ZfdG

Zeitschrift für digitale Geisteswissenschaften

Beitrag aus:
Sonderband 4 der ZfdG: Die Modellierung des Zweifels – Schlüsselideen und -konzepte zur graphbasierten Modellierung von Unsicherheiten. Hg. von Andreas Kuczera, Thorsten Wübbena und Thomas Kollatz. 2019. DOI: 10.17175/sb004

Titel:
The Codex – an Atlas of Relations

Autor/in:
Iian Neill

Kontakt: Iian.Neill@adwmainz.de
Institution: Akademie der Wissenschaften und der Literatur, Mainz
GND: 1082253677

Autor/in:
Andreas Kuczera

Kontakt: andreas.kuczera@adwmainz.de
Institution: Akademie der Wissenschaften und der Literatur, Mainz
GND: 1167802993 ORCID: 0000-0003-1020-507X

DOI des Artikels:
10.17175/sb004_008

Nachweis im OPAC der Herzog August Bibliothek:
1037074203

Erstveröffentlichung:
08.05.2019

Letzte Überprüfung aller Verweise:
08.05.2019

GND-Verschlagwortung:
Graphdatenbank | Semantisches Datenmodell | Textanalyse | Wissensrepräsentation |

Zitierweise:
Iian Neill, Andreas Kuczera: The Codex – an Atlas of Relations. In: Die Modellierung des Zweifels – Schlüsselideen und -konzepte zur graphbasierten Modellierung von Unsicherheiten. Hg. von Andreas Kuczera, Thorsten Wübbena und Thomas Kollatz. 2019 (= Sonderband der Zeitschrift für digitale Geisteswissenschaften, 4). PDF Format ohne Paginierung. Als text/html abrufbar unter DOI: 10.17175/sb004_008.

Iian Neill, Andreas Kuczera

The Codex – an Atlas of Relations

Abstracts

This paper looks at how deep integration between text and data is attempted in The Codex project. Standoff properties are used to mediate between the plain text stream and entities modelled in the Neo4j graph database. A dynamic standoff property text editor was constructed to enable real-time changes to text and annotations without invalidating standoff property indexes. An examination of the multidimensional affordances offered by standoff properties is explored, with reference to how annotations and graph entities can combine to construct an ›atlas of history‹ using Codex.

In diesem Beitrag wird The Codex vorgestellt, ein Projekt, in dem basierend auf Standoff Properties Texte multidimensional annotiert und die Annotationen in eine Graphdatenbank eingebettet werden können. Darüberhinaus sind Basistext und Annotationen im Unterschied zu vielen anderen Standoff-Markup-Systemen editierbar. Auch Annotationen selbst können wieder annotiert werden, was den Forschungsdiskurs leichter nachvollziehbar machen könnte.

1. Introduction

The Codex is a project that aims to achieve deep integration between text and structured data.[1] Although data can be ›extracted‹ from texts and stored in a database – whether that data be persons or places referenced, events recounted, numerical quantities reported, etc. – the non-sequential nature of databases can make it difficult to put the data back into context. While becoming amenable to computational analysis, the crucial narrative or argumentative structure is usually lost. Text itself can be considered to be a kind of database, one that is on the one hand constrained by its sequential presentation but on the other hand makes capable the modelling of thought in its multidimensional complexity. XML is a powerful tool for the modelling of text by allowing regions of text to be marked up with semantic tags or elements, and languages such as XPATH can be used to query the XML document model.[2] However, the use of markup itself introduces a discontinuity between the text and the data annotated. Markup changes the very text it marks up by creating a new marked-up document. Also, overlapping annotations – such as are commonly required in manuscript presentation annotations – cannot be directly expressed in XML's tree-like hierarchical structure, necessitating workarounds such as standoff markup which further degrade the readability of the XML document. Without freely overlapping annotations the most essential multidimensional aspects of text (its multitude of meanings) cannot be adequately marked up.

[1] We would like to thank Elena Spadini and Joris van Zundert for their invaluable comments in preparing this article.

[2] XPATH is arguably not a substitute for database query languages such as SQL or Cypher, however. Although XML bears some similarities to a database, in that its elements can be used to represent entities, a database can represent entities from thousands of documents and allow complex querying of them, whether with set-like operations such as SQL JOINs, filtering across tables, ordering and grouping, etc. Fundamentally, XML is a document markup format while a database is a relational system (whether the entities related are tables as in SQL databases or nodes and edges as in graph databases).

The Codex aims to bridge the divide between database and text – to achieve ›deep integration‹ – by eschewing markup entirely. Standoff properties are used, instead, to represent annotations. As defined by Desmond Allan Schmidt, the use of standoff properties is a technique for recording textual properties that do not conform to a context-free grammar, and can freely overlap.[3] While standoff properties have been proposed in the digital humanities several times,[4] Codex offers a practical solution to adding annotations to text, allowing the user to *make changes to the text in real-time without breaking existing annotations*. In addition, Codex offers a selection of stylistic, presentation, and semantic annotation types, as well as tools like named entity recognition (NER) and pronoun selection. The user can easily link annotations to entities in the graph database, or create new entities and their dependencies, entirely within modal windows, allowing the user to capture and construct data within the editor. They can also add footnotes, marginalia, etc., within a modal window editor, offering the same features as the editor in the window before. Annotations *themselves* can be annotated with editorial commentary entered via the modal editor.[5] In sum, through the use of a real-time standoff properties editor, a powerful modal-window entity management system, and the Neo4j graph database, Codex aims to provide an environment suitable for annotating all varieties of text, and linking annotations back to the text on a character-level.

One of the main goals of Codex, building on the deep integration of text and data, is to construct an ›atlas of history‹ from primary source texts. This phrase is a metaphor for both the *kinds* of annotations used as well as the graphical tools employed to *visualise* connections: ›atlas‹ here refers to a network of relations that are amenable to projection in various ways.[6] In other words, the purpose of Codex is not simply to annotate entities in text, but to represent these entities as nodes and relationships in the Neo4j graph database. The end result is both an annotated document *and* a graph dataset, which means that the more entity annotations are added to texts the richer the network of relations *between* texts becomes. As a basic example, if the corpus were the collected letters of Michelangelo[7] and references to Michelangelo's father Lodovico Buonarroti were annotated, it would be trivial to query the database to find all other texts referring to Lodovico.[8] But to make clearer what can be annotated and represented in the Codex system, we propose to give an overview of the main annotation types.

[3] Schmidt 2016a, pp. 63–69; Schmidt 2016b.
[4] Schmidt 2016a, p. 64.
[5] For example, if you wished to question the attribution of an agent annotation (such as a person) without deleting or changing the annotation, you could add comments against the annotation itself.
[6] Perhaps a more accurate metaphor for Codex would be an ›atlas of relations‹ as data doesn't have to contain spatial or temporal information to be visualised.
[7] An ongoing Codex corpus, along with the diary entries of Luca Landucci, a 15th century Florentine citizen, see del Badia 1883. Imported into Codex was the English translation by de Rosen Jervis 1927, see del Badia 1927. Additional Letters from Michelangelo where transcribed and annotated in Codex, see Carden 1913.
[8] With the proper nouns also the pronouns are annotated, one can get an accurate idea of the referential density in these letters.

2. Annotations

The types of annotations in Codex currently fall into three main categories: stylistic; presentation; and semantic.[9] Stylistic annotations include commonly used typographical styles like italics, bold, underline, strikethrough, subscript, superscript, and forms of emphasis like spaced and uppercased text. The size, colour, and font type can also be set.

Fig. 1: Stylistic, layout, and semantic annotation buttons in the Codex editor. [Neill / Kuczera 2019]

However, the more interesting categories are presentation and semantic. These are broken down as follows.

2.1 Presentation

2.1.1 Page, line, paragraph, sentence, column, etc.

These annotations denote regions of text corresponding to the presentation (or layout) of a page in a manuscript or a publication. Because standoff properties can overlap freely, presentation annotations in Codex pose no danger of truncating text with presentation markup.

2.1.2 Hyphens as a zero point annotation

Hyphens present a challenge for annotating medieval manuscripts; while the editor wishes to record the location of the *hyphen*, it is not desirable to intersperse the plain text with hyphens as this will confound literal text searches. The hyphen annotation in Codex avoids this problem by representing hyphenation with *zero-dimensional annotations*. A zero-dimensional annotation is a special case of a standoff property that has a *start index* value but no *end index* value; in this way, an annotation effectively refers to a position in the text *between characters*. The hyphen itself is not stored in the text (leaving the words unhyphenated), but the annotation indicates the location of the hyphen in the original. Zero point annotation is a generalizable feature of standoff properties and can be used for other cases as well.

[9] There is a fourth category in use in Codex that is out of the scope of this paper: machine-generated syntaxical annotations. These are annotations generated by Natural Language Processing (NLP) libraries such as Google's Cloud Natural Language API which essentially add a syntaxical substrate to standoff property documents. Their potential, when combined with other annotation layers, may be explored in a future paper.

2.2 Semantic

Before proceeding we should note that for each semantic annotation there is a corresponding semantic entity in the system. The types of annotations and entities available in Codex is defined by the application code and not by an existing standard, meaning that the system can be configured by a programmer with more annotations and entities as desired.[10] Each entity is modelled as a combination of nodes and edges in the graph database, sometimes expressed in hypernode structures (in other words, an entity modelled over a cluster of nodes).[11] Creating a semantic annotation for a text is tantamount to either selecting a pre-existing entity, or creating a new, corresponding entity in the modal window interface. Conversely, entities can also be created and managed outside of the Codex editor in sections of the Codex interface specific to that entity type. Therefore, entities can be considered as an independent data-set from texts but capable of integration with texts via semantic annotations. This means that entities can be exported or imported irrespective of texts in which they may or may not be mentioned: entities don't have to be mentioned in (or inferred from) texts to exist in the system.

2.2.1 Agents

An agent refers to any kind of entity that is mentioned in the text.[12] A non-exhaustive list of agent types includes people, places, and objects (natural and man-made).[13] Collective agents like organisations, families and other groups can be represented,[14] as well as *aspects* of an agent at a point in time.[15] Metadata about an agent can be recorded in associated property nodes which are dynamically defined in the interface as needed. For example, one can record the gender of a person, their height, weight, etc., and create new property types as required.[16]

Fig. 2: A record of Lorenzo de' Medici's gender as a property record. If desired, this can even be linked back to a statement in the text via a property annotation. [Neill / Kuczera 2019]

[10] Extending the standoff property editor requires some (arguably basic) knowledge of JavaScript, while creating new types of entities requires grounding in the C# and Cypher programming languages and the ASP.NET MVC framework.
[11] According to Wikipedia, a hypernode is: A kind of graph whose node set can contain other graphs as well as basic nodes. (Wiktionary: hypernode.)
[12] Although an agent was initially defined in Codex as just a person or group that causes events (i.e., an ›historical agent‹), this definition proved too restrictive to capture the variety of entities actually referred to in texts, and which serve agent-like functions in parts of speech.
[13] An agent is stored as an (:Agent) node in the graph database. Additional information specific to the agent type (for example, latitude and longitude for place agents) is recorded by storing the data in (:Property) nodes, and relating them to the (:Agent) node they pertain to.
[14] A collective is a group of agents, such as an organisation, a family, etc. Third-party pronoun parties are often represented as agent collectives in Codex.
[15] Agent aspects can be temporal or quantitative; for example, ›the Christ Child‹ agent is a temporal aspect of ›the Christ‹ agent, while an amount of ›3,000 moggia‹ is a quantitative aspect of (a portion of) ›7,000 moggia‹ in Landucci's diary entry of April 6th, 1484 (del Badia 1883, p. 39).
[16] Property data can also be time-specific for variable properties, such as height and weight to name two.

Agents can also be related to each other via dynamically-created relations. (In Codex, these are called meta-relations, for reasons which are discussed later in the section on this entity.) The example below shows some of Lorenzo de' Medici's genealogical relationships, as well as his connection to transient agent collectives like his presence in a group of six Florentine ambassadors to Rome in 1471, and in another embassy to Rome in 1483.

RELATIONSHIP ID	RELATION	AGENT 2
2e0a3d13-8c5a-4762-903f-2d83df11ca2b	Part of	Six ambassadors [1471/09/23]
46a56b46-f962-4735-ba82-b7ff7dae6922	Brother of	Giuliano de' Medici
980feab9-ed6f-4b7d-b13d-0518333e1283	Part of	The Medici family
993428e1-efb9-43b3-a299-7ba23460f730	Child of	Cosimo de' Medici
cb1ad5d7-e02e-4393-8ca9-2ffa997b607e	Child of	Madonna Lucrezia [de' Medici]
fbc97384-90df-4a2e-8fbb-d0917cca45dd	Part of	Three Florentine ambassadors [1483/11/10]
2afb8754-1705-456a-ae32-97087991fe27	Married to	Madonna Clarice
00700f9b-5369-4a5e-a426-d1dead67d710	Parent of	Giovanni de' Medici

Fig. 3: Screenshot from ›Codex‹ of a list of Lorenzo de' Medici's relationships in the system. [Neill / Kuczera 2019]

2.2.2 Claims

A claim refers to a statement concerning one or more agents, usually with respect to a place and a time. A claim is essentially a statement that usually takes the form of an event (an event claim), but can also represent a thought or an opinion. A claim entity in Codex is not taken as a statement of fact, substantiated or otherwise, but is rather a data-structure resembling a verb-phrase with prepositional agents. For example, the statement that »Lorenzo de' Medici died on his estate at Careggi« made by Luca Landucci in his diary entry of April 8th, 1492, is modelled in Codex as a ›(Subject) Lorenzo de' Medici, ›(Event) died‹, ›(At) Careggi‹, visualised in the Codex interface and Neo4j database browser in Figure 4 and Figure 5.[17] Our approach is not to assert whether or not the statement reports a fact, merely to allow the editor to annotate the statement.

	ROLE	TYPE	NAME	AGENTS	TIME
View	Event	died	Lorenzo de' Medici died on his estate at Careggi	(According To) Luca Landucci (b.1436 - d. 1516; apothecary) (At) Careggi (Subject) Lorenzo de' Medici (b. 1449/01/01 - d. 1492/05/08; active from 1469/12/02)	On 1492/April/8 - -:--:--

Fig. 4: The event claim by Luca Landucci about Lorenzo de' Medici's death in the Codex interface. [Neill / Kuczera 2019]

[17] del Badia 1883, p. 53-54.

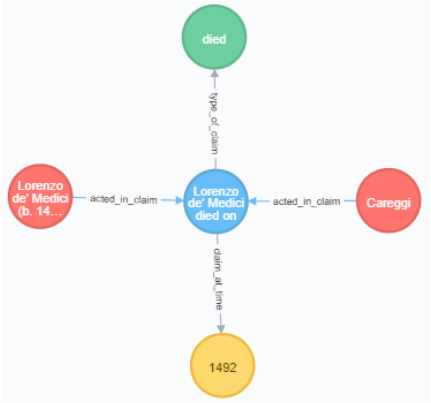

Fig. 5: A representation of the above event claim as nodes and edges in the Neo4j browser. The blue node is a (:Claim); the red nodes are (:Agent)s; the green node is a (:Concept); and the yellow node is the (:Time). [Neill / Kuczera 2019]

2.2.3 Texts

A text entity in Codex is composed of plain text and a collection of standoff-properties.[18] For convenience, texts can be assigned a ›type‹ indicating their function (such as ›body‹, ›footnote‹, ›margin note‹, etc.) but there are no limitations about the kind of text stored. Therefore, a text can contain as much of or as little of the source text as appropriate. In the case of the Luca Landucci Diary, each diary entry (of which there are several per page in the source) is stored in a separate text node, whereas each Michelangelo letter as a whole is stored in a text node.[19] Presentation annotations are used to mark sections of the text corresponding to the source (e.g., pages, columns, etc.), and the structure annotation can be used to link texts to structure entities which represent arbitrary sections of a publication (e.g., the chapters the text belongs to).

A text annotation in Codex is an annotation that relates a region of text in a text entity to a different text entity. There are two annotation topologies available to text annotations: they can either be applied to a region of text, like most annotations; or they can be inserted ›invisibly‹ between characters.[20] We can think of these topologies as one-dimensional and zero-

[18] Entity means here the node or cluster of nodes and edges that model the entity in the graph. For example, a text entity consists of a (:Text) node and a collection of (:StandoffProperty) nodes connected via the [:text_has_standoff_property] edge. We use the term standoff property document (SPD) to refer to the combination of plain text and standoff properties that makes up an annotated text in Codex.
[19] There is currently a practical limitation to the length of an individual text node in Codex of about ten kilobytes per node, due to a technical restriction related to how many HTML elements a browser can render on the screen at one time without noticeable lag. Other web applications have implemented solutions to this rendering issue, however, which can also be applied to Codex at a later date.
[20] A zero-dimensional annotation is ›invisible‹ with respect to the plain text stream (output) but is displayed within the Codex editor. The editor freely supports text characters that are either ›in-stream‹ or ›out-of-stream‹. ›In-stream‹ characters appear within the plain text stream, ›out-of-stream‹ characters do not; but both kinds can be annotated.

dimensional annotations.[21] It will be remembered that zero-dimensional annotations are also used to represent hyphen annotations; in the case of text annotations they could function as footnote numbers that the editor wishes to position in a text, but doesn't want to be included in the text per se.

2.2.4 Meta-Relations

A meta-relation entity is a relationship between agents with a number of features distinguishing it from simple relationships or edges in a graph database:

1. It is dynamically definable in the Codex interface. We have found the ability to create new relationship types in an *ad hoc* way to be invaluable for capturing the fluidity of relationships in texts. When the selection of relationship types is hard-wired into a program it becomes burdensome to create new ones; enabling the user to create them freely in the interface encourages the spontaneous creation of relationship types that are more fit-for-purpose;
2. It is bidirectional, meaning the user is able to specify both directions of the relationship (e.g., ›parent of‹/›child of‹) rather than being forced to adopt a single direction (e.g., ›parent of‹) imposed by graph edges. This encourages the user to think in terms of the overall relationship – e.g., ›parentage‹ – rather than forcing them to arbitrarily choose a single relationship to represent by implication a bidirectional relationship.[22] One advantage of this in constructing Cypher queries is that an agent's participation in a meta-relation can be found *without needing to consider their role in the relationship*, although that role is recorded and can still be explicitly queried if desired;[23]
3. They are composable within a *hierarchy*, effectively allowing *relationships themselves to be treated as a graph*. For example, one can define an overarching relation type like ›interpersonal relationships‹ and nest subordinate types beneath it, like ›social relationships‹, ›family relationships‹, ›professional relationships‹, etc., allowing one to query relationships between agents (e.g., people) at an abstract level. Rather than being limited to finding simply the ›friends of‹ a person, one can expand a query to also retrieve ›associates of‹, ›acquaintances of‹, ›confidantes of‹, etc.

A meta-relation annotation, therefore, is an annotation that refers to a meta-relation entity. Its practical purpose is to allow the editor to annotate agent relationships from texts, extending the network of relationships between agents. Ultimately, such networks allow the user to *find indirect connections between texts* on the basis of the relationships between agents

[21] A one-dimensional annotation contains both start index and end index values equivalent to a line segment, whereas a zero-dimensional annotation contains only a start index equivalent to a point.

[22] Whether or not one models ›parentage‹ with a ›()-[:parent_of]->() ‹ edge or a ›()-[:child_of]->() ‹ edge is to some extent an arbitrary choice; and with inherently unidirectional relationships such as ›married to‹ either two relationships need to be created at all times -- (a)-[:married_to]->(b), (b)-[:married_to]->(a) -- or the read query needs to accommodate either possibility, i.e., (a)-[married_to]-(b).

[23] Because a meta-relation contains more information than a unidirectional edge, meta-relation hypernodes can be programmatically decomposed into unidirectional edges as a final projection, or as a Cypher optimisation step.

established in the network. In Figure 6, the orange line beneath ›son of Antonio‹ is a meta-relation annotation indicating the source of the statement that Luca Landucci was the son of Antonio Landucci.

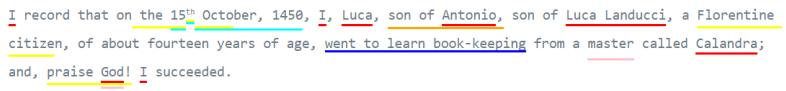

Fig. 6: An example of a meta-relation annotation (orange underline) in the Codex interface. [Neill / Kuczera 2019]

2.2.5 Concepts

A concept in Codex is a class or type that, taken as a whole, is part of a common ontology in the system. Note that the ontology is *not* common in the sense of being a ›universal‹ or ›world‹ ontology, but merely functions as a subgraph that is shared among other entity types (such as agents, claims, meta-relations, etc.) for the purposes of a common, reusable reference. Rather than constituting a universal, top-down ontology, the concept subgraph is in practise composed of any number of open or idiolectic ontologies defined by the user.[24] Codex already contains a number of idiolectic ontologies, such as ontologies for types of events, places, relationships, professions, etc. For example, claim entities reference the ›Events‹ subset of the concept subgraph. The concept ›Event‹ is the root node of the Events ontology and contains all types (and subtypes) of events that occur in the project domain.[25] Note that concepts can have more than one parent, if desired, as a graph is not bound by the limitations of a tree. Changing the structure of the ontology (e.g., moving a child concept to a different parent) is easily done through the Codex interface, meaning that ontological structures can be kept fluid to suit the evolving understanding of the project domain.

A concept annotation, therefore, is an annotation that refers to a concept entity in the common ontology. In Figure 7, the green underlines represent concept annotations on the words ›flautist‹ and ›prodigy‹.

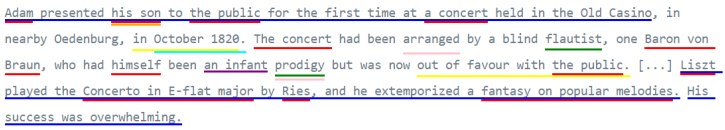

Fig. 7: Concept annotations (green) from Alan Walker's biography of Franz Liszt in the Codex interface, Walker 1983. [Neill / Kuczera 2019]

[24] It is planned to extend the system with a feature to import ontologies – such as OWL ontologies – into the concept subgraph to save the user the labour of entering them manually.
[25] A *project* in Codex is a collection of texts that belong together. As a practical example, the Diary of Luca Landucci (del Badia 1883) and the collected letters of Michelangelo (Carden 1913) are two projects in Codex. The project domain is the set of all texts and their associated entities (e.g., agents, claims, concepts, etc.) in the project. Because more than one project can share the same graph database, associated entities across project domains may overlap. For example, the ›Landucci Diary‹ and ›Michelangelo letters‹ projects have some agents in common, such as Lorenzo de' Medici and his family, various 15th century popes, Italian cities, etc. If overlap is not desired, associated entities *can* be segregated – but in our experience project domain overlap is one of the most exciting features of Codex as it can lead to the discovery of obscure connections between domains. Of course, the easiest way to segregate project domains is simply to reserve one Neo4j database instance per project.

2.2.6 Data Sets and Data Points

A data point in Codex is defined as a quantity with a unit of measurement that is attributable to a place and a time. A data set is a collection of data points. In the example below, the statement that »three people fell dead on this day« translates to a data-point where the value is »3«, the unit of measure is »people«, the place is »Florence«, and the time is »April 23rd, 1483« (the date of Landucci's diary entry).[26]

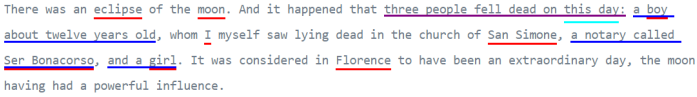

Fig. 8: The data point annotation (dark purple underline) indicates a statement that can be represented as an independent numerical quantity in the Codex interface. [Neill / Kuczera 2019]

The idea of a data point is to enable the editor to extract numerical data from a text that may be of statistical interest.[27] As indicated in the above Figure 8, a data point annotation links the text to a data point entity.

Some practical examples of data sets that can be extracted from historical sources include epidemiological data, weather records, census figures, crime statistics, etc.

2.2.7 Times

A time entity in Codex is a representation of a date and time in various degrees of precision. The entity is composed of nine components, which are all optional. Options for c. include ›on‹, ›before‹, ›after‹, and ›circa‹; options for Section include ›early‹ and ›late‹; and options for Season include ›Winter‹, ›Summer‹, ›Autumn‹, ›Spring‹. The variety of options is meant to reflect the realities of date representation in historical texts. (We intend to review the W3C Time Ontology for guidance on refining this model.)

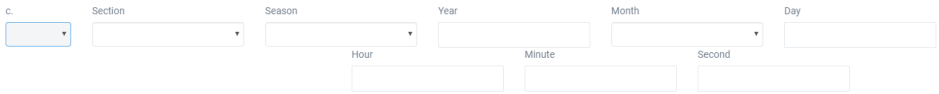

Fig. 9: The data entry modal window for a time entity in the Codex interface. [Neill / Kuczera 2019]

[26] del Badia 1883, p. 37.
[27] Data points can also store text values as well as numbers, so they can be used to represent state values. An example would be states of weather, such as rain, snow, floods, etc.

A time annotation is applied to any part of the text with an identifiable date / time, even if the text does not state a numerical date. For example, in Figure 8 above the time annotation (blue underline) links the text »this day« to the stated date of the diary entry (April 23rd, 1483).

Now that an overview of the main annotation types in Codex has been given, it is necessary to examine the standoff properties model as it forms the basis of Codex's approach to text-as-graph.

3. Standoff Properties

Aside from markup formats, word chains present another approach to annotation. A word chain is a graph model of a text where each word is treated as a token node and structure is indicated by relating each node to its next sibling in the sentence. Lexical and presentational annotations can also be linked to token nodes to model the structure of paragraphs and larger units.

Fig. 10: Text as a chain of word nodes in the Neo4j browser. [Neill / Kuczera 2019]

Like standoff properties, word chains represent a markup-free alternative to XML document formats, and offer a solution to the overlapping annotations problem. However, at present updates to word chains are managed through graph database queries, which requires some programming expertise. The Codex standoff property editor offers a trade-off between the multidimensional affordances of the graph and the technical simplicity, endurance, and sustainability of the text stream. Another distinction between word chains and standoff properties is that word chains take the word as the smallest token, which poses challenges for annotations inside of words (let alone how one chooses to define word boundaries).

The simplest solution from an annotation standpoint is conceivably to treat the *character* and not the *word* as the smallest token unit; however, managing a chain of character nodes as a graph data-structure would be exponentially more unwieldy than a chain of word nodes. However, this assumes that the characters themselves need to be represented as nodes; in fact, what defines an annotation is that it is a region of text with a certain intention (whether stylistic, presentational, semantic, etc.). If one moves away from the token node concept to an annotation node concept, then the text of the document can be stored in a plain text format (sans markup) and annotations can annotate the text by using *start* and *end* character indexes.

The removal of embedded markup makes the text stream easily readable to both humans and machines. It also solves the overlapping annotation problem because the properties are stored apart from the text, and not subject to hierarchical encoding conflicts. Multiple properties can refer to the same regions of text – or overlapping regions – by virtue of the start and end character indexes. Standoff properties are inherently discrete objects which

coexist in a ›flat‹ hierarchy, that is to say, with no imposed hierarchy at all. If a standoff property references a linked entity in the database, it can be easily connected via an edge, allowing full traceability from entities to the regions of text they are referenced by.

A standoff property, then, is essentially a data structure (tuple) that models the following attributes:

1. Type. A string representing the name (i.e., the type) of the annotation.
2. StartIndex. An integer representing the index position of the first character of the annotation: *0* <= *x* < *n*, where *x* is the index and *n* is the length of the text.
3. EndIndex. An integer representing the index position of the last character of the annotation within the length (same rule as the StartIndex).

In practise one would wish to extend this with a fourth attribute:

4. Value. A string representing data specific to the annotation, such as the unique identifier of a referenced entity, or alternatively a colour value, text size, font, etc.

At this point, one might object that standoff properties are not practical in the context of a text editor due to the likelihood of ›breaking‹ standoff property indexes upon text changes: deleting or adding a single character would necessitate the real-time recalculation of potentially every standoff property index in memory. Although this may not even be an issue in practise with texts of a certain size and number of standoff properties (given the efficiency of JavaScript interpreters in modern browsers), the Codex standoff property editor takes an approach that effectively sidesteps this issue. Text in the Codex editor is represented as a NodeList of HTML SPAN elements connected via reference pointers to an array of standoff property JavaScript objects.[28] Because a linked list is used to represent the text,[29] and because the standoff properties are linked to the text with pointers rather than indexes, characters can be freely added to or removed from the text without the need to recalculate indexes while editing.[30] Only when the user is ready to export the data from the editor (e.g., for saving) are the dynamic node pointers converted to static index numbers. As this step happens *after* all changes have been made to the text in the editing session, there is no danger of the indexes being out of alignment. The plain text and properties are exported as a JSON object which can be saved to a text file or converted to a data storage format of the user's choice. Codex translates the JSON export into a standoff property graph, linked to entity nodes as directed by the annotation type. These standoff property nodes can be converted to various formats such as text as a chain of word nodes.[31]

[28]The JavaScript NodeList data type combines a linked list with a tree structure, in that each element is connected to its previous and next siblings, and also connected to its children and parent nodes.
[29]Incidentally, this is basically the same principle as using a chain of character-token nodes to model a text in a graph database, except that it is done in memory in JavaScript in a web application.
[30]Deletions are a special case where the deleted text selection traverses the start or the end of a standoff property text range. In this case, affected properties are updated to ensure that the start and end pointers refer to the correct (remaining) characters.
[31]Because standoff properties are equivalent to character nodes in resolution, there is no difficulty in converting them to word nodes, which are lower in resolution.

Building on our technical definition of a standoff property, Codex extends this model further to include attributes that aid with database integration.

5. GUID. A 32-character string functioning as a unique identifier of the standoff property. This is required for saving the property to the database.
6. UserGUID. A GUID (see above) representing the user who created the annotation.
7. Index. An optional integer representing the order in which the standoff property was created.
8. Text. An optional string representing the source text referred to by the annotation. This is an optional attribute to make it easier to view standoff properties in the database. The StartIndex and EndIndex attributes are the source of truth with respect to the location of the annotation in the text.
9. Layer. An optional string representing arbitrary groups (layers) that the annotation is assigned to. For example, if the user wanted to group several agent annotations referencing artists, they could assign the annotations to an ›artist‹ layer. This grouping could be used for filtering either in the database, or in the editor itself.
10. IsZeroPoint: A boolean value indicating whether the standoff property is a zero-point annotation; that is, an annotation that refers to an invisible index point in the text. This can be thought of as an annotation that refers to the space between two characters.
11. IsDeleted: A boolean value indicating whether the standoff property has been marked as deleted.

To give a real-world example, below are images of a text in the Codex editor as well as a representation of a portion of it in as a JSON export.

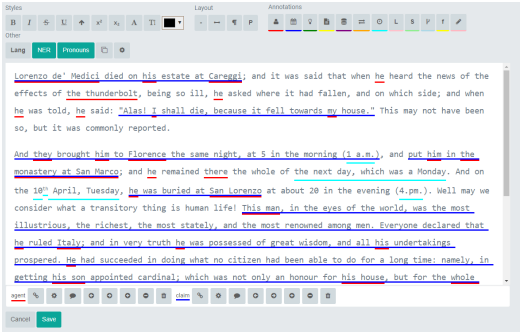

Fig. 11: Luca Landucci's diary entry for April 8th, 1492 on the death of Lorenzo de' Medici in the Codex interface, del Badia 1883, p. 53–54. [Neill / Kuczera 2019]

Following is an extract from the JSON export of the above text, with yellow highlights showing the typical parts of the standoff property data structure. The green parts show which text the annotation covers. The blue part shows the text itself.

{

"text": "Lorenzo de' Medici died on his estate at Careggi; and it was said that when he heard the news of the effects of the thunderbolt, being so ill, he asked where it had fallen, and on which side; ... ",

"properties": [{

"index": 23,

"guid": "cde24f38-81cb-4110-b368-b5b1f4ed4d53",

"type": "agent",

"layer": null,

"text": "Lorenzo de' Medici",

"value": "e45fed44-17a0-4c2c-9c00-858667a17904",

"startIndex": 0,

"endIndex": 17,

"isZeroPoint": false,

"isDeleted": false,

"userGuid": "fb067f75-a121-47c1-8767-99271c75cfc0"

}, {

"index": 25,

"guid": "5e33d2a8-dea0-4bbf-b4f6-8ccf40dac4d9",

"type": "agent",

"layer": null,

"text": "Careggi",

"value": "d8eda97c-79d7-43ad-b43f-fd3e3a05c68e",

"startIndex": 41,

"endIndex": 47,

"isZeroPoint": false,

"isDeleted": false,

"userGuid": "fb067f75-a121-47c1-8767-99271c75cfc0"

}, {

"index": 46,

"guid": "94eebe67-0136-4f54-a4c6-6e7072dfb3de",

"type": "claim",

"layer": null,

"text": "Lorenzo de' Medici died on his estate at Careggi",

"value": "175742e2-a342-455d-a1b9-3fe3a96e3fd9",

"startIndex": 0,

"endIndex": 47,

"isZeroPoint": false,

"isDeleted": false,

"userGuid": "fb067f75-a121-47c1-8767-99271c75cfc0"

}]

}

4. The Modelling of Doubt

Before proceeding to reflect on the possibilities of deep integration between text and data, it is important to review how the above discussion bears on the subject of the January 2018 graph-conference, »The modelling of doubts«,[32] hosted by the Akademie der Wissenschaften und der Literatur, Mainz. Although Codex wasn't designed with the intention of modelling doubt in the strict sense of quantifying it, it can be said that it aims at *modelling interpretation* in the following ways:

1. The ability to freely overlap annotations means that the same text regions can be annotated multiple times, allowing multiple interpretations to be captured. For instance, if two editors disagree about the identity of a person in the text, they could *each* add their own agent annotation to the same text region as overlaps are permitted;
2. Comments can be added to *annotations themselves*, enabling editorial discussions about the annotation validity to be recorded;
3. The ability to annotate agent properties, event claims, and meta-relations leads to *full transparency* around statements that are often just presented as established fact. For example, rather than simply accepting as fact the claim that Lorenzo de' Medici died at Careggi on April 8th, 1492, the claim annotation in Codex is traceable back to the precise section of the Luca Landucci text it occurs in.

5. Implications

The full potential of the standoff property model on the integration between text and graph data structures has yet to be documented. Aside from the convenience of plain text free of markup, of overlapping annotations, and of annotations whose sources are traceable back to precise text ranges, there are two features of standoff properties that suggest possibilities for computational analysis:

1. Standoff properties can be *grouped into layers*, where a layer is defined either implicitly by the annotation type or explicitly by a value stored in the Layer attribute;[33]
2. The StartIndex and EndIndex attributes offer the possibility of *combining annotations* that are contained by or overlap the same text range.

[32] »Die Modellierung des Zweifels« – Schlüsselideen und -konzepte zur graphbasierten Modellierung von Unsicherheiten am 19. und 20. Januar 2018 in der Akademie der Wissenschaften und der Literatur, Mainz.

[33] Incidentally, these layers could be exported into the standoff property JSON format, or a module could be written to export them into an XML format of choice. It should also be technically possible to convert a Codex standoff property document into an XML format such as TEI-XML, but this has not been actively developed yet.

These qualities of layering and combination have already led to useful features in the Codex editor (such as modal windows for managing named-entity and pronoun annotation candidates), but they offer computational insights as well. It is trivial, for example, to write a Cypher query to find all annotations in a text (or in all texts) that overlap each other in various ways. Overlapping annotations have the potential to enrich the text they annotate with their combined meanings. One approach that we are exploring in Codex is the comparison of manually-entered annotations (such as agents and event claims) with machine-generated syntaxical annotations, providing a natural language analysis of texts. This is an example of how standoff properties can support a multidimensional analysis of the text, allowing human-generated and machine-generated annotations to exist together.

6. Conclusion

The Codex uses standoff properties to integrate annotations with text in a graph database. Annotations map back to text at the character level and can be overlapped without constraint. The ability to overlap and comment annotations offers a convenient system for capturing discussions around doubt. The variety of semantic annotations – including event claims, meta-relations, agent properties, etc. – leads to a system where ›factual data‹ can be easily traced back to its text sources. Beyond the modelling of interpretations, Codex seeks to enable project editors to build an ›atlas of relations‹ from their source texts, integrating graph entities with text on a character level. The intended result is not so much a marked-up document (although this is a given) but a graph dataset with ›deep roots‹ in its constituent source texts, mediated through layers of standoff property annotations. Codex's real-time standoff property editor and modal-window entity management system are tools that we hope will assist editors in exploring the connections between structured data and text in their own digital editions.

Bibliographic References

Luca Landucci: Diario fiorentino dal 1450 al 1516. Continuato da un anonimo fino al 1542. Pubblicato sui codici della Comunale di Siena e della Marucelliana. Publ. by Iodoco del Badia. Florenz 1883. [Nachweis im GBV]

Luca Landucci: A Florentine Diary from 1450 to 1516. Publ. by Iodoco del Badia. Transl. by Alice de Rosen Jervis. New York 1927.

Michelangelo Buonarroti: Michelangelo: A Record of His Life as Told in His Own Letters and Papers. Publ. by Robert Walter Carden. London 1913.

Desmond Allan Schmidt (2016a): Using standoff properties for marking-up historical documents in the humanities. In: it – Information Technology 58 (2016), H. 2, S. 63–69. DOI: 10.1515/itit-2015-0030

Desmond Allan Schmidt (2016b): Standoff properties as an alternative to XML for digital historical editions. 2016. PDF. [online]

Alan Walker: Franz Liszt. 3 Vol. New York 1983. Vol. 1: The virtuoso years: 1811–1847. [Nachweis im GBV]

List of Figures with Captions

Fig. 1: Stylistic, layout, and semantic annotation buttons in the Codex editor. [Neill / Kuczera 2019.]

Fig. 2: A record of Lorenzo de' Medici's gender as a property record. If desired, this can even be linked back to a statement in the text via a property annotation. [Neill / Kuczera 2019.]

Fig. 3: Screenshot from ›Codex‹ of a list of Lorenzo de' Medici's relationships in the system. [Neill / Kuczera 2019.]

Fig. 4: The event claim by Luca Landucci about Lorenzo de' Medici's death in the Codex interface. [Neill / Kuczera 2019.]

Fig. 5: A representation of the above event claim as nodes and edges in the Neo4j browser. The blue node is a (:Claim); the red nodes are (:Agent)s; the green node is a (:Concept); and the yellow node is the (:Time). [Neill / Kuczera 2019.]

Fig. 6: An example of a meta-relation annotation (orange underline) in the Codex interface. [Neill / Kuczera 2019]

Fig. 7: Concept annotations (green) from Alan Walker's biography of Franz Liszt in the Codex interface. [Neill / Kuczera 2019.]

Fig. 8: The data point annotation (dark purple underline) indicates a statement that can be represented as an independent numerical quantity in the Codex interface, Walker 1983. [Neill / Kuczera 2019.]

Fig. 9: The data entry modal window for a time entity in the Codex interface. [Neill / Kuczera 2019.]

Fig. 10: Text as a chain of word nodes in the Neo4j browser. [Neill / Kuczera 2019.]

Fig. 11: Luca Landucci's diary entry for April 8th, 1492 on the death of Lorenzo de' Medici in the Codex interface, del Badia 1883, p. 53–54. [Neill / Kuczera 2019.]