The Cross-Disciplinary Repository for Covid-19 Research

LTS is founded by Amy Neustein, PhD, Series Editor of **Speech Technology and Text Mining in Medicine and Health Care** *(de Gruyter); Editor of* **Advances in Ubiquitous Computing: Cyber-Physical Systems, Smart Cities, and Ecological Monitoring** *(Elsevier, 2020); and co-author (with Nathaniel Christen) of* **Cross-Disciplinary Data Integration Models for the Emerging Covid-19 Data Ecosystem** *(Elsevier, forthcoming).*

The Cross-Disciplinary Repository for Covid-19 Research (hereafter called **CR2**) is an archive of open-access research data related to Covid-19 and SARS-COV-2. This repository will accompany the volume *Cross-Disciplinary Data Integration Models for the Emerging Covid-19 Data Ecosystem*, scheduled to be published in October of this year (2020). The current paper will discuss the goals and structure of **CR2**. It will also explain how scientists or organizations can contribute data to **CR2**. The authors are hoping to find and/or republish data sets representing a diverse spectrum of scientific fields which contribute to our medical, biological, and policy-oriented knowledge about the Covid-19 pandemic. Our mission is to centralize Covid-19 data more effectively than is achieved by existing platforms, either general-purpose data hosting frameworks (Dryad, DataVerse, Mendeley) or research corpora focused on Covid-19, but not necessarily placing a priority on data mining and data integration (an example of the second category is **CORD-19**, which is a large corpus of scientific publications related to Covid-19). Although a significant body of relevant scientific literature can be found on platforms such as Dryad and **CORD-19**, mining these resources for usable raw data can take considerable effort. **CR2** seeks to streamline this process by aggregating Covid data into a central corpus and by integrating data sets, as much as possible, into a common representation and technological infrastructure.

With that said, the mission of **CR2** is not specifically to curate a large volume of data for its own sake. Indeed, the core of **CR2** will be available as a package that can be downloaded and used on an ordinary computer — analogous to **CORD-19**, which covers over 33,000 research papers but is encoded in such a way that the total size of the corpus is not prohibitively large. We also believe a more substantial (and not necessarily fully open-access) Covid-19 information space would be beneficial to the scientific community, but this larger project is not the direct mission of **CR2**. Ideally, **CR2** can be paired with a larger technology sharing a similar implementational strategy but with different accession paradigms, allowing for an open-ended collection of Covid-19 data which users may selectively access, instead of a single package that users may assess as an integrated resource. However, the focus for **CR2** — and for the *Cross-Disciplinary Integration* volume — is to serve as a precursor to such a larger ecosystem and/or to use Covid-19 as a case-study in how such larger technologies may best be engineered.

One consequence of this background is that small data sets can be equally valuable for **CR2** as larger ones — indeed, given a volume of data which would be impractical to include in one downloadable package, it is better to extract a representative sample for inclusion in **CR2**, with links to allow researchers to access the entire resource as desired. Relatively small data sets (including samples standing in for larger ones) serve several scientific and computational purposes: (1) they can provide researchers with a mental picture of how data in different disciplines, projects, and experiments is structured; (2) they can serve as a prototype and testing kernel for technologies implemented to manipulate data in relevant formats and encodings; and (3) they can lay the foundation for data-integration strategies. For example, when designing a representation format and/or implementing code to merge different data formats into a single structure (or meta-structure), it is useful to work with small, representative examples of the data structures involved, so as not to complicate the integration logic with computational details solely oriented to scaling up the data-management logistics.

Data associated with Covid-19 comes in many different structures and formats. This diversity in and of itself presents technological challenges: if a Covid-19 information space encompasses files representing 25 different incompatible formats, users need 25 different technologies to fully utilize this data. The point is not to envision one single application used for all Covid-19 data — to the contrary, scientific software generally needs to hone in on the data visualization and analytic requirements of particular disciplines; biochemists use different programs than astrophysicists. However, nor is it appropriate for software components to be so balkanized that common data-management functionality is duplicated

across many isolated projects. The ideal data-management architecture is a compromise between the domain-specific needs of scientific applications and the desire for a centralized technology to handle data logistics at the foundational level where disciplinary domains are not consequential: data accession, provenance, user validation, searching, and so forth.

In short, the ideal technology for a cross-disciplinary information space involves a central technological core which is common to, and serves as an entry point for, diverse domain-specific resources. This entry point can provide preliminary functionality such as user validation and "logging in" (to the degree that access to parts of a repository may be restricted), searching for data and/or data sets, file management, updating information, and so on. The central software may then be complemented with narrower applications that provide more complex capabilities, but within the context of specific disciplines or scientific workflows.

In the Covid-19 context, some of the relevant data (despite superficial differences) has a common table-like structure, which may be manifested in different ways (comma-separated values, spreadsheets, word-document tables, Numeric Python, **XML**, **JSON**, **SQL** dumps, etc.). In general, disparate tabular data can be straightforwardly merged. One level of integration simply encodes tabular structure into a common representation: any field in a table can be accessed via a record number and a column name and/or index. In some cases, more rigorous integration is also possible, for example by identifying situations where columns in one table correspond semantically or conceptually to those in a second table. In either case, it is reasonable to assume that a single abstract data format lies behind surface data-expression in forms like spreadsheets and **CSV**, so that all files in an archive encoding spreadsheet-like data can be migrated to a common model.

Other forms of clinical and epidemiological inputs are more amenable to graph-like representations: for example, trajectories of viral transmission through person-to-person contact is obviously an instance of social network analysis. Similarly, models of clinical treatments and outcomes can take graph-like form insofar as there are causal or institutional relations between discrete medical events: a certain clinical observation *causes* a care team to request a laboratory analysis, which *yields* results that *factor* into the team's desicion to *administer* some treatment (say, a drug *from* some provider *with* some chemical structure), which observationally *results* in the patient improving and eventually *being* discharged. In short, patient-care information often takes the form — at least conceptually — of a network comprised of different "events", each event involving some observation, action, intervention, or decision made by care providers, and where the important data lies in how the events are interconnected: both their logical relationships (e.g., cause/effect) and their temporal dynamics (how long before a drug leads to a patient's improvement; how long before admission to a hospital and discharge). These graph-like representations are a natural formalization of "patient-centered" data models as outlined, for example, in [4].

As the previous two paragraphs have outlined, a significant subset of Covid-19 data (or, more generally, any clinical/biomedical information) conforms to either tabular or graph structures, and so it is feasible to unify all of this information into a common framework to the degree that one works with a meta-model which incorporates both record-sets and graph structures (node-sets and edge-sets) in its representational arsenal. A graph-plus-table architecture is generally considered some form of Hypergraph model, and indeed **CR2** uses a hypergraph paradigm to merge many different sorts of information into a common structure. In particular, **CR2** introduces a new "Hypergraph Exchange Format" (**HGXF**) which can provide a text encoding of many files that, when originally published, embodied a diverse array of file-types requiring a corresponding array of different technologies. **CR2** will include computer code to read **HGXF** files and use them to create hypergraph-database instances.

Not every format relevant to Covid-19 can be realistically translated to **HGXF**. In particular, fields requiring substantial quantitative analysis — e.g., biomechanics or genomics — express data via encodings optimized for relevant mathematical operations. In this scenario, **CR2** will not attempt to migrate *all* of a data file to **HGXF**. However, even for these files **CR2** will generally provide a supplemental **HGXF** encoding supplying data *about* the original file, with information about the file type, preferred software components for viewing/manipulating its data, and so forth. In this manner the contents of non-**HGXF** files can be indirectly included into the **CR2** hypergraph-based ecosystem.

As this summary has implied, hypergraph data models and the **HGXF** format are a significant aspect of how **CR2** is designed and how it technologically achieves its stated goals. The **HGXF** format will therefore be examined in greater detail in the following section.

I DigammaDB and the HGXF Format

CR2 will introduce a new database engine for preparing the information provided within the repository (called DigammaDB, or ϚDB for short) as well as the Hypergraph Exchange Format (HGXF). In general, HGXF files can be generated from ϚDB instances, capturing the state of the database at a moment in time. ϚDB could therefore be used to store research data. When scientists choose to publish data, they would then output the information from their database into HGXF, using the resulting files as the published raw data. In the CR2 context, ϚDB will be used to merge data from multiple files into a single database, yielding HGXF output forming most of the raw data republished in CR2.

Operationally, ϚDB is designed to emulate the programming interface provided by several existing databases and hypergraph libraries — for instance, HyperGraphDB ([12]), WhiteDB ([18]), and HgLib ([8, p. 9]). That is, ϚDB is designed so that existing code using these technologies can be adopted for ϚDB with relatively little effort. ϚDB also introduces some new concepts and structuring features, which will reviewed below. In addition to prior technologies such as HyperGraphDB, ϚDB draws on theoretical work connected to hypergraphs and their value as multi-paradigm, general-purpose metamodels. The reach of hypergraph theory encompasses several paradigms for modeling scientific data, such as Conceptual Graph Semantics (see [22]) and Conceptual Space Theory (which is linked to hypergraphs in work summarized by [5], where Conceptual Space semantics is paired with hypergraph categorial grammar). ϚDB therefore introduces modeling elements designed to capture scientific details (such as dimensional analysis and units of measurement). When discussing graph structures and programming techniques, ϚDB draws terminology from these scientific perspectives as well as from existing Hypergraph database engines.

1.1 Scientific Features of DigammaDB

ϚDB has no specific connection to SARS-COV-2 or Covid-19; it is conceived as a general-purpose database engine that can facilitate application development across many domains and industries. However, ϚDB is designed with exceptional attention to scientific research and software, with respect to metadata, GUI integration, its representation of files and file-types, and interoperability with technologies related to publishing and open-access research data. A full enumeration of ϚDB's scientific focus is outside the scope of this outline, but certain features are specifically relevant to CR2, so they can be discussed here:

From-the-Ground-Up Qt Integration All ϚDB classes natively interoperate with QT (a leading cross-platform application-development and GUI framework). Programmers then have access to features like QDataStream and QT meta-objects for binary serialization of C++ objects for persistence in the database. Robust QT support also makes it easy to design GUI classes for interacting with ϚDB data, and for employing ϚDB as a technology for managing and storing application state. This is important in the scientific computing context because most scientific software is designed as native desktop-style applications needing special-purpose GUI classes (in this environment it is usually not possible to adopt techniques such as HTML page templates which are commonplace in the web-programming context for creating user views onto database objects). Therefore, implementing custom GUI logic is an important part of the development requirements for scientific applications, and it is helpful to interface with a database engine prioritizing that aspect of software engineering.

Projections and Scientific Annotations ϚDB adopts the HyperGraphDB notion of *projections*, or descriptions of individual fields within a complex object that can automate the extrapolation of binary-serialization algorithms (which in turn allows complex objects to be stored in ϚDB alongside primitive values like strings and numbers). ϚDB also uses projections as a basis for introducing metadata associated with Conceptual Spaces and Conceptual Space Markup Language (CSML) as mentioned earlier. In particular, projections can be annotated with statistical/quantitative details such as dimensions (e.g., base quantities and units of measurement), scales or "measurement levels" (in CSML terms, particularly the distinction between nominal, ordinal, interval, and ratio axes), minima and maxima, probability distributions, and the aggregation of isolated units into complex dimensions, or CSML "domains" (vectors, tensors, area/volume spans, and so forth).

Data Microcitations Most scientific data is formally or informally linked to peer-reviewed publications which present research findings. Published data sets make these connections explicit, by joining document identifiers and URLs with the corresponding identifiers for designating and locating data sets. These connections are however coarse-grained, applying only to documents and data sets in their entirety. By contrast, an emerging trend in publishing is to

link portions of publications — such as individual sentences, paragraphs, or figure illustrations — with smaller parts of a data set (which could be an individual record/sample, or a table column, or some conceptually related group of records or columns). To support microcitations, there must be some formal mechanism to individuate small parts of a data set. This structure is provided internally by ꝺB: there are multiple microcitable "zones" in a database, which introduce encodings for microcitation targets that become exposed to publications via HGXF. Here "zones" refer to aspects or portions of a database that may be cited individually, apart from the database as a whole: records/atoms, types, projections, subgraphs, software components implementing a digamma-application interface (analogous to HGApplication in HyperGraphDB), among others. These entities can be given identifiers which are unique not only in the context of a single database, but if needed over a larger corpus (e.g. an archive of research data) so that the relevant atoms/types/projections can be referenced from publication texts (and similar resources). Such references can also be linked to GUIs; one can for instance say that the data currently visible in a GUI window or dialog box is a view onto a specific database atom, information that could be used for debugging, storing application state, inter-application networking (e.g. linking PDF viewers for research papers with applications to view research data), and so forth.

Memory and Persistence Models ꝺB can be run in several different "modes", which determine how data is stored in memory while an application is running and/or is stored in files. If desired, ꝺB can be used as an "in-memory" database which does not maintain persistent file state at all. This feature can be useful in a research-data context where all information derives from data files. In this scenario, a ꝺB instance can be loaded from parsed serializations (e.g., HGXF files) without using other file-system resources. The database engine can then be adopted as a tool for accessing and manipulating the "infoset" derived from these data files, analogous to XML libraries interfacing with a Document Object Model.

Scientific-Computing Interface Description Following HyperGraphDB, ꝺB establishes a "type system" unique to each database, which is responsible for translating application-level types to database atoms. The type system used by ꝺB is actually more detailed (compared to HyperGraphDB) and allows for the description of procedure signatures and the declaration of conceptually related procedural groupings, which collectively enable interface definition as persistent data within the database itself. ꝺB likewise supports a notion of "meta-procedures", which are indirectly derived from Alexandru Telea's *metaclasses* and *dataflow interfaces* (see [24]). Moreover, ꝺB introduces a notion of "channels", which are a higher-level graph structuring feature (see [14, Chapter 3]) that may, in particular, be applied for the semantic annotation of function signatures so as to refine interface documentations. Collectively, these features enable each ꝺB database to store information about procedures used to access and interact with their stored information, to facilitate the implementation of scientific workflows and GUIs for accessing and using ꝺB data.

1.2 Programming with HGXF Files

The prior outline described some of the principle features of ꝺB, which in turn influenced the design of HGXF, insofar as one goal of HGXF is to serialize ꝺB instances. However, HGXF files are also generic resources for serializing research data (or any other technical information), and ꝺB includes libraries for using HGXF (not necessarily in the context of ꝺB instances). Therefore, HGXF deserves a brief overview in its own right.

Like any hypergraph meta-model, HGXF represents information on two levels. On the one hand, each "node" is (in genral) a synthesis of multiple smaller parts. In HGXF, the parts of a hypernode are *structure fields* and *array fields*, where the former are roughly analogous to named columns, and the latter are usually either units within expandable collections (such as lists, queues, and dictionaries) or fixed-size arrays of numeric fields with similar dimensional qualities (e.g., vectors representing point in space). HGXF files define "types", which are internal to the file (but may be associated with ꝺB types) and describe the layout of fields within the containing hypernode (in particular, the number and names of structure fields, and restrictions on the number of array fields, if any). HGXF types may also provide hints about how to best encode the parsed data in running memory.

The structure of HGXF files is specified by a Reference Implementation using C++ to parse and manipulate HGXF data. In general, this code library mixes functional and object-oriented programming styles. The central procedures in this library are ones to isolate single hypernodes, by graph traversal or queries, and then procedures to operate on array and structure fields contained within hypernodes. The basic interface for this latter context involves passing criteria for selecting one or more fields, along with a callback function which will be called with the vector of matching fields (or else a procedure to operate on fields one at a time). In this functional style, HGXF hypernodes can be seen as monads

encapsulating access to their internal structure and array fields (or "hyponodes").

On a higher level, applying not to the scope of a single hypernode but to groups of hypernodes, **HGXF** introduces several higher-level structuring elements (mostly derived from corresponding GDB features). In particular, *channels* are aggregates of edges, and *frames* are contexts for defining edges and nodes. In GDB, every node and edge is constructed in the context of a frame, which *may* (but need not) offer either additional semantic detailing or callback functionality to add code affecting how graph data is constructed. All graphs have a "default" frame that implies no special semantic context (i.e., all edges are asserted relations deemed to apply in general, with no contextual filter) and that utilizes general-purpose algorithms for edge and node construction. Developers can introduce more specialized frames as desired, and construct nodes and edges in these tailored contexts. Such frames can then be identified in **HGXF** as the context where a given node or edge serialization applies. Hypernodes may also be annotated within frames to identify a conceptual, operational, or semantic role of a node within some larger node-collection. Indeed, GDB introduces the concept of a *virtual frame* to capture logical patterns implicit in certain node-annotations which are not explicitly designated by frame objects allocated within the database.

The default behavior for edge-construction (which can be overridden within individual frames) conceives edges as **RDF**-style "triples", where a directed pair of nodes is annotated with a simple label (which may or may not be defined within a controlled vocabulary) and/or a more complex object; edges can be marked with hypernodes of their own. The source and target hypernodes of a hyperedge correspond to the source and target node-sets defined in most mathematical treatments of hypergraph theory. In libraries such as HgLib, which (compared to graph/hypergraph databases) are more directly based on this mathematical theory) hyperedges are defined by providing one or two (for undirected or directed edges respectively) sets of nodes. GDB supports this construction as well, essentially constructing hypernodes "on the fly", but in this case the fields within these auto-constructed hypernodes are "proxies" for hypernodes (possibly three or more) linked by the edge. Proxies can also be defined for more complex objects, such as frames and subgraphs, allowing multi-dimensional, nested graph structures if these are desired for a particular domain model.

HGXF also supports microcitations, which are discussed above. By designating parts of a data set as microcitation targets, **HGXF** can be integrated with annotation systems designed for scientific publications. These features allow document annotations — which are used, for instance, to connect text in a research paper with scientific concepts and Ontologies — to connect also with microcitable elements within data sets and Research Objects which serialize data via **HGXF**. In this use-case, **HGXF** works in conjunction with a related protocol, the Annotation Exchange Format (**AXF**), which is also initially developed to support the **CR2** implementation. In **CR2**, **AXF** will be used to connect Covid-19 research data with publications where the corresponding data sets are introduced to the scientific community. The **AXF** format and design principles will be discussed further in the next section.

II The AXF Platform

The **AXF** Platform (hereafter called **AXF**) is a toolkit for hosting full-text, open access publications, with an emphasis on scientific, academic, and technical documents. At the core of an **AXF** Publication Repository is a collection of files in a machine-readable **AXF** Document Format (**AXFD**), which are paired with human-readable **PDF** documents as well as supplemental multi-media and metadata files. Depending on institutional requirements, an **AXF** repository may be the primary storage resource for the contained publications, or an adjunct resource whose documents are linked to publications hosted elsewhere. In the second scenario, the primary goal of an **AXF** repository is to host manuscripts in **AXFD** format, along with software to aid viewing and text-mining of the associated publications.

AXFD therefore has two distinct purposes: (1) to aid in text and data mining (**TDM**) of full publication text (along with research data that may be linked to publications), and (2) to enhance the reader experience, given e-Reader software (canonically, **PDF** viewers) which are programmed to consume **AXF** information. To (1) aid in text mining, **AXFD** documents can be compiled into different structured representations, yielding document versions that can be registered on services such as CrossREF **TDM** and SemanticScholar. Given a Document Object Identifier, text-mining tools can therefore readily obtain a highly structured, machine readable version of the publication, which may then be used as the basis for further text-mining and **NLP** operations. Simultaneously, to (2) improve reader experience, the **AXF** platform generates numeric data linking semantically significant text locations to **PDF** viewport coordinates (such text locations include annotation, quotation, or citation start/end points and paragraph or sentence boundaries — collectively dubbed a Semantic Document Infoset, or **SDI**). This **SDI**+Viewport (**SDIV**) information can then be used by **PDF** applications

to provide contexts for word searches, to localize context menus, to activate multi-media features at different points in the text, and in general to make PDF files more interactive. Data sets composed with the aid of AXF tools may include source code for a PDF viewer (an extension to XPDF) capable of leveraging AXF data.

In addition to the AXFD document format, the AXF platform includes the Annotation Exchange Format itself, a protocol for defining and sharing annotations on full-text publications. AXF differs from other annotation-representation strategies by (1) providing more detail concerning the location of annotated text segments, in the surrounding publication context, and (2) supplying annotation data in multimedia or microcitation formats which extend beyond conventional "controlled vocabularies". In terms of (1) publication context — that is, the annotation *target* (using terminology from the Linguistic Annotation Framework, or LAF) — AXF represents SDIV information as introduced above; this data supplements the node/index coordinates used by traditional annotation mechanisms. With respect to (2) annotation metadata — or the annotation *body* (again using LAF terminology) — AXF introduces models for multimedia assets, software components, and data set content, which may be linked to annotation targets. With this additional metadata, annotations may be used in application-development environments, not only for text mining.

The following sections will (1) outline AXF and AXFD in greater detail, (2) describe how AXF repositories can unify publications sharing similar themes, scholarly disciplines, or coding requirements, and (3) describe features for data-set publication in the context of AXF repositories.

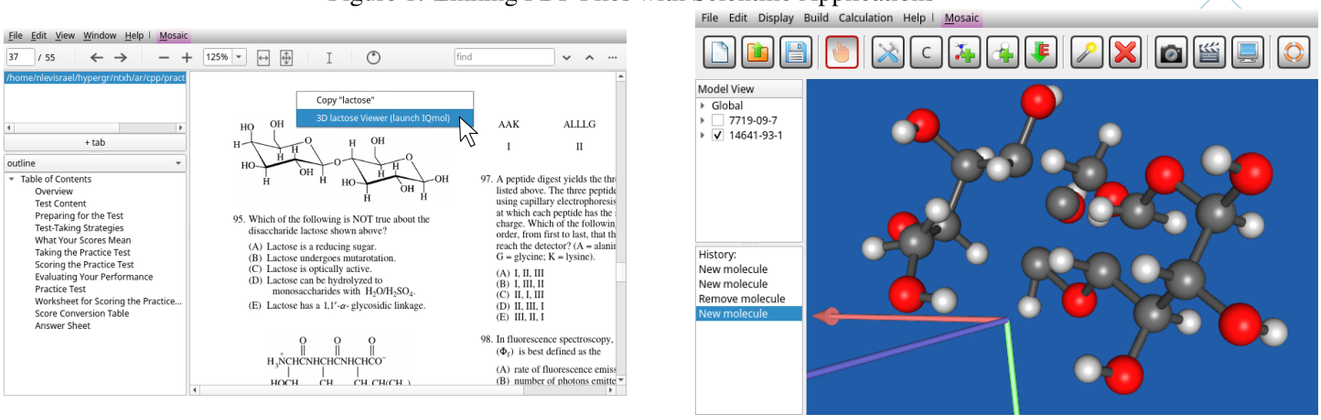
2.1 AXF Documents

The AXFD format for describing document content and structure is designed to be a "Pivot Representation" in the sense of LAF (see [11]). In particular, AXFD can represent the structure of both XML (including several XML flavors used in publishing) and LaTeX. Technically, AXFD does not prescribe any specific input format; instead, a document is considered an instance of AXFD if it can be compiled into a Document Object satisfying interface requirements. A C++ reference implementation anchors the AXF Document Object Model; nodes in this implementation have facets combining LAF, XML, and LaTeX. In practice, AXFD manuscripts are then converted via LaTeX to PDF, and simultaneously compiled to XML representations so as to generate machine-readable, structured full-text versions of the manuscripts. Authors can choose to compose AXFD papers to conform with several common publication XML standards, such as JATS (Journal Article Tag Suite), SciXML [21], and IeXML [20] (the latter is an annotation-oriented XML language used by the BeCAS project [16]). Authors or editors may further define or model scientific concepts via strategies focused on computer simulation (in the broad sense as defined, for instance, in [26]) or statistics (e.g., Predictive Model Markup Language or Attribute-Relation File Format), and/or conceptual semantics, such as Conceptual Space Markup Language (which was inspired by the linguist Peter Gärdenfors, but developed in a scientific/mathematical framework in e.g. [1], [2], [9], and [5]).

One distinct feature of AXF is that LaTeX and XML generation are chained in a pipeline: the LaTeX and subsequent PDF generation steps yield auxiliary data, which includes PDF viewport data, that can be subsequently incorporated into XML views onto the documents. Specifically, AXFD-generated LaTeX files include notations for semantic annotations and for sentence boundaries, implemented via LaTeX commands which, as one processing step, write PDF coordinates to auxiliary files. The resulting data is then read by a C++ program which collates annotations and sentence-boundaries into a vector of data structures indexed by PDF page numbers, creating a distinct file for each page, and zips those files into an archive which can be distributed alongside (or embedded inside) the PDF publications. Simultaneously, sentences, paragraphs, annotations, and other semantically significant content (such as quotations and citations) are assigned unique ids and compiled into their own data structures (from which machine-readable XML full-text may be generated). These XML files may then be hosted and/or registered on TDM-oriented services such as CrossRef. At the same time, unique identifiers unify this XML data (focused on text mining) with PDF viewport data (focused on reader experience). The goal of such integration is to incorporate text-mining results so as to enhance reader experience. For example, Named Entity Recognizers might flag a word-sequence as matching a concept within a controlled vocabulary. Via the relevant paper's SDI model, this annotation may be placed in a proper semantic context — for example, obtaining the text of the sentence where the Named Entity occurs. This semantic information may then be used by a PDF viewer — e.g., providing a context menu option to select the sentence text, when the context menu is activated within the rectangular coordinates of the annotation itself.

As a representation of annotation data structures, AXF ensures that SDI and viewport data is included among annotations

Figure 1: Linking PDF Files with Scientific Applications



wherever this data is available. This facilitates the integration between text-mining tools and **PDF** viewer software, which in turn enhances reader experience. As mentioned earlier, every annotation can be placed in a semantic context (e.g., the text of the surrounding sentence), which provides useful reader features such as one-click copying of sentences to the clipboard. Other reader-experience enhancements involve multimedia assets. As a concrete example, suppose a paper includes mention of a chemical; that particular keyword can accordingly be flagged for annotation. As one encoding of the corresponding scientific concept, the annotation can include the chemical's Chemical Abstract Service Reference Number, via which it is possible to obtain Protein Data Bank (**PDB**) files to view the relevant molecular structure in **3D**. In sum, annotations supply a constellation of data — in this example, concepts may be linked not only to identifiers in cheminformatic ontologies, but also to **CAS** reference numbers and thereby to **3D** graphics files — which facilitate interactive User Experience at the application level, not only document classification at the corpus level. Once a chemical compound (mentioned in a publication) is linked to a **PDB** file (or any other **3D** format) the **PDF** viewer may include options to for the reader to connect to software or web applications where the corresponding visuals can be rendered. Via **AXF**, the relevant document-to-software connections are asserted not only on the overall document level, but on the granular scale of the precise character and **PDF** viewport coordinates where the relevant annotation is grounded (Figure 1 illustrates such capabilities in the context of a chemistry publication — specifically, test-preparation materials for the Chemistry **GRE** exam).

To support this kind of multimedia functionality, **AXF** standardizes a Plugin Framework, dubbed "**MOSAIC**", allowing programmers to embed code which can parse and respond to **AXF** annotations in different scientific and document-viewer applications. **MOSAIC** allows different applications to inter-operate; in particular, **PDF** viewers can share data with scientific applications that can render files in domain-specific formats such as **PDB**. This application networking protocol is considered part of the **AXF** annotation model, because application-oriented information is computationally relevant for many concepts encountered in scientific and technical environments. For instance, one aspect of cheminformatic data is that many chemical compounds are modeled by **PDB**, **MOL**, or **CHEMXML** files, which in turn are associated with software applications that can load those file types. Such inter-application networking data is relevant to **PDF** viewers when they display manuscripts with annotations that suggest links to special file types and their applications; the viewers can employ this information to launch and/or communicate with the corresponding software. **AXF** is designed to facilitate implementation of application-networking protocols as an operational continuation of processes related to obtaining and consuming annotation data.

The **AXF** document model, at the manuscript-structure level, is paired with a novel "Hypergraph Text Encoding Protocol" (**HTXN**) operating at the character-encoding level. Within the **HTXN** protocol, an annotation target is a character-index interval in the context of an **HTXN** character stream. On that basis, **HTXN** treats documents as graphs whose nodes are ranges in a character stream, where text can be recovered as an operation on one or more nodes (e.g., the text of a sentence is derived from a pair of nodes representing the sentence's start and end). **HTXN** code-points are distinguished in terms of their semantic role, which may be more granular than their visible appearance — for example, a period glyph is assigned different code-points depending on whether it marks a sentence-ending punctuation, an abbreviation, a decimal point, or part of an ellipsis. Procedures are then implemented to represent text in different formats, such as **ASCII**, Unicode, **XML**, or LaTeX. In contrast to a format such as Web Annotations, any particular human-readable text presentation (including **ASCII**) is considered a *derived* property of the annotation, not a foundational representation.

AXFD manuscripts do not need to utilize **HTXN** for character data, but **HTXN** simplifies certain **AXF** operations, such as

identifying sentence boundaries. In particular, HTXN provides distinct code-points for end-of-sentence punctuation, so that sentence-boundary detection reduces to a trivial search for those particular code-points. Proper HTXN encoding requires that authors follow certain simple heuristics — e.g., that end-of-sentence periods should be followed by two spaces and/or a newline, whereas other uses of a period character should precede at most one space. Aside from the goal of preparing documents for text-mining machine-readability, such conventions are appropriate even for basic typesetting, because non-punctuation characters have their own kerning rules (this is why LaTeX provides a distinct command for non-punctuation glyphs that would otherwise be read as punctuation characters). HTXN hides these typesetting details within its character-encoding schema, which is then useful both for producing professional-caliber LaTeX output and for identifying SDI details (such as sentence boundaries) which with less rigorously structured text would need elaborate text-mining or NLP algorithms.

Each AXFD document is, in sum, associated with an aggregate of character-encoding, annotation, document-structure, and PDF viewport information. The AXF platform uses code libraries to pull this information together as a runtime object system, so that any application which loads an AXFD manuscript can execute queries against the corresponding collection of AXF objects (queries such as obtaining the sentence text around an annotation, obtaining the concave-octagonal viewport coordinates for a sentence,[1] obtaining application-networking information for an annotation, etc.) In addition to such runtime data, AXF platforms can compile the full suite of information into machine-readable files for text and data mining. These files, collected across a corpus of multiple documents, then form the backbone of an AXF publication repository, as will be discussed next.

2.2 AXF Publication Repositories

The AXF platform is designed for hosting collections of publications sharing a common academic or technical focus. When AXF is used in the context of a general-purpose text and/or data repository, the AXF platform is designed to work with collections that are organized into separate projects or topics, each giving rise to an archive or corpus of publications. Insofar as these corpora internally share a common theme or focus, they can be associated with their own ontologies, code libraries, annotation models, and application-networking protocols, based on the sorts of applications and data structures commonly used in the corresponding scholarly discipline. In some cases, publishers may choose to package an entire archive of research papers (perhaps along with research data) as a single downloadable resource. AXF allows publishers to construct such Research Archives following the structure of existing examples such as the ACL (Association for Computational Linguistics) Anthology or the recent CORD-19 corpus. This latter archive is a useful case-study in both the possibilities and limitations of existing publication-repository technology, so it is reviewed here in more detail.

CORD-19, curated by the Allen Institute for Artificial Intelligence, was spearheaded by a White House initiative to centralize scientific research related to COVID-19 (see [6]). The collection was formulated with the explicit goal of promoting both *text mining* and *data mining* solutions to advance coronavirus research, so that CORD-19 is intended to be used both as a document archive for text mining and as a repository for finding and obtaining coronavirus data for subsequent research. Although novel research is being incorporated into CORD-19, many of the articles reproduced in this corpus are older publications related to coronaviruses and to SARS in general, not just to the current pandemic. As a result, the full-text versions of these publications were retroactively aggregated into a single archive due to the unanticipated emergence of a coronavirus crisis, with the full text often obtained from PDF files rather than from structured representations (such as JATS) explicitly intended for text mining.

This archival methodology results in CORD-19 being limited as a TDM framework. These limitations include the following:

Transcription Errors Transcription errors can easily result from trying to read scientific data and notations based on PDF files — or on full-text representations using relatively unstructured formats such as XOCS (the response-encoding format for the ScienceDirect API). Transcription errors cause the machine-readable text archive to mis-

[1] In the general case, sentence coordinates are concave octagons because they incorporate the line height of their start and end lines; in the general case sentences share start and end lines with other sentences, while also including whole lines vertically positioned between these extrema. A sentence octagon roughly corresponds with the screen area where a mouse/pointer action should be understood as occurring in the context of that sentence from the user's point of view — implying that the user would benefit from context menu options pertaining specifically to that sentence, such as copy-to-clipboard.

represent the structure and content of documents. For instance, there are cases in CORD-19 of scientific notation and terminology being improperly encoded. As a concrete example, "2'-C-ethynyl" is encoded incorrectly in one CORD-19 file as "2 0 -C-ethynyl" (see [7] for the human-readable publication where this error is observed; the corresponding index in the corpus is 9555f44156bc5f2c6ac191dda2fb651501a7bd7b.json). To help address these sorts of errors — which could stymie text searches against the CORD-19 corpus — it is obviously preferable to archive structured, machine-readable versions of publications, using a platform such as AXF.

Converting Between Data Formats Although the CORD-19 corpus is published as JSON files, many text-mining tools such as those reviewed in [15] recognize inputs or produce outputs in alternative formats, such as XML, BIOC, CONLL (Conference on Natural Language Learning), or JSON trees with different schema than CORD-19. For this reason, rather than providing data with one single representational format, it is better to encode the data along with code libraries that can express the data in different formats as needed for different TDM ecosystems.

Inconsistent Annotations The structure of CORD-19 allows text segments to be defined via a combination of JSON file names, paragraph ids, and character indices. This indexing schema is used for representing certain internal details of individual articles, such as citations, but is not explicitly defined as an annotation target structure for standoff annotations against the archive as a whole. This problem could also be rectified with code libraries that map index targets to file handles and character pointers.

Limited Support for Research Data-Mining Even though many papers in CORD-19 are paired with published data sets, there is currently no tool for locating research *data* through CORD-19. For example, the collection of manuscripts available through the Springer Nature portal linked from CORD-19 includes over 30 COVID-19 data sets, but researchers can only discover that these data sets exist by looking for a "supplemental materials" or a "data availability" addendum near the end of each article. These Springer Nature data sets encompass a wide array of file types and formats, including FASTA (which stands for Fast-All, a genomics format), SRA (Sequence Read Archive, for DNA sequencing), PDB (Protein Data Bank, representing the 3D geometry of protein molecules), MAP (Electron Microscopy Map), EPS (Embedded Postscript), CSV (comma-separated values), and tables represented in Microsoft Word and Excel formats. To make this data more readily accessible in the context of CORD-19, it would be appropriate to (1) maintain an index of data sets linked to CORD-19 articles and (2) merge these resources into a common representation (such as XML) wherever possible. This research-data curation can then be treated as a supplement to text-mining operations. In particular, queries against the full-text publications could be evaluated *also* as queries against the relevant set collection of research data sets.

Wrappers for Network Requests Scientific use of CORD-19 will often require communicating with remote servers. For example, genomics information in the COVID-19 data sets (such as those mentioned above that are available through Springer Nature) is generally provided in the form of accession numbers which are used to query online genomics services. Similarly, text mining algorithms often rely on dedicated servers to perform Natural Language Processing; these services might take requests in BIOC format and respond with CONLL data. As another case study epidemiological studies of COVID-19 may need to access APIs or data sets such as the John Hopkins University "dashboard" (see https://coronavirus.jhu.edu/map.html, which is paired with a GIT archive updated almost daily). To reduce the amount of "biolerplate code" which developers need for these networking requirements, an archive's text-mining code could provide a unified framework with which to construct web-API queries, one that could be used across disparate scientific disciplines (genomics, NLP, epidemiology, and so forth).

Many of these limitations observed in CORD-19 reflect the fact that this corpus was prepared as raw (text) data, without any supporting code. By contrast, recent initiatives — such as the Research Object protocol (see [3]) and FAIR ("Findable, Accessible, Interoperable, Reusable"; see [25]) — encourage authors to publish code and data together, so that the computing environment needed to process published data is provided within the data set itself. The CORD-19 limitations accordingly provide an example of why Research Objects, rather than raw data sets, should be preferred for data publication in the future. The Research Object model is usually defined in the context of a single publication, but the paradigm applies equally well to corpora encompassing many single articles. That is, AXF is structured so that Research Archives can be designed as higher-scale Research Objests, wherein the document collection is bundled with supporting code and an overall computing and software-development environment. Such archive-specific SDKs would include AXF-specific code as well as libraries or applications often utilized in the academic disciplines relevant to the archival subject areas. The AXF platform especially promotes the design of domain-specific SDK which are *standalone*

and *self-contained*, with minimal external dependencies. As much as possible, users should not have to install external software to utilize data provided along with an **AXF** repository; instead, the needed data-management tools should be provided in source-code form within the archive itself.

Each **AXF** repository, then, should bundle numerous applications used for database storage, data visualization, and scripting. The goal of this application package would be to provide researchers with a self-contained computing platform optimized for scientific research and findings related to the archived publications. Archival **SDK**s should try to eliminate almost all scenarios where programmers would need to perform a "system install"; for the most part, the entire computing platform (including scripting and database capabilities) should be compiled from source "out-of-the-box". While the actual libraries and applications bundled with an archive would depend on its topical focus, the following is an example of components that would be appropriate in many different **SDK**:

- **XPDF**: A **PDF** viewer for reading full-text articles (augmented with **CORD-19** features, such as integration with biomedical ontologies);
- **Qt**: The **Qt** library is a cross-platform Application-Development framework and **GUI** toolkit commonly used for scientific applications (**XPDF** is one example of a **Qt**-based document viewer). Almost any data set can be accompanied with **Qt** code for data visualization, so that readers would not have to install additional software. For its part, **Qt** can be freely obtained and, once downloaded, resides wholly in its own folder (there is no install step which modifies the user's system); as such, **Qt** along with individual archive **SDK**s function as standalone packages, although optimally the **SDK**s would be updated along with new **Qt** versions.
- AngelScript: An embeddable scripting engine that could be used for analytic processing of data generated by text and data mining operations on **CORD-19** (see [13]);
- WhiteDB: A persistent database engine that supports both relational and **NoSQL**-style architectures (see above);
- MeshLab: A general-purpose **3D** graphics viewer;
- LaTeXML: a LaTeX-to-**XML** converter;
- PositLib: a library for use in high-precision computations based on the "Universal Number" format, which is more accurate than traditional floating-point encoding in some scientific contexts (see [10]).

To this list one might add components specific to various scientific fields: **IQmol** for chemistry and molecular biology, for example, or open-source libraries such as EpiFire or Simpact (for Epidemiology), **UDPipe** (for **CoNLL**), and so forth. Here again the priority would be for self-contained components with few external dependencies — particularly libraries programmed in **C** or **C++**, which are the languages best positioned to be a common denominator across diverse research projects (of course, many scientific **C++** libraries have wrappers for languages like **R** or Python that researchers may be more comfortable using). In general, Research Archive code should be (1) *self-contained* (with few or no external dependencies, as emphasized above); (2) *transparent* (meaning that all computing operations should be implemented by source code within the bundle that can be examined as code files and within a debugging session); and (3) interactive (meaning that the bundle does not only include raw data but also software to interactively view and manipulate this data). Research Archives which embrace these priorities attempt to provide data visualization, persistence, and analysis through **GUI**, database, and scripting engines that can be embedded as source code in the archive itself.

It is worth noting that a data-mining platform requires *machine-readable* open-access research data (which is a more stringent requirement than simply pairing publications with data that can only be understood by domain-specific software). For example, radiological imaging can be a source of **Covid-19** data insofar as patterns of lung scarring, such as "ground-glass opacity," are a leading indicator of the disease. Consequently, diagnostic images of **Covid-19** patients are a relevant kind of content for inclusion in a **Covid-19** data set (see [19] as a case-study). However, diagnostic images are not in themselves "machine readable." When medical imaging is used in a quantitative context (e.g., applying Machine Learning for diagnostic pathology), it is necessary to perform Image Analysis to convert the raw data — in this case, radiological graphics — into quantitative aggregates. For instance, by using image segmentation to demarcate geometric boundaries one is able to define diagnostically relevant features (such as opacity) represented as a scalar field over the segments. In short, even after research data is openly published, it may be necessary to perform additional analysis on the data for it to be a full-fledged component of a machine-readable information space.[2] To deal with this

[2] This does not mean that diagnostic images (or other graphical data) should not be placed in a data set; only that computational reuse of such data will usually involve certain numeric processing, such as image segmentation. Insofar as this subsequent analysis is performed, the resulting data should wherever possible be added to the underlying image data as a supplement to the data set.

Figure 2: Data Microcitations via Tabular Columns

sort of situation, **AXF** equips **SDK**s with a *procedural data-modeling vocabulary* that would both identify the interrelationships between data representations and define the workflows needed to convert research data into machine-readable data sets.

Another concern in developing an integrated Research Arcive data collection is that of indexing documents and research findings for both text mining *and* data mining. In particular, **AXF** introduces a system of *microcitations* that apply to portions of manuscripts *as well as* data sets. In the publishing context, a microcitation is defined as a reference to a partially isolated fragment of a larger document, such as a table or figure illustration, or a sentence or paragraph defining a technical term, or (in mathematics) the statement/proof of a definition, axiom, or theorem. In data publishing, "data citations" are unique references to data sets in their entirety or to their smaller parts. A data microcitation is then a fine-grained reference into a data set. For example, a data microcitation can consist of one column in a spreadsheet, one statistical parameter in a quantitative analysis, or "the precise data records actually used in a study" (in the words adopted by the Federation of Earth Science Information Partners to define microcitations; see [17]). As a concrete example, a concept such as "expiratory flow" appears in **CORD-19** both as a table column in research data and as a medical concept discussed in research papers; a unified microcitation framework should therefore map *expiratory flow* as a keyphrase to both textual locations and data set parameters. Similarly, a concept such as *2'-C-ethynyl* (mentioned earlier, in the context of transcription errors) should be identified both as a phrase in article texts and as a molecular component present within compounds whose scientific properties are investigated through **CORD-19** research data. In so doing, a search for this concept would then trigger both publication and data-set matches at the same time.

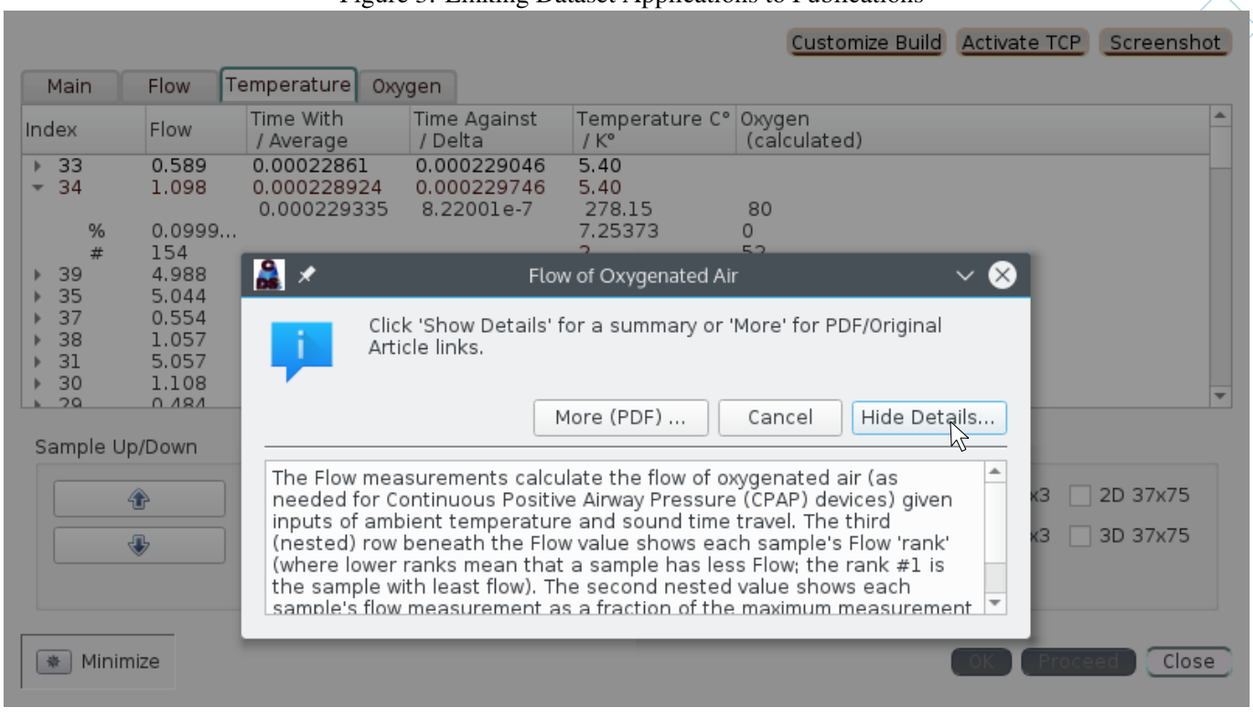
Further discussion on data microcitations depends on how data sets are structured, which is addressed in the next section.

2.3 Research Objects and Data Microcitations

The design of **AXF** assumes that many Research Archives, comprising of multiple publications sharing an academic focus, will also include open-access research data. The primary motivation for publishing research data is to ensure transparency and reusability — open-access data allows readers to verify that scientific/technical claims are warranted, and/or to reuse or incorporate existing data into new research. Open-access data has other purposes as well: data sets, for example, can serve as pedagogic tools helping readers understand publications' concepts experimentally and interactively (a good example is [23], which pairs data sets with its individual chapters to illustrate principles in data visualization). Moreover, the theory informing how data sets are organized can serve as a technical exposition of research principles and methodology. For all of these reasons, open-access data is an increasingly important part of the publishing ecosystem. This means that well-curated archives will often need to prepare data sets for data mining, alongside the preparation of text materials for text mining.

In contrast to text mining, however, it is not feasible, in the general case, to assign one single format (like **AXFD** or **JATS**)

Figure 3: Linking Dataset Applications to Publications



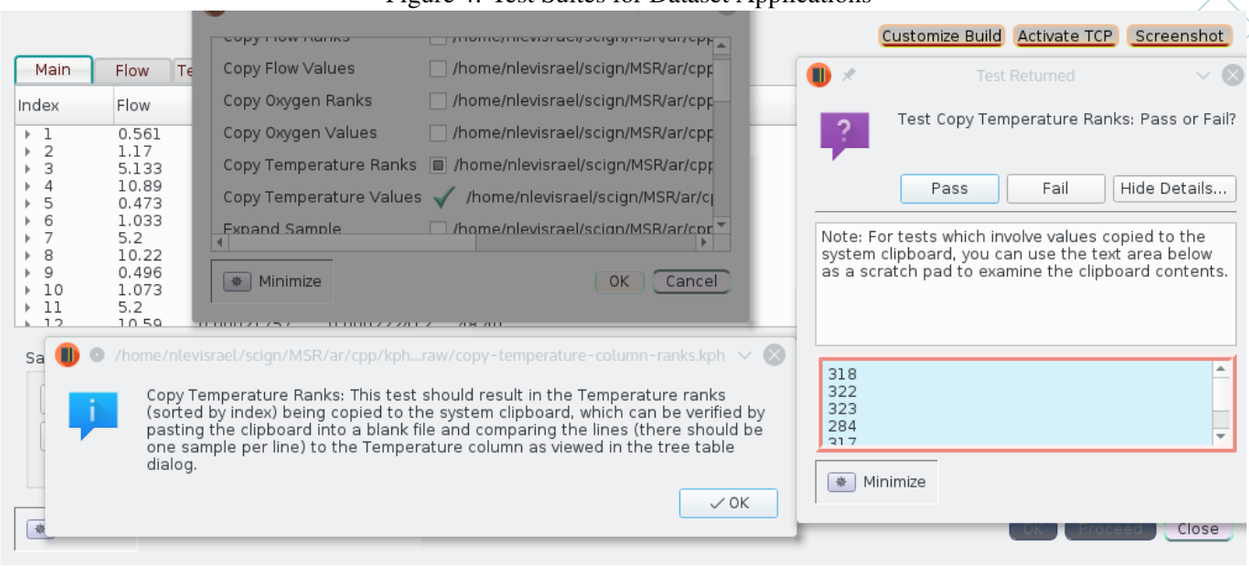
for all data sets published within an archive. Precisely how data sets can be annotated depends on the data models, programming languages, and analytic methodologies which they utilize. Because this variability prohibits a single data-annotation protocol from being required, **AXF** adopts a strategy of defining a rigorous protocol for **C++** code bases, which can then be emulated by other languages.

As outlined above, data citations refer to parts within a data set — such as individual data records, but also larger-scale aggregates such as table columns or statistical parameters. The complication when defining data citations is that a concept such as a table column, although it may have an obvious technical status as a discrete conceptual unit from the point of view of scientists curating, studying or reusing a data set, does not necessarily correspond to a single coding entity that could be isolated as an annotation target. It is therefore the responsibility of *code base annotations* to provide annotations for computational units — such as data types, procedures, and **GUI** components — that have an annotatable *conceptual* status relative to the data set on which the code operates. Often this will involve mapping one concept to several computational units (for instance, several procedure implementations).

For a concrete example of these points concerning data citations, consider the data set pictured in Figure 2, representing cyber-physical measurements used to calculate oxygenated airflow. The data-set application (interactive-visualization code deployed within the Research Object) displays tabular data via a tree widget (which functions as a generalized, multi-scale spreadsheet table), with tabular columns expressing quantities — such as air flow and oxygen levels — in several formats (raw measures as well as sample rankings and min-max percentages). Conceptually, these columns have distinct methodological roles and therefore can be microcited; indeed, the application links the columns to article text where the corresponding concepts are presented (see Figure 3). However, the implementation does not introduce a distinct **C++** object uniquely designating individual columns. Instead, the individual columns can be annotated in terms of **C++** methods providing column-specific functionality. In the current example, these methods primarily take the form of features linked to context-menu actions (copying column data to the clipboard, sorting data by one column, etc.). In general, rather than a rigid protocol for data-set annotations, **AXF** proposes heuristic guidelines for how best to map programming constructs to scientifically salient data-set concepts.

Defining an annotation schema for data sets can potentially be an organic outgrowth of software-development methodology — viz., the engineering steps, such as implementing unit tests, which are essential to deploying a commercial-grade application. This point is illustrated in Figure 4, which shows a **GUI**-based testing environment for the data set depicted in Figures 2 and 3. For this data set, the context menu actions providing column-specific functionality are also discrete capabilities which can be covered by unit tests, so the set of procedures mapped to the citeable concept correspond with a set of unit-test requirements. In this data set, these procedures are also exposed to scripting engines via the **Qt** meta-object system. In general, there is often a structural correlation between scripting, unit testing, and microcitation, so that an applications' scripting and testing protocol can serve as the basis for annotation schema. For data sets which use

Figure 4: Test Suites for Dataset Applications



in-memory or persistent databases, evaluable queries against these databases provide an additional grounding for annotations. In general, data-annotation should be engineered on the basis of a dataset applications' scripting, testing, and/or query-evaluation code. However, this is only a heuristic guideline, and **AXF** does not presuppose any data-annotation scheme *a priori*.

III Conclusion

This outline has focused on **AXF**, **HGXF**, and DigammaDB, which are complementary technologies designed in particular for creating and managing scientific or technical data sets. Although applicable for many use-cases, the initial motivation for implementing these technologies in their current form is to enable the curation of a Covid-19 data repository in conjunction with the volume *Cross-Disciplinary Data Integration Models for the Emerging Covid-19 Data Ecosystem*.

AXF uses a two-tier node structure similar to **LAF**; at one level is an extensible text-encoding methodology (outlined earlier in the context of **HTXN**), while a higher level defines annotations in terms of directed hypergraphs. Programmatically, **AXF** aims in the canonical case for a level of detail that is intermediate between linked-data-oriented projects like Web Annotations (which tend to focus mostly on isolatable semantic resources such as citations and named entities) and **NLP**-oriented paradigms such as **LAF**. That is, **AXF** does not natively serialize fine-grained **NLP** data at the level of individual words (the kind of data asserting semantic and morphosyntactic details: lemmatization, Part of Speech, dependency relations, and so forth), although it does support queries which return sentences as (unparsed) word-sequences. On the other hand, **AXF** offers some granular information about small-scale linguistic units, such as the role of non-alphanumeric characters, or **PDF** coordinates of sentence start and end points. In short, **AXF** occupies a unique space in the landscape of annotation tools at the intersection of application-development, **NLP**, and document-preparation requirements.

HGXF is first and foremost a tool for serializing ϛDB data sets, but can also be employed as a generic system for describing research data. Finally, ϛDB is a **C++** database engine which brings sophisticated hypergraph modeling capabilities to the **C++** and **Qt** ecosystem. Unlike many high-level databases, ϛDB is mostly self-contained and "transparent": it can be distributed in source-code fashion along with applications and/or research data sets.

Although a number of open-access Covid-19 data sets have already been published, the **CR2** repository can be a site for sharing new Covid-19 data as well as republishing existing data in the rigorous data-modeling framework established for **CR2**. Scientists can propose new data sets for inclusion in the corpus through an official GitHub repository. Depending on the size and nature of these resources, the submitted information will then be translated in whole or in part into hypergraph data models and the **HGXF** format; moreover, open-access publications, if applicable, may be annotated with **AXF** to properly link data sets to their corresponding research papers. **CR2** can then be used as a portal for hosting Covid-19 related manuscripts as well as the collection of research objects tailored to the pandemic and its proper medical and governmental response.

References

[1] Benjamin Adams and Martin Raubal, "A Metric Conceptual Space Algebra". https://pdfs.semanticscholar.org/521a/cbab9658df27acd9f40bba2b9445f75d681c.pdf

[2] Benjamin Adams and Martin Raubal, "Conceptual Space Markup Language (CSML): Towards the Cognitive Semantic Web". http://idwebhost-202-147.ethz.ch/Publications/RefConferences/ICSC_2009_AdamsRaubal_Camera-FINAL.pdf

[3] Khalid Belhajjame, *et. al.*, "Workflow-centric research objects: First class citizens in scholarly discourse". https://pages.semanticscholar.org/coronavirus-research

[4] Eran Bellin, "Riddles in Accountable Healthcare". https://streamlinehealth.net/wp-content/uploads/2016/02/Riddles-In-Accountable-Care_Defining-Your-Outcomes.pdf (summary presentation)

[5] Joe Bolt, ıet. al., Interacting Conceptual Spaces I: Grammatical Composition of Concepts. https://arxiv.org/pdf/1703.08314.pdf

[6] "COVID-19 Open Research Dataset (CORD-19)". 2020. Version 2020-03-13. Retrieved from https://pages.semanticscholar.org/coronavirus-research. Accessed 2020-03-20. doi:10.5281/zenodo.3715506 https://pages.semanticscholar.org/coronavirus-research

[7] Ludëk Eyer, *et. al.*, "Nucleoside analogs as a rich source of antiviral agents active against arthropod-borne flaviviruses". https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5890575/

[8] Project-Team ERABLE: "Activity Report 2018" (European Research Team in Algorithms and Biology, Formal and Experimental). https://raweb.inria.fr/rapportsactivite/RA2018/erable/erable.pdf

[9] Peter Gärdenfors and Frank Zenker, Theory Change as Dimensional Change: Conceptual Spaces Applied to the Dynamics of Empirical Theories. Synthese 190(6), pp. 1039-1058, 2013. http://lup.lub.lu.se/record/1775234

[10] John Gustafson, "Beating Floating Point at its Own Game: Posit Arithmetic", http://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf

[11] Nancy Ide and Keith Suderman, "GrAF: A Graph-based Format for Linguistic Annotations". https://www.cs.vassar.edu/~ide/papers/LAW.pdf

[12] Borislav Iordanov, "HyperGraphDB: A Generalized Graph Database". http://www.hypergraphdb.org/docs/hypergraphdb.pdf

[13] Andreas Jönsson, "AngelCode Scripting Library", www.AngelCode.com/AngelScript/

[14] Amy Neustein, *ed.*, *Advances in Ubiquitous Computing: Cyber-Physical Systems, Smart Cities and Ecological Monitoring*, https://www.elsevier.com/books/advances-in-ubiquitous-computing/neustein/978-0-12-816801-1

[15] Amy Neustein, *et. al.*, "Application of Text Mining to Biomedical Knowledge Extraction: Analyzing Clinical Narratives and Medical Literature", https://www.researchgate.net/publication/262372604_Application_of_Text_Mining_to_Biomedical_Knowledge_Extraction_Analyzing_Clinical_Narratives_and_Medical_Literature

[16] Tiago Nunes, *et. al.*, "BeCAS: biomedical concept recognition services and visualization". https://www.ncbi.nlm.nih.gov/pubmed/23736528

[17] Mark A. Parsons and Ruth Duerr, "Data Identifiers, Versioning, and Micro-citation", https://www.thelancet.com/action/showPdf?pii=S1473-3099%2820%2930086-4

[18] Enar Reilent, "Whiteboard Architecture for the Multi-agent Sensor Systems", https://www.thelancet.com/action/showPdf?pii=S1473-3099%2820%2930086-4

[19] Heshui Shi, *et. al.*, "Radiological findings from 81 patients with COVID-19 pneumonia in Wuhan, China: a descriptive study". https://www.thelancet.com/action/showPdf?pii=S1473-3099%2820%2930086-4

[20] Dietrich Rebholz-Schuhman, *et. al.*, "IeXML: towards an annotation framework for biomedical semantic types enabling interoperability of text processing modules". https://www.semanticscholar.org/paper/IeXML%3A-towards-an-annotation-framework-for-semantic-Rebholz-Schuhmann-Kirsch/1d72a56b6576117c62f388a5f2193965e4c7e293

[21] C. J. Rupp, *et. al.*, "Flexible Interfaces in the Application of Language Technology to an eScience Corpus". https://www.cl.cam.ac.uk/~sht25/papers/Rupp_et_al.pdf

[22] Matt Selway, "Formal Models from Controlled Natural Language via Cognitive Grammar and Configuration". https://aise.unisa.edu.au/wp/wp-content/papercite-data/pdf/selwaythesis2016.pdf

[23] Alexandru Telea, *Data Visualization — Principles and Practice*. https://www.semanticscholar.org/paper/Data-visualization-principles-and-practice-Telea/c853f4b8aee67cd749e03b3a4413769792222776

[24] Alexandru Telea and Jarke J. van Wijk, "VISSION: An Object Oriented Dataflow System for Simulation and Visualization". https://www.rug.nl/research/portal/files/3178139/1999ProcVisSymTelea.pdf

[25] Alina Trifan and José Luís Oliveira, "FAIRness in Biomedical Data Discovery". https://www.researchgate.net/publication/331775411_FAIRness_in_Biomedical_Data_Discovery

[26] Eric Winsberg, *Science in the Age of Computer Simulation*. https://www.press.uchicago.edu/ucp/books/book/chicago/S/bo9003670.html