The Socinian Correspondence: A Graph-Based Digital Scholarly Edition

Patrick Toschka[1] Julian Jarosch[1]

Andreas Kuczera[2]

[1] Academy of Sciences and Literature | Mainz
Mainz, Germany
[2] University of Applied Sciences (THM), Gießen, Germany

Abstract

The Socinian Correspondence Project uses a graph database as a leading persistent data layer to create a digital scholarly edition (DSE) of letters written by and to followers of Socinianism, a Unitarian belief system founded by Lelio and Fausto Sozzini in the sixteenth century. In addition to more conventional elements, such as transcribed texts and an accompanying critical commentary, our DSE will allow scholars to freely explore the collected data in any time-, place-, or content-based context, without the impediment of preset categories. In order to achieve this goal, we are in the process of developing two generic open source web applications that follow the current software standards and form the backbone of our innovative approach.

In: Tara Andrews, Franziska Diehr, Thomas Efer, Andreas Kuczera and Joris van Zundert (eds.): Graph Technologies in the Humanities - Proceedings 2020, published at http://ceur-ws.org.

1 Introduction

1.1 The Socinians and their Correspondence Network

The Socinians were a Unitarian religious community of protestant origin that emerged in the sixteenth century and considered the subjective rationality of the individual as the determining principle in matters of religious faith. Socinians could be found all across Europe (Daugirdas, 2016, p. 11) and were actively engaged in developing a close-knit correspondence network with leading figures in theology (e.g. Remonstrants like Philipp van Limborch and Johann Jakob Wettstein), astronomy (e.g. Johannes Hevelius and Johannes Kepler), and politics (e.g. Magnus Gabriel De la Gardie and Carl Gustav Wrangel). Surviving letters of the adherents of Socinianism are an important resource for anyone interested in the transconfessional struggle that defined Europe in the Early Modern period, especially with regards to the latter's efforts to strike a balance between theology, science, and politics. Socinian theology introduced an air of radical tolerance into contemporaneous debates on such issues by affording its practitioners the freedom to engage in evidence-based scientific discussions unhindered by traditional metaphysical considerations.

Dedicated to the period between roughly 1580 and 1740, the DSE prepared by the Socinian Correspondence Project[1] encompasses the time-frame from the Age of Confessionalism to the Enlightenment. The 2047 letters cataloged to date were addressed to other Socinians or to scholars and politicians of other confessions, and reflect the complex interdependencies between historicizing, rationalist approaches to religious subject matter, early modern scientific research and its methodology, and political correspondence. These letters also provide valuable insight into the wider scholarly networks that spanned across Europe during the period under investigation.

Currently, no comprehensive historical-critical edition of the Socinian correspondence exists. Our project aims to rectify this situation by identifying letters in which at least one correspondent is a Socinian and by gathering the said letters into a DSE. Featuring textual-critical commentaries and descriptions of the subject matter and content of the letters, our graph-based edition (Kuczera, 2016) will also enable analysis of the material using criteria such as geography, persons involved, and topics discussed by the correspondents through a systematic indexing of the text. The DSE also incorporates images of astronomical phenomena that were enclosed with the letters and were often published in the form of copperplate engravings, as they help to il-

[1] "Die sozinianischen Briefwechsel," URL: http://www.adwmainz.de/projekte/zwischen-theologie-fruehmoderner-naturwissenschaft-und-politischer-korrespondenz-die-sozinianischen-briefwechsel/informationen.html. (https://sozinianer.de)

lustrate the subject matter discussed in the corresponding letters. Previously unpublished political reports, which accompanied more than 250 letters, are likewise included.

It is our hope that by making the primary sources and supplementary materials accessible in the manner described above, our edition will enable researchers to gain new insights into interdisciplinary fields of research, such as theological history, philosophical history, the history of science, and cultural history. As the Socinian correspondence network is one that traverses boundaries of religion, geography, subject matter, and media, we believe that the DSE itself should reflect this pervasive interconnectivity (Figure 1).

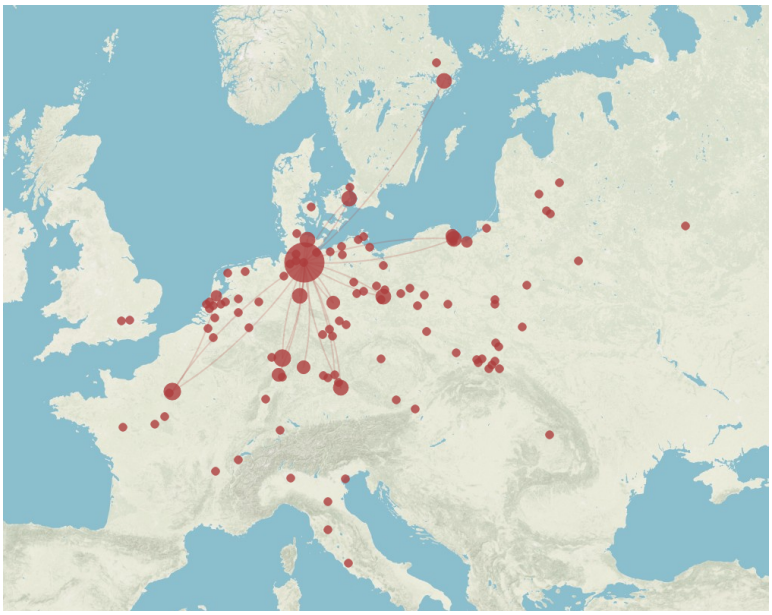

Figure 1: Locations from where Socinian letters were sent and their corresponding destinations. Dot size represents number of letters. Created using Palladio `http://hdlab.stanford.edu/palladio/`.

1.2 The Digital Scholarly Edition

The development of the Digital Scholarly Edition (DSE) follows the current software standards (Schrade, 2018). So the text component uses a classical XML-based format and draws on established technologies for editing and publication.

The letters and their accompanying materials are encoded in a subset of TEI P5 XML, the DTA-Basisformat (Haaf et al., 2014), which offers an unambiguous schema for encoding. The editing work is carried out in ediarum, a framework for the Oxygen XML editor (Dumont and Fechner, 2014), and

is stored in an eXist XML database.

Correspondence metadata is captured in the TEI element <corresp Desc> (Stadler et al., 2016), while the text itself is annotated (e.g. with clarifications concerning dates, abbreviations, etc.) and furnished with a critical commentary. Furthermore, the letters are linked to indexes of persons, places, concepts, things, and astronomical entities, which were first created as lists in XML in order to begin the editing work as early as possible, while the graph component of the DSE continues to be developed.

The DSE's publication frontend is realized with the TYPO3 Open Source Content Management System[2] and various extensions[3] needed for the implementation of Linked Open Data (LOD) standards[4] and the creation of the framework required for scientific publication. This includes tasks such as versioning and the assignment of stable Uniform Resource Identifiers (URIs) that enable the citation of scholarly work.

These components of the DSE make use of current technologies and best practices based on the tree model of XML and the relational database underlying TYPO3. Neither of these technologies fully account for the distinctive network character of this edition's subject, however, which is why the DSE will be augmented by a graph component.

1.3 The Graph Component

Our project aims to adequately model the closely interlinked Socinian correspondence network, and to break new ground in bringing XML and graph technologies together by using a graph database as the leading persistent system for one component of the DSE, alongside and connected to the XML text data.

In order to achieve this goal, it is necessary to complete two major tasks: first, the data that currently resides in XML must be transferred into a neo4j graph database; second, the graph database must be given a user-friendly interface that can be edited by historians. As can be seen in Figure 2, both tasks are to be completed using a generic, open source web application: eXGraphs for the extraction of XML data into a graph, and GRACE for the subsequent editing of the graph.

[2] URL: https://typo3.org/.

[3] E.g. TYPO3 Extension Beaconizer, URL: https://extensions.typo3.org/extension/beaconizer/.

[4] As presented on the TYPO3 University Days 2019, Slides: https://github.com/digicademy/2019_typo3_goes_lod.

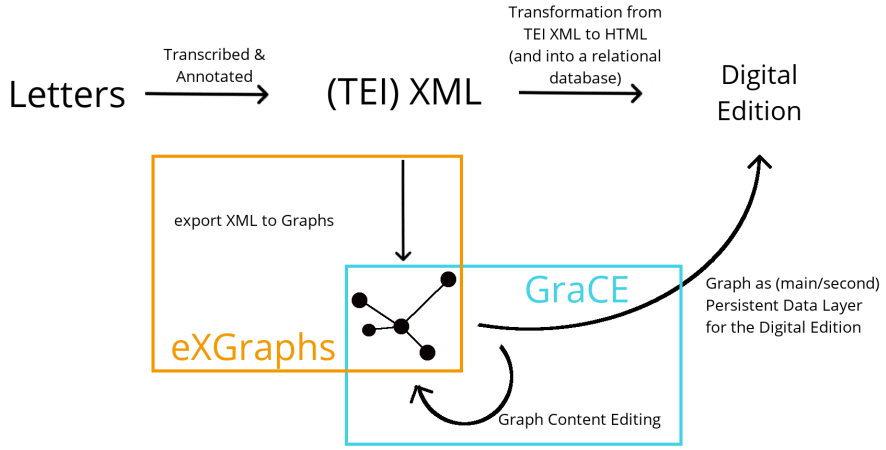

Figure 2: Abstract workflow for an XML-based digital edition using our tools

2 The Tools

2.1 Step 1: Generating a Graph from XML Data

As noted above, we use eXGraphs[5] to extract the existing XML data from its XML representation and to generate a graph structure. One of the advantages of assuming such an approach is that users can freely configure elements and attributes from their own XML schema which they wish to generate as nodes. The nodes can be then be connected by relations that are created in the same way, such as the places that are mentioned in letters, or the persons who send and receive them.

eXGraphs can be installed locally, or used via a public web server. Written in PHP, the tool relies on a configuration file in XML format composed by the user. Once the default template has been adjusted to the user's own data with the help of the supplied documentation, the configuration file can be submitted to the application. The file consists of three parts: the data source, the output format and target for the result, and the transformation rules.

The source of the data can be any local XML file or folder, RESTful API, or eXist-db API URL. As a target, eXGraphs can either directly export the data to a neo4j database, or can provide a text file with cypher queries for privacy and debugging purposes. As far as the modeling step is concerned, users can write a blueprint structure in which it is possible to freely create nodes from XML elements referenced by XPath and to link them by creating edges (Figure 3).

After submitting the necessary information, every XML file that is avail-

[5] eXGraphs (**e**xport **X**ML to **Graphs**), see also: https://lod.academy/site/tools/digicademy/exgraphs.

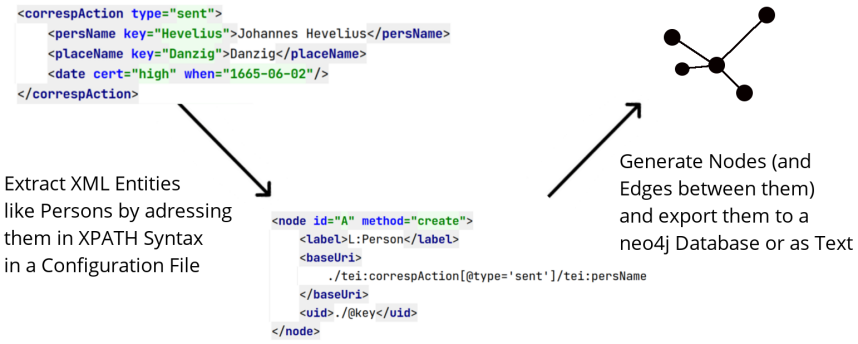

Figure 3: Simplified eXGraphs configuration instruction for extracting data from XML

able through the selected API, database, or folder is processed, and the output is either exported or written to a neo4j database. In the process, eXGraphs first collects all XML source files provided in the configuration file and converts them into PHP objects. Afterwards, it applies the provided transformation rules to each object and collects the resulting nodes and edges. In the last step, all collected nodes and edges can be exported, either as a download or into a neo4j database.

Whether or not this process is successful depends on the specifications provided by the users in the configuration file. Since the input can be reduced to a single test file and the output can be a plain text file download, users can experiment with different specifications without having to wait for their results. They can troubleshoot the resulting graph database structure by iteratively adapting the blueprint data model in the configuration file and downloading the cypher query text file. To facilitate abstraction, the whole transformation process can also be divided into several configuration files, so that users can, for example, first import all entities named in a text, followed by the texts as nodes together with the edges to or from the entities.

2.1.1 Configuring Input and Output

The input XML may either be retrieved from an URL, or passed to the application directly. The URL may either point to a single XML file, or to a collection of files via REST API, in which case the collection will be (recursively) crawled and all XML files ingested. Alternatively, the input XML can be passed directly as the content of the <resource> element within the configuration. Finally, if the application is installed locally, local data folders can be used as input. After the extraction process, all provided sources are collected prior to processing.

The application offers three ways of returning the graph data, one of

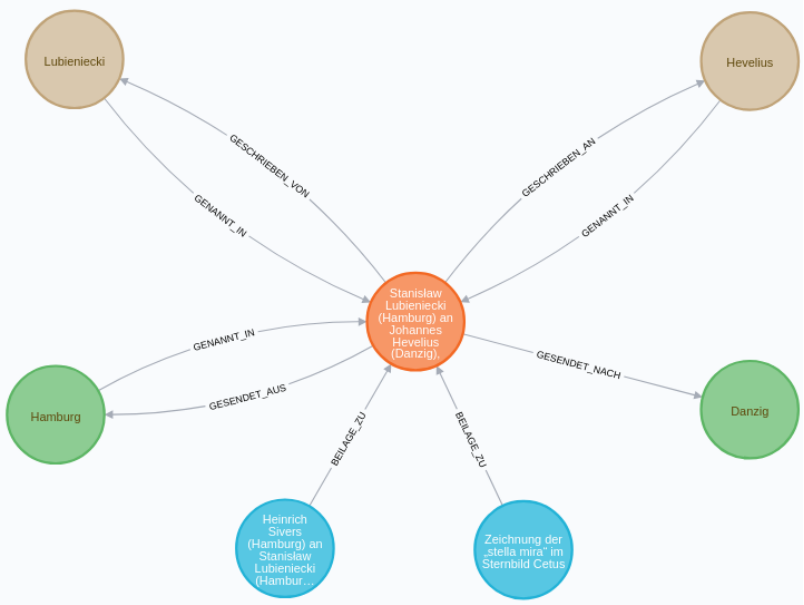

Figure 4: Sample dataset with a Person node and its context in the graph database

which must be specified in the REST API URL invoked by the user. If the user chooses to transmit the credentials of a neo4j database to eXGraphs, the application can write the results directly to the neo4j database. As a more privacy-oriented alternative, the application can return the data in one of two formats: either as cypher queries in a text file, which can then be manually executed in a neo4j database, or in a json format.

2.1.2 Configuring XML Extraction Rules

In the process of adjusting the transformation rules in an eXGraphs configuration, the nodes and edges of the graph are specified separately – nodes are obligatory, while edges are optional. The XML data to be extracted is specified using the XPath 1.0 syntax. For each node, one XPath expression serves as the basic iterator, while other values can be specified as relative XPath expressions. For example, as shown in Figure 5, the node label can be set using an XPath relative to the current node – the use of a constant string is optional. In a similar vein, the attributes of the graph node can be populated using (relative) XPath expressions.

```
<nodes>
    <node id="A" method="create">
        <label>L:Dokument</label>
        <baseUri>/*</baseUri>
        <uid>//tei:publicationStmt/tei:idno[1]/text()</uid>
        <attributes>
            <attribute name="title">./tei:title
            </attribute>
        </attributes>
    </node>
</nodes>
```

Figure 5: Sample eXGraphs instruction to import XML entities as Document nodes with the attribute *title*

```
<nodes>
    <node id="A" method="CREATE">
        <label>L:Brief</label>
        <baseUri>../*[@corresp]</baseUri>
        <uid>./@corresp</uid>
    </node>
    <node id="B" method="CREATE">
        <label>L:Person</label>
        <baseUri>
            ./tei:correspAction[@type='sent']/tei:persName
        </baseUri>
        <uid>./@key</uid>
    </node>
</nodes>
<relations>
    <relation from="A" to="B">
        <label>WRITTEN_BY</label>
    </relation>
</relations>
```

Figure 6: Sample eXGraphs instruction to generate nodes labeled *Brief* ("letter") and *Person*, and to add relations between each letter and its sender

The node specification supports three methods provided by the cypher query language: CREATE (always create a node), MERGE (create a node only if no identical node exists yet), and MATCH (only select the node for future reference), the latter being crucial for selecting the source and target nodes of edges.

The node configurations can be nested recursively. To give just one example of a likely usage scenario, for each instance of an XML element, its children can be iterated and imported, or selected as edge targets.

As shown in Figure 6, the edges of the graph are specified first by con-

figuring nodes with identifiers. In this example, we want to generate nodes for each letter occurring in the data source. Simultaneously, we create Person nodes for each sender of a letter. In the last step, we reference the created nodes as the source and target in an edge specification to create the edge named `WRITTEN_BY`.

2.2 Step 2: GRACE – Making the Graph Editable after XML Extraction

The second generic web application used in our workflow, GRACE,[6] makes it possible to edit the persistence data layer represented by a neo4j graph database. With GRACE, necessary corrections and future additions can be written directly to the graph. The same holds true for relations that may connect sub-graphs, and thereby yield additional context for certain data in the future. Thanks to GRACE's graphical user interface (GUI), working in the graph database involves a comparatively gentle learning curve.

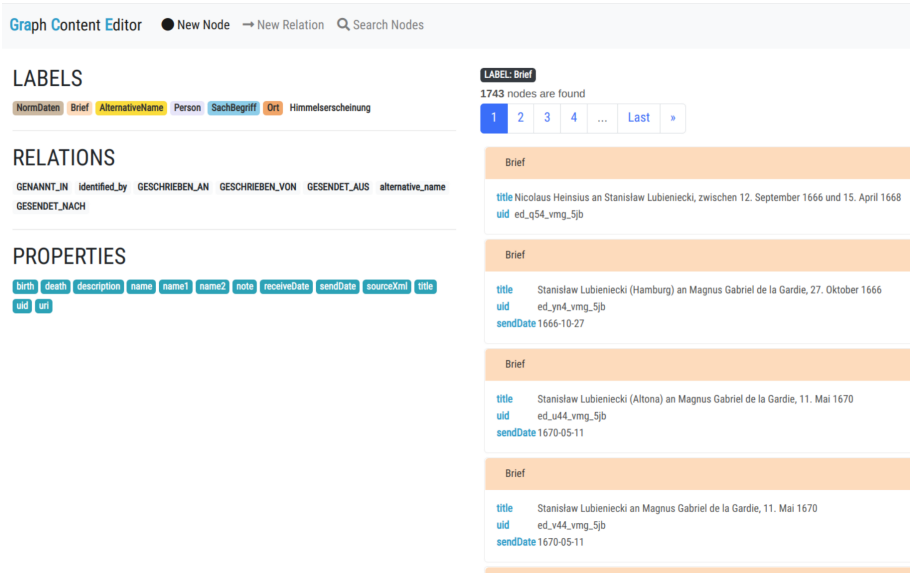

Figure 7: Graphical user interface of GRACE (proof of concept)

Our current proof of concept for GRACE is written in Vue.js and PHP. Once a neo4j database with its credentials has been configured in the tool's settings, it sets up a connection to the neo4j database via the GraphAware neo4j framework.[7] The database can be explored in a web browser in a more convenient and table-like view (Figure 7) than the one offered as a default

[6] GRACE (**Gra**ph **C**ontent **E**ditor), URL: `https://lod.academy/site/tools/digicademy/grace`.
[7] See also `https://github.com/graphaware/neo4j-framework`.

by neo4j. Furthermore, nodes and relations – including properties – can be edited or added.

All changes and additions are directly written to the provided neo4j database. Our aim is to provide a software that is capable of editing a graph database in the same way as a software like the Oxygen XML editor does for XML databases.

Ultimately, we would like the application to support two basic user actions: searching the data, and modifying the data.

2.2.1 User Action: Searching the Data

A basic prerequisite for making GRACE a useful editing tool is to make the data navigable and retrievable. To this end, we plan to implement a list view of nodes that will include a faceted search and a text search.

Since nodes represent the most relevant entities in our model (and, presumably, also in many other models), this is what the list view focuses on, with edges and related nodes appearing as subordinate information. The nodes' labels and properties are displayed by default. However, edge information is also accessible from the list view by simply toggling it open.

Furthermore, the list view will include faceted filtering options based on the graph database structure. Users may filter the nodes by their label, their properties, and/or connected edges. The filtering options are generated and updated from the graph database.

As a final instrument, the list view includes a search engine which mainly operates on node attribute values.

With these tools, navigating the graph database is bound to result in a user experience that is similar to that offered by current web technologies. The list format, in particular, is intended to make the database easily accessible to researchers from the humanities who are used to working with indexes, bibliographies, and catalogues, while still realizing the full potential of the graph model.

2.2.2 User Action: Modifying the Data

Besides making data easily searchable, one of the primary goals of our project is to provide an editing interface for graph data that can be readily applied to digital humanities projects. The action of modifying or editing the data is therefore implemented through a node single view, which encompasses edges (and related nodes).

This single view displays existing nodes, but is also used for the creation of new nodes. The interface is consistent with the list view in its layout of node properties and edges, but also encompasses editing options. The options

that are available to date include adding/changing/deleting node properties with autocomplete suggestions for the property names, and adding/deleting edges (primarily by selecting existing nodes as targets). We hope to further develop the latter function to recursively call a node creation dialogue.

In order to contextualize the node single view, a graph diagram panel is also included, which shows the immediately surrounding subnetwork of the selected node. This diagram can be updated to reflect edits to the edges, and thereby helps to immediately visualize and support editorial work.

3 The Socinian Correspondence Network in the Graph Database

In this section, we would like to demonstrate the tools that we have outlined above in action by taking a closer look at sample applications for the research data collected in the Socinian Correspondence Digital Scholarly Edition. More specifically, we would like to show the steps involved in transforming the basic correspondence metadata to a graph model and in connecting our research data to external data sources within the graph.

3.1 Graph Model

As the whole project revolves around the Socinian correspondence network, our main focus lies on processes of communication: a person writes a letter from a particular place at a particular moment in time, and then sends it to a correspondent, who assumes the role of the recipient. Figure 8 illustrates this process in graph form. The yellow nodes represent persons that fulfill either the role of the sender or the recipient in this particular communication context, even though in other contexts their roles might be different. This strict distinction between an entity, like a person or a place, and its role in a specific situation is what makes the graph model so flexible and powerful, as can be seen in Figure 9.

3.2 First Examples in the Graph

Figure 9 shows Stanisław Lubieniecki's correspondence.[8] This sub-graph illustrates that the individuals involved (represented by yellow nodes) could either be the sender or the recipient of a letter, while the brown nodes represent index entries related to the subject of comets, which are discussed in various contexts in the said letters.

One of the biggest advantages of graph technologies is that they greatly facilitate the linking of research data to other Linked Open Data sources. The

[8] Cf. https://sozinianer.de/id/Lubieniecki_Stanislaw

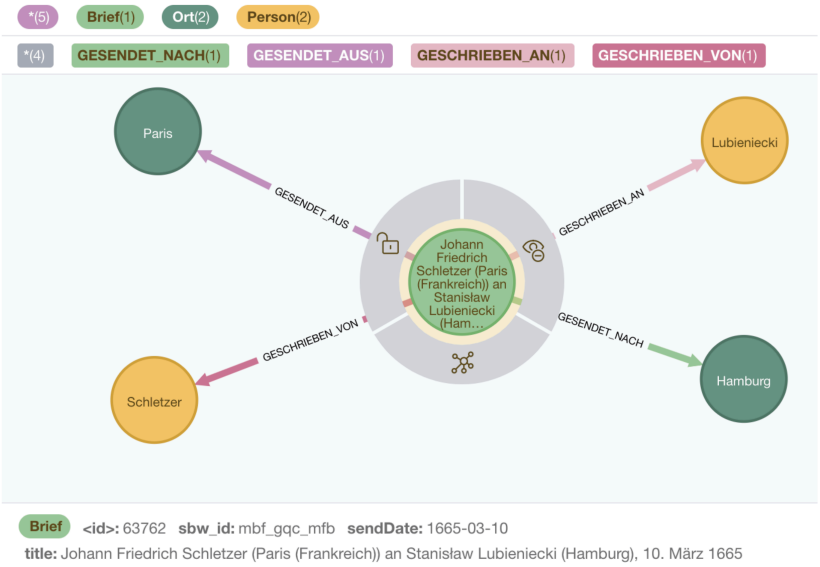

Figure 8: Data model of a letter

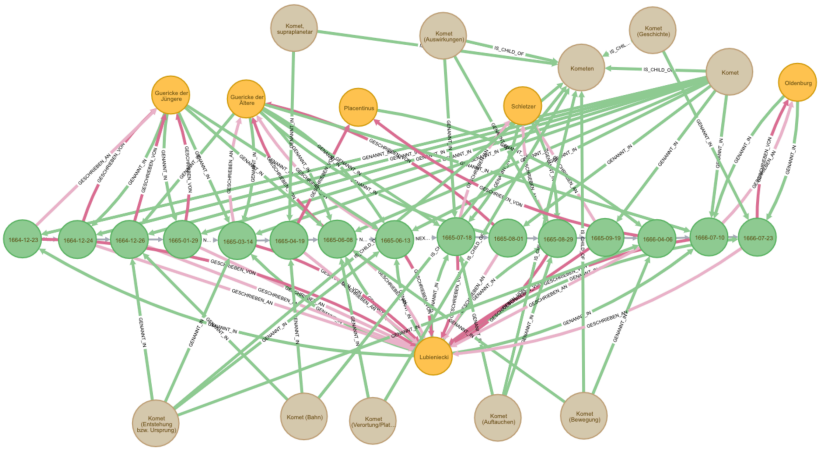

Figure 9: Letters sent by Lubieniecki (lower central yellow node) to his correspondents (yellow nodes in the upper part of the graph) on topics related to comets (brown nodes).

indexes of the Socinian Correspondence Project contain authority data for the entities mentioned. Figure 10 shows three people (yellow nodes), all of whom are mentioned in each of the three letters shown (green nodes). All three have their Wikidata ID stored in the graph as properties, which makes it easy to retrieve information on their kinship relations. It immediately becomes clear that they are closely related.

Figure 10: Persons (yellow nodes) mentioned in letters (green nodes) and their kinship relations according to information retrieved from Wikidata (red)

While such an exercise helps to illustrate the power of connecting different sources of research data, it is important to keep in mind that the primary purpose of this type of information is to provide clues as to which sources might prove useful for a given research question. It does not, however, provide direct answers to that question. Figure 11 shows a visualization of the Socinian letters from correspSearch metadata, which we hope will lead to further insights in the future — for example when researchers can combine the information from our project with additional data from other projects on correspSearch.

3.3 correspSearch

The metadata of the letters is also published at correspSearch,[9] which allows users to "...search within the metadata of diverse scholarly editions of letters. One can search according to the letter's sender, addressee, as well as place and date of the letter's creation" (Dumont, 2016).

[9] Cf. https://correspsearch.net.

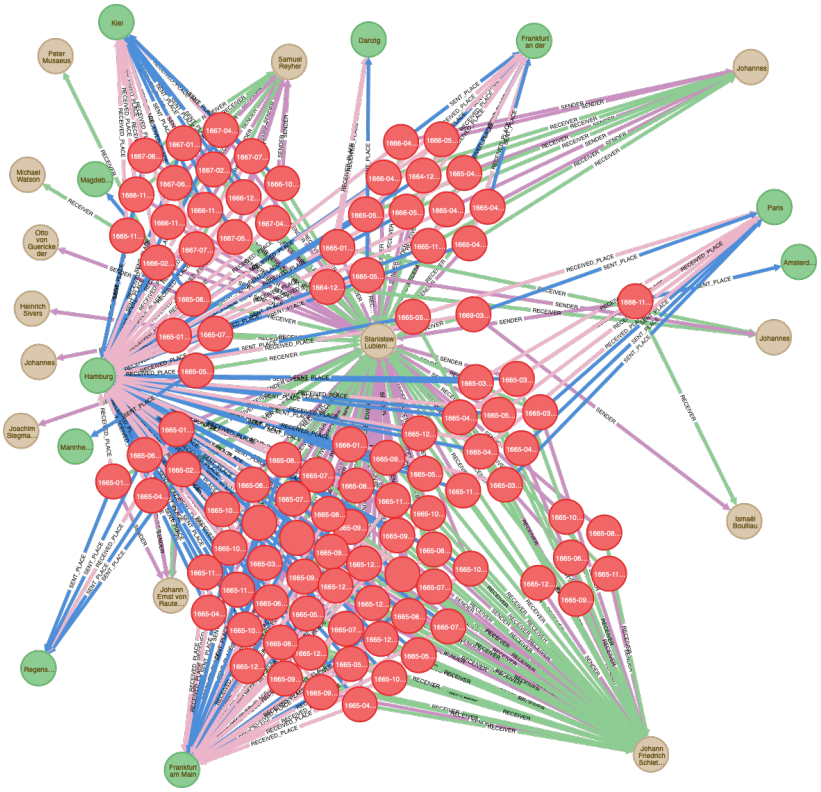

Figure 11: The Socinian letters in correspSearch

4 Remaining Work and Future Goals

The development of our tools and DSE is financed for six years by the DFG.[10] Our plan is to first publish eXGraphs in a beta version in order to obtain feedback from the digital humanities community and then to further test our proof of concept for GRACE.

Beside more conventional elements, such as the transcribed texts, facsimiles, and indexes, we plan to include graph visualizations in the context of certain entities in the frontend of the DSE. These will initially address a relatively objective context, like other letters in the chain of correspondence of the currently viewed letter. It is our hope that the resulting graph visualizations will then enable users to explore the data in a context of their own choosing, with their own research questions in mind.

We also wish to use the tools that we have outlined in this essay to put the graph as a persistent data layer for Digital Scholarly Editions to the test in a production environment. After transferring our data from XML to neo4j with the help of eXGraphs, our editors will be able to test GRACE when

[10] Deutsche Forschungsgemeinschaft https://www.dfg.de/.

editing or adding entries to our indexes. Furthermore, we seek to explore other workflows related to the integration of neo4j in the scholarly editing process by, for example, developing an extension for the Oxygen XML editor that makes it possible to retrieve and send data to a neo4j database.

Other questions that remain to be answered include how data in a neo4j database can be versioned and appropriately cited, and how the API for accessing the graph data created in the Socinian Correspondence Project can be implemented.

References

Daugirdas, K. (2016). Die Anfänge des Sozinianismus: Genese und Eindringen des historisch-ethischen Religionsmodells in den universitären Diskurs der Evangelischen in Europa. Number 240 in Veröffentlichungen des Instituts für Europäische Geschichte Mainz. Vandenhoeck & Ruprecht, Göttingen, DOI: 10.13109/9783666101427.

Dumont, S. (2016). correspSearch – Connecting Scholarly Editions of Letters. *Journal of the Text Encoding Initiative*, 10, DOI: 10.4000/jtei.1742.

Dumont, S. and Fechner, M. (2014). Bridging the Gap: Greater Usability for TEI Encoding. *Journal of the Text Encoding Initiative*, 8, DOI: 10.4000/jtei.1242.

Haaf, S., Geyken, A., and Wiegand, F. (2014). The DTA "Base Format": A TEI Subset for the Compilation of a Large Reference Corpus of Printed Text from Multiple Sources. *Journal of the Text Encoding Initiative*, 8, DOI: 10.4000/jtei.1114.

Kuczera, A. (2016). Digital Editions Beyond XML – Graph-Based Digital Editions. In Düring, M., Jatowt, A., Preiser-Kappeller, J., and van Den Bosch, A., editors, *Proceedings of the 3rd HistoInformatics Workshop on Computational History (HistoInformatics 2016) co-located with Digital Humanities 2016 conference (DH 2016)*, volume 1632 of *CEUR Workshop Proceedings*. http://ceur-ws.org/Vol-1632/paper_5.pdf.

Schrade, T. (2018). Annotate, Generate, Test, Deploy. Aktuelle Software-Engineering Methoden zur Steigerung der Nachhaltigkeit Digitaler Editionen. https://digicademy.github.io/2018-sustainable-editions/#/step-1.

Stadler, P., Illetschko, M., and Seifert, S. (2016). Towards a Model for Encoding Correspondence in the TEI: Developing and Implementing <correspDesc>. *Journal of the Text Encoding Initiative*, 9, DOI: 10.4000/jtei.1433.