b
UNIVERSITÄT
BERN

Visually Exploring Scientific Communities

Extending EggShell's Model and Visualization

Bachelor Thesis

Silas David Berger
from
Hilterfingen BE, Switzerland

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

31. August 2017

Prof. Dr. Oscar Nierstrasz
Leonel Merino

Software Composition Group

Institut für Informatik
University of Bern, Switzerland

Abstract

Researchers within a community gather in scientific conferences periodically. As a result, the scientific output, reflected in the papers published in conferences, carries valuable knowledge about the underlying community, such as collaboration groups and the history of the evolution of topics. Researchers can use this knowledge to identify *i*) candidates for collaboration for future projects, *ii*) active topics of research, and *iii*) relevant papers in their field of research. However, to obtain this knowledge, researchers would have to collect and analyze data such as titles, authors, and keywords that might be spread across thousands of papers.

The size and the number of relations in such data sets can make the analysis hard using tabular representations such a spreadsheet. Instead, visualizations provide users a graphical representation of the attributes and relations of data on which they can reflect. Through visualizations researchers can obtain an overview of a scientific community, analyze patterns of evolution, and identify entities of interesting.

We propose ExtendedEggShell (EES), a unified framework for extracting, modeling and visualizing scientific communities. EES enables users to visualize the collaboration network of a community based in node-link diagrams, and interact with the graphs by posing queries inspired by meaningful keywords in word clouds.

We evaluate the performance of EES by analyzing the complete set of 366 papers published in the software visualization community (VISSOFT). We demonstrate the tool via selected usage examples, on which we analyze 1084 papers from the object-oriented, systems, languages and applications community (OOPSLA). We found that visualizing scientific communities as bigraphs using node-link diagrams helps users to better understand the collaboration within these communities.

Contents

1 Introduction

The collection of papers published in a conference carries valuable knowledge about its underlying scientific community, such as the network of collaboration. However, this knowledge can be spread out across thousands of papers. A user who wants to obtain such a knowledge would have to collect, extract and model data such as paper titles and authors. The size of the resulting data and its complexity in terms of the number of relationships make them hard to analyze through tabular representations such as spreadsheets. Instead, we believe users could benefit from visualization that can provide them a suitable support for analysis. We wonder:

RQ.1) What factors influence the accuracy of data extraction and modeling of scientific communities?

RQ.2) How can visualization support the understanding of the dynamics of scientific communities?

Previous research has proposed an approach to extracting and modeling scientific communities. EggShell [9] transforms PDF files into plain text, for which it recovers the semantic structure of papers based on ad-hoc heuristics. Although this strategy can give good results for papers that use well-known format, it can be time-consuming and error prone. Furthermore, the adaptions made for a collection of papers may not be applicable to future collections, requiring to repeat the effort.

CommunityExplorer [7] builds on top of EggShell. CommunityExplorer models scientific communities as bigraphs of author and paper nodes, and visualizes them using node-link diagrams.

We expand on the idea of EggShell and propose ExtendedEggShell (EES) — a new framework for the visualization of scientific communities. EES relies on *1*) third-party tools to recover the semantic structure *2*) heuristics to improve the accuracy of the model, and *3*) visualization of collaboration networks modeled as a bigraph.

As opposed to EggShell, ESS relies on third-party tools for data extraction, which have proven to produce more accurate data and also expand the data available. ESS not only extracts paper titles and authors but also author affiliations, and e-mail, section titles and citations.

Although ESS adopts bigraphs to model collaboration networks similar to CommunityExplorer, in EES we opted to clean the data once modeled and not when it is extracted. ESS first models the extracted data as-is, then relies on heuristics to improve the accuracy of the model. Then, visualizing the model users can iteratively expand and improve the cleaning heuristics to incrementally improve the accuracy of the model.

We evaluate ESS by analyzing the complete set of 366 papers published in the software visualization community (VISSOFT). We demonstrate the tool on a collection of 1084 papers from the object-oriented, systems, languages and applications community (OOPSLA)

2
ExtendedEggShell

We introduce ExtendedEggShell (ESS), a framework for the visualization of scientific communities. ESS is implemented in Pharo (a modern Smalltalk[1]). ESS uses the Moose platform [8] for data analysis and the Roassal engine [1] for visualization. We made ESS source code publicly available to facilitate reproducibility[2]. In the following we describe the design and implementation of the framework pipeline.

2.1 Framework Pipeline

The publication record of scientific communities contains valuable data that can help researchers to understand how collaboration occurs. Collections of papers (typically in PDF format) contain data such as paper titles, and author names and affiliations. Consequently, we designed EES to extract, model and visualize these data from the scientific community paper collection. Figure 2.1 shows the 5-step pipeline implemented by the framework. ESS (1) extracts plain text from PDF files, (2) recovers semantic structures from the plain text, (3) models community entities based on the recovered semantic structures, (4) cleans the data in the model, and (5) builds visualizations based on the model.

The first two steps focus on extracting data from the PDF files. These data include author names, affiliations, e-mail addresses; and also paper titles, section titles, and

[1] http://pharo.org

[2] http://smalltalkhub.com/#!/ SilasBerger/ExtendedEggShell

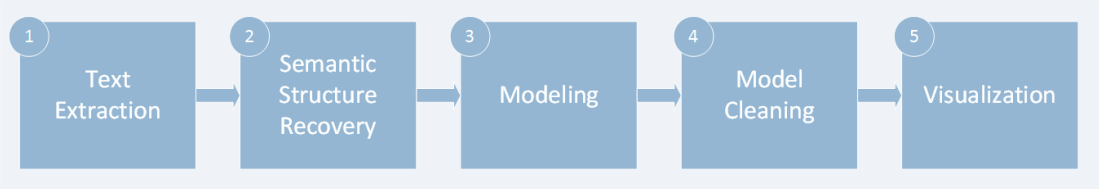

Figure 2.1: steps of the framework pipeline

paragraphs. In step 3, ESS models these data. We observe that the data extraction shows numerous difficulties. For example, a common error with the extracted name of authors is including in the name also the author's affiliation. In step 4, ESS uses heuristics to resolve these problems in the model. In step 5, we visualize the model.

In the remainder of this chapter, we examine the implementation details of the framework pipeline. We implemented EES using Pharo as a unified environment for the framework. Being a live programming environment, it supports inspection and modification of of the state of objects at any point during runtime. This allows users to modify and improve the model without having to restart the program execution after changing its code.

2.2 Text Extraction

We decided to extract the text from PDF files using Apache PDFBox[3] — an open source Java library for PDF manipulation. We designed ESS to provide users a unified environment for modeling, analyzing, and visualizing scientific communities. In consequence, we wanted to use PDFBox from the ESS implementation in Pharo. We used `PipeableOSProcess`[4] to link ESS and PDFBox using the command line.

Listing 1 shows an example of the text extraction result from one paper [6]. Notice that even though for a human reader to identify the semantics of the structure of the paper from the extracted plain text can be straightforward, for a machine it imposes a great obstacle.

```
Finding Refactorings via Change Metrics
Serge Demeyer
LORE - University of Antwerp
Universiteitsplein 1
B-2610 WILRIJK (Belgium)
sdemey@uia.ua.ac.be
http://win-www.uia.ac.be/u/sdemey/ÃƒÂ©
Stphane Ducasse
```

[3]https://pdfbox.apache.org
[4]http://pharo.gemtalksystems.com/book/PharoTools/OSProcess/

```
SCG - University of BerneÃƒÂ¼
Neubrckstrasse 10
CH-3012 BERNE (Switzerland)
ducasse@iam.unibe.ch
http://www.iam.unibe.ch/~ducasse/
Oscar Nierstrasz
SCG - University of BerneÃƒÂ¼
Neubrckstrasse 10
CH-3012 BERNE (Switzerland)
oscar@iam.unibe.ch
http://www.iam.unibe.ch/~oscar/
ABSTRACT
Reverse engineering is the process of uncovering the design and
the design rationale from a functioning software system. Reverse
```

Listing 1: extracted plain text from the example paper [6] PDF file

2.3 Semantic Structure Recovery

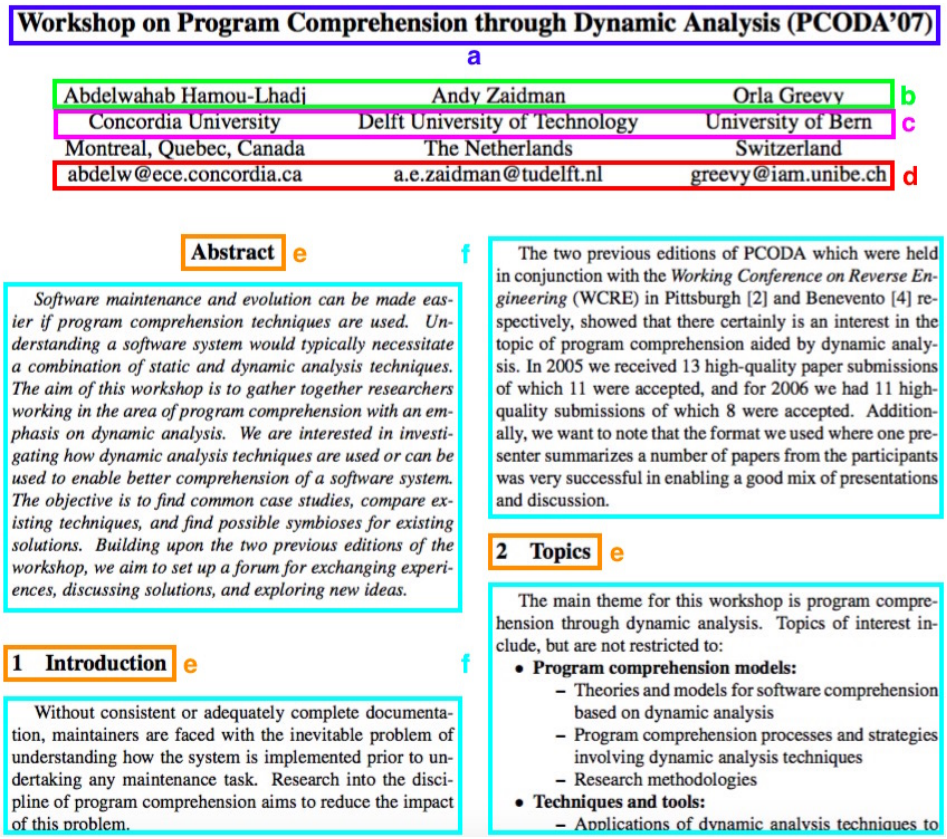

Figure 2.2: semantic structure of a paper

ESS recovers the semantic structure using the extracted plain text as input. Figure 2.2 shows the structure of a paper: (a) title, (b) author names, (c) affiliations, (d) e-mail, (e) section titles, and (f) paragraphs.

ESS relies on ParsCit [5] —an open source tool for semantic document structure recovery. ParsCit relies on machine learning algorithms to extract data such as paper titles and authors names from papers in plain text format. It offers a command line interface, which takes the text file as input and produces as output information of the semantic structure of a paper in XML format. Uniformly, we link ParsCit to ESS using ParsCit's command line tools through PipeableOSProcess.

ParsCit includes three algorithms: *SectLabel*, *ParsHed*, *ParsCit* (shown in Listings 2, 3, and 4 respectively) that target various aspects of the semantic structure of papers.

SectLabel focuses on recovering elements such as paper title, section titles, paragraphs, and author names.

```
<algorithm name="SectLabel" version="110505">
   <title confidence="0.758802">
      Finding Refactorings via Change Metrics
   </title>
   <author confidence="0.715038">
      Serge Demeyer
   </author>
   <affiliation confidence="0.654289">
      LORE - University of Antwerp
   </affiliation>
   <figure confidence="0.900427125">
      ...
   </figure>
   <sectionHeader confidence="0.813482" genericHeader="abstract">
      ABSTRACT
   </sectionHeader>
   <bodyText confidence="0.9983555">
      Reverse engineering is the process of...
   </bodyText>
   ...
</algorithm>
```

Listing 2: example result of the SectLabel algorithm

The `confidence` attribute is a normalized metric which indicates the certainty to which the respective element was recovered correctly. A confidence of one means that an element was recovered correctly with absolute certainty.

In our experience using the tool, we observed that SectLabel even though it performed well for extracting section titles and paragraphs, it did not perform well for elements such as paper titles, and author names, affiliations and e-mail. In only a few cases the algorithm was able to extract all these elements without any errors. We observe this issue

in Listing 2, in which two co-authors of the paper (*i.e.,* Stéphane Ducasse and Oscar Nierstrasz) are missing.

ParsHed focuses on recovering paper title, and author names, affiliations, and e-mail. Listing 3 shows an extract of the ParsHed node of the example paper [6].

```
<algorithm name="ParsHed" version="110505">
   <title confidence="0.999597">
      Finding Refactorings via Change Metrics
   </title>
   <author confidence="0.99976">
      Serge Demeyer
   </author>
   ...
   <author confidence="0.643633">
      Stéphane Ducasse
   </author>
   ...
   <author confidence="0.760562">
      Oscar Nierstrasz
   </author>
   <affiliation confidence="0.91192">
      SCG - University of Berne
   </affiliation>
   <address confidence="0.9574545">
      Neubrückstrasse 10 CH-3012 BERNE (Switzerland)
   </address>
   <email confidence="0.987768">
      oscar@iam.unibe.ch
   </email>
   <web confidence="0.998921">
      http://www.iam.unibe.ch/~oscar/
   </web>
   <abstract confidence="0.999847411764706">
      Reverse engineering is the process of ...
   </abstract>
   ...
</algorithm>
```

Listing 3: example result of the ParsHed algorithm

We think that ParsHed performs better than SectLabel at extracting titles and authors. In the case of the example paper [6], all three authors of the paper are recovered correctly, including additional data, such as affiliation, e-mail and web address.

ParsCit is an algorithm to recover the list of citations from a paper. Each citation contains a paper title, the author names, and the publication date. Listing 4 shows the ParsCit node for the example paper [6].

```
<algorithm name="ParsCit" version="110505">
   <citationList>
```

```
      <citation valid="true">
         <authors>
            <author>Robert S Arnold</author>
         </authors>
         <title>Software Reengineering</title>
         <date>1992</date>
         ...
      </citation>
      ...
   </citationList>
</algorithm>
```

Listing 4: example result of the ParsCit algorithm

2.4 Model

ESS models the extracted entities in a way such that they reflect the main elements of the scientific community. We designed the model having in mind that it will be the base to create visualizations and to pose queries that will help users to analyze the scientific community.

Figure 2.3 shows a class diagram of the model design. In it, the scientific community is represented as a collection of papers and authors. The two collections provide methods to query the model. A `Paper` contains the paper title, authors, publication year, its paragraphs, a collection of its section titles, and a collection its of cited papers. An `Author` object contains the author's name, affiliation and e-mail address. Both `Paper` and `Author` inherit from `AbstractCommunityItem` that holds the state of three variables: `communityItemId`, `isColleted`, and `additionalInfo`. The variable `isCollected` is a boolean that identifies when a paper is part of a source collection – a collection of papers in PDF format. Hence, we distinguish `Paper` objects that originate from a paper in the source collection to the papers created from citations, which we call *absent papers*. Similarly, `isCollected` identifies authors who contributed to at least one paper in the source collection. The `additionalInfo` dictionary variable is used to store collection specific properties of a paper or author. For example, to store data of the source of funding of papers.

We extend the meaning of the words *paper* and *author* to include instance of the `Paper` class and the `Author` class respectively.

By default, the `ModelLoader` (the class responsible for loading the model) creates a new `ScientificCommunity` model upon initialization. However, in many cases users may want to extend an existing model. This can for example be useful for the collection of papers of a conference, which is extended each year the conference is held. Instead of reloading the model every time the collection is extended, users should be able to extend an existing model. They can achieve this by setting the `ModelLoader`'s

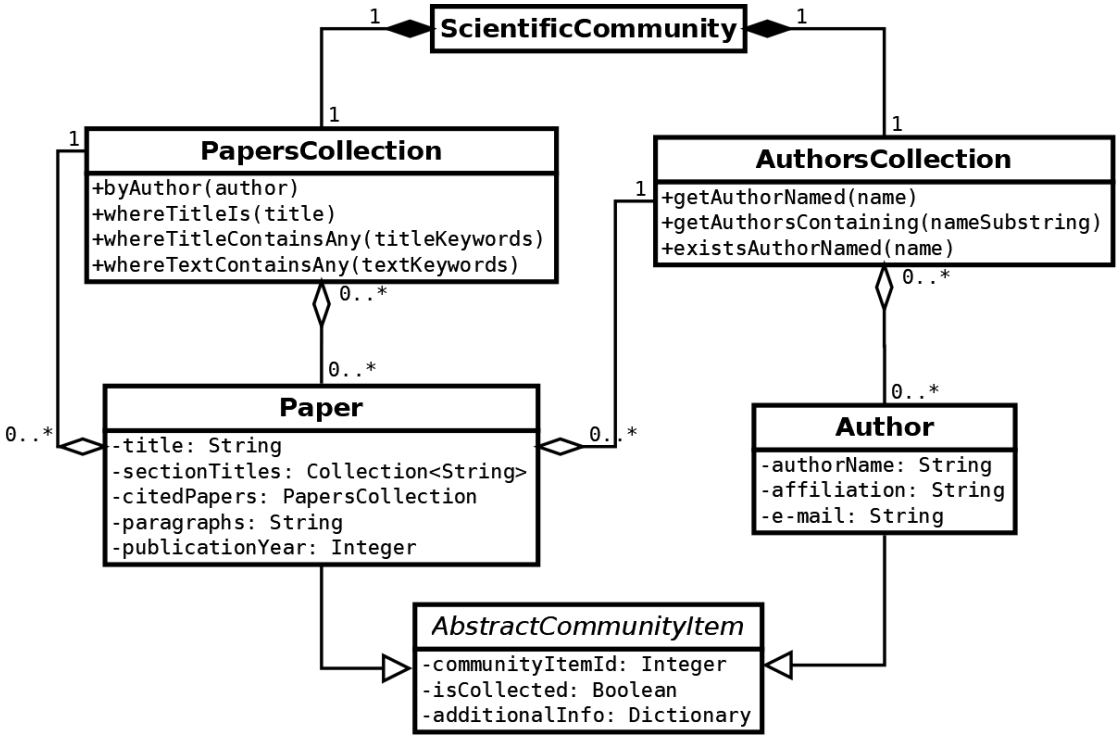

Figure 2.3: class diagram of the Scientific Community model

`ScientificCommunity` property to the existing model they wish to extend, before starting the modeling process.

2.4.1 Enriched Model

Although none of ParsCit's algorithms extract the publication year of a paper, we notice that sometimes the year is encoded in paper files or directories. Listing 5 shows an example of the file structure of the collection of papers from the various editions of the Object-Oriented Programming, Systems, Languages and Applications conference (*OOPSLA*):

```
OOPSLACollection
|__oopsla1986
|   |___p1-moon.pdf
|   |___p9-schaffert.pdf
|   |___...
|__oospla1987
|__...
|__oospla2015
```

Listing 5: folder structure encoding the papers' publication year

We therefore enrich the model of this collection by including the publication year of each paper. Listing 6 shows the script that extracts the year from the folder's name and uses it to set the paper's publication year property during modeling:

```
1  sc := ScientificCommunity new.
2  collectedPapers := PapersCollection new.
3  1986 to: 2015 do: [ :year |
4      | folder newPapers |
5      folder := rootFolder / ('oopsla', year asString).
6      ModelLoader new
7          scientificCommunity: sc;
8          modelPDFs: folder.
9      newPapers := sc papers difference: collectedPapers.
10     newPapers do: [ :newPaper | newPaper year: year ].
11     collectedPapers addAll: newPapers].
```

Listing 6: modeling publication year information encoded in folder structure

The `rootFolder` variable holds a `FileReference` to the *OOPSLACollection* folder (see structure outline above in Listing 5). For each year of a conference edition (line 3), the algorithm obtains the list of PDF paper files (line 5), and populates a model by creating author and paper objects. Then, it calculates newly added papers with respect to a previous paper collection (line 9). Finally, the algorithm writes the publication year to paper objects to update the paper collection (lines 10 and 11).

2.5 Cleaning the Model

Oftentimes the output data resulting from step 2 of the pipeline (*i.e.,* semantic structure recovery, Section 2.3) contains errors, such as misspellings of an author name or paper title. Typically, they relate to author names and paper titles include additional wrong words or characters. We found *1*) additional non-alphanumeric characters in author names and paper titles; *2*) words in author names that are not part of the name. For example, for an author such as "Oscar Nierstrasz" we found his name followed by his affiliation "Oscar Nierstrasz SCG University of Bern"; and *3*) multiple author names extracted as a single author name, such as "Oscar Nierstrasz Stéphane Ducasse". Other times, we found errors that relate to the use of accents that are extracted as two separate characters: a non-accented letter and its accent.

Figure 2.4 shows the results of looking for all authors whose names contain "Oscar Nierstrasz" in a model[5] that has not yet been cleaned. The figure shows examples of the discussed errors, such as multiple author being extracted names as a single name.

[5]This model corresponds to a collection of 100 randomly selected papers from the SCG PDF archives `http://scg.unibe.ch/archive/papers`

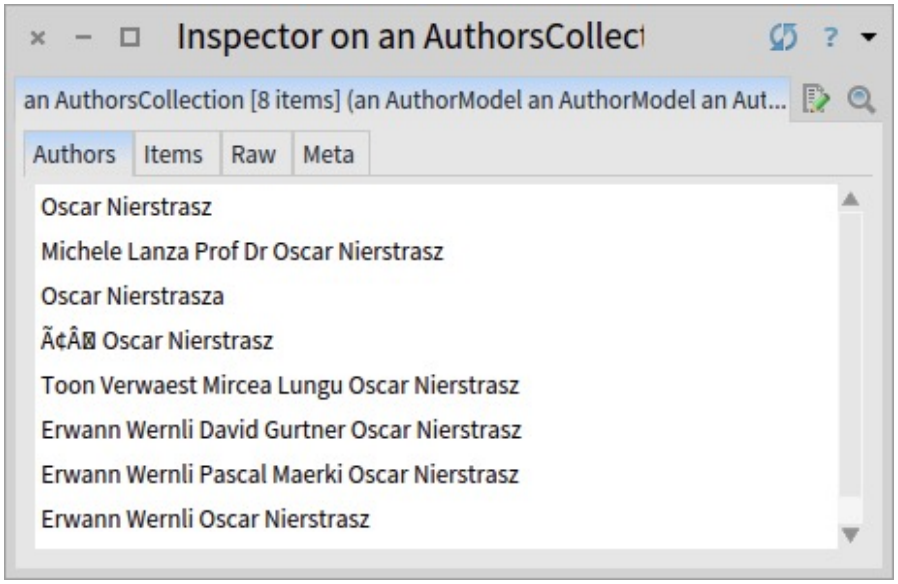

Figure 2.4: authors whose names contain ”Oscar Nierstrasz”, before cleaning the model

Although these errors could be removed in step 2 (*i.e.,* semantic structure recovery), we decided to clean the data already modeled as first-class objects.

We observe that cleaning the model is an iterative process. Although we start by removing a set of type of name errors, when we visualize the data (at the end of the process) we might detect new errors. We then would want to return to this step and expand the number of cleaning heuristics. We therefore introduce the EES' model cleaning system, which relies on heuristics to clean the model from errors.

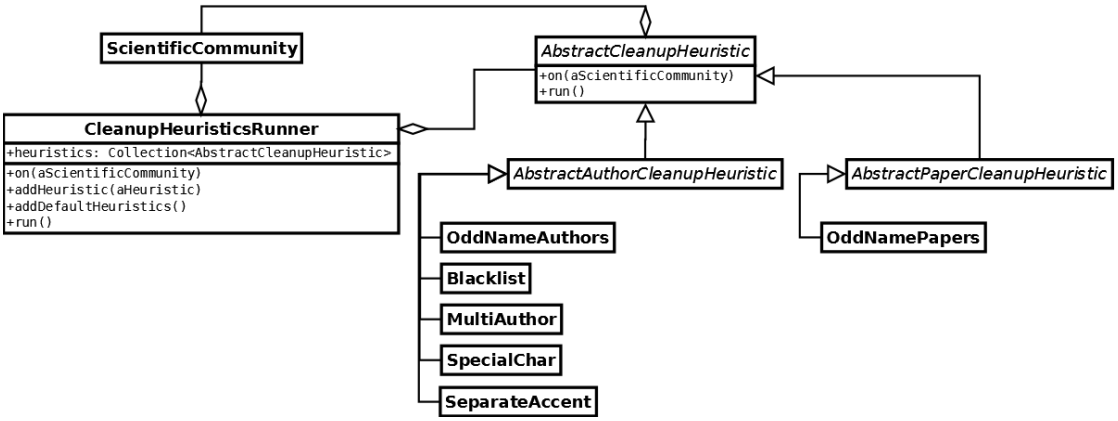

Figure 2.5: class diagram of the model cleaning system in EES

Figure 2.5 shows a class diagram of the extensible design of the cleaning step. `AbstractCleanupHeuristic` is an abstract class that holds a reference to a model (*i.e.,*

`ScientificCommunity`). A number of particular heuristics that fix errors in extracted names and titles are implemented as an extension to this class. Each extended class has to override the `run` method to implement its logic. Heuristics that tackle errors in author names extend `AbstractAuthorCleanupHeuristic` and the one that tackle paper title errors extends `AbstractPaperCleanupHeuristic`. The execution of these heuristics is managed by the `CleanupHeuristicsRunner`, referred to as the *runner*. `CleanupHeuristicsRunner` class holds a reference to a model and to a list of heuristics (*i.e.,* subclasses of `AbstractCleanupHeuristic`).

Users can add heuristics classes using the runner's `addHeuristic:` method. The runner will only run the explicitly added heuristics, in the sequence they were added. Users need to be able to choose which heuristics to run, because not all heuristics may perform well on every model. Once users have added all desired heuristics, they call the runner's `run` method, which iterates through its collection of heuristics. For each heuristic, it will *i*) instantiate the class, *ii*) pass it the reference to the targeted `ScientificCommunity` model, and *iii*) call the instance's `run()` method. In the following we examine the heuristics that we implemented (shown in Figure 2.5).

2.5.1 Blacklist

We found some frequently occurring names containing words that come from authors' affiliations, such as *University*, *Institute*, *Informatics*. We therefore defined a blacklist of words that are unlikely to belong in a name. The `Blacklist` class not only removes the words in the blacklist but also the words in between two blacklisted word. We implemented it in this way since we did not find a case where proper name parts were located between blacklisted words.

2.5.2 MultiAuthor

We often observed that papers written by multiple authors (*e.g., Erwann Wernli* and *Oscar Nierstrasz*) are extracted as a single author (*e.g., Erwann Wernli Oscar Nierstrasz*). Algorithm 1 shows how this heuristic attempts to clean the model from such errors.

Note that, in certain cases, this heuristic might behave in an unexpected manner. For example, if a paper has an author named *Joe Ortega Zisman Jon Doe*, and the model contains another author named *Joe Ortega*. The heuristic will detect *Joe Ortega* as an additional author of that paper, which is correct, assuming that *Joe Ortega* and *Joe Ortega Zisman* are variations of the same name. It will however reduce the name *Joe Ortega Zisman Jon Doe* to *Zisman Jon Doe*, instead of *Jon Doe*. In Chapter 2.8.1 we show that, in our evaluation dataset, the heuristic had a positive impact on model accuracy nevertheless.

Algorithm 1 pseudo-code algorithm of the MultiAuthor heuristic

```
1: procedure MULTIAUTHOR
2:     for each author ∈ Model.authors do
3:         containedAuthors ← new Collection
4:         for each candidate ∈ Model.authors do
5:             if author.name contains substring candidate.name then
6:                 remove substring candidate.name from author.name
7:                 add candidate to containedAuthors
8:         add containedAuthors to papers by author
```

2.5.3 OddName

The main intent of this heuristic is to address *i*) casing issues, *ii*) incorrectly extracted characters, and *iii*) names consisting of only a single word, in both author names and paper titles. For each author name and paper title this heuristic removes all characters above Unicode code point 8188. We found that this is a good threshold to filter out most incorrectly extracted characters automatically, without removing correctly extracted ones. We also found that some incorrectly extracted characters fall below that threshold and often even fall into the range of common alphanumeric characters. However, we observe that they usually appear as sequences of characters, rather than just a single one. An example of such an often observed case is *Â©*. Although we could exclude the © symbol, we observe that some occurrences of Â are correct. In the future, we plan to use regular expressions that can better match the multiple cases. The algorithm also 1) removes all words containing numbers, 2) normalizes words leaving only the initial letter in capital, and 3) removes names consisting of only a single word.

2.5.4 SpecialChar

This heuristic looks for accented versions of vowels (lower and upper case). For example, the following is the class of equivalent characters (char class) for E, È, É, Ê, Ë, and e, è, é, ê, ë. That is, given two author names n_1, n_2 as *equal within char class*, if and only if they have the same length n and $\forall i \in [1, n] \subset \mathbb{N} : n_1[i] \in C \wedge n_2[i] \in C$, where C is a char class. Hence, we would for example consider *Stephane Ducasse* and *Stéphane Ducasse* as equal within char class. Algorithm 2 shows how this heuristic groups together authors whose names are equal within char class.

2.5.5 SeparateAccent

We sometimes found errors that relate to the use of accents that are extracted as two separate characters: a non-accented letter and its accent. This heuristic defines a dictio-

Algorithm 2 pseudo-code algorithm of the SpecialChar heuristic

```
1: procedure SPECIALCHAR
2:     charClassIdMap ← map char classes to integer numbers
3:     for each author ∈ Model.authors do
4:         hash ← empty string
5:         for each char ∈ author.name do
6:             hash ← hash + (charClassIdMap at char class of char) + "."
7:         sort author into char class bucket of hash
8:     for each bucket ∈ char class buckets do
9:         replace all authors in bucket by any representative
```

nary that translates such cases to the respective single accented character. The heuristic iterates through all author names and fixes such errors, using this dictionary.

2.6 Visualization

The last step of the pipeline is to generate visualizations that users can utilize to augment their understanding of the scientific community data. Through visualizations, users can gain an overview of the scientific community and identify entities of interest. We propose two visualization approaches, one based on node-link diagrams and a complementary one based on word clouds.

2.6.1 Node-Link diagrams

We opted for visualizing the bigraph model of papers and authors using the node-link technique. We configure the technique so that rectangles depict paper nodes and circles depict author nodes (shown in Figure 2.6). The size of an author node encodes the number of papers to which the author has contributed. Author nodes have a label on top to display their family name. The default color of author nodes is dark gray, while paper nodes are black. The bigraph is laid out using a force-based layout[6]. We selected this layout because it groups parts of the graph with more edges closer together, making it easier to spot groups of collaborators. During development, we experimented using different colors for paper nodes and author nodes. We also tested multiple ratios for rectangular paper nodes. We found that a width-to-height ratio of *2:1* resulted in easier distinction between paper and author nodes. The visualization can include not only papers and authors, but also absent authors. Absent papers (and their absent authors) are the result of a paper being cited by another paper, but for which there is not a PDF

[6]charge: -80, strength: 1

file in ESS. We observed that in all our models, absent entities accounted for the vast majority of all nodes. We think that they might improve the visualization for the analysis of collaboration. However, we face the problem that most absent entities are incorrectly extracted or cleaned, which leads to a huge number of nodes and a great deal of cluttering. The graph does therefore not include absent entities by default.

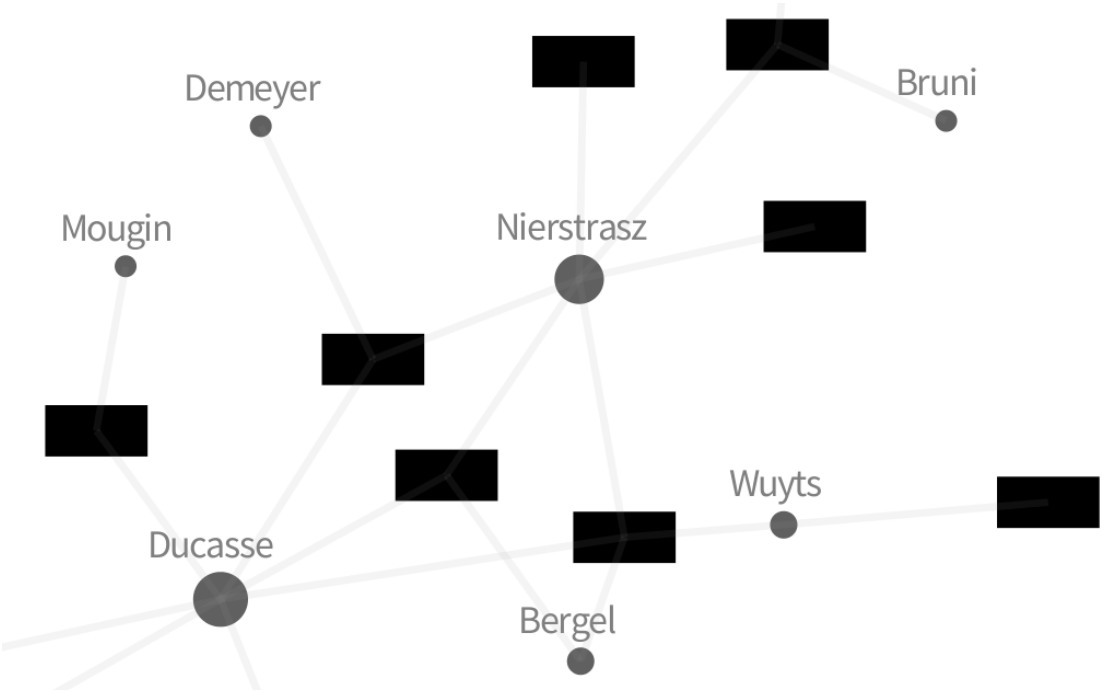

Figure 2.6: extract of an SCGraph – squares depict papers, circles depict authors

In the following we describe the characteristics of our visualization by adopting Shneiderman's popular classification [10]. Once users trigger the visualization of the model built from a collection of papers. They take the following steps:

1) Gain an overview: we consider an important characteristic of our visualization its ability to display the community as a whole. Thus, users can gain an overview over the community.

2) Zoom: Users can zoom-in to focus on interesting elements of the graph, and zoom-out to analyze how those elements connect with the the rest of the graph. Users trigger the zoom feature by using the keyboard and mouse.

3) Filter: Users define which sub-collections of papers, authors, absent papers and absent authors are included in the graph.

4) Show details on demand: Users hover over a paper node in the graph to display its title. Uniformly, they hover over an author node to display their name. Users can also select an element in the graph to inspect its properties, such as a paper's title and paragraphs, and an author's name, affiliation and e-mail.

5) Relate: Edges connect authors and papers. Users can hover over an author to highlight their immediate and transitive connections. Users can explore how authors connect to others by analyzing the highlighted edges reachable from that node. We decided to map the distance to the node to the color intensity of the edge. We believe this might help users to identify authors that have a close relationship but might be far apart in the graph. Users can also analyze the relationship among attributes by encoding them into the intensity of the color of nodes, for instance, to analyze how recent are papers.

6) Keep a history: When highlighting elements in the graph, an `SCGraphHighlighting` object is created. Users can interact with it to remove, change and apply the highlighting.

7) Extract: Users define which sub-collections of papers, authors, absent papers and absent authors are included in the graph.

Analyzing collaboration networks is complex due a large number of nodes and edges. Hence, users would benefit from expressive and flexible API to filter data from the model during visualization. ESS therefore lets users highlight the results of queries to the model. We decided to use a predefined color palette of seven qualitative colors, so users do not have to spend time deciding what color to assign to the results of a query. Figure 2.7 shows a graph with the results of a query highlighted. The system automatically assigns a color from the predefined palette. A legend appears on the right-hand side of the graph, explaining the color coding. Figure 2.8 shows the result of executing a second query in the same graph. The color automatically assigned to the new query is added to the legend. The left-hand part of Figure 2.9 shows the highlighting of the results of a more complex query.

Users can encode a numerical property in the intensity of a highlighting color, as shown in the right-hand side of Figure 2.9.

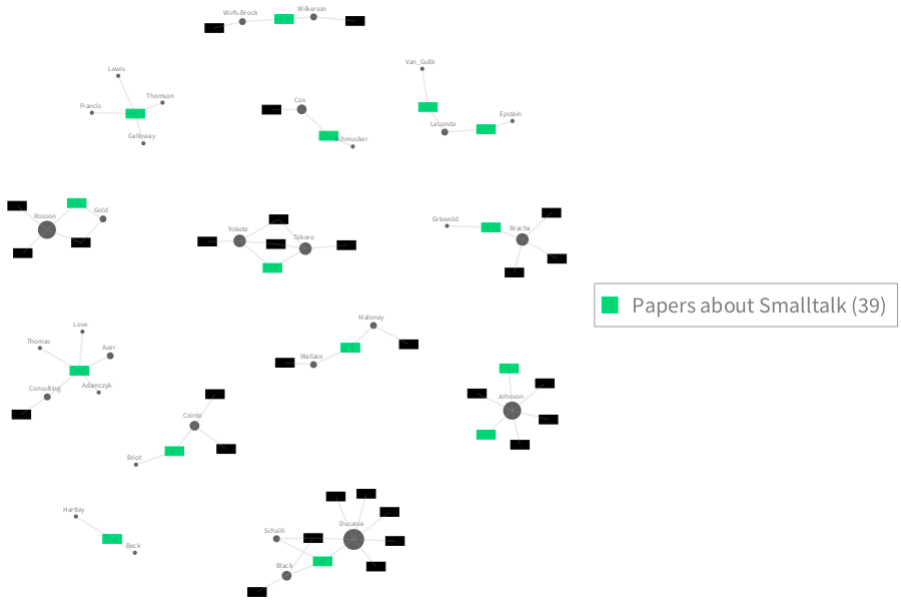

Figure 2.7: Highlighting a set of query results. 39 elements are highlighted, as indicated by the number in parentheses, in the legend on the right-hand-side.

Figure 2.8: Multiple sets of query results can be highlighted at the same time. Only two elements match this second query.

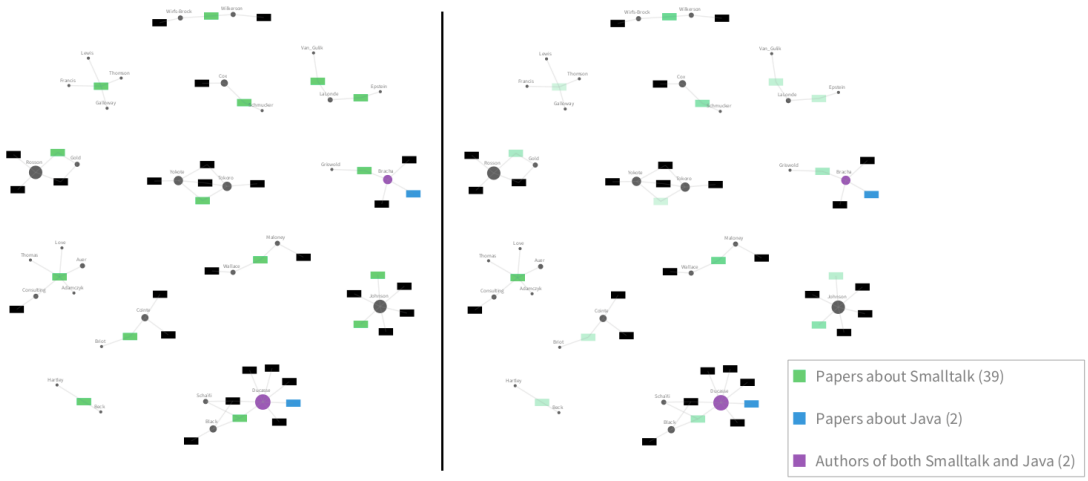

Figure 2.9: Left-hand side: highlighting all authors who have published papers about both *Smalltalk* and *Java*. Right-hand side: in the same graph, encoding into color intensity the number of occurrences of the word *Smalltalk*, in all papers about Smalltalk

2.6.2 Word Cloud

Often when users need to formulate queries to match papers that relate to a certain topic, they struggle to identify keywords. We cope with this problem by designing a companion visualization based on a word cloud. The word cloud is useful to quickly assess the most commonly used words in the content of a set of papers. ESS builds the word cloud by collecting the full text of the whole set of papers. In the word cloud each word from the papers is laid together, and the size of a word corresponds to its frequency in the set of papers. The more frequently a word occurs, the larger it is displayed. Figure 2.18 shows an example built from papers that contain the word *refactoring* in their title.

2.7 Modular Design

We observed that steps 1 and 2 of the ESS pipeline (text extraction and semantic structure recovery described in Section 2.2 and 2.3 respectively) represent the source of most issues. Therefore, we designed these steps in a modular fashion, so they can evolve over time by allowing users to replace them by other tools. Although the current implementation of EES uses PDFBox for text extraction and ParsCit for semantic structure recovery, users could substitute these tools to obtain the benefits of other tools. Figure 2.11 shows the modular design. The only constraint for a text extraction tool is to generate plain text from PDF files. On the other hand, a candidate to replace the semantic structure recovery tool has to extract at least the paper title and author names from a plain text input. In such

Figure 2.10: word cloud from papers whose title contains the word *refactoring*

a case, a user would need to convert the XML results of the semantic structure recovery tool into the ESS schema. For our convenience we adopted ParsCit' schema. Listing 7 shows the required standard schema for the minimal data a semantic structure recovery tool has to provide (*i.e.,* paper title and author names). The schema requires at least a paper title (line 4) and one author name (line 5). Further author names (lines 6-8) are optional.

```
1 <algorithms>
2    <algorithm name="ParsHed">
3       <variant>
4          <title>Paper Title</title>
5          <author>First Author Name</author>
```

```
6           <author>...</author>
7           ...
8           <author>...</author>
9       </variant>
10    </algorithm>
11 </algorithms>
```

Listing 7: required standard XML schema for data extraction results

We propose the XSLT language [4] to transform the results from a particular schema to ESS schema. Next, the user would define a new tool chain by adding a new method to `PDFXMLImporter`, which defines *i*) the use of a text extraction tool and a semantic structure recovery tool, *ii*) their interaction, and *iii*) the conversion of the semantic structure recovery result to the standard XML schema.

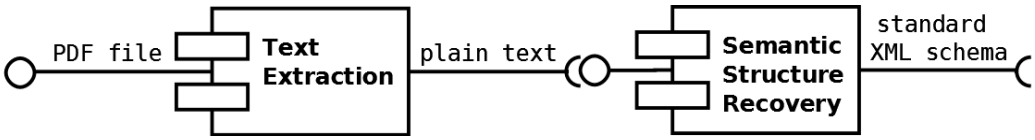

Figure 2.11: modular design of the data extraction

2.8 Evaluation

We evaluate the tool by measuring the accuracy of the model and the time performance. For this, we use the complete set of 366 papers published in the software visualization community (VISSOFT[7]). We demonstrate the tool via selected usage examples on a collection of 1084 papers from the object-oriented, systems, languages and applications community (OOPSLA), published between 1986 and 2013.

2.8.1 Accuracy and Time

We compare the model loaded with EES to the ground truth reference model. The data in the reference model does not include accented characters. We therefore removed accents before comparing paper titles or author names from both sources. For example, we considered the names *Stéphane Ducasse* and *Stephane Ducasse* to be equal. We also ignored whitespace and case. We considered a paper's title to be recovered correctly, if it was equal to the title of its reference model counterpart. Similarly, we considered that an author of a paper in the reference model was correctly recovered if its counterpart in the model loaded with EES contains an author with the same name.

[7] `http://www.vissoft.info/`

Table 2.1 shows a summary with the results. Notice that we did not include the SpecialChar heuristic in the Table, since the reference model does not include accented characters. When ignoring accents, this heuristic does not have an impact on the accuracy of the model. We observe that OddName brings the largest increase in author precision, suggesting that that most common error is incorrectly extracted characters. We find that Blacklist only results in a minor improvement. A possible reason is that this list was tailored training models during development. In the roughly 22% of authors that are still incorrectly modeled after applying all heuristics, we find multiple names with words that could belong on the blacklist. Although MultiAuthor contributes to the author accuracy, we also still find cases where multiple authors were extracted as a single one. We further observe several author names that are extracted incompletely.

Heuristic	Author Accuracy	Title Accuracy
[none]	64.2%	74.6%
OddName	74.6%	76.5%
Blacklist	75%	-
MultiAuthor	76.4%	-
SeparateAccent	77.8%	-

Table 2.1: model accuracy impacts of the cleaning heuristics

We analyzed the time for each step of the pipeline, using the 366 papers of the VISSOFT collection. We ran the analysis in a MacBook Air (2015) with a 1.6 GHz Intel Core i5 CPU and 8 GB RAM. Table 2.2 shows the results of the analysis. We observe that text extraction and semantic structure recovery take particularly long. It is therefore advisable for users to save the data extraction results as XML files, if they want to model the same collection again in the future. Table 2.3 breaks the time consumption of the model cleaning step down to the used model cleaning heuristics.

Step No.	Pipeline Step	Time (seconds)
1	Text Extraction	604
2	Semantic Structure Recovery	3736
3	Modeling	21
4	Model Cleaning	12
5	Visualization (SCGraph, no absent elements)	30
5	Visualization (SCGraph, including absent elements)	595
5	Visualization (SCWordCloud[8])	36

Table 2.2: time consumption of each pipeline step

Heuristic	Time (seconds)
OddName (paper titles)	1.4
OddName (author names)	7
BlackList	0.3
MultiAuthor	0.6
SpecialChar	2.4
SeparateAccent	0.01

Table 2.3: model cleaning time consumption broken down to the separate heuristics

We revisit our research question RQ.1): *What factors influence the accuracy of data extraction and modeling of scientific communities?* We observe that incorrectly extracted characters is the factor that influences the accuracy of paper titles and author names the most. Some of these are incorrectly extracted special characters, others come from marks used to associate authors names with their affiliation. Further errors result from multiple authors being extracted as a single one. While the MultiAuthor heuristic is able to clean some of these errors, others remain in the model. We also find that oftentimes author names contain words that do not belong in a name. This problem can be addressed by defining a blacklist. However, such a blacklist must be tailored to a collection of papers to produce accurate results.

2.8.2 Usage Examples

Example 1: Candidates for Collaboration for Future Projects We work through a fictitious scenario where author *Oscar Nierstrasz* attempts to find possible future co-authors. Listing 8 shows how the user can clean and visualize the OOPSLA collection, which was loaded as shown in Listing 6. This collection is organized in sub-folders for each year, which follow the naming convention *oopsla[year]*.

```
1 CleanupHeuristicsRunner new addDefaultHeuristics runOn: sc.
2 graph := SCGraph new
3     on: sc;
4     title: 'OOPSLA collection SC graph';
5     load.
```

Listing 8: cleaning and visualizing the OOPSLA collection

Oscar Nierstrasz first looks at the entire graph. He uses the search function to search for the node representing him in the graph, shown in Figure 2.12. The search function subsequently zooms in on that part of the community, shown in Figure 2.13. To find

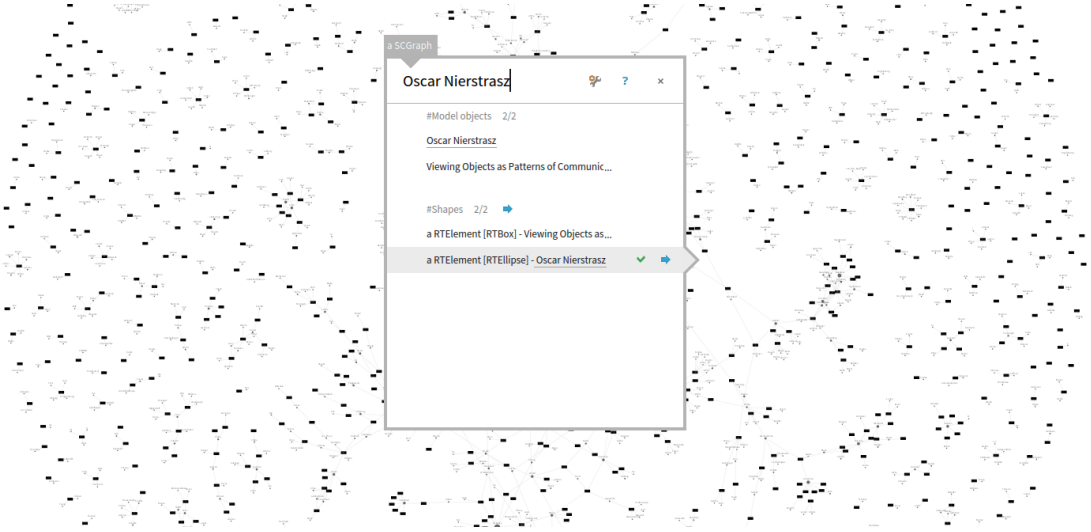

Figure 2.12: searching the graph for *Oscar Nierstrasz*

possible future co-authors, he starts by highlighting relevant papers in his field: object-oriented programming. Consequently, he queries the model for all papers containing the words *programming*, *object-oriented*, or *software* Figure 2.14 shows the results of the query. Highlighted nodes that represent papers that matched the query are in green. He observes that almost all papers matched this query and that the query needs to be refined.

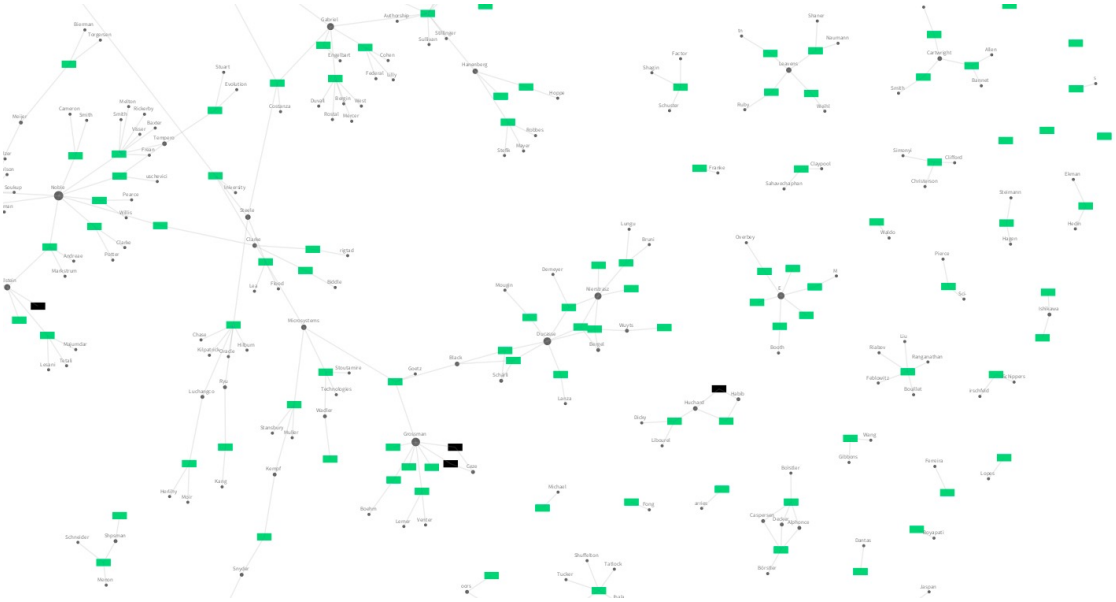

Figure 2.14: too general query, most nodes are highlighted

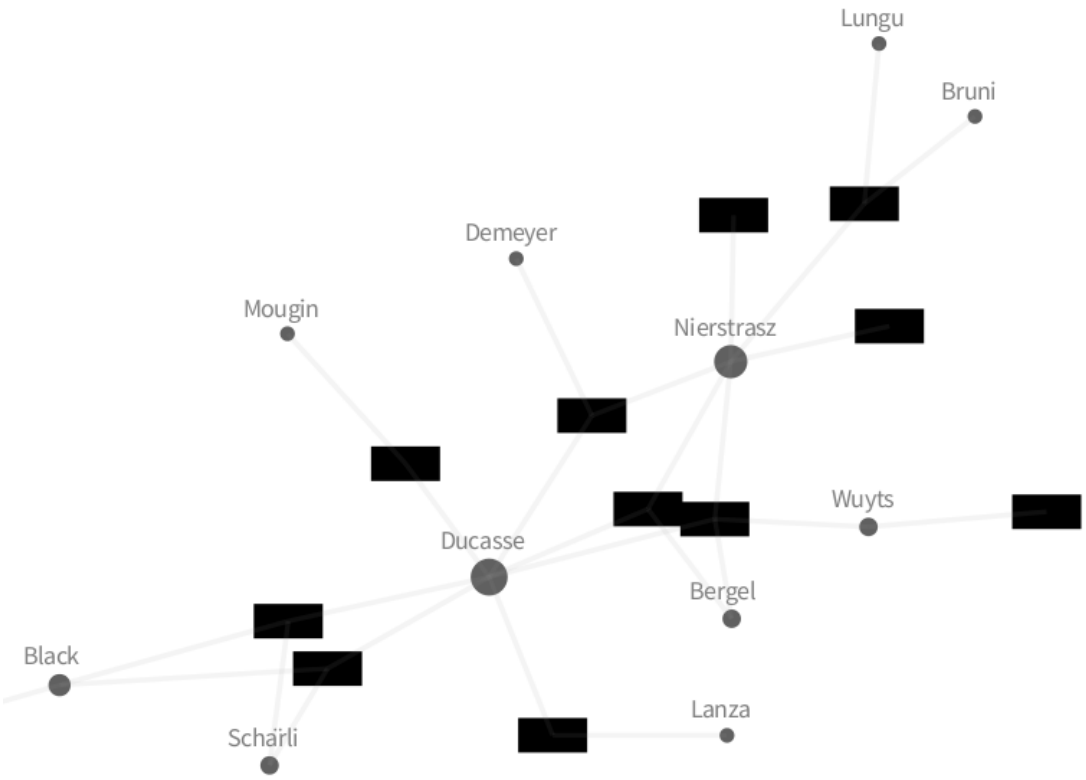

Figure 2.13: close-up on a group of collaborators

Since *refactoring* is one of Oscar's research interests, he poses a new query, to highlight all papers whose title contains the word *refactoring*. Figure 2.15 shows the result of this query in Oscar's closer community.

He subsequently zooms out, to analyze other highlighted nodes. He identifies a different part of the community where a number of papers are highlighted, shown in Figure 2.16. Because numerous authors in this community have published about *refactoring*, Oscar decides that this community might include potential candidates for future collaboration. However, he wants to be sure that these authors have published about refactoring recently. He changes the highlighting of the query results to encode the publication year in the color intensity. The stronger the color, the more recently a paper was published. Figure 2.17 shows the results of this configuration. By selecting a highlighted paper he can analyze when the paper was published.

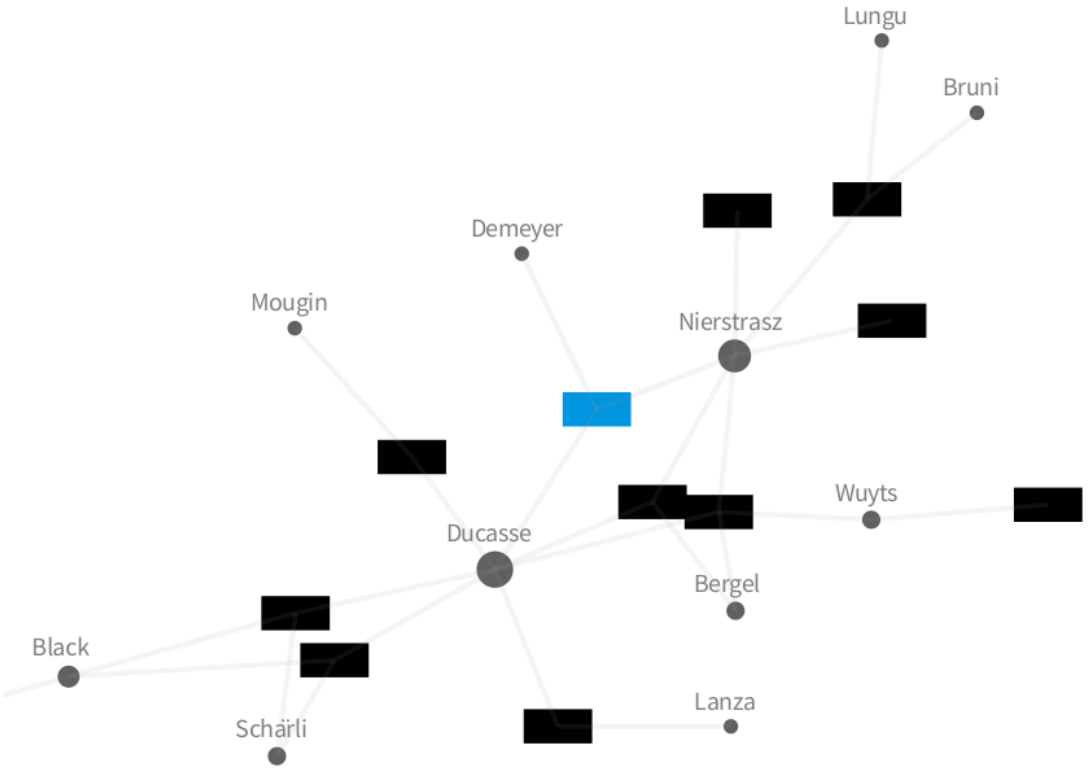

Figure 2.15: one paper matches the query

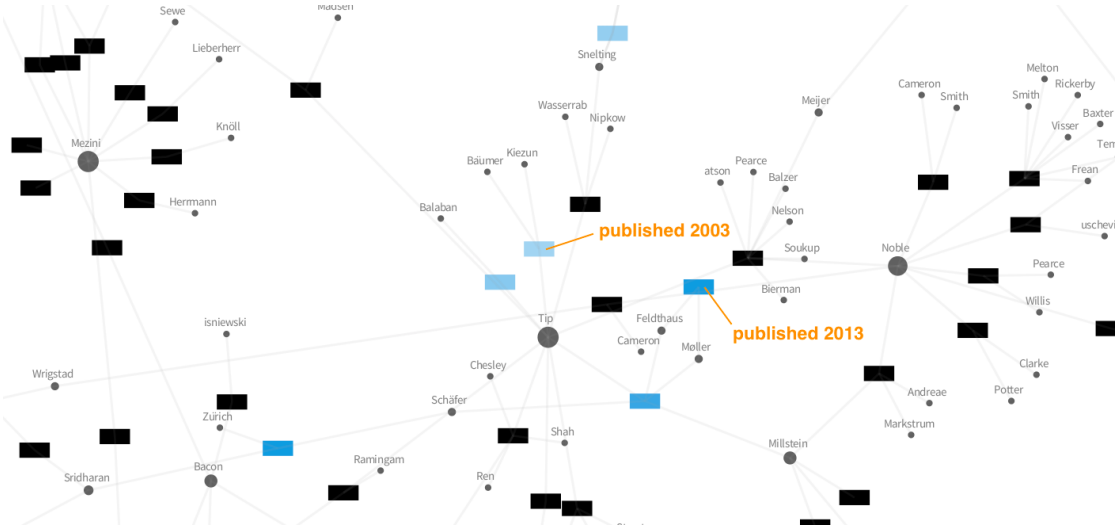

Figure 2.17: matching papers appear stronger, the more recently they were published

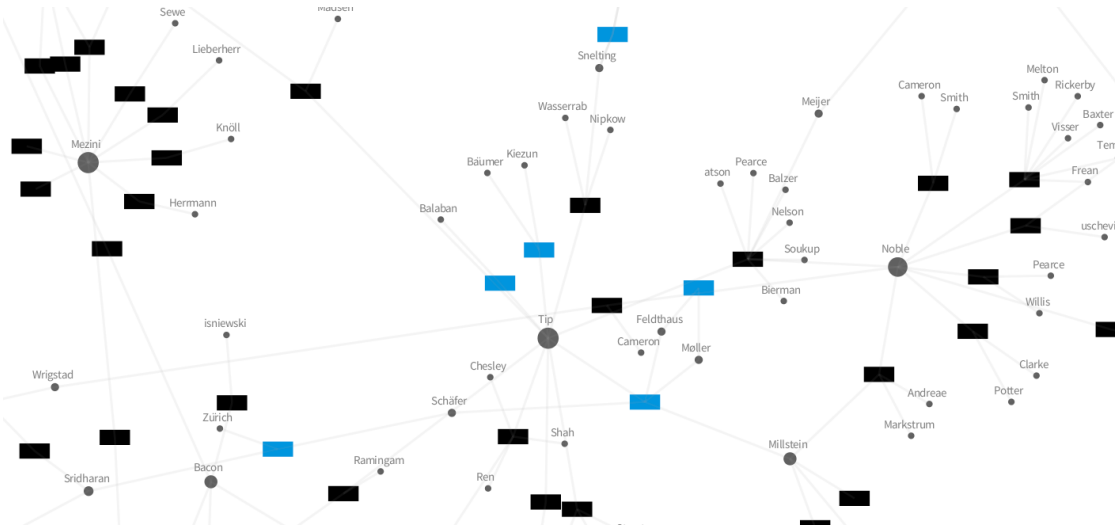

Figure 2.16: a larger group of authors working on *refactoring*

He selects *Anders Møller* and *Asger Feldthaus* as candidates for future collaboration because both have published papers about *refactoring* recently and have collaborated in numerous publications in the OOPSLA community.

Example 2: Active Topics of Research

Oscar assesses what are the main topics of research of these suggested two authors. He queries the model for all papers that belong to these two authors and builds a word cloud from the content of these papers, shown in Figure 2.18. Since these papers are about *refactoring*, this is the most commonly used word. Other prominent words include *javascript*, *renaming* and *related*.

Example 3: Collaboration and Research Topics in Java vs. Smalltalk

We first compare the collaboration among authors of papers that focus on the Java language versus the ones that used Smalltalk dialects: Pharo [2] and Squeak [3]. And then we investigate how active these research topics are.

We query the model for all papers whose title contains *Java*, and for those whose title contains *Smalltalk*, *Pharo*, or *Squeak*. To assess how recent these papers are, we split each of these results into two sub-collections: papers published before the year 2010 and papers published in 2010 or after. We first look at the legend of the resulting graph, shown in Figure 2.19, to understand the color coding. We then study the resulting graph and observe that there are large groups which heavily focused on Java. Figure Figure 2.19 shows such a group. The Figure also shows that this group is still active after the year 2010.

We do the same for authors who used Smalltalk. However, we only find small separate groups of authors. This search shows scalability of the visualization (shown in Figure 2.20). Notice that the different colors of highlighted nodes become harder to

Figure 2.18: word cloud for all papers by a group of possible future co-authors

distinguish on a low zoom level.

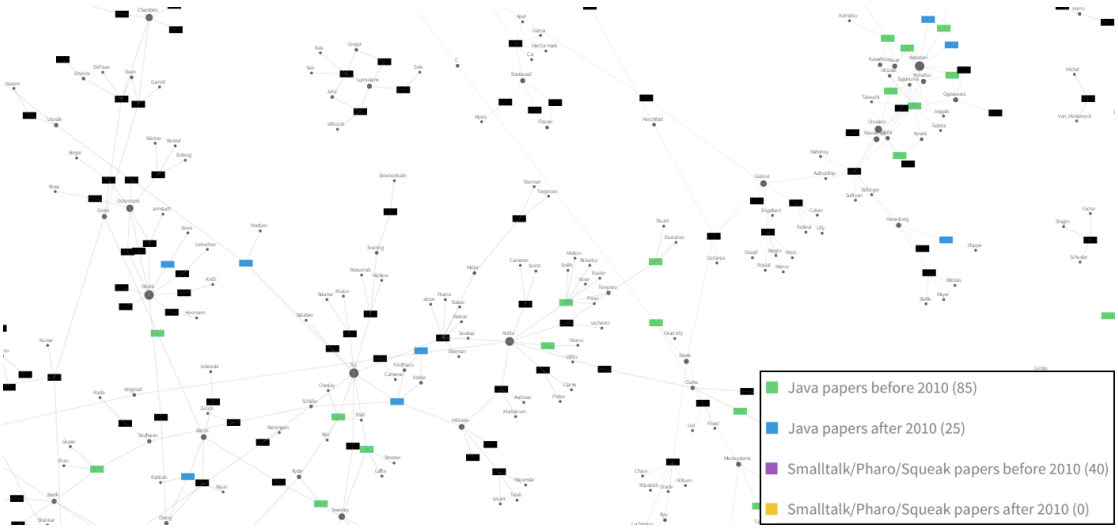

Figure 2.19: a community of authors publishing about Java

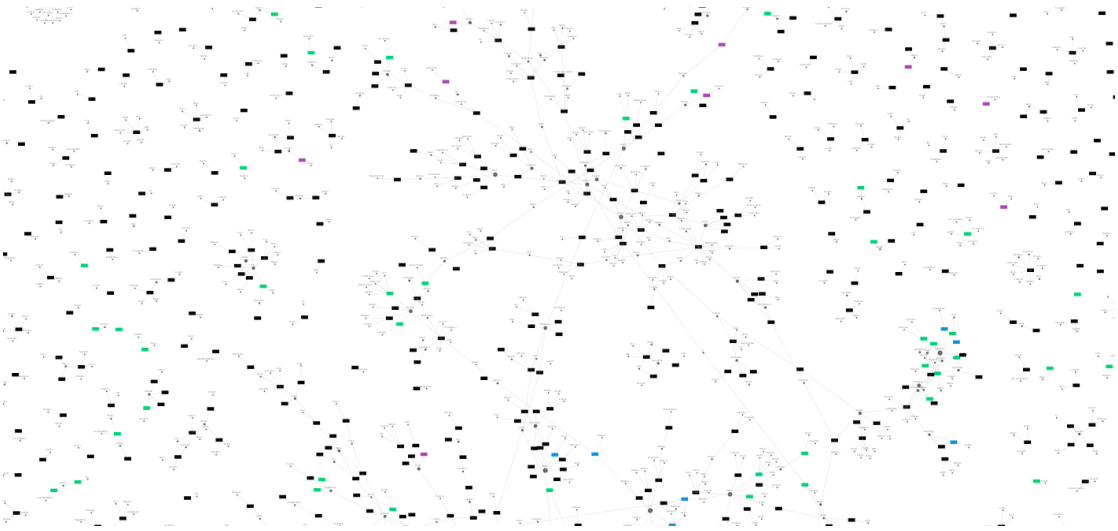

Figure 2.20: limited scalability: large communities require low zoom levels to gain an overview

We notice from the legend (shown in Figure 2.19) that no papers on Smalltalk are published after 2010. We want to find out whether these authors have stopped publishing or have changed their focus of research. However, we observe that the rather small community of Smalltalk authors is difficult to analyze in the large graph. Hence, we decide to create an additional, smaller graph, that only displays the Smalltalk authors and their papers. We query the model for *a*) *all authors who have published papers whose*

title contains the words Smalltalk, Pharo or Squeak; and *b*) *all papers contributed to by these authors*. Listing 9 shows how we instrument EES to visualize such filtered groups.

```
smalltalkPapers :=
    scientificCommunity
        papers
        atTitleSubstrings: #('pharo' 'smalltalk' 'squeak').
smalltalkAuthors := (smalltalkPapers flatCollect: #authors) asSet.
papersByStAuthors :=
    (smalltalkAuthors flatCollect: [ :each |
        scientificCommunity papers atAuthor: each ]) asSet.

stGraph := SCGraph new
    on: scientificCommunity;
    title: 'Smalltalk Authors Graph';
    includeAuthorElements: [ :e | smalltalkAuthors includes: e ];
    includePaperElements: [ :e | papersByStAuthors includes: e ];
    load.

nonStPapers := papersByStAuthors difference: smalltalkPapers
hlFactory := GraphHighlightingFactory new graph: stGraph.
hLFactory
    highlight: smalltalkPapers
    callIt: 'Papers about Pharo/Smalltalk'.
hLFactory
    highlight: (nonStPapers select: [ :e | e year < 2010 ])
    callIt: 'Other papers, before 2010'.
hLFactory
    highlight: (nonStPapers select: [ :e | e year >= 2010 ])
    callIt: 'Other papers, after 2010'.
```

Listing 9: visualizing the isolated community of Smalltak authors

Figure 2.21 shows the graph of the isolated community of Smalltalk authors and their papers. We observe that there are only a few authors who have published several papers of Smalltalk. Most authors have either published only a single paper, or published most of their papers on other topics and incidentally have a paper that mentions Smalltalk. We now build a word cloud based on the paper titles to analyze the topics covered by such papers that do not relate to Smalltalk. The result is shown in Figure 2.22. The most frequently used word in the titles of these papers is *objectoriented*. Other prominent words include *design*, *panel*, *environment*, and *inheritance*.

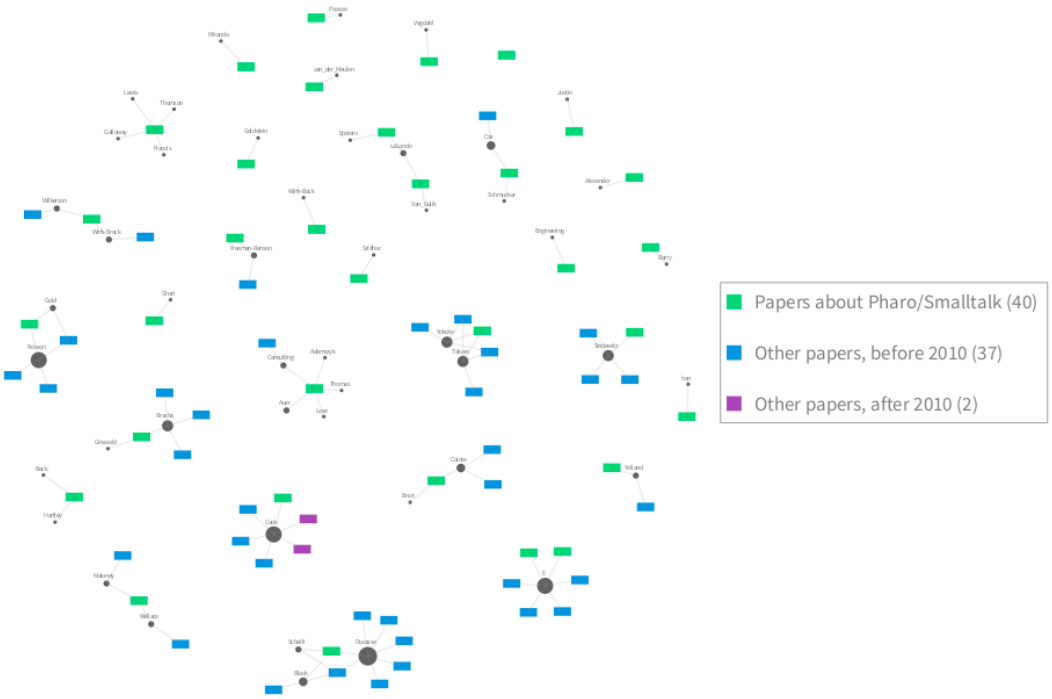

Figure 2.21: graph of the isolated community of Smalltalk authors

Figure 2.22: most frequently used words in the titles of papers not about Smalltalk

Figure 2.21 also confirms what we suspected from analyzing the large graph: most Smalltalk authors are not connected in large communities, but spread out across small, unconnected groups. The figure also shows that there is only a single author who has published after 2010. The other authors have not changed their field of research, but have not published at all since then.

We then analyze what other topics the Smalltalk authors cover by creating a word cloud with the most frequently used words in their papers which are not related to Smalltalk. Figure 2.22 shows the result. Since Smalltalk is an object-oriented language, the most prominent keywords are *object-oriented* and *design*.

We revisit our research question RQ.2): *How can visualization support the understanding of the dynamics of scientific communities?* We found that visualizing scientific communities as bigraphs using node-link diagrams helps users to better understand the collaboration within these communities. Users can gain an overview of the whole community, as well as analyze smaller sub-communities. However, the scalability of this visualization is limited. We also found that the word cloud lets users quickly determine the most frequently used words in a set of papers. In combination, both visualizations can for example help users to find possible future co-authors and determine active fields of research.

3
Conclusion and Future Work

EggShell proposed a workbench for defining pipelines that model scientific communities from collections of papers in PDF format. We built on top of this idea, and defined our own modelling pipeline. Thereby, we relied on the two third-party tools to extract metadata from PDF files. Further, we adapted EggShell's data model of scientific communities and extended it by data such as a paper's paragraphs, citations, and publication year, as well as an author's affiliation and e-mail. We also added methods to the model that facilitate complex queries.

Since both PDFBox and ParseCit make errors, we faced the challenge of misspelled paper titles and ambiguous author names. We developed a model cleaning system, which relies heuristics to tackle such problems. With our data extraction pipeline, combined with these heuristics, we were able to correctly recover 76.5% of all paper titles. On average, 77.8% of a paper's authors were recovered correctly. We designed the model cleaning system in a way that allows users to defined which heuristics should be applied to a model, and to add additional cleaning algorithms to further improve a model.

We visualized the model using the node-link diagram technique. We found that this visualization helps users understand collaboration in a community, and gain knowledge on active fields of research.

For future work we see multiple possibilities to improve both the model and its underlying collection of papers. One particular opportunity to improve the model is matching normalized author names with non-normalized ones, if there is reasonable confidence that they both refer to the same person. Further, it could be possible for ExtendedEggShell to crawl web locations for missing papers, namely referenced papers that are not yet in the respective model's underlying corpus of PDF files. Apart from

improving the model, it would also be useful to automate certain exploration processes. We see large automation potential in the search for related papers for a paper, and for possible future co-authors for an author.

We also see large potential in citations. When appropriately visualized, they might help users find relevant papers, and gain further knowledge on the dynamics of collaboration in a community.

4

Anleitung zu wissenschaftlichen Arbeiten

4.1 How to Install ExtendedEggShell

In this section we describe how to install ExtendedEggShell (EES). The source code of EES is available on SmalltalkHub[1]. Additional resources, such as the required third party tools, can be found on GitHub[2]. Note that EES was developed and tested only on OSX operating systems. This installation guide is therefore only targeted towards Mac users.

1) Download Moose 6.0 from the Moose website[3].

2) verify that the installation path does not contain white-spaces

3) Open the Moose image.

4) Within Moose open a Playground and execute the following code:

```
Gofer new
    package: 'ConfigurationOfExtendedEggShell';
    url: 'http://smalltalkhub.com/mc/SilasBerger/ExtendedEggShell/main'
    username: '' password: '';
    update.
(ConfigurationOfExtendedEggShell project version: '0.1') load
```

This will download and install ExtendedEggShell in the Moose image.

[1] http://smalltalkhub.com/#!/ SilasBerger/ExtendedEggShell
[2] https://github.com/SilasBerger/ExtendedEggShell-tools
[3] http://www.moosetechnology.org/#install

4.1.1 Installing the Additional Tools

Users first need to download the contents of the ExtendedEggShell-tools[4] repository on GitHub and put the contained *tools* folder next to the ExtendedEggShell image. They also need to make sure to have Ruby, Perl and Java installed. In the following, all lines with a leading $ sign need to be executed in the Terminal.
Users then need to install the cpanm utility:

```
$ cpan App::cpanminus
```

Then, they install the required Perl libraries as follows:

```
$ sudo cpanm Class::Struct
$ sudo cpanm Getopt::Long
$ sudo cpanm Getopt::Std
$ sudo cpanm File::Basename
$ sudo cpanm File::Spec
$ sudo cpanm FindBin
$ sudo cpanm HTML::Entities
$ sudo cpanm IO::File
$ sudo cpanm POSIX
$ sudo cpanm XML::Parser
$ sudo cpanm XML::Twig
$ sudo cpanm XML::Writer
$ sudo cpanm XML::Writer::String
```

Then, users navigate to the previously downloaded *tools* folder, then further to /mac/parsecit/crfpp, and issue the following commands, to install CRF++:

```
$ rm -Rf CRF++-0.51
$ tar -xvzf crf++-0.51.tar.gz
$ cd CRF++-0.51
$ ./configure
$ make
$ cp crf_learn crf_test ..
$ cd .libs
$ cp -Rf * ../../.libs
```

4.2 Basic Usage

4.2.1 Modeling a Collection of Papers in PDF Format

To download a set of example papers, users can execute the following line:

```
FileDownloader downloadExamplePDFs
```

[4] https://github.com/SilasBerger/ExtendedEggShell-tools

This creates a folder called *examplePDFs* in the current working directory (*i.e.,* the location of the current Moose image), where the example papers are subsequently downloaded to.

To model a collection of papers in PDF format, users can run the following code:

```
scientificCommunity := ModelLoader new modelPDFs: folderReference
```

The `folderFileReference` needs to hold a `FileReference` to the folder containing the PDF files collection. The reference to the aforementioned examplePDFs folder can be obtained as follows:

```
folderFileReference := FileSystem workingDirectory / 'examplePDFs'
```

Users should then perform the model cleaning step to reduce the number of errors in the model. They can run the default cleaning heuristics as follows:

```
CleanupHeuristicsRunner new
   addDefaultHeuristics
   runOn: scientificCommunity
```

4.2.2 Visualizing a ScientificCommunity model

Users create an graph for a `ScientificCommunity` model as follows.

```
graph := SCGraph new
   on: scientificCommunity;
   title: 'Title of the Graph';
   load.
```

The `scientificCommunity` holds a reference to such a model. To highlight query results in the graph, users need to create a highlighting factory:

```
hlFactory := GraphHighlightingFactory new graph: graph.
```

They can subsequently highlight query results as follows:

```
queryResults := scientificCommunity
   papers
   atTitleSubstring: 'refactoring'.
exampleHL := hlFactory
   highlight: queryResults
   callIt: 'Example Highlighting'.
```

To create a word cloud, a word cloud factory is needed. This factory can then be used to create word clouds from collections of papers:

```
wordCloudView := SCWordCloudView new.
wordCloudView buildCloudFor: aCollectionOfPapers.
```

Bibliography

[1] Vanessa Peña Araya, Alexandre Bergel, Damien Cassou, Stéphane Ducasse, and Jannik Laval. Agile visualization with Roassal. In *Deep Into Pharo*, pages 209–239. Square Bracket Associates, September 2013.

[2] Andrew Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. *Pharo by Example*. Square Bracket Associates, 2009.

[3] Andrew P. Black, Stéphane Ducasse, Oscar Nierstrasz, and Damien Pollet. *Squeak by Example*. Square Bracket Publishing, 2007.

[4] James Clark et al. Xsl transformations (xslt). *World Wide Web Consortium (W3C). URL http://www. w3. org/TR/xslt*, page 103, 1999.

[5] Isaac G Councill, C Lee Giles, and Min-Yen Kan. Parscit: an open-source crf reference string parsing package. In *LREC*, volume 2008, 2008.

[6] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *Proceedings of 15th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '00)*, pages 166–178, New York NY, 2000. ACM Press. Also in ACM SIGPLAN Notices 35 (10).

[7] Leonel Merino, Dominik Seliner, Mohammad Ghafari, and Oscar Nierstrasz. CommunityExplorer: A framework for visualizing collaboration networks. In *Proceedings of International Workshop on Smalltalk Technologies (IWST 2016)*, pages 2:1–2:9, 2016.

[8] Oscar Nierstrasz. Agile software assessment with Moose. *SIGSOFT Softw. Eng. Notes*, 37(3):1–5, May 2012.

[9] Dominik Seliner. EggShell — a workbench for modeling scientific communities. Bachelor's thesis, University of Bern, August 2016.

[10] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.