WHITE PAPER

New Database Engineering and Archive Construction Technology to Accelerate Covid-19 Research

LTS (Linguistic Technology Systems) is founded by Amy Neustein, Ph.D., 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 and Conceptual Space Models for Covid-19** (Elsevier, forthcoming).

Executive Summary

LTS is building SARS-CoV-2/Covid-19 repositories to accelerate scientific discovery and therapeutic intervention. To do so, LTS is developing a pair of data and code repositories to advance both Covid-19 research and new methodologies for data-integration and AI research in general. The first of these repositories, the "Cross-Disciplinary Repository for Covid-19 Research" (CR2), is *empirically-focused* — aggregating published data sets into a common research platform so as to promote Covid-19-related data mining. The second of these repositories, "AI Methodology and Conceptual Space Theory" (AIM-Concepts), is *methodology-focused* — providing a collection of code libraries implementing techniques applicable both to data integration and to Artificial Intellegence research. AIM-Concepts prioritizes analytic methods and data representations — such as hypergraphs, fuzzy sets, and conceptual spaces — which have broad applications in software engineering as well as AI. Both CR2 and AIM-Concepts are companion resources to the forthcoming Elsevier volume (authored by the LTS team) titled *Cross-Disciplinary Data Integration and Conceptual Space Models for Covid-19.* In this volume, we will provide concrete examples of Covid-19 data integration/analytics by examining data sets included in the CR2 repository; we will likewise demonstrate analytic techniques by examining computer code included in the AIM-Concepts repository.

AIM-Concepts, with respect to methodology, will focus on Conceptual Space Theory, which is a valuable unifying framework connecting to both fuzzy sets as an AI analytic strategy and to hypergraph models as a data representation strategy. In addition to AIM-Concepts providing AI code libraries based on Conceptual Spaces, CR2 will likewise feature conceptual space models, applying this paradigm as a framework for describing research data. To this end, we will be introducing an updated version of Conceptual Space Markup Language (CSML) — expanded in order to serve as a general-purpose dataset-description language. CR2 and AIM-Concepts will be connected by a new hypergraph database protocol, which we are calling "Transparent Hypergraph Query Language" (THQL). The CR2 archive will include an implementation of this protocol used to aggregate CR2 data into a common format. Likewise, the methodology presented in the AIM-Concepts archive will be examined via demonstrations of how techniques and algorithms of the AIM-Concepts libraries can be applied to data hosted in a THQL database. The THQL technology has many concrete applications outside the context of Covid-19, penetrating vertical markets such as pharmaceuticals, manufacturing, fintech, healthcare, educational software, and bioinformatics.

One goal shared by both CR2 and AIM-Concepts is to provide a common programming infrastructure which facilitates the implementation of algorithms and GUI components that synthesize data across different scientific fields and methodologies. This common infrastructure is based primarily on Qt, a C++ application-development framework, and secondarily on established scientific/medical code projects such as CBICA (Center for Biomedical Image Computing & Analytics)

and ReproZip (a tool for packaging scientific software dependencies). In addition to utilities and build tools, the programming core for CR2 and AIM-Concepts will include C++ libraries to natively read and manipulate data in formats commonly used for clinical, diagnostic, and medical outcomes reporting, such as OMOP (from the Observational Medical Outcomes Partnership), PCORNET (defined by the Patient-Centered Clinical Research Network), FHIR (Fast Healthcare Interoperability Resources), RADLEX (Radiology Lexicon), and LOINC (Logical Observation Identifiers Names and Codes). While these data formats are normally used in the context of relational databases, the CR2 and AIM-Concepts programming tools allows developers to manipulate clinical data in formats natively recognized by data-mining and GUI code, which enables programmers to apply sophisticated algorithms to clinical data and also to present human users with convenient desktop-style applications to initiate and view the results of these analytic processes. Appendix 1 to this paper presents more information about Medical Outcomes code incorporated into CR2 and AIM-Concepts, with specific focus on integration radiological and Clinical Effectiveness data.

In the specific field of medical and diagnostic imaging, recent research has focused on how to effectively connect radiological data with medical outcomes and Comparative Effectiveness Research (CER). Traditionally, medical imaging formats such as PACS (Picture Archiving and Communication System) and DICOM (Digital Imaging and Communications in Medicine) have been largely separated from clinical records. As a result, it was difficult for researchers to cross-reference radiological data with clinical reports indicating subjects' subsequent treatments and corresponding outcomes. Attempts have been made to remedy this information gap by the semantic annotation of radiological findings, leading to the development of imaging-related ontologies such as RADLEX and SEDI (Semantic DICOM Ontology). One goal of CR2 is accordingly to generalize these semantic-imaging frameworks to the more flexible paradigms of *hypergraph* ontologies, which define schema on hypergraph instances by analogy to how conventional ontologies define constraints on directed, labeled, RDF (Resource Description Framework) style graphs. Compared to RDF ontologies, hypergraph ontologies are more expressive and also more complex, because there is a larger space of distinct graph elements which are candidates for semantic annotation (conventional ontologies are primarily composed of only two sorts of concepts — "classes," or node-annotations, and "properties," or edge-annotations.

Another crucial research domain for Covid-19 is vaccine development and immunization, particularly as scientific focus begins to shift from containing the pandemic to developing an effective vaccine. Conducting vaccine trials and vaccination campaigns generates a diverse spectrum of data profiles, ranging from observational reports of candidate vaccines' effectiveness during their development to reports on the extent and effectiveness of immunization projects, which often depend on socioeconomic, geographic, and political information as well as biomedical data. Moreover, scientific understanding of vaccines' effectiveness depends on complex assessment of vaccinated individuals' immunological response, which may incoporate data from blood tests, molecular imaging, cytometry, serological profiling, and so forth. Comprehensive support for vaccine-related data sets therefore requires technologies that can integrate and represent information from a wide range of laboratory and in-the-field methods. As with radiology/outcomes integration, this heterogeneous data profile has led to the design of RDF ontologies, notably the "VIOLIN" Vaccine Ontology. Here again, CR2 will generalize this technology to the domain of hypergraph ontologies (more information on hypergraph ontologies, with specific reference to radiological and vaccine-related lexicons, is provided in Appendix 2).

Introduction

In an effort to accelerate scientific discovery and therapeutic intervention, LTS is engaged in the curation of the CR2 ("Cross-Disciplinary Repository for Covid-19 Research") and AIM-Concepts ("AI Methodology and Conceptual Space Theory") repositories. The purpose of such repositories is

to centralize *disparate* Covid-19-related data and code into a common research platform. This code and data, as much as possible, will be marshaled into a common, hypergraph-based representation format, and a common C++-based programming environment. In building the **CR2** repository, LTS will develop and demonstrate new database engineering and dataset/repository construction technologies. Part I of this paper will outline the **CR2** and **AIM-Concepts** repositories; Part II will discuss the new technologies in greater detail. These technologies have deep market penetration-potential in many areas, including, but not limited to, pharmaceuticals, manufacturing, fintech, healthcare, educational software, and bioinformatics.

Some of the **CR2** code/data will be hosted on GitHub at Mosaic-DigammaDB/CRCR (for data aggregation) and Mosaic-DigammaDB/LingTechSys (for code previews). The latter repository includes preview code sampling database engine features, such as the logic for constructing a database in shared memory, encoding data types for persistence, and so forth. The data management tools developed for the **CR2** repository have a broad range of use-cases, and can be customized for different projects. Companies or research groups interested in a more substantial code preview are invited to contact LTS to discuss their projects and requirements in greater detail.

The main challenge when curating a data repository such as **CR2** is reconciling heterogeneous data formats. In response to this challenge, LTS has focused on hypergraph-based data models which can unify many different information structures into one common structure. In particular, **CR2** will introduce a special "Hypergraph Exchange Format" (**HGXF**) which can take the place of disparate tabular or graph file formats (comma-separated values, numeric python, spreadsheets, graph-network serializations, etc.), so as to merge data sets into a common *machine-readable* archive. In addition, **CR2** will introduce a new protocol for engineering hypergraph databases, called "Transparent Hypergraph Query Language" (**THQL**). Databases conformant to the **THQL** model will be able to export data in hypergraph-based formats, thereby generating data sets which can be used as published, citable Research Objects. **CR2** will demonstrate **THQL** via a new database engine called "DigammaDB" (or ϚDB) which serves as a "Reference Implementation" for **THQL**. In **CR2**, ϚDB functions as a prototype and reference example for **THQL**, used to curate data sets before their final form is exported into the main data repository. LTS can also customize commercial versions of a **THQL** engine tailored to the requirements of individual projects.

THQL is designed with a priority on application development. In particular, any instantiation of **THQL** should provide data persistence capabilities through (as much as possible) self-contained code libraries that can be included in source-code form within an overall application. **THQL** is designed to integrate seamlessly with native, desktop-style standalone applications. In short, **THQL** represents an unprecedented combination of native desktop-style software development and hypergraph database engineering.

Complementing **THQL**'s application-development focus, **CR2** will also introduce "Dataset Creator," (dsC), a new tool for curating research data sets. The main feature of Dataset Creator is its use of native software components (called "Dataset Applications"), allowing researchers to view, manipulate, and reuse research data. In sum, the typical Research Object built with dsC will include self-contained source code implementing a customized desktop application providing access to the accompanying data set. In addition to **GUI** code, each Dataset Application will supply "data-access" code libraries for parsing the raw data-set files, so as to obtain the information visualized within the **GUI** classes of the Dataset Application. These data-access libraries offer machine-readable access to the raw data, permitting subsequent researchers to reuse the data-access software libraries so as to transform, filter, or analyze the published data in the context of replication studies and/or novel research projects.

Development of **THQL** and dsC is concomitant with **CR2**; the **CR2** repository will provide a practical test-bed for validating this new technology. Accordingly, the following sections will

describe **CR2** in greater detail, followed afterward by sections offering more information about **THQL** and dsC.

Part I: The Covid-19 Repository

The Cross-Disciplinary Repository for Covid-19 Research

The sudden emergence of Covid-19 as a global crisis has cast a spotlight on computational and technological challenges which, in the absence of a catastrophic pandemic, would rarely rise to public attention. In particular, an effective response to the dangers of SARS-CoV-2 requires coordinated policy making integrating diverse modes of scientific inquiry. Genomic, biomolecular, epidemiological, socio-demographic, clinical, and radiological information are all pertinent to Covid-19. In this environment, it is important that the empirical foundations for expert recommendations — which in turn drive public policies of enormous social and economic consequence — be transparently documented and critically examined. The proper synergy between government and science depends on data centralization: given the gaps in our current Covid-19 knowledge, it is understandable that different jurisdictions will craft responses to the pandemic in different ways. There is no central authority with sufficient epistemic force to legitimize homogeneous mandates across the entire country. However, such policy differences should be a consequence of alternative interpretations of scientific knowledge or the diverse needs of local communities — rather than being a haphazard consequence of governments working with divergent, competing, and poorly integrated data.

The current administration, along with numerous corporate and academic entities, has clearly recognized the need for a more centralized paradigm for sharing Covid-19 data. For example, the White House spearheaded a scientific initiative to develop **CORD-19**, an open-access corpus of over 46,000 peer-reviewed publications related to Covid-19, which were transformed into a common machine-readable representation so as to promote text and data mining. Similarly, large institutions such as Google, Johns Hopkins, and Springer Nature have all implemented some form of coronavirus data-sharing platform targeted to both scientists and policy makers. However, these two aspects of the corporate/academic contributions to Covid-19 data sharing (exemplified by the **CORD-19** White House initiative and by institution-generated portals, respectively) have been incomplete, for opposite but complementary reasons. Specifically, **CORD-19** is highly structured and tightly integrated, but it focuses primarily on text mining and scientific documents, not *research* data. While it is possible to find data sets about Covid-19 through **CORD-19**, the techniques to do so are both cumbersome and non-scalable. On the other hand, projects such as the Johns Hopkins coronavirus "dashboard" provide accessible data sets, yet these projects are isolated and do not offer the level of structure and integration evinced by **CORD-19**. In short, an optimal Covid-19 research platform would merge the structural text-mining rigor of **CORD-19** with the data-centric focus of isolated projects that share Covid-19 data with the scientific community, policy makers, and the general public.

The benefit of **CR2** is that it can accelerate Covid-19 research by (1) pooling a diverse collection of data sets into a single resource which scientists can utilize; (2) serving as the prototype for larger research portals that can aggregate new Covid-19 data that will emerge from hospitals, labs, and academic institutions in the future; (3) formalizing a framework for aggregating patient narratives to accurately capture first-hand subjective symptomatology of the patient suffering from Covid-19; and (4) accelerating the implementation of novel data-integration and software-development technologies which can contribute to scientific progress vis-à-vis Covid-19 in particular, and biomedical/scientific computing methodology in general. These principles, in turn, will shape the design of **CR2**. An ideal data-sharing ecosystem should merge data from multiple sources, but should do so in a fashion which yields a machine-readable totality, analogous to **CORD-19**'s structuration

with respect to text mining. The merit of **CR2** therefore lies not only in the data which it will encompass but also in novel technology that it will concretize for constructing data repositories adhering to these goals — aggregating data, but also instantiating novel data-integration and database engineering strategies.

Given these varying goals, **CR2** can provide value at different scales of realization. Relatively small data sets 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. As a result, **CR2** can provide a testbed for implementing data-integration technologies which can scale up as needed. To fulfill this mission, **CR2** can aggregate relatively small data sets which have previously been published on academic and research portals, such as Springer Nature, Dryad, and DataVerse. At the same time, a more substantial (and not necessarily fully open-access) Covid-19 data-set collection would also be beneficial to the scientific and policy-making community. Ideally, then, **CR2** will be paired with a larger technology which shares 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 acquire as an integrated resource). The common denominator in both cases (whether the focus is on relatively smaller or larger data sets) is the importance of deploying novel and contemporary data-integration techniques to centralize Covid-19 research as much as possible. Accordingly, this summary will briefly explain how **CR2** can accelerate Covid-19 data integration on both a practical and technological level.

Methodology for Covid-19 Data Integration

As indicated above, pertinent Covid-19 data is drawn from multiple scientific disciplines. On a technological level, Covid-19 data is documented via a wide array of file types and data formats. This diversity presents technological challenges: if a Covid-19 information space encompasses files representing 25 different incompatible formats, users would need 25 different technologies to fully benefit from this data. In many cases, however, data incompatibilities are merely superficial — an important subset of Covid-19 data, for example, has a common tabular meta-model, even if the data is realized in discordant technologies (spreadsheets, relational databases, comma-separated-value or Numeric Python files, and so forth). Applying **CR2**'s technology, one level of data integration can thus be achieved simply by encoding 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 another table. In either case, it is reasonable to assume that a single abstract data format lies behind surface data-expression in patterns such as spreadsheets and comma-separated values (**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 often more amenable to graph-like representations. For instance, 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 decision to *administer* some treatment (e.g.,

a drug *from* a particular provider *with* a specific 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 much time elapses between admission to a hospital and discharge). These graph-like representations are a natural formalization of "patient-centered" data models.

Using **CR2**'s associated software (for example, importing Covid-19 data sets into a **THQL** database), a higher level of data integration can then be achieved by merging tabular and graph-like models into a single *hypergraph* format. A significant subset of Covid-19 data (or, more generally, any clinical/biomedical information) conforms to either tabular or graph structures; thus it is feasible to unify all of this information into a common framework. A graph-plus-table architecture is generally considered some form of Hypergraph model, and indeed **CR2** adopts 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 specialized computer code that would enable machine-readability of the **HGXF** files, and use them to create hypergraph-database instances. In short, **CR2** will promote Covid-19 data integration by translating a wide range of files into a common **HGXF** format.[1]

Hypergraph Data Models and Multi-Application Networks

As has been outlined thus far, via the **CR2** technology most Covid-19 data can be wholly or partially integrated into a single hypergraph framework, which accordingly simplifies the process of designing software applications and algorithms to analyze and manipulate this data. Specifically, software components can employ a single code library to obtain, read, consume, and store data, rather than needing to re-implement this logic for a large number of different file formats and/or database models.

Quality software (especially in the clinical and biomedical context) demands a balance between applications which are either too broad or too narrow in scope. On the one hand, doctors often complain that homogeneous Electronic Health Record systems (where every digital record or observation is managed by a single all-encompassing application) are unwieldy and hard to work with. This is understandable, because the clinical tasks of health care workers with different specializations can be very different. On the other hand, doctors also complain about software and information systems which are so balkanized that they must repeatedly switch between different, non-interoperable applications. In short, clinical, diagnostic, and research software should be neither too homogeneous nor too isolated; finding the proper balance between these extremes is, no doubt, a major challenge to the usability of electronic health systems going forward.

Against this background **CR2** demonstrates novel solutions to this problem: it focuses on the dimensions of data acquisition and management that are specific to individual scientific or medical specializations, while also identifying requirements that are consistent across domains. Scientific software generally needs to hone in on the data visualization and analytic requirements of

[1] **CR2** data sets are not required to compile all files to a hypergraph format; in particular, sciences requiring substantial quantitative analysis — e.g., biomechanics or genomics — express data via encodings optimized for relevant mathematical operations, and have parser libraries optimized for these specific formats. For these files **CR2** will generally provide an **HGXF** encoding supplying data *about* the original file, with information concerning the file type, preferred software components for viewing/manipulating its data, etc., so that the contents of non-**HGXF** files can be indirectly included into the **CR2** hypergraph-based ecosystem.

particular disciplines; for example, biochemists use different programs than astrophysicists. However, much of the code underlying scientific applications has nothing to do with these high-level models or theories, but is simply a fulfillment of basic data-management functionality — data storage, accession, provenance, searching, user validation, and so forth. In effect, the computational requirements of scientific and biomedical software can be partitioned into two classes: (1) domain-specific logic which reflects the quantitative or theoretical models of narrow scientific fields; and (2) data-management logistics which can be realized within a central access hub, rather than being re-implemented by each application in isolation.

In short, **CR2** architecture conceives of a central hub responsible for storing data and serving as a common access point — providing the "gateway" where authorized users can gain access to heterogeneous information spaces utilized by an array of domain-specific software applications. Since peer applications would not be directly responsible for data persistence or user identity management, they can focus on their specific data analysis and visualization capabilities. The central hub, serving multiple peer applications, is then a heterogeneous data space managing information from multiple applications while also tracking information about the applications themselves: helping users to identify and launch the software which is most directly relevant to their clinical or research needs at the moment. Meanwhile, because peer applications are jointly connected to a central hub, it is possible to implement scientific workflows where one application may send and receive data from its peers, allowing applications to complement each others' capabilities.

This multi-application networking architecture has precedents in some of the current database and engineering technologies. For example, many hospitals and medical institutions employ some version of a "Data Lake," pooling disparate data sources into a heterogeneous aggregate which is then accessed by multiple client applications. Similarly, Machine Learning and Artificial Intelligence often adopts "software agents" or analytic modules in contexts such as Online Analytic Processing, which again represent semi-autonomous software components sharing an originary data hub. Web applications, too, often act as domain-specific subsidiaries deferring operational requirements, such as user authentication or transaction processing, to a central web service. The limitation of multi-application networks in these existing contexts are that the software agents involved are generally "lightweight," with relatively primitive user-interface design. By contrast, the hypergraph technology introduced with **CR2** will support multi-application networking in the context of more substantial desktop-style scientific applications. In sum, the novel hypergraph technology developed by LTS offers a hybrid of the development methodologies employed for desktop scientific software and those applicable to multi-agent heterogeneous data stores, like a Semantic Data Lake. To accomplish these goals, **CR2** will utilize a new hypergraph database engine, coded in the **C++** programming language, which has a unique focus on supporting native **GUI** applications from the ground up, including persisting application state and storing application documentation within the database itself.

A New Paradigm for Data Sharing and Data Transparency

One exceptional feature of Covid-19 research is the extent of public attention focused on scientific discoveries about the disease. Academic and commercial research teams find themselves in an unprecedented situation where there is unusual pressure to accelerate the Research and Development process, and a concomitant demand for a novel level of transparency and openness. For example, vaccine development protocols are being fast-forwarded to take months instead of years, and information about the development process (such as trial results and scheduling) will likely be shared with the public much more than is standard practice. This new reality, in turn, calls for a commensurate evolution in the technology for public data-sharing.

In conventional biomedical R&D, much of the research data is proprietary, and revealed only in restricted contexts to select parties (such as the Food and Drug Administration). Data which *is* then publicly shared tends to be tied to published research papers in peer-reviewed literature, primarily read by a relatively small, specialist audience. All of this is changing with SARS-CoV-2: companies pursuing Covid-19 R&D (in the context of vaccine trials, for example) are facing pressure to publicly share their results as soon, and as transparently, as possible; and policy makers, scientists, and journalists are no less looking for quick access to research data directly, rather than circuitously through academic publications.

CR2 will introduce the new Dataset Creator technology targeted toward this new environment of direct, transparent data-access (dsC will be discussed in greater detail below). Data sets created via this technology therefore implement the "Research Object Protocol," which mandates that research data be bundled with code allowing scientists to analyze and manipulate the information in the corresponding data set. The Research Object framework was designed by a consortium of academic and governmental entities, such as the National Institutes of Health, to promote a paradigm for data publishing which prioritizes multi-faceted research tools over "raw" data that can be difficult to reuse in the absence of supporting code. In particular, Research Objects should be (as much as possible) *self-contained*, which means that scientists do not need external software dependencies to access and study the data — any special code which is a prerequisite to using this data should be included, alongside the raw data, as part of the Research Object itself.

Dataset Creator enables standalone, self-contained, and full-featured native/desktop applications to be uniquely implemented for each data set, distributed in source-code fashion along with raw research data (dsC Dataset Applications use **Qt** by default to provide native **GUI** classes, tailored to the relevant Research Object). Adopting such a data-curation method makes data sets easier to use across a wide range of scientific disciplines, because the data sets are freed from having to rely on domain-specific software (software which may be commonly used in one scientific field but is unfamiliar outside that field). In addition, Research Objects composed with dsC can be integrated into Multi-Application Networks (which are described in the previous section) because the dataset applications are autonomous native **GUI** applications that can easily interoperate via **Qt** messaging protocols.

Of course, most of the **CR2** data sets are previously-published work composed via older technology. Many of these resources, created with a wide range of software products, predate (or fail to apply) contemporary specifications such as the Research Object Protocol; not every **CR2** data set will have the full set of features described in this section. However, **CR2** will try to maximize the value of each data set by translating them into a **Qt**-based format — in particular, **CR2** will provide **Qt** code for reading **HGXF** files, as well as a **Qt**-based hypergraph representation library. Following the data integration methods outlined earlier, much of the **CR2** data can be merged into a **Qt**-based framework, which can facilitate the implementation of new, more sophisticated Dataset Applications as the information in **CR2** gets reused for subsequent research. **CR2** will also include **Qt**-based software, such as a customized **PDF** viewer, which will help researchers utilize the corpus in its entirety. For example, **CR2**'s **PDF** viewer will include special code to connect **PDF** files with data sets via "micro-citations," as discussed in the next section.

Supporting Data Micro-Citations to Improve Machine Readability

The **CR2** database engine supports annotating individual components of a database — a technology sometimes referred to as "micro-citation." Data micro-citations are references to integral parts of a data set, such as an individual table, or a single row/record or column in a table. Micro-citations allow these integral parts within the data set to be cited by and linked to publications, for purposes of machine readability and attribution. As an example, preliminary vaccine trials often

target a patient cohort selected for demographic or medical criteria matching the population who would most benefit from the vaccine. These criteria for selecting the cohort for the vaccine study are usually described in the texts of the articles. However, these criteria are also identified within the data set by socio-demographic data which is part of the information generated by the trial. By making these connections between criteria discussed in the article and those represented in the corresponding data set explicit, text and data mining can be *merged* as analytic tools targeting a data repository, so that machine reading is able to mine not just article text but the corresponding data.

One reason why micro-citations are important is that they clarify the scientific meaning attributed to data set elements by connecting these elements to scientific concepts and "controlled vocabularies" (such as a list of drug names, diseases, proteins, etc.). For instance, micro-citations allow table columns to be mapped to statistical parameters, enabling their empirical properties (such as min/max values and distribution) to be queried by text and data mining software. Likewise, **CR2** enables dimensional and measurement annotations to describe the empirical and experimental significance of the measured or calculated quantities which are stored in a database. Such quantity dimensions model the conceptual roles which particular parameters perform: e.g., the axiation "mJ/cm^2" (millijoule per square centimeter) indicates the intensity of ultraviolet light — any table (or other data aggregate) having a column or field with this dimension is intrinsically associated with observations or experiments pertaining to **UV** light. Consequently, to locate data sets relevant for research about the clinical uses of antiviral **UV** radiation, one method is to search for data fields dimensionalized in terms of joule or millijoule per square centimeter. As this example illustrates, data micro-citation — via annotations on data fields, statistical parameters, and table columns — is an important data-mining tool. In short, constructing micro-citations within a database serves two distinct benefits: (1) to aid data mining; and (2) to enable granular links (joining specific parts of articles to corresponding parts of the data set in the data set repository — analogous to hyperlinks between web pages) to be established between publications and data sets, making it *easier* for researchers to find the specific information most relevant to their own research.

Code Libraries and Data Sets for Analytic Methodology

As a companion to **CR2**, whose essential purpose is to present *empirical* observations concerning Covid-19, we are curating the **AIM-Concepts** archive, which is focused on code that supports Artificial Intellegence and concrete data-integration methodology. At the core of **AIM-Concepts** is a collection of code libraries implementing analytic techniques and representations used by **AI** researchers as well as by software engineers — in particular, different varieties of hypergraphs; fuzzy sets/logic; and conceptual spaces. In order to integrate this code into a common programming framework, the **AIM-Concepts** libraries are ported or modified when necessary, with **C** or **C++** as the primary development language and **Qt**/qmake as the primary development framework and build system. **CR2** and **AIM-Concepts** will be interconnected by employing **CR2** data sets as concrete examples for demonstrating how **AIM-Concepts** libraries may be used within software applications. Here are some of the methodological approaches (based in part on how Covid-19 research can provide concrete use-cases) which will be important components of **AIM-Concepts**:

Computational Epidemiology Methods in this category generally concern the simulation of disease transmission given a set of initial parameters (such as an infectious agent's reproduction rate and an average degree of person-to-person contact), often concomitant with empirical data, tracking how a disease has spread within some observable subpopulation. The empirical findings, therefore, suggest *a posteriori* which statistical model best fits the actual nature of the disease in question. The accuracy of this analysis depends, in part, on how well the observed subpopulation mirrors the susceptible population as a whole; but it also depends on the accuracy of the mathematical formulae translating initial parameters into projected epidemiological

simulations to be compared against *a posteriori* data. As such, concrete expositions of these mathematical frameworks — in particular, computer code implementing the calculations which drive a simulated model — constitute a computational asset in their own right. **AIM-Concepts** will include several code libraries that have been published as tools investigating different quantitative epidemiological models. Although some epidemiological simulations rely primarily on mathematical equations, most epidemiology libraries internally use graph-based models, often employing weighted graphs where edges denote, for example, the probability of viral transmission between people in close contact. As a result, distinct epidemiological models can sometimes be merged into a common analytic framework, using custom quantitative algorithms that are composed within a common graph-traversal framework.

Event Modeling Event modeling, sometimes called "Entity-Event Modeling," is an emerging data-analytic trend which focuses on events, rather than objects, as the most fundamental form of observation within a data space. Conceptually, the rationale for this paradigm is that every property which is attributed to some object can be tied to a specific event wherein the given property was measured, observed, discovered, or inferred. For instance, asserting that a patient is *infected* with SARS-Cov-2 implies that a *test* for SARS-Cov-2 was positive. Focusing attention on the *event* (viz., the test and its result) allows the data model to accrue information in a more detailed manner: in the case of SARS-Cov-2 tests, the relevant testing method/kit used; false positive/negative rates in the population; the time ellapsed between the onset of symptoms (if any) and the test being ordered and the interval until the results are provided; the observations or factors (such as the subject's exposure to an infected family member) which caused health care providers to order the test; and so on. Many of these details might be included in a conventional database as well; however, focusing on events allows the relevant information to be obtained more easily in that the event model supplies a temporal and operational organization for the underlying information. Event models are especially useful when temporal sequencing and duration are important considerations for uncovering scientific facts about the phenomenon being studied. In the context of infectious diseases, intervals such as the length of time between exposure and contagiousness, length of time between exposure and the appearance of symptoms, or the duration of hospital stays, are essential to our understanding of the disease's biology.

Fuzzy Sets and Fuzzy Logic Mathematically, fuzzy logic is often conceived in terms of replacing a simple binary logic ("true"/"false") with a more complex multi-valued logic, wherein properties may be true or false to varying degrees. Thus, on the surface, this introduces quantitative models (such as the calculation of the conjunction or disjunction of fuzzy predicates) as a substitute for purely logical reasoning. In practice, however, fuzzy sets often have the opposite effect: this theory gives rise to methods which can simplify empirical analyses by *eliminating* quantitative formulae — in particular, parameters of a continuous variable. Fuzzy set models tend to simplify data spaces by collapsing continuous quantities into discrete cases (e.g., *low*, *medium*, *high*), or grouping objects into similarity clusters. Via these operations, data models that depend on computationally intensive mathematical variables can be replaced by qualitative models, often representable in graph form (where graph edges may designate membership in a prototype-class, or the evolution of some observable according to several property classes, each of which collapses a spectrum of granular cases into a single prototype). Fuzzy methods can be shown to be effective simplications of complex data models by demonstrating that analyses conducted via qualitative reconstructions of a data space, for a given set a observations, are comparable to results obtained from more quantitative methods and/or are a good fit to empirical data. Research data sets which emerge from fuzzy methods can be qualitative models derived from quantitative data spaces, as well as code libraries which perform the relevant transformations.

Conceptual Space Theory Conceptual spaces, which have some similarities to Fuzzy Sets, have

likewise been proposed as a paradigm for modeling both scientific knowledge and Artificial Intelligence. Conceptual Space Theory is rooted in cognitive and linguistic investigations of how humans formulate and understand concepts (extending to scientific theories, and in particular how we "process" scientific information to infer facts about the world). Conceptual spaces have, therefore, been proposed as a language for representations or descriptions of scientific knowledge, with an emphasis on how scientific models are built up from individual conceptual parameters (such as points in space/time or notions of length, heat, speed/acceleration, electric charge, etc.). As scientific ideas become formalized, our intuitive conceptualization of spatial or observational quantities gets translated into physical, mathematical, or statistical details: scales and units of measurement, observational value ranges, statistical levels (Nominal, Ordinal, Interval, Ratio), and so forth. Formal implementations of conceptual spaces, therefore, focus attention on the dimensional and measurement properties of data parameters (e.g. scales/units of measurement) — information which is usually not made explicit in the publishing of data sets — and on how these parameters aggregate into conceptual units. Such aggregates include things like how two geographical coordinates that define a geospatial location, in Geographical Information Systems; or how shape and color, in combination, characterize visible objects, such as in image segmentation. One conceptual-space represenation is Conceptual Space Markup Language (**CSML**), which **CR2** will update in order to document scientific parameters within data sets. Other conceptual-space models and analytic libraries have been developed in the context of Artificial Intelligence, where the goal is to simulate the patterns of human conceptualization within **AI** engines; this work can then be incorporated into the **AIM-Concepts** repository.

Cognitive Discourse Analysis and Conceptual Role Semantics Cognitive discourse theory is similar to Conceptual Space Theory in investigating the overlap between conceptualization and scientific knowledge — or, more generally, all of our observed facts or empirical beliefs as they are encoded in language. Although it is obvious that most of our language is based in concrete beliefs about the world around us, this basic linguistic principle is not usually captured with full rigor within formal reconstructions of linguistic expressions. The essential paradigm in cognitive linguistics is the notion of cognitive *grounding* — that is, how every object and event included in a sentence connects to the speaker's fundamental understanding of the situation around them, and its relevant facts and observables. While all linguistic theories acknowledge grounding, cognitive analysis develops a detailed theory of how grounding works in all of its aspects: there are multiple dimensions of grounding, because we connect objects/events to situations in many different ways. For example, objects relevant to a given event play distinct empirical roles (the *agent*, which *causes* something — some change — to happen; the *patient*, which is thereby changed/affected; and potentially secondary participants such as the instrument by which the change is effected; the "benefactor," i.e., the object for whose benefit the change is caused, etc.). These conceptual differences are rigorously treated within Conceptual Role Semantics, which shares a similar philosophical orientation to Cognitive Discourse Analysis. As a practical tool for analyzing language, these two methodologies provide a matrix of classifications for objects and events (and their corresponding linguistic units) in terms of the situational background where every object and event (in a given discursive context) is perceived. Formally, these classifications then provide a layer of annotation which capture linguistic details at a cognitive level more rigorous than usually found in Natural Language Processing. **AI** methods in this field focus on automatically identifying cognitive-discursive patterns in language, although data sets can also be formed by manually annotating language samples according to cognitive-discourse and conceptual-role vocabularies. In either case (whether via manual or **AI**-driven annotations), Cognitive Discourse Analysis represents one emerging technique for the representation of natural-language assets — such as Patient Narratives — for the purposes of applying advanced text/data mining strategies (in the case of Patient Narratives, to extract critical patient symptomology often buried in circumlocutory discourse).

Adding Patient Narratives to Covid-19 Data

In addition to aggregating published data sets, CR2 may be used as a repository for collecting new Covid-19 information. With that in mind, we are prioritizing the design of a standard for storing and accessing natural-language text representing patients' subjective symptom descriptions, which is quite useful for diagnostic/prognostic assessments of patients infected by Covid-19.

Just as CR2 envisions a curation of published data sets for data mining to improve machine-readability of Covid-19 research, LTS also sees the benefit of a repository of patient narratives prepared for text mining, to improve machine readability of the open-ended symptom descriptions offered by patients. While CR2 does not need to specify how these narratives should be collected, it will implement a common representational format so that patient narratives can be pooled, similar to to how CORD-19 research texts are merged and encoded with a system that permits annotation.

In modeling patient narratives, this technology will be oriented toward the scientific-computing ecosystem outlined in the previous section. In particular, we assume that GUI-based desktop applications will be the primary instruments for data collection and analysis; this means that the encoding of patient narratives may, at times, need to be paired with GUI or multi-media content. For example, the software for patients to submit medical history information could also allow them to pair (text-form) narratives with graphics indicating the location of their pain or discomfort. Furthermore, the software could allow narratives to be accompanied by an audio file where patients could cough/speak into a microphone. Given this range of possible inputs, patient-narrative encodings must be flexible enough to include diverse multi-media content.

As described earlier, an information space adapted for multiple peer applications should encompass capabilities for saving application state (the current visual appearance of the program), which includes features for modeling instances of GUI classes. This technology provides the necessary infrastructure for managing patient narratives. For example, consider a multi-media intake form where patients may describe symptoms by placing icons (representing pain or discomfort) against anatomic silhouettes (head/body, back/front, extremities, and so forth). As patients use such a multi-media form, GUI application state corresponds to the patient's subjective symptomology; in this way the graphics-based representation of symptoms could then be incorporated into the overall patient narrative. This is an example of how application-persistence logic can be marshaled to the related project of curating patient narratives.

Part II: The new Database Engineering and Data-Set Construction Technology

Native/Desktop Application Development with THQL

As described earlier, THQL is a database engineering protocol which prioritizes data-persistence components that can be included in source code fashion within application-development projects. THQL is "transparent" in that all layers of data persistence and query processing logic are provided via self-contained source-code libraries. The complete database functionality can then be statically examined via the source-code files, and dynamically examined by running the client application through a debugger. Moreover, because all THQL source code is bundled with application code, THQL can be configured to integrate seamlessly into its environing client-application logic. For example, THQL can be extended to natively recognize client-specific datatypes as data fields, or to execute client algorithms as query parameters.

As a query *language*, THQL can be instantiated either by special languages with their own syn-

tax and semantics (analogous to SQL or SPARQL), or as an interface and pattern specified for a conventional language, such as C++. In the latter guise, THQL provides a common protocol for essential database tasks, such as constructing, updating, querying, and backing up database instances. Each procedure comprising the THQL protocol is assigned a specific role, so that the protocol can be abstractly modeled as a set of data-management roles mapped to corresponding procedure implementations. On this basis, custom query languages can be constructed by exposing each role-procedure to a scripting interface. For example, THQL's Reference Implementation (DigammaDB) exposes the protocol-specific procedures via a set of pointers to C++ functions. Consequently, parsing a query language is then rendered a straightforward process of mapping query expressions to the requisite procedures, whose corresponding handle can then be obtained via C++ interop.

As suggested by its name, THQL is centered around the operations to define and store hypergraph-form data: information which has several levels or scales of structuration. This means that the THQL protocol includes procedures for registering individual data fields (representing, in general, primitive types such as integers and decimals) in a database; aggregating fields into "hypernodes," or groups of interrelated information; connecting pairs of hypernodes by identifying a specific connection which they have; adding contextual details or annotations (via so-called *frames* and *channels*) which refine assertions of hypernode connections; and constructing "proxies" to database elements (e.g. hypernodes, frames, and subgraphs) which can be referenced (via unique identifiers) as individual data fields. Since proxies can then be aggregated into hypernodes in turn, THQL graphs can have, if desired, arbitrarily deep nested structures.

THQL also recognizes additional structures corresponding to conceptual details described earlier — for example, fields within hypernodes can be linked with dimensional attributes (e.g. scales and units of measurement) and identified as micro-citation targets. THQL likewise supports a genre of controlled vocabularies applying to hypernode-types and/or connection labels (called "dominions," for "Domain-Specific Mini-Ontologies"). Consequently, a graph can then be, if desired, configured to only accept hypernodes which conform to one of the dominion-defined types; and/or to only allow connections to be asserted between hypernodes when these connections can be labeled from a dominion-specific list of connectors. Less restrictively, graphs can be defined in a more free-form style but use dominions to filter or query nodes and edges.

Another feature of THQL, relevant to application integration, is the notion of configuring each database to support different "modes" of data persistence. It is possible to use THQL for completely in-memory data management, with no direct data persistence at all. This would be an appropriate solution when data can be read all at once from a static source, such as a data set. In this guise, THQL would be used to build a structural model of the data set, which can then be queried by application code. Conversely, it is possible to employ THQL as a continuously-updated data store, where changes to an underlying THQL graph are persisted to disk as soon as they are registered. Between these extremes, THQL graphs can hold dynamically changing representations of a persistent database, which are only incorporated into the underlying database when instructed by the client application. To support these different operational modes, THQL engines need the capability to represent each data type in several different formats, tailored to different stages of processing through which values are routed before they can be stored persistently.

In DigammaDB (the THQL Reference Implementation), persistent data storage is implemented via the WhiteDB database engine. WhiteDB is a hybrid graph/record database which allocates a persistent data store in shared memory (allowing each database to be accessed from multiple applications). ϛDB encodes hypernodes in WhiteDB records (although programmers can interface with the underlying WhiteDB instances if desired). ϛDB can then use WhiteDB's index and query mechanism as the foundation for its own higher-level query system. ϛDB provides a convenient interface for binary-encoding user-defined C++ types, so that arbitrary application-level data can

be stored via THQL (we can supply more documentation describing hypergraph-encoding with WhiteDB if needed). In addition — as an alternative to writing serializers for bonafide C++ types — it is possible to construct "ghost" types, which are QT data structures built via the "QVariant" class, so that all (or most) hypernodes in the corresponding database have a single C++ type — this technique is appropriate when, for example, the purpose of a THQL instance is to read and then update HGXF files with new information. To support different THQL operational modes, ϚDB organizes a stage-structure based on encoding WhiteDB values: at the ground level, values are simply pointers to in-memory C++ objects. At an intermediate level, values are encoded (via the QDataStream class in QT) into structures which recognize the WhiteDB encoding scheme but do not themselves interact with WhiteDB. Finally, values may be recorded as WhiteDB fields and records ready for persistent storage.

WhiteDB also allows databases to be shared (including being sent over a network) by storing all database information in a special file format. ϚDB instances can be shared via this same mechanism, although another option is to export the contents of a ϚDB database to HGXF files, which in turn form the core of a research data set representing the database contents at a specific moment in time. In this guise ϚDB works in conjunction with dsC, serving as the engine to construct a data set through which research data curated via a DigammaDB database can be published. Our dsC technology will be described in the next section.

Dataset Creator

Dataset Creator (dsC) is a framework for constructing data sets which include computer code based on the QT application-development platform. Dataset Creator takes advantage of the QT platform to construct Research Objects with exceptional GUI and data-mining capabilities. QT, the leading native cross-platform development toolkit, is a comprehensive framework encompassing a thorough inventory of programming features — networking, GUI implementation, file management, data visualization, 3D graphics, and so forth. Data sets based on QT require users to obtain a copy of the QT platform, but QT is free for non-commercial use and easy to install — importantly, QT is wholly contained in its own folder and does not affect any other files on the user's computers (in this manner QT is different than most software packages, which usually demand a "system install").

By leveraging the QT platform, dsC enables standalone, self-contained, and full-featured native/desktop applications to be uniquely implemented for each data set, distributed in source-code fashion along with raw research data. Adopting such a data-curation method makes data sets easier to use across a wide range of scientific disciplines, because the data sets are freed from having to rely on domain-specific software (software which may be commonly used in one scientific field but is unfamiliar outside that field). In addition, Research Objects composed with dsC can be integrated into Multi-Application Networks (which are described above) because the dataset applications are autonomous native GUI applications that can easily interoperate via QT messaging protocols.

Because every data set is unique, each Dataset Application will necessarily include some code specific to that one Research Object. However, dsC will provide a core code base and file layout which is shared by all dsC data sets by default. This common core is structured in part by the goal of developing Dataset Applications in a QT context; for instance, dsC projects are structured to use QT's "qmake" build system as the primary tool for compiling data-set code. The common dsC code therefore includes qmake project files which support compiling application with several build configurations. In this framework, data-set users are classified into several different roles — in addition to ordinary users (specifically, researchers who want to work with and draw information from data sets but have no development connection to these data sets themselves), dsC recognizes

roles for authors, editors, testers, and other users who are responsible for bringing data sets into publication-ready form to begin with. Depending on the administrative role, data set code can be compiled with additional features (e.g., unit testing features).

Another core component of dsC is LaTeX code that authors may use when preparing documents accompanying their data set. These LaTeX files encompass special functionality for defining code annotations and semantically significant points in article text, such as sentence and paragraph boundaries. This LaTeX code can be used in conjunction with a pre-processor that generates LaTeX files from a special input language. The goal of these text-processing technologies is to improve the interoperability between research papers and data sets. In particular, the LaTeX pre-processors (and subsequent LaTeX-to-PDF converters) generate HGXF files which store information about textual and PDF viewport coordinates specifying the location of semantically meaningful elements such as sentences and annotations. These files are then zipped and embedded in PDF files. With dsC, the resulting PDF files can then be loaded into a customized PDF viewer capable of reading the embedded HGXF data. This allows the PDF application to utilize the embedded information so as to provide a more interactive reading experience — for instance, viewing annotations or copying sentences via context menus, where viewport data maps cursor position to textual elements visible on the PDF page. These features provide an application-level interface between the PDF viewers (considered as GUI components) and the corresponding GUI components in Dataset Applications.

With proper customization, both the PDF viewers and the dsC Dataset Applications can interoperate, with PDF context menus calling up windows in the Dataset Application's GUI implementation, and vice-versa. For example, researchers reading the PDF version of a scientific article can launch the Dataset Application to explore some detail mentioned in the text. This is an example of where micro-citations are practically useful: any microcitable element in a document (such as a table, column, row/record, or analytic procedure formalized as a procedural asset associated with a data set) can be linked to a corresponding GUI element in the Dataset Application. For example, a statistical parameter — mentioned by name in the text, and perhaps represented in serialization within raw data — can be mapped to a GUI table column, and specifically the column header; this is then an annotation target, in the sense that for readers to gain more information it is proper to link mentions of the relevant scientific concept in article text to the column header as a graphical element that can be made visible. The link is operationalized by implementing a procedure to show the GUI window where the table is located, and ensure that the column header lies in the visible portion of the screen, as a response to readers on the PDF side signaling a desire for information on the annotated text element. In the opposite direction, database elements can be annotated with links that can used by the Dataset Application to launch a PDF window opened to the page and location where the corresponding concept is discussed in the article.

In order to properly model this semantic, viewport, and data set data integration, dsC uses a new document-representation format called HTXN (Hypergraph Text Encoding Protocol). With dsC, HTXN files are not only associated with data set assets; they are also machine-readable document encodings that can be introduced into publication repositories and other corpora oriented toward machine-readability. Authors can host HTXN files within data sets and link to them via services such as CrossRef, thereby ensuring that a highly structured, machine-readable version of their papers is available for text and data mining. The HTXN protocol is also useful for encoding natural-language content which becomes part of a data set as data assets in themselves; for example, patient narratives.

Further documentation of text-encoding methodology (applicable to both patient narratives and publications associated with CR2 research data) is available on the CR2 web site: such as here (this is a downloadable PDF link; visit the repository to see the larger archive structure). The document referenced in this hyperlink contains more information on dsC, HGXF, HTXN, and the other technologies discussed here.

Appendix 1: Hypergraph Ontologies

The notion of *hypergraph* ontologies generalizes the idea of *ontology* in the context of the Resource Description Framework (**RDF**). In the Semantic Web, an "ontology" is essentially a graph schema, defining metadata for any information that can be serialized or represented in graph-like fashion, as well as criteria which graphs must satisfy when they are used to encode a specific sort of data. To encode information about a *clinical trial*, for instance, one may require that graphs include one node (representing the trial itself) which is linked to other nodes representing patients enrolled in the trial, as well as one node (or a pair of nodes) representing the trial's start and end dates. These requirements constitute both a structural mandate on the graphs — specifying how the nodes should be connected — and a *semantic* requirement on the nodes, each of which must be tagged with metadata clarifying that the corresponding node embodies a trial, person, or date. An ontology then supplies a fixed vocabulary with which to rigorously declare this necessary metadata, as well as the relevant graph-construction rules. Insofar as graphs are employed to encode data, ontologies are analogous to Document Type Declarations (**DTD**s) in the context of **XML**.

In conventional **RDF** ontology, metadata is primarily associated with graph nodes and edges. In particular, nodes are referenced to Uniform Reference Identifiers (**URI**s), such as web addresses, and edges are labeled with concepts formally defined in one or more ontologies. Concepts which are used to annotat graph-edges, and which are given a fixed meaning in some controled vocabulary, are often called "properties." One special "is-a" property is often used to connect nodes with concepts that classify entities into one of many categories defined in an ontology, often called "classes." So most **RDF** ontologies are primarily composed of *classes* and *properties*, each assigned a unique label. The metadata for a given graph then links nodes to classes (for example, specifying that one node represent a clinical trial and the second represents a patient) and links edges to properties (for example, specifying that a patient-node is connected to a trial-node in that the patient is *enrolled in* the trial).

Hypergraph ontologies are similar to conventional **RDF** ontologies in that they likewise provide constraints and metadata for graphs, but hypergraph ontologies are more complex because hypergraphs are more complex than ordinary graphs — in particular, hypergraphs have different layers of structure. Whereas **RDF** nodes are intended to represent a single concept or value — such as a number, date, name, or **URL** — a *hypernode*, within a graph, typically encompasses multiple pieces of information inside it (often called *hyponodes*, *projections*, *inner elements*, *roles*, or just *nodes*).[2] In general, when analyzing hypergraphs it is necessary to distinguish at least to "tiers" of nodes, *hypernodes* and *hyponodes*, such that hyponodes are contained within hypernodes. As a result, hypergraph ontologies need a corresponding distinction for node and edge annotations: insofar as nodes are categorized via classes, and edges via properties, it is neccesary to stipulate whether these classifications apply to hypernodes, hyponodes, or some combination.

A further complication arises because hypergraphs represent nested or hierarchical structures, but these hierarchies are often partial or overlapping. For example, a patient is *part of* a clinical trial, but a patient is also included in other collections as well; for instance, a patient may be enrolled in a specific health plan (for insurance coverage). One technique for modeling overlapping hierarchical data via hypergraphs is to employ "proxies," which are digital identifiers encoding a multi-faceted concept into a single value that can be part of a hypernode (proxies are similar to "foreign keys" in **SQL**). So each patient, represented by its own hypernode, has an identifier which can be a

---

[2] The term "projections" is used by HyperGraphDB (see http://www.hypergraphdb.org/?project=hypergraphdb&page=RefCustomTypes); "roles" is used by Grakn.ai (see https://dev.grakn.ai/docs/schema/concepts); "inner entity" is used by the biointelligence project (see https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3555311/), where the corresponding notion of "external entity" refers to what in other contexts might be called other hypernodes linked to a given hypernode via hyperedges.

proxy-value for the patient assigned to a hyponode included in the hypernode encoding the list of patients enrolled in a clinical trail, and/or a hyponode included in the hypernode encoding the list of patients enrolled in a health plan. Hypernodes can then be linked to other hypernodes by virtue of proxies (e.g., the trial-to-patient connection), and also by virtue of overlap (e.g., the set of all patients both enrolled in a given clinical trial *and* enrolled in a given health plan).

In sum, compared to **RDF** — where there is one single sort of node-to-node relationship, based on whether or not an edge exists betweel nodes and how this edge is labeled — hypergraphs have multiple genres of node-to-node relationships: the relation between hypernodes and their inner hypnodes; between hypernodes and one another; between hyponodes in different hypernodes; and variations on each of relation-types wherein relations are defined indirectly through proxies. Moreover, in addition to hypernodes and hyponodes, hypergraphs may have additional levels of detail, such as *frames*, *channels*, and *axiations*. All of these details provide different "sites" where hypergraph annotations and metadata may be defined. This means that formats for describing hypergraph ontologies have to be more expressive than **RDF** ontologies, because **RDF** ontologies need only to classify metadata as node-annotations or edge-annotations; by contrast, hypergraph ontologies need to distribute annotations among multiple sites of graph structure.

An addition distinction within the Semantic Web is the contrast between *reference ontologies* and *application ontologies*. In general, *reference ontologies* are general-purpose schema intended to establish conventions shared by many different applications, to ensure that a large collection of data-producing software in a given domain is interoperable. By contrast, *application ontologies* are narrower in scope, more tightly integrated into applications that directly send and receive data. Ontologies of either variety are used by software to interoperate with other software: so long as two applications are using the same ontologies, it is possible to ensure that one application can understand the data produced by a second, and vice-versa. However, such inter-operability is only potential; it is the responsibility of programmers to actually implement code which produces and/or consumes data that conforms to the relevant ontology specifications. In general, application ontologies are structured in such a way that these concrete implementations are more straightforward to produce, compered with reference ontologies. Reference ontologies offer little guidance to developers vis-à-vis how to directly support the ontology within application code. Application ontologies, conversely, more rigorously describe the data structures which applications must implement in order to properly manipulate data structured according to the ontology's specifications.

Most of commonly used ontologies published by large-scale Semantic Web projects, such as the **OBO** (Open Biomedical Ontology) Foundry, are *reference* ontologies (see for example the **OBO** tutorial at https://github.com/jamesaoverton/obo-tutorial/blob/master/docs/obo.md). By contrast, hypergraph schema are closer in spirit to *application* ontologies, although sometimes different terminology is used to express this idea (see the HyperGraphDB tutorial at http://www.hypergraphdb.org/?project=hypergraphdb&page=IntroStoreData). In particular, there is often a direct correlation between hypernode types and the data types understood by code libraries implemented in a given programming language, and these type-level correlations need to be modeled by hypergraph ontologies. As a result, hypergraph ontologies require capabilities to model programming languages' type systems (by contrast, conventional **RDF** ontologies do not intrinsically interact with type systems outside a limited group of **rdf:type**s). Moreover, because programming language types cover a range of use-cases — including functional and **GUI** types — hypergraph ontologies need to directly model details about how applications interact with human users, including the visual objects (windows and their components) which users see on-screen, and the "operational semantics," or patterns of human-software interaction, which govern how people use applications to achieve their desired goals. For example, if a software tool exists in part to help researchers define a patient cohort from an aggregate of clinical data, a relevant application ontology can include a description of the concrete steps which users use, interacting with the software, to achieve

this end.

Because hypergraph ontologies are more directly structured around software operations and programming language type systems, these ontologies can be a more rigorous guide to those implementing novel software solutions than reference ontologies intended for Semantic Web data sharing. Hypegraph ontologies can therefore be especially important in the context of multi-application networks (which are discussed in the main paper) where collections of distinct software applications share a common technological and information infrastructure. In general, hypergraph ontologies are more appropriate for modeling data in contexts where custom software implementation is an important business requirement of the overall data space — that is, a particular data ecosystem is engineered with a priority on enabling developers to implement semi-autonomous software "agents" that provide specialized views or analyses of a data space shared by many such applications.

Appendix 2: A Hypergraph Ontology for Radiological Outomes