are the steps:

1. The project source code is hosted in github repository. To clone this project, simply use the following command:

   git clone git@git@github.com:cmusv-sc/18656-Fall2017-Team5.git
2. Install Neo4J database and use **neo4j** as username and **123456** as password.
3. You need to populate the database with pre-processed data we have. Please go to backend_data_load folder and see how to populate the neo4j database with our pre-processed data.
4. Install activator/sbt to run the backend code. To run the backend code, use this following command:
   ```
   activator run
   ```
5. Install node and npm to run the frontend code. To run the frontend code, use this following command:
   ```
   npm install
   npm start
   ```
6. You can access the project from your browser by using **http://localhost:3000** as the URL.

Conclusions and future work

Conclusion

1. Application consists of a backend and frontend application. The backend is supported by Neo4J database to store all graph informations.
2. All basics and advanced features have been implemented and work as expected.

Future works:

1. Currently, this application doesn't have any cache used to store the backend response. Using cache surely will improve the system performance because every time multiple similar requests hit the API endpoint, backend application doesn't has to perform the same query multiple times to the database.
2. Currently, the login credentials are stored in Neo4j data and they are represented as Neo4j node label. From system design point of view, this approach is not good. Usually login credentials such as username and password will be stored in transactional database such as MySQL, PostgreSQL and etc. By doing it this way, we can limit the information that our user can have when they want to pull some informations from our Neo4J database. In the other words, we will not easily leak our user credentials information to all users who have access to Neo4J database and network or graph data.

Contribution of each Team Member

Imre Nagi

Basic Requirements:

1. Generate a Paper-Paper network, while edges representing their citation relationships.
2. Generate a Person-Person network, while edges representing their direct collaboration relationships (co-authorship in papers).
3. Generate a Paper-Person network, while edges representing their authoring relationships.
4. Given an author name, show a knowledge card with his/her metadata (name, title, affiliation, email, research interests, publication list in the field);

Advanced Requirements & Workflow:

1. Categorize all research papers (given a time period).
2. Given two researchers' names, find out a path that will lead them to connect to each other (prove the small-world theory through co-authorship).
3. Categorize the research papers (given time period, publication channels, keywords).
4. Given some keywords, generate a graph of top k related papers together with their authors.

Mashup API:

1. Given a topic keywords, retrieve all informations about the experts of that topics and their co-authors.

General:

1. Refactor JSON format for graph data
2. Refactor backend code with Neo4J OGM Driver

Shreyansi Kumar

Basic Requirements:

1. Given a time period of years for an author, generate a histogram showing her publications per year in the field. Use Tableau to generate the histogram and upload.
2. Given a journal name, generate a histogram showcasing the number of authors contributing to each of its volume, taking volume as the base axis.
3. Given a publication channel (a journal or a conference) and a year, list the focused topics of the papers (based on topics and abstracts) and show in Tableau generated diagram. (Hint: use LDA to find topic distribution)

Advanced Requirements:

1. Show the evolution of focused topics of a journal, in a given year frame - generated the visualization in tableau and excluding workflow implementation.

2. Categorize all research papers (given a time period) - generated the visualization in tableau and excluding workflow implementation.

3. For each major publication channel (e.g., a journal or a conference), list the top 20 cited papers - include feature implementation and exclude workflow implementation.

4. For each major publication channel (e.g., a journal or a conference) and a given year or year period, list the top 10 most cited papers-include feature implementation and exclude workflow implementation.

Workflows:

1. Given a publication channel name (journal or conference) and a time frame, showcase in Google map the publications distribution.

Mashup API:

Given a topic keywords, retrieve all informations about the experts of that topics and their co-authors.

Siva Rama Krishna

Basic Requirements:

5. Given a time period of years for an author, generate a histogram showing her publications per year in the field. Use Tableau to generate the histogram and upload.

7. Given some keywords, search for researchers that you may like to follow (who published significantly in the related topics as yours). Provide a ranked list of the researchers. Compared to question 8, this item should also consider your own interests and profile.

Advanced Requirements

7. Given the name of a researcher, generate a graph* showing a multi-depth collaboration network of the author (her co-authors and their co-authors).

15. Show a graph of their relationships (e.g., citation)

Workflows

7. Given the name of a researcher, generate a graph* showing a multi-depth collaboration network of the author (her co-authors and their co-authors).

8. Given some keywords, generate a graph of top k related papers together with their authors.

9. For each major publication channel (e.g., a journal or a conference), list the top 20 cited papers.

10. For each major publication channel (e.g., a journal or a conference) and a given year or year period, list the top 10 most cited paper

15. Show a graph of their relationships (e.g., citation)

Mashup Service

Allow users to mashup existing APIs for more comprehensive functions. Build a use case example.

Sheersha Kandwal

Basic Requirements :

1. In a publication network, click on a paper, show a knowledge card summarizing its publication information (metadata) and citation data.
2. Allow to view statistics and detailed information of one's own followers.

3. Allow to view statistics info of a person's followers.

Advance Requirements :

1. Given some keywords, find out a network that can link them together.
2. Given a research topic, form a possible research team, taking into consideration of their expertise in the field, their past collaboration relationships, and their possibility of willingness to join the team.
3. Show the evolution of focused topics of a journal, in a given year frame.
4. Given some keywords, return a collection of papers that the reader may be interested in reading (sorted by relevance).

Workflows:

1. Given some keywords, return a collection of papers that the reader may be interested in reading (sorted by relevance).

Mashup Service

Allow users to mashup existing APIs for more comprehensive functions. Build a use case example.

References

[1]
http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=57192&copyownerid=7812
[2] https://en.wikipedia.org/wiki/Neo4j
[3] https://www.playframework.com