Analyzing and Visualizing Projects and their Relations in Software Ecosystems

Van Tuan Tran[1], Cheng Cheng[1], Fabio Petrillo[2], and Yann-Gaël Guéhéneuc[1]

[1]Concordia University, Montréal, Québec, Canada
[2]Université du Québec à Chicoutimi, Québec, Canada

***Abstract*—More and more software projects are being consolidated into ecosystems to increase their discovery, usability, and usefulness. Some of the most popular ecosystems exist in npmjs, Python Package Indexing, and Apache Maven Repository. It is difficult for developers to relate these projects and use them to their full potential because of their number, the spread and depth of their features, and their intrinsic and accidental complexities. We present a technique—SECO Storms Maker—to capture and present the essential information from projects in an ecosystem to help developers join, use, and contribute to the ecosystem. We generate word-clouds based on the projects' documentation via tokenization and distribution frequency. We identify relations among projects using grammar patterns scanning after part-of-speech tagging. We put these word-clouds into a graph to ease navigation and exploration. We evaluate our technique by manually building a ground truth and comparing a randomly-selected project with SECO to show its benefits.**

I. Introduction

Software Ecosystems (SECOs) have been expanding in recent years. For example, the number of packages in the NuGet package manager went from about 3,600 packages in 2011 to more than 220,000 packages in 2020.

The growth of SECOs in number and size is due to the many developments in various application domains, from personal assistants to self-driving cars, and the open-source movement. It is a boon for software developers, who can choose and use many ecosystems, but also a curse: developers struggle to identify the "best" ecosystems and, in it, the projects that could fulfill their requirements, i.e., the Paradox of Choice.

Indeed, Loewenstein argued that more choices is not always better [1]. If the choices promote sane competition among providers, then developers, projects, companies, and ultimately users, benefit. However, too many choices hide sustainable libraries and frameworks, fragment development, yield the "bandwagon effect"[1]. **Hence, developers need help to identify, relate, compare, and contrast SECOs.**

Previous work [2, 3] discussed the information needed by developers when exploring SECOs: (1) a standard visualization of the SECOs, (2) compatibility with other systems or software, and (3) comparisons among the projects in the SECOs. Merino *et al.* [4] discussed the importance of visualization and multiple methods to visualize software but these are not directly applicable to SECOs. Indeed, developers often only consider the SECO documentation and that of its projects. **Dedicated visualizations for SECOs must help developers in exploring, comparing, and choosing a SECO and projects in a SECO using available documentation.**

For example, the Internet of Things (IoT) has dramatically expanded in recent years with many different software projects, from run-time libraries to communication protocols to databases running on Cloud servers, which form many competing but overlapping SECOs, i.e., from Google Cloud IoT to Bosch IoT Suite. In particular, the Eclipse IoT SECO[2] covers many IoT developers' needs, sometimes with different solutions to similar problems, e.g., CoAP vs. MQTT. Thus, (new) developers must spend time and effort studying, understanding, and relating, via their documentation, the various projects available in the Eclipse IoT SECO before using them. **We devise a visualization technique to show and contrast software projects in/across SECOs to help developers.**

There are multiple ways to visualize textual data from the documentation. While bubble charts or treemaps may not apply, word clouds (aka tag clouds) are more appropriate, especially when a metric can be associated with each word, e.g., importance, frequency, or distance. They were used in multiple software-engineering research work [5, 6] and in practice. e.g., to compare the USA presidential candidates' speeches in 2008. Barth *et al.* [7] stated that word clouds are a standard tool for abstracting, visualizing, and comparing text documents.

We propose an approach, SECO Storms Maker, to generate a word-cloud-based visualization of a SECO from its documentation and that of its projects. We describe a set of grammar rules for analyzing any SECOs when their projects implement or use predefined standards/protocols. We apply our approach on the Eclipse IoT SECO.

Section II summarises the related works. Sections III, IV and V describe our approach, implementation and application on the Eclipse IoT SECO. Sections VI and VII evaluate our approach and its limitations. Finally, Section VIII concludes.

II. Related Work

Text Analysis Techniques: Manning *et al.* [8] designed and developed a more accurate Natural Language Processing (NLP) toolkit than the NLTK toolkit. It is widely used by the NLP research community, businesses, and governments. We use the PoS annotators for tagging the words in sentences.

[1]https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/ [2]https://iot.eclipse.org/projects/

Previous Visualizations of Words: Shaw *et al.* [9] displayed words in connected graphs, where words are nodes and edges are the similarities between words. The length of the edges is based on the word metadata, such as their distributions, their relations, etc. We adopt the idea of connected graphs with edges showing the similarities among nodes. The nodes in our visualization are the projects in a SECO.

There are other ways to visualize words. Dubinko *et al.* [10] arranged word in a timeline while Wattenberg *et al.* [11] assigned colors to each word that carries an activity and displayed them in a Chromogram graph to show patterns.

Seifert *et al.* [12] proposed multiple techniques for creating word cloud where font sizes depend on the importance of words. The most appropriate technique minimizes redundant white space, arranges the words "nicely" in a predefined space, but takes more time. We use their Shift-Scale-Trunc (SCT) algorithm, without the "Truncation" part.

Previous Visualizations of SECOs: Lungu *et al.* [13] presented the Small Project Observatory (SPO) to build interactive visualizations of SECOs, using projects metadata, like sizes, activities, developers' activities. Their visualizations can show inter-project dependencies using a graph of the projects.

Castella *et al.* [14] introduced *word storms* as groups of word clouds where related words have the same color, size, and position. We adopt this idea with a different approach and application to visualize and compare projects in SECOs.

III. Approach

Often, many projects solve intersecting subsets of the same functional requirements, providing other features, with different non-functional quality. Developers must read all the projects' documentation to know if one fits their needs, which is time and effort-consuming and uncertain. **We believe that a graph with word cloud as nodes could help developers in identifying, evaluating, and choosing projects within a SECO.** Indeed, they would provide (1) word cloud for each project, summarizing their main topics, and (2) relations among projects, showing complementary, substitutable, and incompatible projects.

Our SECO Storms Maker divides into four steps:

Step 1 Collecting Data. Using Scrapy[3] to crawl text from the projects' homepage and documentations.

Step 2 Finding keywords, generating word clouds. Using Python NLTK[4] library to analyze sentences and words and Wordcloud[5] to generate the clouds from the words.

Step 3 Finding relationships between projects, using PoS tagging and rule-based grammatical scans of the sentences.

Figure 1 shows the resulting graphs for the Eclipse IoT SECO. Blue nodes are standards/protocols. Word-cloud nodes are the projects. We use the projects' names as shapes of the nodes because a project's name usually has the highest frequency, would take most space inside the node, but with little information and comparability. Edges are relationships: with red for "interchangeable" relationships and blue for "use". A legend assists first-time viewers. Each step presents implementation challenges, which we discuss in the next section.

[3]https://scrapy.org/

[4]https://www.nltk.org/

[5]https://github.com/amueller/word_cloud

IV. Implementation

A. Collecting Data

We first need a list of all the projects in a SECO, which we can build in multiple ways. Besides manually adding each project, we can use a SECO public API or some JavaScript code to analyze a SECO homepage.

We use Scrapy, an open-source and collaborative framework for extracting data from some projects' homepages/documentation. Scrapy supports multi-threading well, compared to Selenium or BeautifulSoup, for performance.

From the list of projects' homepages/documentation, we create a JSON file containing the following fields for each project: `IsCrawled`, `CrawlDepthLevel`, `IsWordcloudGenerated`, `SiteUrl`, `ProjectName`.

Using this JSON file, we can track which project is crawled, on how many levels, and if that project's word-cloud is generated. The results of this step are text files containing all texts and links originating from the projects' homepages.

B. Finding Keywords and Generating Word Clouds

We use the Python NLTK library to analyze the texts. For each project, the input is its text file from Step 1; its output is a list of keywords and their frequencies. The NLTK library maintains a list of stop words to remove them from the output. The resulting lists contain words of different types (verb, noun, adjective, etc.). We use the Porter algorithm for stemming words [15]. The stemming does not always create meaningful words usable in word clouds. (For example, the word "variable" is stemmed to "variabl".) We maintain a link to their original forms while stemming words. We also keep a count of the original forms of the stemmed words.

a) Generating Word Clouds: Word clouds have properties that can create very different looks: (1) the shapes of the clouds and (2) the sizes of the words. We observed that the names of the projects are constantly repeated, which takes valuable space for little information. Thus, we use the projects' names as shapes of the word clouds.

The space between characters of the project's name should be negative, so the characters overlap and create a connected space, which is filled using words. For each project, we display the top 50 most frequent words. Given a frequent stemmed word to display in a word cloud, we show its related, most frequent unstemmed word, as shown in Figure 2.

With the Python Wordcloud library[6], we create the word clouds whose shapes are the names of the projects and content are the more common words in decreasing sizes.

C. Finding Relationships between Projects

We split this step into four sub-steps:

1) Identify the possible relationships between projects.
2) Find sentences containing the relationship keywords.

[6]https://github.com/amueller/word_cloud

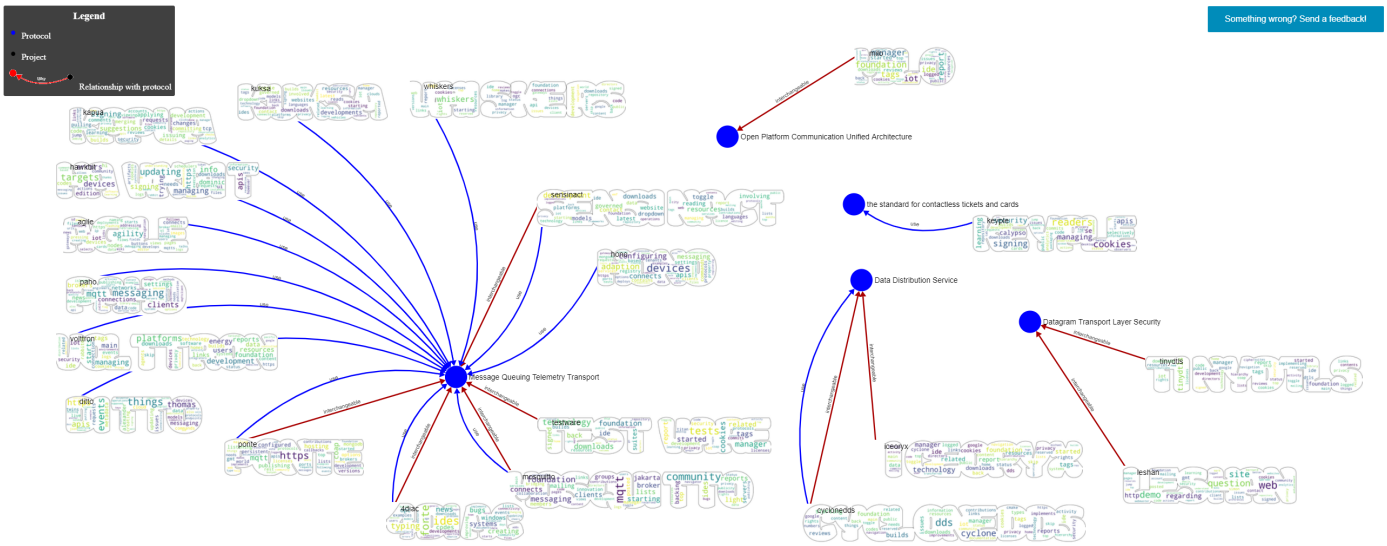

Fig. 1. The graph of Eclipse IoT projects' relationships

Fig. 2. The generated word-cloud for Keyple project

3) Tag sentences using Part-of-Speech (PoS).
4) Use grammar patterns to scan the result.
5) Add matches to the relationship list.

a) Identifying Relationships: Manikas *et al.* [16] defined the symbiotic relationship between projects in a SECO, inspired by natural ecosystems, which we call the "Communicable" relationship: two projects can communicate with each other via the same protocols. Basole *et al.* [17] described the "Dependency" relationship: one project depends on the other. Batory *et al.* [18] introduced the "Interchangeable" relationship: the projects can replace each other. Users can use any of these projects to get the same features/behavior.

We add the "Other" relationship to describe relationships that only exist in a specific SECO. For example, in the Eclipse IoT SECO, some projects implement the same standards but provide different features. In every SECO, these relationships are different and must be identified manually. We cannot detect such relationships automatically, as illustrated in Section V.

b) Finding Relationship Keywords: Once we know the possible relationships between projects in a SECO, we can detect their occurrences based on the sentences in the documentation of each project. For example, a sentence could read "Project A depends on project B".

c) Tagging Words and Building Grammar Patterns: We then can construct a list of grammar rules that match sentences describing relationships. The grammar follows a set of predefined rules for each type of relationship. If a sentence satisfies a rule, we add the relationship type between the projects. For example, one rule for "Communicable" is:

Rule 1: if two projects support the same protocol, they can communicate with each other.

The grammar rules can be straightforward. For example, the pattern for the sentence "Project A depend on project B" is: `Noun + depend + preposition + Noun`.

We store the detected relationships between projects in a JSON file for visualization on the Web.

V. Case Study: Eclipse IoT SECO

We apply our approach on the Eclipse IoT SECO. Tool and data are at https://github.com/huntertran/seco-storms-maker.

A. Collecting Data

We extract a list of projects' homepages from http://iot.eclipse.org. While crawling, we see that the AGILE IoT project's homepage at http://agile-iot.eu/, contains little information and links, so we increase the crawling level to 2.

B. Finding Keywords and Generating Word Clouds

Besides the list of common English stop words, we create an additional list specifically for IoT SECO. We then apply the steps in Section IV-B to break the texts into keywords and generate individual word clouds for each project.

C. Finding Relationships between Projects

1) Identifying Types of Relationship: We observed that no project has the "Dependency" relationship, possibly because the number of projects is relatively small, compared to other SECOs like NuGet. For the "Interchangeable", "Communicable", and IoT-specific "Standard Implementation" relationships, we use a list of predefined protocols and standards, which must be implemented/supported by IoT projects to communicate with one another. We treat these protocols/standards as relationship keywords as described in Section IV-B.

To identify these protocols/standards, we read the project documentation and FAQ and manually identify abbreviations. We share these keywords/abbreviations at https://git.io/JnC4f.

Using this list of standards/protocols, we identify the relationships among projects. For example, projects that support MQTT are interchangeable. We process the crawled text files of each project to obtain all sentences describing the relationships among projects, tagged with PoS tagging.

2) Identifying Projects Relationships: Based on the list of keywords, we draw relationships among the projects with a simple ruleset for Eclipse IoT projects: (1) if two projects implement the same protocol, they are interchangeable, and (2) if two projects support the same protocol, they can communicate.

We build grammar patterns common in English to describe the meaning of an IoT project related to the keywords, available at https://git.io/JnCEf.

The resulting visualization is illustrated in Figure 1 and available at https://huntertran.github.io/seco-storms-maker/. The website contains visualizations for other SECOs. It uses the SigmaJS library[7] to draw the relationships. Users can zoom and drag the node to move it.

VI. Evaluation

To assess the visualization of the case study, we randomly choose a project and perform a manual analysis to build a ground truth. Then, we compare the result of this manual analysis with that of our approach. We choose Keyple with the parameters: (1) project homepage: https://projects.eclipse.org/projects/iot.keyple and (2) CrawlDepthLevel: 1. We define the following assessment criteria: (1) words in the word cloud, (2) frequency of the words, and (3) project's relationship to implemented/used protocols.

We use the URLs in the project homepage for a deeper analysis. The URL must contain the project homepage to be valid for further process. Then, we do a search for each word/phrase to see how many times they are used on the current page. We add this number to a list. We skip unimportant words based on our understanding of the project. See https://git.io/JnClh for the list of words.

To identify the relationships of Keyple with other projects, we read and pick all sentences in the crawled text that clearly define the capability of Keyple:

- *The Keyple Calypso API provides a high-level interface to implement fast and secure contactless ticketing transactions based on the Calypso standard.*—Keyple only provides interfaces to implement the Calypso standard.
- *Scope: Eclipse Keyple provides generic librairies for simplifying the development of contactless applications based on the Calypso standard, and for facilitating integration with the secure elements typically involved in a secure contactless solution.*—The sentence claimed that Keyple provides libraries to develop contactless applications but does not say which standards it supports.
- *This is the repository for the Java implementation of 'Calypso API' for the 'Eclipse Keyple' project.*—Keyple has a repository with source code that implement Calypso standards and usage examples.
- *Description: The goal of Eclipse Keyple is to allow developers to easily implement fast and secure off-line contactless transactions (using NFC cards, mobile phones, ...) based on the Calypso standard.*—Keyple allows users to implement Calypso standard quickly

Base on these sentences, we concluded that Keyple *uses* the Calypso standard, as was concluded by our approach. Comparing the ground truth with the result generated from our automation tool, we observed that:

- Some words do not appear in the tool result (ticketing, terminal) because they have low frequencies while important in the project.
- There are words that should not appear in the result of the automation process like `go`, `tags`, `back`. These words, while meaningless for the Keyple, could bear meaning in other projects. For example, `go` appears in the sentence `Go back` but, in some other project, it could be the Go programming language.
- The relationships drawn from manual inspection and automation tool are the same.

VII. Discussion and Threats to Validity

While our approach could generate word clouds for the projects in the Eclipse IoT SECO and their relationships, it has some limitations, which reduce its usefulness.

A. Discussion

Compared to the "Small Project Observatory", which shows the dependencies between projects, our tool shows if projects can interchange and communicate with each other via a specific protocol. Compared to the "Word Storms", which synchronize color, size, and drawing position of the same word in different word clouds, our tool presents a different approach, showing the "interchangeable" and "communicate able" relationships of the projects in a SECO. However, our approach depends heavily on the quality of the documentation. But it could be used to assess the documentation's quality.

B. Internal threats

1) Bias in the Manual Evaluation: Because we design the tool, we know how it works. This knowledge could create the wrong inception while doing the manual inspection on the selected project to build the ground truth. To address this problem, on our visualization website, we put a simple feedback form. Any researchers or users could use that form to report a problem back to us.

2) Word, Not Word Phrase: The automation code counts each word in the text separately. For example, the phrase *smart card* in Keyple mentioning the NFC cards. Each word in the phrase has meaning when standing alone and may have other meanings in other projects. To address this problem, we can include a list of phrases for each project in future work. The tool can check this list when counting words.

[7]http://sigmajs.org/

C. External threats

1) Abbreviation: In the Keyple project, the word `Secure Element` was abbreviated to `SE` and used multiple times. This is specific hardware on Android devices that store cryptographic data. The word `SE` also appeared in the phrase `Java SE`, which is a version of Java. For other abbreviation words, they can have other meanings or not an abbreviation in other projects. To address this problem, in future work, we can use the grammar rule with Part-of-Speech tagging as described in section IV-C, with some modifications. The rule must be customized for each project.

2) Words Without Meaning in a Project: When comparing the list of words generated by automation code and the list by manual inspection of Keyple project, we found some words that have no meaning in the context of the project, such as `learning`, `downloads`, `tags`, `desktop`. These words may have meanings in other projects. Therefore, we cannot add them to the stop words list to be excluded when counting.

3) Detection of Project Relationships: In our case study, we rely on a list of standards–protocols to identify the relationships between each project. We compose this list manually, as there is no efficient and accurate way to detect or extract them from the text. Researchers need to extract this list once for each SECO domain.

VIII. Conclusion

In this paper, we proposed an approach to analyze and visualize the contents and relations of software projects in an ecosystem. The result is a word-cloud for each project, visualized in a graph with relations to the standard or protocol that the project supports/implements. We used the projects in the Eclipse IoT ecosystem as a case study. We verified the results by conducting a manual inspection on a randomly chosen project and showed that our approach could indeed summarise and relate projects in the ecosystem.

In the future, we could evaluate the tool more thoroughly by interviewing developers. Another improvement is allowing the project's owner and maintainer to modify the generated word-cloud and the relationship to archive the highest accuracy, combined with the automation approach.

References

[1] George Loewenstein. Is more choice always better. *Social Security Brief*, 7(1):7, 1999.

[2] Alexander Serebrenik and Tom Mens. Challenges in software ecosystems research. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, pages 1–6, 2015.

[3] Nicole Haenni, Mircea Lungu, Niko Schwarz, and Oscar Nierstrasz. Categorizing developer information needs in software ecosystems. In *Proceedings of the 2013 international workshop on ecosystem architectures*, pages 1–5, 2013.

[4] Leonel Merino, Mohammad Ghafari, and Oscar Nierstrasz. Towards actionable visualization for software developers. *Journal of software: evolution and process*, 30(2):e1923, 2018.

[5] Gillian J Greene and Bernd Fischer. Interactive tag cloud visualization of software version control repositories. In *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, pages 56–65. IEEE, 2015.

[6] Rylan Cottrell, Brina Goyette, Reid Holmes, Robert J Walker, and Jorg Denzinger. Compare and contrast: Visual exploration of source code examples. In *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 29–32. IEEE, 2009.

[7] Lukas Barth, Stephen G Kobourov, and Sergey Pupyrev. Experimental comparison of semantic word clouds. In *International Symposium on Experimental Algorithms*, pages 247–258. Springer, 2014.

[8] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[9] Blake Shaw. Utilizing folksonomy: Similarity metadata from the del. icio. us system. *Project Proposal, December*, 2005.

[10] Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Visualizing tags over time. *ACM Transactions on the Web (TWEB)*, 1(2):7–es, 2007.

[11] Martin Wattenberg, Fernanda B Viégas, and Katherine Hollenbach. Visualizing activity on wikipedia with chromograms. In *IFIP Conference on Human-Computer Interaction*, pages 272–287. Springer, 2007.

[12] Christin Seifert, Barbara Kump, Wolfgang Kienreich, Gisela Granitzer, and Michael Granitzer. On the beauty and usability of tag clouds. In *2008 12th International Conference Information Visualisation*, pages 17–25. IEEE, 2008.

[13] Mircea Lungu, Michele Lanza, Tudor Gîrba, and Romain Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, 75(4):264–275, 2010.

[14] Quim Castella and Charles Sutton. Word storms: Multiples of word clouds for visual comparison of documents. In *Proceedings of the 23rd international conference on World wide web*, pages 665–676, 2014.

[15] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[16] Konstantinos Manikas and Klaus Marius Hansen. Characterizing the danish telemedicine ecosystem: Making sense of actor relationships. In *Proceedings of the Fifth International Conference on Management of Emergent Digital Ecosystems*, pages 211–218, 2013.

[17] Rahul C Basole. Visualization of interfirm relations in a converging mobile ecosystem. *Journal of information Technology*, 24(2):144–159, 2009.

[18] Don Batory and Sean O'malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1(4):355–398, 1992.