CURIO: TOWARDS ORGANICALLY CONSTRUCTED ACTIVE ONTOLOGIES FOR SCIENTIFIC RESEARCH

by

MATTHEW R. EAVENSON

(Under the Direction of Krzysztof J. Kochut)

ABSTRACT

Curio is a software framework intended to help researchers aggregate, integrate, organize, enhance, share, and analyze their scientific data. Most current research systems are built around predefined data management schemas that scientists must use in order to do their work, even if their data must be reorganized. Instead, Curio allows these schemas to evolve from researchers' personal organizational preferences by providing the ability to enhance research data with metadata, including properties, relationships, and types. This allows the construction of ontologies starting with research data, and without requiring experience in data modeling. Curio also supports binding "behaviors" (built-in actions, or pre-programmed plugins) directly to ontological patterns, providing more powerful ways to utilize enhanced data. Although this project specifically focuses on scientific research rather than personal or business use, the features it offers could be employed in any context where data management and utilization is desired. Furthermore, even though we mainly use glycobiology as our primary domain for case studies, the features present in Curio could support research in any domain.

INDEX WORDS: Scientific Information Management, Ontology, Active Ontology, Ontology Construction, Ontology Behaviors

CURIO: TOWARDS ORGANICALLY CONSTRUCTED ACTIVE ONTOLOGIES FOR SCIENTIFIC RESEARCH

by

MATTHEW R. EAVENSON

B.A., University of Georgia, 2005

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2015

CURIO: TOWARDS ORGANICALLY CONSTRUCTED ACTIVE ONTOLOGIES FOR SCIENTIFIC RESEARCH

by

MATTHEW R. EAVENSON

Major Professor:	Krzysztof J. Kochut
Committee:	John A. Miller William S. York I. Budak Arpinar

Electronic Version Approved:

Julie Coffield
Interim Dean of the Graduate School
The University of Georgia
May 2015

DEDICATION

For my family, friends, and other loved ones. All of you inspire me.

ACKNOWLEDGEMENTS

Firstly, I would like to thank the members of my committee, from whom I've learned so much. Thanks to my advisor, Professor Krys Kochut, for his seemingly limitless patience and guidance during my years as a graduate student. Thanks to Professor John Miller for his frequent encouragement and for the numerous tennis lessons, and Professor Will York for the interesting projects, the good humor, and for being patient with me as I attempted to grasp the basics of glycomics. Many thanks also to Professor Budak Arpinar for his helpful advice while working on my dissertation.

Furthermore, I greatly appreciated the constructive criticism frequently offered by Dr. René Ranzinger. Many of the projects I have worked on were greatly improved as a result.

Lastly, of course, I must express gratitude to my friends and colleagues. I certainly wouldn't have gotten this far without you.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1 INTRODUCTION 1

1.1 CONTRIBUTIONS 5

1.2 OUTLINE OF LATER CHAPTERS 6

2 BACKGROUND AND RELATED WORK 8

2.1 ONTOLOGIES 10

2.2 THE SEMANTIC WEB 12

2.3 PROVENANCE 15

2.4 GLYCOMICS 16

2.5 INFORMATION MANAGEMENT 17

2.6 THE MEMEX 19

2.7 PERSONAL KNOWLEDGE BASES 20

2.8 SEMANTIC DESKTOPS 21

2.9 RESEARCH OBJECTS 22

2.10 LABORATORY INFORMATION MANAGEMENT SYSTEMS 24

2.11 ELECTRONIC LABORATORY NOTEBOOKS 24

2.12 INFOGRID 25
2.13 FEDORA REPOSITORY 26
3 MOTIVATION 27
3.1 QRATOR 30
3.2 BIOSYNTHETIC PATHWAY BROWSER 32
3.3 GLYCOVAULT 33
4 QRATOR: A SYSTEM FOR CURATING GLYCAN STRUCTURES 35
4.1 KNOWLEDGE REPRESENTATION 39
4.2 STRUCTURE MATCHING ALGORITHM 40
4.3 CURATION WORKFLOW 44
4.4 SUBMITTING STRUCTURES 46
4.5 REVIEWING AND MATCHING STRUCTURES 48
4.6 CURATING STRUCTURES 51
4.7 COMMITTING STRUCTURES 51
4.8 IMPLEMENTATION 52
4.9 USER INTERFACE 53
4.10 WEB SERVICES 53
4.11 DATA STORAGE 54
4.12 RESULTS 54
5 ACTIVE ONTOLOGIES 57
5.1 BEHAVIORS 58
5.2 DATA PROVENANCE 66
5.3 WORKFLOWS 68

6 Organically Constructed Ontologies 70

6.1 Ontology Construction Methods 71

6.2 Resources 72

6.3 Resource Aggregation 76

6.4 Resource Organization 77

7 Curio Prototype Implementation 82

7.1 Challenges 84

7.2 Architecture 85

7.3 Persistence Module 87

7.4 Web Services 92

7.5 Plugins and Behaviors 92

7.6 User Interface 94

8 Evaluation Plan 98

8.1 Qrator 99

8.2 GlycoVault 101

8.3 Pathway Browser 103

9 Conclusions and Future Work 105

References 109

Appendices

A Persistence Module Data Model Example 115

List of Tables

Page

Table 1: Current glycan curation status 55

Table 2: BDL selector patterns 60

Table 3: Attributes of persistence module entities 89

List of Figures

Page

Figure 1: An overview of glycomics bioinformatics applications 29

Figure 2: Qrator's browse interface 31

Figure 3: Biosynthetic pathway browser 32

Figure 4: Overlaying experiment data on a biosynthetic pathway 33

Figure 5: An example of possible paths within a single O-glycan structure 41

Figure 6: The current canonical tree for GalNAc initiated O-glycans 42

Figure 7: The curation workflow 44

Figure 8: An example of an N-glycan structure that has been matched against a canonical tree 48

Figure 9: Visualizing a resource representing a glycan 64

Figure 10: An example of contextual actions 65

Figure 11: Data provenance scenario using the qRT-PCR workflow 67

Figure 12: An example of resources before and after metadata enhancement 74

Figure 13: Curio's data model 78

Figure 14: Curio's architecture 85

Figure 15: Persistence module's code generation capability for a standard software stack 88

Figure 16: Behavior pattern indexing structure 93

Figure 17: GlycoVault's data model, minus attributes for classes 102

CHAPTER 1

INTRODUCTION

Scientific research methodologies continue to evolve, incorporating advanced information management technologies and inventing new ones as required. Simply coping with the amount of data being generated continues to be a particularly difficult problem in the age of "Big Data", where terabytes of information can be produced in a single day. However, big data still has not resulted in big knowledge, big insights, or big wisdom. Analysis techniques are continually being developed and refined to extract valuable information from this vast body of data. Meanwhile, efforts are progressing to link information from different domains of scientific interest together, and to foster collaboration between research groups. These trends indicate that researching better ways to do research is a growing field in and of itself.

The ability to effectively manage data has always presented unique and difficult challenges, with scientific data consistently presenting the most challenging scenarios. The needs of scientists vary widely amongst different fields of research, as each field has its own specific requirements. This often leads to highly customized solutions for specific tasks, which can lead to information fragmentation. The needs of biologists keeping track of cell samples are different from the needs of climate scientists researching global weather patterns, or the needs of oceanologists studying oceanic currents and the behaviors of sea life, even though requirements sometimes overlap between fields. Many systems developed to analyze, store, or visualize research data under these circumstances are typically built with a very narrow focus in mind, and

rarely contain the ability to adapt to information from other domains without significant alterations. Others are meant to accommodate widely used data types or support generic visualizations for general-purpose usage, but do not offer the ability to customize their capabilities with features that certain domains require. Thus, many current scientific data systems are either too generic to satisfy the needs of more complex scientific domains, or are too narrowly focused to be of use when integration of research data from multiple domains is needed.

The tradeoff when creating flexible systems for dealing with scientific research is ever present, and presents a challenge when attempting to implement systems that can adapt to a wide variety of research requirements. Making a system too generalized usually limits its effectiveness within specific domains of interest. Conversely, focusing a system too exclusively on a certain domain often causes it to lose applicability in others. Flexibility limitations hamper many such systems when attempting to integrate data from external sources, or when attempting to read formats they were not designed for, which are typically from domains of interest outside their focus. While such systems could certainly be altered to incorporate new sources of data, this is a labor-intensive and cost-prohibitive process.

Another common tendency with existing scientific data systems is that they require scientists to switch to a different method of organizing and interacting with data than they are accustomed. Instead, these systems should be adaptable enough to accommodate the methods scientists prefer when doing their research. As Tim Berners-Lee commented in [1]:

> *"What I was looking for fell under the general category of documentation systems – software that allows documents to be stored and later retrieved. This was a*

> *dubious arena, however. I had seen numerous developers arrive at CERN to tout systems that "helped" people organize information. They'd say, "To use this system all you have to do is divide all your documents into four categories" or "You just have to save your data as a Word Wonderful document" or whatever. I saw one protagonist after the next shot down in flames by indignant researchers because the developers were forcing them to reorganize their work to fit the system."*

Many scientific communities do not have standardized methods for dealing with research data, leaving it up to individual labs, or even individual scientists, to develop their own standards for handling research data. While this often causes problems when attempting to integrate data, scientists should still not be forced to adopt a different method of data management without clear, objective advantages.

Although some may consider the formation of a data model to be a one-time process [2], data modeling in many instances is an iterative process that produces designs which are constantly being revised and corrected as new knowledge is discovered. Thus, expecting a data management system with a static data model focused on a single domain to be flexible is optimistic at best. Data models, especially for rapidly changing scientific data, should be able to be modified easily and grow dynamically to fit the needs of the scientists who use them.

Since research requires the ability to organize, analyze, and visualize data, scientists are frequently required to use multiple systems to make sense of the data they produce from experimentation. Information fragmentation can become a major hassle when desiring to view an overall picture of research progress or results. This places a burden on scientists to keep up

with where certain pieces of data are, and how the pieces fit together. Often, this can result in an incomplete representation of their research if part of the data is misplaced, or if one system containing data is incompatible with others. Valuable research directions or other serendipitous discoveries may be lost if data cannot be pieced together effectively. As the complexity and volume of research data continues to increase, it is realistically infeasible to expect scientists to keep track of where specific data fragments are held. Thus, better data aggregation and integration are required.

Also of concern is the large number of domain-specific data exchange formats that are present in many areas of scientific research. Integrating data encoded in different formats can be challenging within a single domain, let alone when combining data from several domains of interest. One example is GLYDE-II [3], which is intended to encode information about the structure of glycan molecules. GLYDE-II is used in the glycomics domain, and is written in eXtensible Markup Language (XML). Climate scientists work with another XML-based format developed as part of the Thematic Realtime Environmental Distributed Data Services (THREDDS) project [4]. NetCDF [5] is another format developed by the creators of THREDDS, and is intended for general use, as it excels at encoding array-oriented scientific data. These are but a few of the extensive number of data formats in use by researchers.

Data provenance is also an increasingly important part of scientific data systems. The ultimate goal of provenance in scientific research is to sufficiently document experimentation in order to ensure reproducibility. Provenance systems attempt to keep track of the path data takes as it is altered through transformations, analyses, and interpretations. Often, the unfortunate tendency in scientific research is to "publish and forget", with the data used to arrive at the conclusions presented being misplaced or deleted altogether. Thus, there is a pressing need for

systems that are able to keep up with research data well after papers have been published in order to ensure experiment reproducibility. Systems constructed with this in mind should also store and even generate provenance where appropriate. Provenance of this type could also assist in transferring responsibility for data or projects to someone else in the case of staff turnover or graduating students, which happens frequently in academic settings.

Workflows are also becoming increasingly important to scientific research when performing experiments or data analysis. Scientific workflows are typically process networks used as data analysis pipelines [6]. Workflow systems with a focus on scientific workflows have become more numerous in recent years, with specific examples including Kepler [6], Taverna [7], and Pegasus [8]. These systems support a wide variety of functionality, including data analysis, visualization, management, and modeling.

1.1 CONTRIBUTIONS

To address the aforementioned issues and attempt to better support scientists in their research, we propose a framework to allow researchers to create ontologies organically, with the ontology schema naturally arising over time in a bottom-up fashion. This process begins with research data and can adapt to the user's preferences when organizing data. This framework will also facilitate the binding of behaviors with patterns in the ontology, allowing plugins or other actions to be triggered automatically when certain conditions are satisfied. We believe these two additions can have far-reaching implications for scientific research, especially if provided in combination. The Curio framework could assist with rapid creation of scientific research platforms that allow researchers to share and organize their data more effectively, as well as

create workflows both for research tasks and mundane activities that support research, such as data backup.

1.2 Outline of Later Chapters

The remaining chapters in this dissertation describe the intended capabilities of the Curio framework, discuss possible applications, and provide information on our efforts so far at constructing a prototype system. Chapter 2 discusses related work in different areas including data management, the Semantic Web, provenance management, and scientific workflows. Background information is also provided for glycomics, the domain used for our case studies.

Chapter 3 gives more in-depth motivations for this project, including previous glycomics projects that served as a major inspiration for Curio. These projects support glycomics research and provide good examples of the types of scientific applications that Curio is well suited for. Efforts to build a glycan curation application, a biosynthetic pathway browser, and an experiment repository are discussed.

Chapter 4 discusses an application we developed for glycan curation in greater depth. This section gives detailed context of the usefulness of glycan curation in glycomics, and discusses the specific implementation details of the curation application. Curation efforts by glycobiologists are also provided.

Chapter 5 introduces the concept of an "active ontology". Active ontologies support the binding of behaviors with ontological patterns, and can facilitate a wide variety of research activities. This section describes what behaviors are, and gives possible applications for active ontologies, including sections on data provenance and workflows.

Chapter 6 introduces organic ontology construction, describing what it is, as well as how such construction could be performed with regards to scientific research. Resources and their role in data aggregation and organization are described, as well as the general data organization philosophy used in Curio.

Chapter 7 describes the current status of the prototype implementation of Curio. The architecture of the system is outlined, and a detailed description of the custom-built persistence module is given. The user interface is also described, and how it attempts to provide scientists with an intuitive environment in which to perform their research.

Chapter 8 deals with the evaluation of Curio by describing specific scenarios that are expected to be possible using this framework. Ideas for adapting previous projects are given, including the previously mentioned molecular pathway browser, experiment repository, and the glycan curation application covered in Chapter 4.

Finally, Chapter 9 closes with conclusions drawn from this research and future directions, including planned features for the Curio prototype.

CHAPTER 2

BACKGROUND AND RELATED WORK

Managing data is a basic necessity of scientific research, and the importance of finding efficient ways to organize, extract, and share data cannot be understated. However, acquiring data is not the point of scientific research. The analysis that happens after data is obtained, unveiling new information about what is being studied, is what adds to the sum of human knowledge.

Information science is a broad, interdisciplinary field, with many sub-fields that cover all aspects of dealing with data and information. Our efforts are concentrated within one of these sub-fields, Information Management (IM), which deals with collecting and managing information from diverse sources and then distributing it to various audiences. We also incorporate ontologies, another sub-field of information science which is also heavily involved with the Semantic Web.

The Semantic Web movement has championed the use of ontologies for various roles in scientific research, along with attempting to remake the currently human-readable World Wide Web (WWW) into a machine-processable Web. In a broad sense, the Semantic Web movement shares many of the same goals as IM, since they both aspire to capture, share, and use information effectively. Even though the original vision of the Semantic Web still remains largely unrealized [9], many useful technologies have been produced from it that can help with scientific research.

Data management systems have proposed numerous ways to store and maintain data, with varying degrees of success. Over time, systems dealing with scientific research have become more specialized, as the data formats they support have become more diverse. Highly focused systems are typically only useful in their respective fields of study, and do not offer much flexibility with integrating data formats outside their research area. This is readily apparent in two categories of systems geared specifically towards laboratory data management - Laboratory Information Management Systems (LIMS), and Electronic Laboratory Notebooks (ELNs).

Other broadly focused information management tools also have features related to those in Curio. The Networked Environment for Personal, Ontology-based Management of Unified Knowledge (NEPOMUK) project produced software that blends techniques from the Semantic Web into a desktop environment. InfoGrid is a graph database that has some unusual features compared with other databases of its type, and has certain features similar to what we propose for Curio. Lastly, Fedora Commons is an application that acts as a digital repository, but also has certain semantic features related to what is planned for Curio.

The terms *data*, *information*, and *knowledge* are frequently used interchangeably in literature. There are multiple definitions for each of these words, and sometimes they conflict or overlap. Their meanings are obviously not identical, though when discussing actions like recording scientific *data* or exchanging *data* between sources, *information* would suffice as a substitute. But substituting *knowledge* in either of those instances would not make much sense. Indeed, most often, both *data* and *knowledge* seem to be more frequently interchanged with *information* rather than with each other [10]. This may indicate that *information* is the most general of the three terms. However, Jones makes a compelling case for defining these words in

relation to one another. In his view, information is the key to understanding both data and knowledge. Information can be thought of as "data in motion", and knowledge is "information in action". The conclusion Jones draws from this is that knowledge is not a thing in and of itself, but only arises through information, and so can only be managed by managing information. Perhaps, then, so-called knowledge management systems should not strive to contain actual "knowledge", as we contain it within our minds, but should instead contain information that acts as cues to remind or reinstate knowledge if it is forgotten. Such a system would be a knowledge manager and an information manager at the same time.

2.1 ONTOLOGIES

An *ontology*, as the term is used in information science, is a specification of concepts and related metadata. Ontologies are typically used as a structural framework for organizing and encoding information for use by machines. Usually such information is limited to a single domain, but ontologies may also contain concepts spanning multiple domains of interest, especially in the case of upper level ontologies. Ontologies enable communities to agree on a common vocabulary by describing concepts and the relationships between them, and may also contain information about instances of these concepts and relationships. Furthermore, Uniform Resource Indicators (URIs) allow concepts and instances, referred to as resources, to be reused by other ontologies. URIs function as unique identifiers for resources within an ontology, and referencing concepts by their URIs is intended to be unambiguous. By using URIs and linking resources together with relationships, ontologies are also useful as bridges for integrating information from

different domains, as well as providing a means for different software applications to interoperate.

The term *ontology* was originally used to describe a part of metaphysics, one of the branches of philosophy, which deals with questions about the nature of reality and existence. However, in information science, ontologies initially appeared as a construct in the field of Artificial Intelligence (AI) that provided a way to encode knowledge for use by artificially intelligent systems. AI researchers started creating ontologies with different forms of automated reasoning, allowing new information to be inferred automatically by a machine from existing information, given a set of rules. Later, Tom Gruber more formally defined an ontology as currently used in information science, separate from its usage in philosophy [11]. Today, ontologies are used extensively in AI, the Semantic Web, systems engineering, software engineering, biomedical informatics, library science, and information architecture, among other fields.

Typically, an ontology consists of statements in the form of triples. Triples are expressions comprised of a *subject*, a *predicate*, and an *object*. The subject represents a resource, with the predicate representing a trait of the subject and relating the subject to the object. To present a simple examples of triples, we could describe the properties of a particular pizza:

<Pizza1> <isA> <Pizza>

<Pizza1> <hasTopping> <Pepperoni>

<Pizza1> <hasTopping> <MozzarellaCheese>

<Pizza1> <hasDiameter> 16”

Collectively, these triples describe a 16" pizza with mozzarella cheese and pepperoni toppings. The first triple states that Pizza1 is an instance of class Pizza, while the following two triples define relations between Pizza1 and instances of its toppings. The fourth triple is an attribute specifying that the pizza has a diameter of 16". Triples can express information at an organizational level (in the ontology schema), or at an instance level (as ontology instances and metadata).

2.2 THE SEMANTIC WEB

The Semantic Web is a collaborative movement led by the Word Wide Web Consortium (W3C), an international standards organization. The term "Semantic Web", coined by Tim Berners-Lee, the creator of the World Wide Web, describes a web of data joined together by relationships and processable by machines [12]. The W3C promotes common data formats on the World Wide Web and the Semantic Web, and is attempting to convert the current, mostly unstructured web into semantically structured information - a "web of data", as opposed to a web of documents. This will allow users to search for, share, and combine information more easily, and allow machines to automatically accomplish many of the same tasks without human intervention or direction. Berners-Lee expressed his original vision for the Semantic Web as follows [1]:

> *"I have a dream for the Web (in which computers) become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web", which makes this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy, and our daily*

lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize."

Unfortunately, much of this vision has yet to be realized, but research continues.

The Semantic Web makes heavy use of ontologies to formalize and share information. One of the Semantic Web's main goals is to allow easy interconnection of data from different domains, making ontologies an ideal tool to enable data integration through the use of relationships between concepts. When used in the Semantic Web, ontologies are typically serialized in either a variant of the Resource Description Framework (RDF) language, the Web Ontology Language (OWL), Turtle (Terse RDF Triple Language), or N3 (Notation3), a shorthand form of RDF. RDF and OWL began as variants of XML. However, they now have definitions independent of XML, and RDF is now analogous to UML in XMI.

Recently, there has been much discussion and adoption of Linked Data and Linked Open Data (LOD) in research, government, and academic institutions. In 2006, Tim Berners-Lee introduced a set of four principles to guide organizations in linking their data [13]:

1. Use URIs as identifiers for resources.
2. Use HTTP URIs so that people can look up those resources.
3. When someone looks up a URI, provide useful information using standard formats (RDF, SPARQL).
4. Include links to related URIs so that they can discover more resources.

These principles serve as a set of best practices for connecting data to the growing Semantic Web. By 2007, datasets in the LOD cloud contained over two billion RDF triples, linked together by more than two million RDF links. By September 2011, this had increased to 31 billion triples, linked by approximately 504 million RDF links.[1] However, an updated report in 2014 did not provide new statistics on the amount of triples present in the current LOD cloud.[2] Furthermore, the authors state that the numbers from 2011 were derived from dataset publishers' numbers, meaning that crawlers were not used. Even though no concrete statistics of triple numbers were given, they state that the number of datasets has increased by 274%. Thus, it appears that many scientific communities are starting to publish their research data as Linked Data.

There are currently a wide variety of non-proprietary data representation vocabularies available to ease the burden of data modeling. General-purpose vocabularies that are useful for scientific domains are offered by ontologies such as the Data Cube Vocabulary [14], SKOS (Simple Knowledge Organization System) [15], SIOC (Semantically-Interlinked Online Communities) [16], or EXPO (EXPeriment Ontology) [17]. Data Cube and SKOS are both vocabularies offered by the W3C. Data Cube is a candidate recommendation that is geared towards modeling multidimensional and statistical data, commonly used in scientific research, so that it can be linked to related concepts and data sets. Data Cube builds upon the SKOS vocabulary, which is intended to model classification systems such as thesauri, classification schemes, subject heading systems and taxonomies, among others. The SIOC vocabulary is designed to integrate online community information from sources such as blogs, wikis, forums, and newsgroups. Lastly, EXPO is an ontology that attempts to model scientific experiments in a

[1] http://lod-cloud.net/state/
[2] http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/

general manner, and subsuming other experiment ontologies that are subject-specific. EXPO links SUMO (Suggested Merged Upper Ontology) [18] with subject-specific experiment ontologies by generalizing the concepts involved in experiment design, methodology, and experiment results. Any or all of these ontologies could be useful in a scientific data-modeling context.

2.3 PROVENANCE

The field of data provenance, especially with respect to provenance in scientific research, is large. Multiple methods of utilizing provenance to enhance data have also been published. Uses of provenance include estimation of data quality and reliability, tracing audit trails of data, steps for replication of data, correct attribution of data, and simply for more information about data [19]. In the context of cloud research, [20] describes using provenance to detect anomalies in business applications, for content-based search, and even for access control purposes. Furthermore, [21] identifies two major types of data provenance: annotations attached to data, and data derivation paths (which reveal how data was constructed).

As mentioned previously, a large amount of research has been devoted to scientific data provenance, with much of that research dealing with scientific workflows [22-25]. Survey literature about provenance exists in many fields, as evidenced by reviews of provenance in e-Science by [26], scientific processing by [27], database provenance by [28], and a paper classifying different approaches to provenance by [29]. Moreau has combined information from all of the previously mentioned works, and has written a comprehensive review of systems and approaches [30].

The W3C has recently published a recommendation for a provenance data model, described in the PROV family of documents.[3] The PROV data model supports accessing provenance-related information expressed in other standards, expressing provenance of provenance, as well as attempting to ensure reproducibility, versioning, and representing activities and derivations (i.e., for transforming one entity into another).

2.4 GLYCOMICS

Glycobiology is still an emerging discipline, and is aimed at understanding the diverse biological functions of complex glycans and the relationships between glycan structure, abundance, biosynthesis, and function. Glycans participate in a broad range of cellular processes including cell-cell recognition and maintenance of cellular integrity. Glycomics is a branch of glycobiology that studies the structure and synthesis of glycans (sugar chains) expressed by cells, tissues, or organisms. Glycomics seeks to identify specific glycan structures and determine how their abundance varies in various tissues, cells and organelles, or as a function of cell development or pathology. Glycans are essential in living organisms, and their study could eventually result in cures for various cancers in humans, new drugs, advances in drug efficacy, or other discoveries. They are composed of monosaccharide residues that can be linked together in several different ways, often resulting in branched structures. This topological complexity distinguishes glycans from proteins and polynucleic acids (DNA/RNA), which are basically linear structures. The added complexity also makes determination and accurate representation of glycan structures a challenging endeavor.

[3] http://www.w3.org/TR/prov-overview/

Glycomics data is sometimes combined with data from transcriptomics, also referred to as gene expression profiling, which studies gene expression patterns in cells in order to create a more complete picture of cellular functions. A laboratory technique commonly used in transcriptomics is qRT-PCR (Quantitative real-time Reverse Transcription Polymerase Chain Reaction), which is used to measure gene expression levels. Paired with qRT-PCR data, glycomics data can be rendered in the form of a biosynthetic pathway, and gene expression levels can be analyzed at each step of the pathway. An example of a biosynthetic pathway browser annotated with actual experiment data has been described in [31].

2.5 Information Management

The information management field has generally been concerned with improving organizations as a whole instead of the individuals making up the organization. This has led to a differentiation between Personal Information Management (PIM) and Organizational Information Management (OIM). PIM is a relatively new area of study, though the act of acquiring, organizing, and maintaining personal information dates back to ancient times with the spoken word, when mnemonics were used to augment human memory [32]. Software applications for PIM are numerous, but they can also add to the growing problem of information fragmentation. As mentioned previously, information fragmentation is the unfortunate side effect of spreading information amongst multiple devices or applications, each with its own approach of storing and organizing information.

Personal Knowledge Management (PKM) is also relatively new, having its origins in a working paper from 1999 [33]. PKM combines ideas from Knowledge Management (KM) with

PIM, and is thus far an under-researched subject [34]. More recent studies into PKM (and KM) have concerned the role of Web 2.0 technologies and the Semantic Web in enhancing the user experience and overall utility of knowledge management [35-37]. However, there are some criticisms of PKM, most notably the question of whether PKM is simply a wrapper around PIM.

Dataspaces [38] are another approach related to Data Management and PIM. Dataspaces can be considered an umbrella term for many of the challenges in modern data management, but it is also an abstraction that intends to mitigate some of the problems faced with data integration. The concept emerged from current practices in data management, which often require multiple data sources to act in concert. Integrating data between multiple, often incompatible systems is a formidable challenge, and much research has been conducted in this area. However, the authors explicitly mention that dataspaces are not a data integration approach, but more of a "data co-existence" approach. To this end, the authors also proposed DataSpace Support Platforms (DSSPs), which are systems intended to allow developers to focus on the specific challenges within their applications, rather than worrying about integrating multiple data sources that are needed by such applications. There are four specific properties that distinguish DSSPs from traditional DataBase Management System (DBMS) approaches.

1. DSSPs are required to support all data within the dataspace, dealing with data and applications in a variety of formats, rather than leaving some parts out, as in a DBMS.
2. Unlike a DBMS, a DSSP is not in full control of its data, since often the same data will be available through another interface native to the system that contains it.

3. A DSSP is not required to support queries to every data source equally, and queries may be returned with approximate answers.
4. Finally, a DSSP must offer the necessary tools to create tighter data integration in a dataspace, as is necessary for the application.

A major goal of dataspaces is to reduce the amount of effort initially required to setup data integration systems. Such effort is delayed until it is required, forming a "pay-as-you-go" system. This allows the implementer to improve data integration as needed.

2.6 THE MEMEX

The history of information management is long, and stretches back as far as the written word. One of the earliest modern efforts to imagine a scientific data management system was Vannevar Bush's ubiquitously cited "Memex". This theoretical machine was described in his 1945 essay "As we may think" [39], in which he meticulously describes the technologies available in his day that could enable the construction of an electromechanical desk. This desk, the Memex device, would be capable of storing photographs, newspaper articles, scientific papers, or other items of scholarly or personal interest as microfilm. More importantly, it would be capable of accessing such records through "associative trails". Bush understood well the mind's propensity to relate concepts together, connecting them through semantic associations. Consequently, this essay was particularly forward thinking for its time, and in some ways predicted (and likely influenced) many later technological developments such as personal computers, hypertext, the World Wide Web and the Internet, and even Wikipedia. Strikingly, even now with our vastly superior

technology to that in Bush's day, we are still attempting to construct devices and software that could provide Memex-like functionality, and are still taking inspiration from his conceptual design. One would think, with the technology now available, we should easily be able to construct devices capable of meeting and in some cases far exceeding the imagined capabilities of the original Memex. However, such a ubiquitously flexible and useful device that allows powerful, standardized methods of preserving knowledge among multiple scientific domains has remained elusive.

2.7 Personal Knowledge Bases

The concept of a "Personal Knowledge Base" (PKB) has been around for many decades, likely originating from the aforementioned Vannevar Bush article "As we may think" [39]. The term itself however, seems to have been coined by Stephen Davies in a technical report in 2005 [40]. PKBs derive their name from three major notions, the first being that they are primarily for personal use. In this regard, they are much like Semantic Desktops, and draw much inspiration from Bush's Memex with their focus on individual use and intent to supplement a user's memory. Secondly, what is stored in a PKB is primarily "knowledge", as opposed to data or information. Davies defines data as raw symbols, devoid of meaning, and information as the parsing of that data via context, and shared conventions, into something meaningful. His definition of knowledge takes this one step further as the "true meaning" that has been internalized inside a person's mind. However, questions must be raised as to how effectively a PKB can truly store this definition of "knowledge". Thirdly, a PKB must function as a "base" of knowledge, which means that it acts as consolidated, integrated storage of knowledge. It must be

able to connect two pieces of knowledge without the kinds of artificial boundaries typically found in information systems.

2.8 SEMANTIC DESKTOPS

The notion of a Semantic Desktop arose from Semantic Web research, but it also takes inspiration from ideas presented many decades ago [39, 41]. Again, Vannevar Bush's Memex device is likely an inspiration, and shares many of the proposed functions of a Semantic Desktop. The primary goal of a Semantic Desktop is to provide a more personal experience regarding the organization, relation, and use of information, and even to supplement human memory. However, the term itself was first used in 2003, after being coined by Stefan Decker. The definition of a Semantic Desktop is given in [42], in which he echoes Bush's sentiments for the Memex:

> *"A Semantic Desktop is a device in which an individual stores all her digital information like documents, multimedia and messages. These are interpreted as Semantic Web resources, each is identified by a Uniform Resource Identifier (URI) and all data is accessible and queryable as RDF graph. Resources from the web can be stored and authored content can be shared with others. Ontologies allow the user to express personal mental models and form the semantic glue interconnecting information and systems. Applications respect this and store, read and communicate via ontologies and Semantic Web protocols. The Semantic Desktop is an enlarged supplement to the user's memory."*

There are many applications that fulfill aspects of a Semantic Desktop, even if they do not specifically claim to be one. Examples include MindRaider [43], TheBrain [44], and Gnowsis [45], though only Gnowsis claims to actually be a Semantic Desktop. All of these applications support the creation of mind-maps through inputting concepts and data, in an attempt to enhance a person's memory. Gnowsis was developed within the scope of the NEPOMUK project, which was initiated to build a Social Semantic Desktop framework. NEPOMUK utilized experts from academia, industry, and open-source backgrounds[4], and spawned an open-source software library with the same name under the KDE platform.

2.9 RESEARCH OBJECTS

Research Objects (ROs) are a recent concept arising from the need to support the exchange of experiment data and other scientific information on the Web [46]. ROs are structured aggregations of resources that are produced and consumed by standard Semantic Web formats and services, and are intended to connect seamlessly with the Linked Data cloud. Various types of scholarly data can be bundled into ROs, including research problems, datasets, workflows, persons performing experiments, and even publications resulting from research. These bundled packages of data are intended to facilitate experiment reproducibility, transferring all essential data for an experiment in a single unit.

Currently, the W3C has adopted ROs in the form of a community group, Research Objects for Scholarly Communication (ROSC). The community charter has stated that their primary goal is "to provide a platform for scholars, librarians, publishers, archivists and policy

[4] http://cordis.europa.eu/project/rcn/79390_en.html

makers to exchange requirements and expectations for supporting a new form of scholarly communication".[5] A number of principles have been outlined for ROs to follow:

1. **Reusable** – ROs support the sharing and reuse of data
2. **Repurposeable** – ROs should expose their constituent pieces for reuse
3. **Repeatable** – ROs should contain sufficient information to allow others to repeat the study
4. **Reproducible** – A special case of repeatability in which prior results are compared to results obtained by repeating a study to verify they are identical
5. **Replayable** – ROs should provide sufficient provenance to allow the ability to analyze individual steps in a study to see what happened
6. **Referenceable** – ROs should be citable, and should support unambiguous references
7. **Revealable** – ROs should support the ability to audit the steps performed in a study to ensure validity
8. **Respectful** – ROs should support user-visibility, credit, and attribution

2.10 Laboratory Information Management Systems

Laboratory Information Management Systems are software-based platforms that provide features for managing laboratory operations. Some of these features may include workflow management, data exchange interfaces, and integration with laboratory equipment. LIMS have also performed

[5] http://www.w3.org/community/rosc/rosc-community-group-charter/

many of the same functions as Laboratory Information Systems (LIS) and Process Development Execution Systems (PDES). Thus, the definition of a LIMS is somewhat controversial, as no two systems seem to offer the same capabilities and the definition varies from the perspective of the user [47].

The first generation of software to be labeled as LIMS was released around 1982, owing to the fact that no mention of "LIMS" was found before 1983 [48]. These early systems were mainly purposed with assisting in laboratory automation [48]. However, newer systems are emerging that attempt to be more configurable and adaptable. Such systems attempt to mitigate past inflexibilities of LIMS systems to adapt to widely varying requirements amongst scientific laboratories [49].

2.11 Electronic Laboratory Notebooks

Electronic Laboratory Notebooks are software intended to be a replacement for paper-based lab notebooks. However, an ELN, owing to its electronic nature, can offer many enhancements over its paper counterparts. Scientists, engineers, and other technicians use ELNs to document research, experiments, and procedures performed in a laboratory environment. Similar to paper notebooks, the data contained within ELNs are considered to be legal documents and may be used in a court of law as evidence. They are often referred to in cases of patent prosecution and intellectual property litigation. This requirement necessitates that they adhere to certain standards of record keeping and security. Some ELNs are built for specific purposes (Specific ELNs), while others are designed with broader applications in mind (Generic ELNs) [50].

Examples of ELNs are as diverse as the environments in which they serve. They can be standalone applications, client-server applications, or be entirely web-based.

eCAT [51] is a prominent example of an ELN, and is offered by ResearchSpace. eCAT is web-based and provides interfaces to allow scientists to manage their samples. Since the system is web-based, it can be accessed from anywhere, and has many features to support collaboration between researchers. It allows scientists to structure records hierarchically, much like in a file system, and also to hyperlink records together.

2.12 INFOGRID

InfoGrid [52] is described as a "modern, Java-based, dual-licensed web applications platform", and is offered by NetMesh. NetMesh claims that InfoGrid allows developers to create REST-ful web applications on top of either a relational database or grid storage technology. The basic component of data in InfoGrid is a MeshObject that can be "blessed" with one or more types (classes). InfoGrid was recently made open-source and contains a number of sub-projects, including a graph database, a user interface project (Viewlet), a Lightweight Identity project (LID, OpenID, etc.), and a project for integrating external data sources from the web (Probe). The Viewlet framework renders MeshObjects for consumption using concrete formats such as HTML. This is useful when dynamically creating graphical user interfaces or renderings for data elements stored in InfoGrid. The Probe framework allows data from external sources on the web to appear as a graph of MeshObjects that can self-update, thus mirroring the content of the data source as it changes.

2.13 FEDORA REPOSITORY

The Fedora Repository [53] is a storage system for digital content provided by DuraSpace, a non-profit organization, and is offered under a Creative Commons license. Originally developed at Cornell University, Fedora (Flexible Extensible Digital Object Repository Architecture) [54] was created for managing and preserving digital resources, and was inspired by the Kahn and Wilensky Framework [55]. It supports RDF, and the repository is integrated with the Mulgara triple store.[6] A SPARQL endpoint is also supplied for querying data. Fedora supports the storage of digital objects and associated metadata, and relations can also be created between objects. Also provided is the ability to associate "behaviors" with digital objects, with "services" being the example given by Fedora. Like InfoGrid, Fedora can reference external resources, and can deliver them through the repository when needed. Examples of use cases for Fedora include digital collections, e-research, digital libraries, archives, digital preservation, institutional repositories, open access publishing, document management, and digital asset management.

[6] http://www.mulgara.org

CHAPTER 3

MOTIVATION

Research methods in many, if not all, scientific fields are changing rapidly due to the swift evolution of technology. New instruments allow the collection of increasingly massive amounts of data, and gleaning useful information from it has become a monumental task. Analysis techniques are continually being developed and improved upon to support research, but scientists still need intuitive ways to organize it for human consumption, share data with other scientists, control access to their research, and perform further analysis. Better ways of organizing, curating, and sharing information are needed to ensure that scientists are not overwhelmed by the data they collect, and to ensure that the data is of sufficient quality and able to be disseminated effectively. Recent techniques in social media, including the ability to share opinions and information easily, have not yet been effectively translated to scientific communities and scientific information systems. It is generally accepted, however, that collaboration is extremely important in understanding experiment results, and discussing future directions for research. Moreover, collaborative data curation by experts can more effectively detect poor-quality data since more people reviewing data reduces the chance that errors will go undetected. While scientific collaboration is often necessary between labs or organizations, security concerns unfortunately sometimes prevent sharing data easily. Thus, more convenient methods of data sharing and collaboration are needed between groups researching common interests, along with the ability to adequately secure data according to the policies of the organizations involved.

The Semantic Web initiative is a great place to begin looking for solutions, as it aims to provide standards for information to be shared, reused, and combined, and to allow machines to make use of such data without the need for human guidance. The central components of this initiative are ontologies, typically described in either RDF, OWL, N3, or Turtle. Ontologies enable communities to agree on a common vocabulary, describe concepts and relations between concepts, encode knowledge relevant to their domain of interest, and to reuse those concepts by referencing them via Uniform Resource Indicators (URIs). Ontologies can be specific to a particular domain, or provide concepts and relations useful to multiple domains. Ontologies are also useful for integrating information from different domains, as well as providing a means for different software applications to interoperate.

Even though ontologies assist in allowing systems some flexibility when modeling data and help with data integration, ontologies are still largely utilized as static creations. They are also often used as alternatives to database systems when high performance is not necessary, and when the schema plays an important role in answering queries. Modern systems that store and manage ontologies typically do not provide the capability to self-populate data, provide analyses of the data they contain, or provide other advanced features that could make them more useful within their domains of interest. With appropriate extensions to an ontology management system, these capabilities and more can be added. This could provide enormous benefits for scientists who want powerful ways to integrate and organize their data, along with ways to visualize, analyze, or otherwise glean further information from raw data files. These capabilities could all be included in the same ontology management system, allowing seamless access to data analysis and visualization tools from the same interface used for organization.

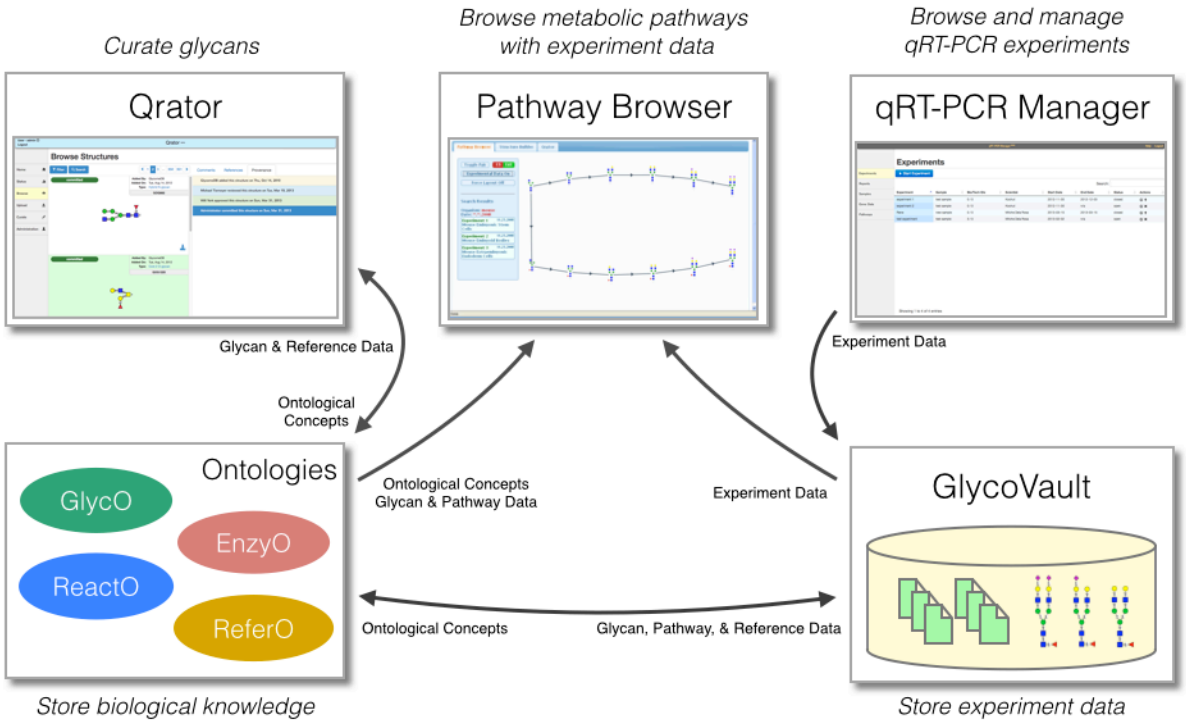

Figure 1: An overview of glycomics bioinformatics applications

As mentioned previously, designing a data management system with enough flexibility to incorporate new types of data, and allow the addition of new analysis methods, is not an easy task. Nevertheless, scientific research that handles a variety of different data types (laboratory data in particular) requires such systems. Multiple systems that facilitate data collection, storage, and analysis for glycomics and transcriptomics have been developed by the University of Georgia's Computer Science Department in conjunction with the Complex Carbohydrate Research Center. Prototype systems of this nature frequently require changes during their development to accommodate new information or ideas about how data should be stored. These changes can range from simple user interface alterations to fundamental changes in the underlying data model of the system. Designing a system that allows data models to be easily altered can help mitigate some of the inherent challenges of handling scientific data in especially

volatile scientific fields (fields whose conceptualizations change frequently). This could also assist in prototyping systems when the data model is not fully fleshed out, or is expected to change at various points in its development. Multiple systems developed within the CCRC's bioinformatics program feature data models that are unable to be easily stabilized because of frequently changing requirements. An overview of a few of these systems and how they relate to one another are shown in Figure 1.

Curio began as a summer internship project at Oak Ridge National Laboratory. The original plan was for a scalable virtual organizational system that allowed researchers to organize references to large amounts of data residing in different places, and then easily publish or share links to the data without having to deal with complicated security protocols in use by various organizations, or other security in place on different storage systems at ORNL. The system would have the capability to create both shared and private workspaces for data to reside in. Another major goal was for scientists to be able to add metadata to the referenced data, thus increasing its value. Logically, this idea was compatible with ontologies, and so the idea was extended to include linking data to one another via named relationships. Lastly, we considered the need to effectively use the data in Curio, and so behaviors were conceived that attached to specific data items, or patterns of data, allowing virtually any conceivable action to be performed contextually.

3.2 QRATOR

Data provenance and curation is important for scientific research in general, and glycomics is no exception. The Qrator application [56] is designed to assist glycobiologists with curating glycan

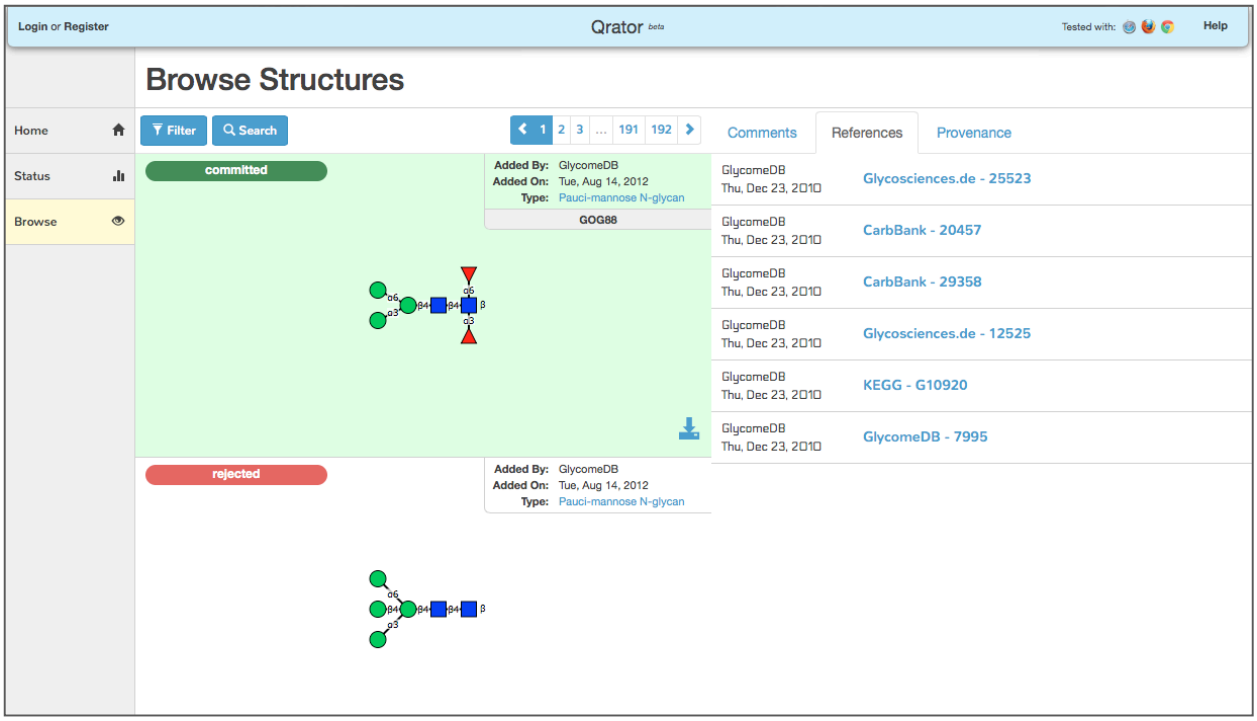

Figure 2: Qrator's browse interface

structures pulled from publicly available databases. Currently, all the structures in Qrator are imported from the GlycomeDB database [57], which also contains imported structures from many other databases. As shown in Figure 2, acquiring structures from GlycomeDB has distinct advantages, including numerous external references that link imported structures to other databases. These references are listed alongside structures as the curator is reviewing them. Provenance data about who submitted the structures, accepted or rejected the structures, or included the structures in the GlycO ontology [58] is also recorded and displayed if the user activates the appropriate tab. Qrator utilizes canonical trees for specific classes of glycans, which are encoded within GlycO, to further aid in curation. New structures are compared against these canonical trees to determine whether their structural configuration fits within the currently established tree structure. If a structure does not fit the canonical tree to which it belongs, this may indicate a problem with the structure and Qrator highlights the differences,

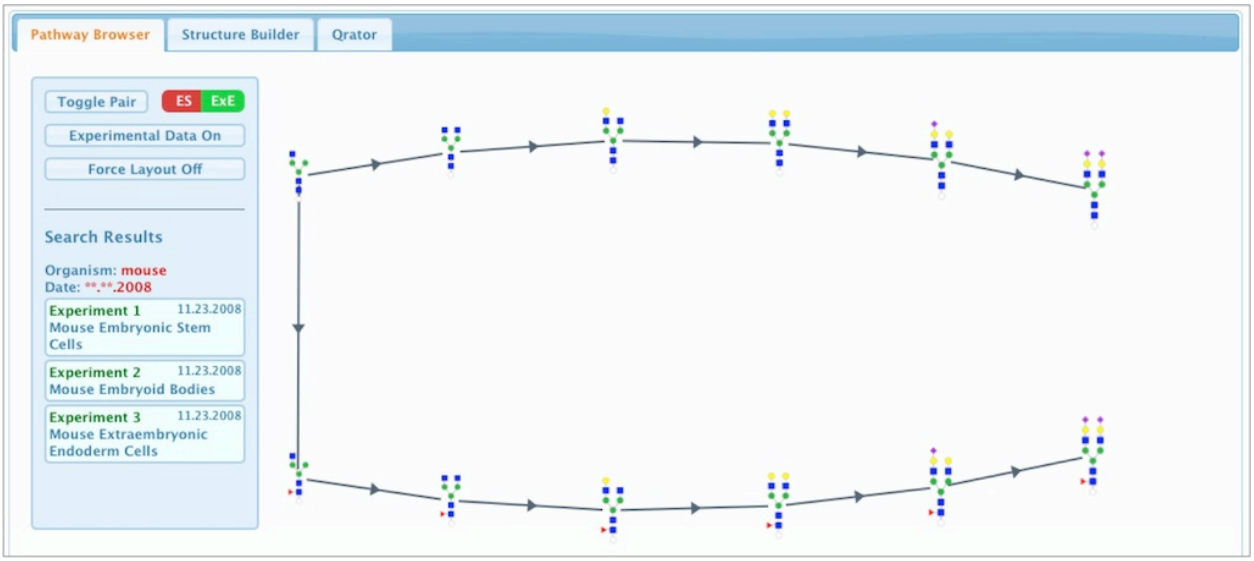

Figure 3: Biosynthetic pathway browser

thereby notifying the curator that closer attention is needed. If the curator decides that the structure is correct in spite of differences with the canonical tree, additions may be made to the tree. Qrator uses a two-stage review and curation process to make it more difficult for biosynthetically incorrect structures to be accepted. When structures are approved, they are included in GlycO.

3.2 Biosynthetic Pathway Browser

Visualization of biosynthetic pathways is another important requirement for glycobiologists. In particular, mapping expression data over biosynthetic pathways can be a powerful method of glycomics data analysis. As shown in Figure 3, we have developed an application that draws from research data stored in GlycoVault and combines it with pathway knowledge contained in GlycO to provide such visualizations. The Pathway Browser is capable of displaying glycan and transcript gene abundance levels at each step of a biosynthetic pathway. An example of this is

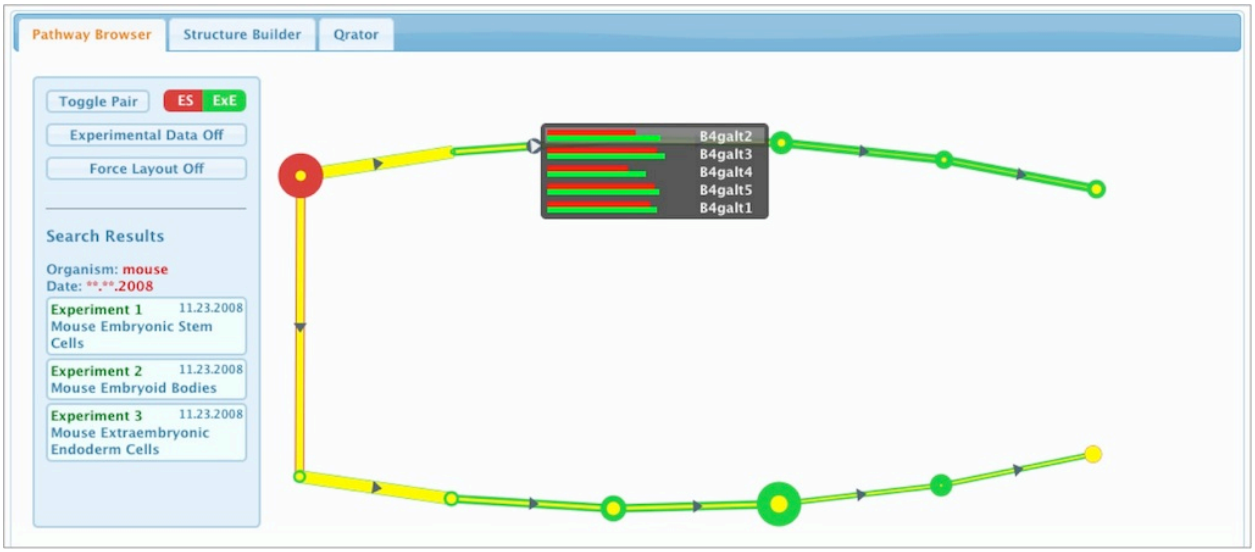

Figure 4: Overlaying experiment data on a biosynthetic pathway

shown in Figure 4. The data displayed is a contrast of two different types of cells that can be a combination of any two of the following: embroid bodies, embryonic stem cells, or extra-embryonic endoderm.

3.3 GLYCOVAULT

GlycoVault was designed to be a flexible system for storing experiment data and research workflows from various types of glycomics-related experiments. In general, GlycoVault supports glycobiologists as they collect and analyze data about glycans. Examples of such data include changes in a glycan's abundance level over a cell's life cycle, as well as correlating the abundance with gene expression levels for proteins that serve as enzymes in the biosynthetic process. Information about this process is available from multiple heterogeneous sources, including several Web accessible databases, published papers, and experimental data produced by glycobiologists. Thus, GlycoVault was designed to be able to accept information from

diverse sources. GlycoVault also manages information about experiment protocols, parameters, samples, and other concepts involved in data collection workflows. However, while GlycoVault was created with flexibility in mind when representing experiments, its primary purpose is to support glycomics experiments, while also being capable of storing transcriptomics, proteomics, and genomics experiments. Though, as with most systems that are designed for a specific purpose, its data model is configured for the kinds of data it is expected to store. For example, an *Experiment* has a set of *Tasks* that it follows, which are based on *Protocols*, with the *Experiment* based on an *Experiment Design*. While such concepts are applicable within glycomics and related fields, this may not be true in other areas of research. The kinds of experiments expected in glycomics are obviously much different than experiments performed in Computer Science, where experimentation is mostly done with algorithms, or on improving computer hardware and software. Thus, a system like GlycoVault, while designed to be flexible within a few related domains, still does not have the capability to accommodate experiment data from domains it was not designed for.

A satellite application designed to work alongside GlycoVault has been developed for Transcript Profiling data collection. The qRT-PCR manager supports record keeping of individual qRT-PCR experiments, and the biological samples used in these experiments. Provenance about samples and who performed the experiments is also recorded. The application supports the input of spreadsheet data generated from these experiments, and can upload data to GlycoVault for long-term storage. However, it contains specific functions that GlycoVault does not possess, including the ability to generate more complex reports about experiments, and the ability to view information about gene expression pathways.

CHAPTER 4

QRATOR: A SYSTEM FOR CURATING GLYCAN STRUCTURES

Glycobiology is an emerging discipline aimed at understanding the diverse biological functions of complex glycans and the relationships between glycan structure, abundance, biosynthesis, and function. Glycans participate in a broad range of cellular processes including cell-cell recognition and maintenance of cellular integrity [59]. As a sub-discipline of glycobiology, glycomics seeks to identify glycan structures and determine how their abundance changes in various tissues, cells and organelles, or as a function of cell development or pathology. The availability of robust and accurate collections of glycan structural data is a key element required for the success of this emerging field. Glycans are composed of monosaccharide residues that can be linked together in several different ways, often resulting in branched structures. This topological complexity distinguishes them from proteins and polynucleic acids (DNA/RNA), which are basically linear structures [60-63]. Unlike proteins and polynucleic acids, glycans are not synthesized using a template-based mechanism, but are generated by glycosyltransferases and glycosyl hydrolases that modify glycan structure by catalyzing the addition or removal of specific monosaccharide residues [64]. This structural and biosynthetic complexity makes the determination and accurate representation of glycan structures a challenging endeavor.

Collecting and storing data is an essential part of every field of scientific research, with databases and ontologies, among other methods, being used to capture scientific data. Ontologies are formal, shared vocabularies of concepts and relations that represent knowledge

within a domain, and are increasingly utilized for capturing scientific knowledge, as evidenced by the popularity of sites like OBO Foundry.[7] Yet, the amount of information that has been recorded in databases and ontologies has not kept pace with the recent surge of data acquisition in biological and biomedical research. Furthermore, the quality of archived data is often compromised as a result of errors in data exchange, translation and annotation. When glycan structures are recorded in databases they are translated into database specific, non-standard formats [65], often leading to inaccuracies ranging from simple typographical errors to fundamental inconsistencies in the structural representation. Thus, standardization and curation of glycan structural representations are important issues that must be addressed for the effective interpretation and utilization of laboratory data and associated metadata, particularly when populating databases or ontologies.

The Complex Carbohydrate Structural Database (CCSD), also called CarbBank [66], established at the Complex Carbohydrate Research Center (CCRC), was the first major international effort to systematically archive structural and meta information of complex glycans. After the discontinuation of funding for CarbBank, other glycan structural databases, including GLYCOSCIENCES.de [67], the Consortium for Functional Glycomics Glycan Structure Database [68], KEGG Glycan [69], the Bacterial Carbohydrate Structure Database (BCSDB) [70], and GlycoSuiteDB [71] were established partially using the CCSD as a source of core data [59]. Unfortunately, CCSD also contained its share of errors, which then propagated to the databases that make use of its data [72]. More recent bioinformatics efforts at the CCRC have emphasized the establishment and population of ontologies, such as GlycO [58], to represent knowledge pertaining to glycan structures. However, the set of glycans to be represented is

[7] http://www.obofoundry.org

potentially very large, encompassing many complex, branched structures that are composed of many residues linked together in distinct ways. Therefore, manual data entry is prone to the introduction of errors, which can be mitigated by the development and implementation of effective curatorial tools.

An effective curation tool, whether used for populating ontologies or more traditional databases with glycan structures, requires a highly intuitive interface for reviewing glycan structures. Such an application should assist scientists in identifying and eliminating errors, and also provide provenance information when possible. Subtle changes in the linkage between two monosaccharide residues can completely change the physical and biological properties of the glycan. As the human eye can easily miss subtle errors of this nature, computational methods are extremely helpful in highlighting potential errors in the representation of the glycan.

The curatorial framework described in this paper takes advantage of canonical trees, each of which represents the emergent structural features of a set of related glycan structures. This concept was first implemented as a "GlycoTree" used to predict the retention times of glycan structures [73]. Such trees are also formally implemented within the schema of the GlycO ontology and as "composite structure maps" in the KEGG database. These are powerful canonical representations assembled by overlaying many glycan structures of a particular class (e.g., N-glycans) to generate a superstructure containing all of the different residues and residue-to-residue linkages included in the glycan structures that were used to generate the tree. Canonical trees are generated using a set of naturally occurring, biosynthetically related glycans, rather than chemically synthesized glycans. Each glycan corresponds to a subset or sub-tree of the canonical tree. Moreover, the structure of the GlycO ontology allows us to place canonical trees within the context of other information that we use to support the curation process. This

information consists of all the individual structures that make up the tree, as mentioned previously, as well as any references or other meta-information they contain. As curation progresses the amount of information increases, which in turn supports further curation. To date, the application of canonical trees for the curation of glycan structures has not been extensively explored, and software applications that use such trees to aid in the curation process are not currently available.

Most of the curation efforts published in the literature (not limited to glycan structure curation) rely on completely manual curation. Curation in GlycoSuiteDB or BCSDB is performed manually by trained glycobiologists, and data is based on scientific literature. The developers of these databases assert that disallowing direct data entry by researchers ensures consistency and integrity in the data, but Qrator offers users the option to upload their own structures for curation. Moreover, their meaning for curation seems somewhat different from ours since they curate structures by extracting them from literature results, while Qrator allows a scientist to submit a structure directly, verify its structural correctness against a canonical tree, and then present the structure to experts for their approval or rejection. GlycoBank [74] is a system that has been used for the curation of glycosaminoglycans (GAGs). The system contains a repository of GAGs, their references, and classification material. Proposed entries or modifications are reviewed and approved manually by GlycoBank appointed curators with no apparent computer assistance, except for the actual display of pending additions. WikiPathways [75] offers wiki-based pathway curation by a community of scientists, allowing domain experts from all over the world to directly collaborate on improving pathway diagrams. However, this system is geared towards signaling pathways and pathways leading to cellular metabolites rather

than glycan structures and does not feature any algorithms to aid scientists in curating glycan specific pathways.

Henceforth, we describe a novel approach for curating glycan structures with the help of a web-based application, Qrator, which assists researchers in the curation of new structures by using existing knowledge of previously curated glycans. After passing through a two-stage human curation process, approved structures are available for download, and stored in the GlycO ontology.

4.1 KNOWLEDGE REPRESENTATION

Glycan structures approved at the end of the curation workflow may be deposited into the GlycO ontology, which contains knowledge about many types of biomolecules including glycans. A key feature of GlycO is the representation of complex glycans as collections of canonical residues that are defined in terms of their local structures and context within a canonical tree. Within GlycO, each canonical tree corresponds to a particular class of glycans (e.g., N-glycan or O-glycan), such that all known structures of that particular class are represented as subtrees of the tree. These trees are constructed by identifying the union of residues and links contained in a validated collection of structures of a particular class. When the tree is generated using a set of biosynthetically related glycans, it not only provides a convenient method to represent glycan structures, but also constitutes a concise basis for inferring a subset of the rules for glycan biosynthesis. That is, adjacent structures in the overall biosynthetic pathway often correspond to valid subtrees that differ by the presence of a single residue; structures that cannot be mapped to a specific subtree cannot be generated by the biosynthetic mechanisms that give rise to the

canonical tree. However, canonical trees are not static constructions. Although they grow over time, canonical trees tend to become more stable as structures are added to GlycO, as subsequent inclusion of new structures is less likely to require extension of the tree by the addition of new residues.

To supplement the structural knowledge contained in GlycO, we have created another ontology named ReferO that contains meta-information for each glycan structure in GlycO. This includes references to other databases that contain the same glycan structure, publications that describe or cite the structure, biological source information and provenance information that is collected at each stage of the curation workflow.

4.2 Structure Matching Algorithm

After a glycan is submitted to Qrator, one of the early steps in the curation process involves matching the glycan against a suitable canonical tree in order to establish its conformance to existing structural knowledge. In order to establish how well a candidate glycan matches within a canonical tree, we check if all of its paths are included within the canonical tree. These paths are enumerated by starting at leaf residues (residues that have no successor residues) and following the linkages to their predecessors, all the way back to the root residue of the glycan. At present, Qrator does not match glycan fragments.

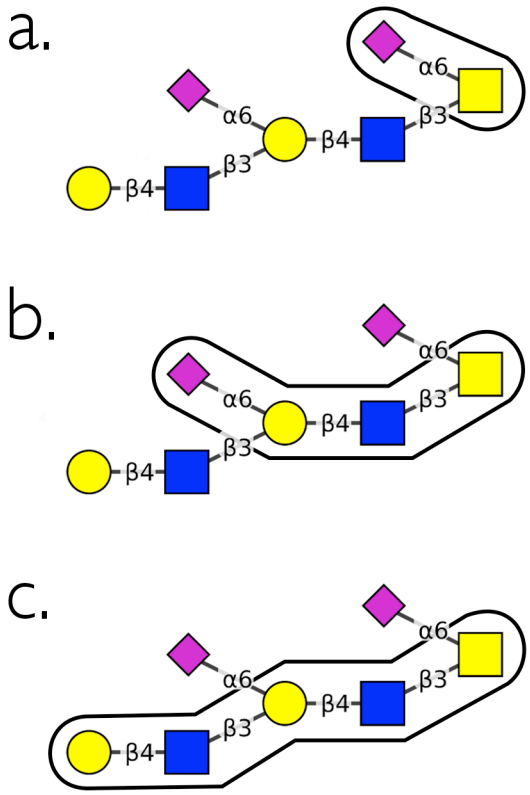

Figure 5: An example of possible paths within a single O-glycan structure

A path in a candidate structure is fully included in a canonical tree if all of its residues and linkages have the same corresponding residues and linkages in the canonical tree. For example, consider the O-glycan paths shown in Figure 5. This glycan has three paths, each starting with a leaf of the structure tree on the left and leading back to its root on the right. Two of the paths (Figure 5a and Figure 5c) are fully included in the canonical tree shown in Figure 6. The third one Figure 5b is included only partially, since the α-linked sialic acid residue connected by a 1-6 linkage (the leaf residue) is not present in the canonical tree.

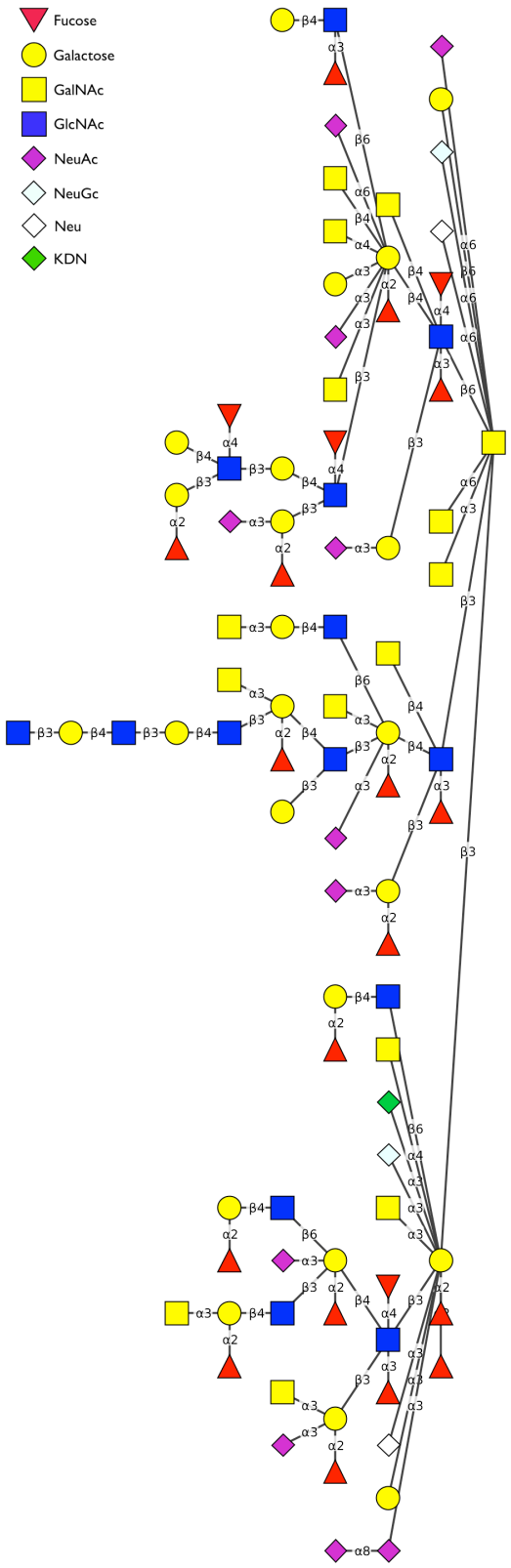

Figure 6: The current canonical tree for GalNAc initiated O-glycans

If any path in a candidate structure is not fully included in the canonical tree, incorporation of the candidate structure will extend the canonical tree, providing new information about residues and linkages that have not been previously reviewed. The structure depicted in Figure 5 is one such example, and will extend the canonical tree if approved.

On the other hand, it is possible that a candidate structure contains errors. Their existence also results in partially included paths. One possibility is to treat non-included residues as correct and assume that they extend the canonical tree. However, our matching algorithm attempts to generate additional alignments in which the partially included paths correspond to potential errors in the representation of the candidate structure, and are returned to curators for consideration.

To measure the quality of a glycan's match within a canonical tree, we compute the match score between the glycan and its inclusion in the canonical tree. Pairs of corresponding residues in the candidate glycan and the canonical tree are compared with respect to their (1) residue types, e.g. mannose, glucose, galactose, etc.; (2) absolute configurations, i.e. D or L; (3) anomeric configurations, i.e. alpha or beta; (4) ring forms, i.e. furanose and pyranose; and (5) parent attachment site. A perfect residue match has a score of 5, which means the residue agrees with the canonical tree on all considered factors. Thus, the match score of an entire glycan is the total score of all residue assignments of all residues in the candidate structure. A perfect candidate glycan match consists entirely of residues perfectly matched to the canonical tree.

Our matching algorithm enumerates the list of all possible matches of the candidate glycan sorted by score in decreasing order and shows the ten best matches to the user. The user has the option to request more matches as needed, though in practice this feature is rarely used. The list may contain perfect matches, matches that contain residues that disagree with the

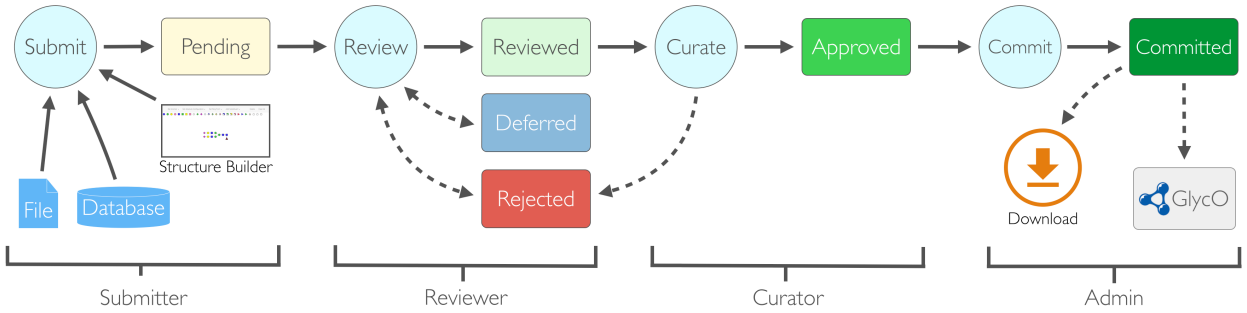

Figure 7: The curation workflow. Solid lines represent the primary path that a structure takes to be included in GlycO. Dashed lines indicate secondary paths that a structure could take, such as being deferred or rejected. Once a structure is submitted to Qrator, it goes through separate review and curation before being added to the GlycO ontology. User roles for different workflow tasks are also shown.

canonical tree, or matches that contain residues not currently found in the canonical tree to which it was compared. The matching algorithm, and consequently the user interface, gives indication of where and how each of these matches differ with respect to a canonical tree in order to assist a human curator in evaluating the structure.

4.3 Curation Workflow

The curation workflow (Figure 7) is a multistep process that requires approval from curators at key points to minimize the possibility of incorrect structures making their way into the GlycO ontology. Combined with computer-aided structure validation, this approach provides an effective means to reduce both the amount of manual labor involved, and the amount of human error. Note that "incorrect" structures, in this context, mean structures that are not consistent with curators' knowledge of biosynthesis.

The process for structure approval is purposely multi-stage to give ample opportunities for reviewers and curators to reject incorrect structures. In Qrator, reviewers are identified as users who initially examine a pending structure and match it against a canonical tree, as opposed

to curators who have the power to approve or reject reviewed structures. In a typical usage scenario, anyone may register to use the Qrator for structure submission and review, but only known and trusted experts are allowed to make final approvals or rejections of structures. Interested labs may download and install their own Qrator software using source code hosted on Google Code (https://code.google.com/p/qrator/). In such cases, the decision of who should be allowed to submit, review, and curate structures, or whether the application should be restricted at all, is up to the lab director. The downloadable version of Qrator will facilitate the implementation of autonomous databases or ontologies that focus on glycan structures that are relevant in a specific context, such as those produced by a particular organism or those related to a particular disease.

During structure submission, Qrator can process a single file, or an archive containing several files. File formats accepted by Qrator are discussed in the next section. The structures parsed from these files are automatically classified using a set of core motifs to identify the canonical tree that most closely fits each one (e.g., the N-glycan tree), and its subtype (e.g., complex N-glycans), if applicable. In general, motifs are substructures that are shared among all glycans of a particular class, but only motifs that contain the root residue of the glycan are used for this classification. After classification, a reviewer determines whether a structure is consistent with his or her knowledge of biosynthesis. Structures that are consistent are computationally compared to the canonical tree that matches their classification. Then, depending on the judgment of the reviewer, one of the matches generated by Qrator may be chosen for a second review stage. However, if a structure is determined to be incorrect (e.g., it contains a structural feature that is biologically improbable), it is not compared to a canonical tree and is moved to the *rejected* status and kept for future reference. This means that the

structure does not make it to the second stage of curation. Retaining rejected structures prevents identical structures from being uploaded in the future, as well as allowing previously rejected structures, along with comments and references, to still be viewed. In certain cases, the decision to reject a structure may be reversed, and structures can be brought out of the rejected state and reviewed again.

In the second stage of curation, a curator examines structures in the *reviewed* state to make sure no errors are introduced to the ontology. In the curation stage, structures may be sent to *approved* status, or to *rejected* status. There is no distinction in status between structures that were rejected during curation and those rejected during review. However, the provenance for such a structure will show that it passed the review stage, but was rejected during curation. Approved structures are eventually moved to *committed* status by an administrator, and added to the GlycO ontology. Reviewers and curators are able to make comments on structures (e.g. correctness, other concerns) at any stage of the curation process, even after a structure has achieved *committed* status.

4.4 SUBMITTING STRUCTURES

Glycan structures can be uploaded to Qrator using GLYDE-II, an XML-based glycan structure format (http://glycomics.ccrc.uga.edu/core4/informatics-glyde-ii.html) that has been accepted as a standard data exchange format [59]. Glycans may also be uploaded as a GlycoWorkbench Structure file (GWS files) [76], or zipped archives of many GLYDE-II or GWS files. Once logged in, a scientist may upload a structure, or review structures already imported from a database. After a structure is imported or uploaded, it is parsed and converted into a Glycan

Object Model (GOM) object, and then converted into a simplified structure representation in JavaScript Object Notation (JSON) (http://www.json.org), which is then stored in the database. GOM is an application programming interface (API) used for parsing GLYDE-II XML files for use by Java applications, while JSON is an information exchange format that is often used as an alternative to XML. Afterwards, both SHA-1 and MD5 hashes of the structure representation are computed to ensure uniqueness among structures. SHA-1 and MD5 are cryptographic hash functions that take potentially large strings of data as input, such as the aforementioned glycan representations, and generate fixed-length (shorter) hashed strings as output. These are useful as unique structure keys in the Qrator's database. After being imported, all new structures are given *pending* status (Figure 7).

Additionally, scientists may use the included structure builder interface to construct glycans for review. The interface allows users to rapidly construct glycans by clicking residues in a graphical menu in order to chain residues together. Default values for ring form and absolute configuration are provided for each residue type, but users may modify them. Users may also select a linkage position for each residue by clicking the link between two residues and selecting the desired position from the provided menu. Parent residues are checked to make sure that they do not have multiple child residues connecting to the same linkage position. Additionally, substituents are created in much the same way as creating a child residue, though users pick substituent types and set their positions from a drop-down menu. Moreover, the structure builder doubles as a substructure search interface. It does not utilize the canonical tree to guide construction, however, as users may want to build structures that are not yet subsumed by a canonical tree.

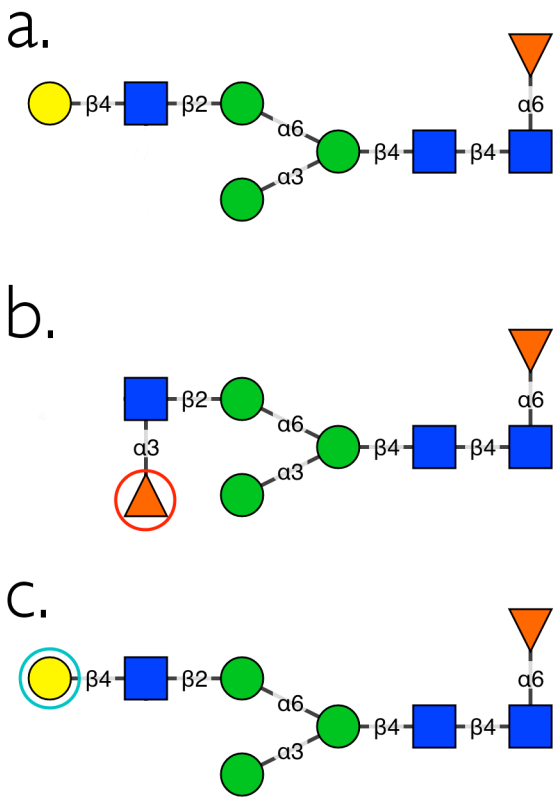

Figure 8: An example of an N-glycan structure that has been matched against a canonical tree. The originally reviewed structure is depicted in (a). An example of an alignment that differs from the original is shown in (b). In this case, Qrator suggests changing the β-galactose to an α-fucose, and the linkage from 1-4 to 1-3. An example of an alignment that will cause a new residue (in this case, β-galactose) to be added to a canonical tree is shown in (c).

Structures in Qrator are rendered in CFG Nomenclature. When a structure contains residues with no CFG equivalent, they are rendered as dark circles. However, the user may mouse over a residue at any point to view the textual description of the residue. Other notations for rendering glycans may be considered in future versions.

4.5 Reviewing and Matching Structures

During the structure review process, a reviewer decides whether a structure is consistent with well-established biosynthetic pathways according to the reviewer's knowledge. Many

biosynthetic pathways are summarized and accessible as public resources (e.g. http://www.ccrc.uga.edu/~moremen/glycomics/) but expert knowledge is still required, especially when considering isomeric complexities that may be known to interfere with specific elongation reactions [64]. Also, no ambiguities are allowed within a structure's definition, since curation is meant to produce a collection of specific, completely defined structures. If, in the judgment of the curator, each linkage and residue is not supported by known biosynthetic capabilities or clear analytic data, the structure may be rejected.

After initial review, the structure may then be compared against one of the canonical trees present in GlycO using the previously discussed matching algorithm. A list of possible matching structural configurations (alignments) is then presented to the reviewer, with differences between the alignment and the submitted structure highlighted as colored circles. For example, the candidate structure shown in Figure 8a has 8 residues, meaning the maximum score can be 40 (5 possible points per residue and its linkage). The alignment shown in Figure 8b has a score of 36/40 because the linkage position, absolute and anomeric configuration, and residue type for the β-galactose residue in the target structure do not match that of the canonical tree. The only matching criterion between the β-galactose and the α-fucose is the ring form (pyranose). In cases where two structures have the same score, they are ranked equivalently and it is left to the reviewer to determine which is correct. However, in practice the number of structures with identical scores is small and does not impair the curation process. A perfect match (an alignment with no differences) appears exactly as the submitted structure does, with no highlights around residues. It should be noted that a perfect match merely means that a structure is completely subsumed by the canonical tree to which it was compared, and that attaining perfect matches is not the ultimate goal of curation. However, perfect matches do

indicate a higher probability that a structure is correct, since the curation process has previously validated other structures containing the same residues and linkages. However, in an imperfect alignment, candidate structure residues may exist that do not completely match a canonical tree residue, or simply are not present in the canonical tree to which the candidate structure was mapped. As shown in Figure 8b, a single residue in the candidate structure has been mapped to a topologically equivalent, yet structurally different, residue in the canonical tree and is highlighted with a red circle. Selection of a match highlighted thus indicates the candidate structure should be edited to correct a mistake in the indicated node in order to match the corresponding node in the canonical tree. Reviewers examining structures manually, or using software that does not attempt to detect errors may overlook such mistakes, since they are not pointed out automatically and are difficult to detect. Residues in the candidate that are not matched topologically or structurally to any residues in the canonical tree are highlighted with blue circles, as shown in Figure 8c. Selection of a match highlighted in this way should be carefully considered, as each highlighted residue requires the addition of a new residue to the canonical tree, fundamentally changing the tree's information content. Alignments such as these are most frequently selected when a canonical tree is initially being built. As a canonical tree becomes larger and more robust, addition of new residues is less often necessary, as the mature canonical tree is more likely to contain all of the residues in each new candidate structure. Structures that have been successfully matched by the reviewer against a canonical tree are assigned *reviewed* status.

The computer-assisted approach for structure review presented here reduces the incidence of human error and the amount of manual labor required in the overall curation process.

4.6 CURATING STRUCTURES

After a structure has been reviewed and matched, a second human curator further assesses the matched structure to determine whether it should be included in GlycO. When viewing a matched structure, all of the references and provenance data associated with it are immediately available to the curator, along with the aforementioned visual cues highlighting possible discrepancies between the structure and the canonical tree it was matched against. With this information readily accessible, the curator can make a well-informed decision as to whether the structure should be approved or rejected.

Another important feature of Qrator is its capability to construct canonical trees for new classes of glycans, provided that the root residue is present and an appropriate set of representative glycan structures is available for each class. In this context, we have built upon early work [73] by expanding the initial N-glycan canonical tree that we had manually imported into the GlycO ontology. We subsequently regenerated and extended this tree, and generated several O-glycan and glycosphingolipid trees from scratch by defining appropriate root residues (i.e., reducing end residues) and adding new residues using the Qrator application.

4.7 COMMITTING STRUCTURES

After a number of structures have been subjected to the curation workflow and have been approved, they are given *committed* status. If Qrator is not in standalone mode (i.e. it is not configured to add structures to GlycO), they are written as GLYDE-II XML files and sent to a separate web application that manages the GlycO and ReferO ontologies. This application

parses the XML and makes the necessary modifications to GlycO. All metadata about the structures are sent in much the same manner and added to ReferO. The local copies of the canonical trees are then updated for future structure matching.

If Qrator is configured for standalone operation, committing structures to GlycO is not enabled. However, approved structures (or structures in other stages of the curation workflow) can always be downloaded as a zipped archive of GLYDE-II files from the *Status* panel. This allows scientists to utilize Qrator to curate structures for use in a specific biological context, or for situations where utilizing the semantic capabilities of the GlycO ontology is not needed.

4.8 IMPLEMENTATION

Qrator has been implemented as a lightweight, web-based application that connects to a series of Representational State Transfer (REST) based web services that are backed by a PostgreSQL database. The web application can be accessed at http://glycomics.ccrc.uga.edu/qrator/. Guest users may browse structures, and view the status page, along with the canonical trees, but they cannot review or curate structures. However, those interested may install a local copy of Qrator, giving them the ability to curate their own structures. The source code and installation instructions are available at https://code.google.com/p/qrator/. Qrator can be configured to interact with our Ontology Web API (still in development), or to function in standalone mode, meaning that it will not be able to add structures to an instance of the GlycO ontology. Even without this capability, curated structures are always available for download from Qrator's status page.

4.9 User Interface

The user interface is implemented in JavaScript, and relies heavily on the jQuery framework (http://www.jquery.com/). Many interface elements and CSS styling also come from the Twitter Bootstrap framework (http://getbootstrap.com/). Asynchronous calls to REST-based web services implemented as Java servlets supply it with data. These calls are made using the AJAX (Asynchronous JavaScript And XML) collection of web techniques. After glycan structures are retrieved from the server side, they are rendered on the client side using D3.js (http://d3js.org/), and further enhanced with supplementary information such as residue and link labels that are displayed on mouse-over. Annotations, references, and provenance for structures are also retrieved when necessary, and rendered in a second pane alongside the list of structures.

Data is encoded in JSON for transmission between the web services and the web interface. JSON offers a more concise data exchange format than XML, and integrates well with a JavaScript user interface. Similar to XML schema, a JSON schema validation method is available in case there is a need, but Qrator does not utilize it at this time.

4.10 Web Services

The service layer has been divided into multiple REST-based web services based on their functionality. Creation, edit, and search functions for most types of objects within Qrator are contained in their own individual services, and all are available by querying them through a suitable Java web server. For example, a user may want to create more references for a structure, edit an annotation for a structure, or search over structures they have uploaded. Individual

services are provided for structures, literature and database references, annotations, user accounts, and administration functions for updating GlycO or downloading structures. Currently, we deploy Qrator on a JBoss Application Server (http://jbossas.jboss.org), but we have also tested the application on an Apache Tomcat web server (http://tomcat.apache.org).

4.11 Data Storage

Qrator utilizes a PostgreSQL database for keeping track of submitted structures in different phases of review, along with references, annotations, and provenance information for these structures. We also keep track of which canonical trees are available in GlycO, and sub-classes of structures within those trees. Furthermore, we retain copies of all canonical trees in the database to compare structures against during the structure-matching step. The GlycO ontology itself is maintained by a separate application, and is updated via web service invocations by Qrator when necessary.

4.12 Results

The Qrator web application has been thoroughly tested by scientists at the CCRC, and the application has evolved significantly based on the feedback we were given. Workflow changes were implemented, making certain stages of the workflow appear under-populated, as in the case of deferred structures. However, this disparity is expected to decrease over time as more curation is done. In all, over 2500 glycans from various classes including N-glycans, O-glycans, and glycosphingolipids have been reviewed thus far, and are in various stages of the curation

CANONICAL TREE	IMPORTED	PENDING	DEFERRED/ REJECTED	APPROVED
N-glycan lipid-linked precursor	14	0	4	10
N-glycan	1911	509	535	883
O-GalNAc	383	36	195	152
Gal-initiated glycosphingolipid	12	2	8	2
Glc-initiated glycosphingolipid	428	108	35	285

Table 1: Current glycan curation status

workflow. These structures were all imported from the GlycomeDB meta-database, which provides access to structural information from several different databases [57, 77], including CCSD, BCSDB and the CFG database. Acquiring structures from GlycomeDB has distinct advantages, including the availability of numerous external references to the other databases that have been integrated within GlycomeDB. In our past curation efforts, the focus was on the curation of mammalian structures, which led to a temporary deferment/rejection of valid glycan structures that are not present in *Mammalia*. Such structures will be reviewed again at a later date. Moreover, Qrator has been designed for curating glycan structures, not glycoconjugates or aglycons, and thus we only import glycans for review.

It is important to note that not all classes of glycans are amenable to curation by Qrator. Each canonical tree used must consist of structurally related glycans (such as N-glycans) that are produced by variations of the same biosynthetic pathway. Thus, curation of N-glycans using Qrator was undertaken before other classes of glycans. This was due in part to the abundance of available structures to curate, as well as the availability of an existing canonical tree (GlycoTree) manually generated by [73]. This tree, containing 74 residues, was initially utilized for curation efforts. After several hundred structures were curated, the N-glycan canonical tree was completely rebuilt using only the curated structures. This newly rebuilt tree has been further

extended by continuous structure curation to 144 residues, as of May 2014. Of the 1911 N-glycan structures submitted for curation, experts have approved 879 for inclusion in GlycO. All numbers are current as of August 2014.

Currently, curation of O-glycans has been primarily focused on GalNAc-initiated O-structures, with limited curation on mannose-initiated structures and plans to start curation of fucose-initiated structures in the near future. Of the 383 GalNAc-initiated structures input for review, 347 have been curated thus far, with 152 structures approved for inclusion in GlycO. Currently, 9 mannose-initiated structures have been curated and approved, out of 466 candidate structures with a mannose residue at the reducing end.

We have also added 440 glycosphingolipid structures including both glucose-initiated and galactose-initiated varieties. In all, 330 glycosphingolipid structures have been curated thus far, with 282 structures approved for addition to GlycO.

All structures are available for download at any stage of curation, either in batches from the status page, or individually. Also, the latest version of GlycO is freely available from the CCRC's Ontology Web API website.

CHAPTER 5

ACTIVE ONTOLOGIES

Information intended for use on the Semantic Web is primarily encoded within ontologies that contain concepts and relations typically expressed in RDF or OWL format. Ontologies model domains of interest, and facilitate the reuse of knowledge by allowing references to concepts within the domain of interest, which are all assigned URIs. They are typically used as a means of capturing knowledge about a domain that does not change often, since most ontologies are not used for daily information storage and retrieval. This contrasts with databases that, for some installations, can see many millions of updates on a daily basis.

Common axioms within ontologies include:

1. **CLASSES** – Classifications of things.
2. **INDIVIDUALS** – Instances of classes.
3. **ATTRIBUTES** – Properties of objects (and classes).
4. **RELATIONS** – Ways in which classes and individuals can be related to one another.
5. **RESTRICTIONS** – Formal descriptions of what must be true in order for an assertion to be acceptable as input.

6. **RULES** – Descriptions of logical inferences that can be made from an assertion.

These axioms enable the specification of concepts, and making assertions about concepts in the form of triples. As mentioned previously, triples are expressions comprised of a *subject*, a *predicate*, and an *object*. The subject represents a concept or an instance of a concept, with the predicate representing a trait of the subject and relating the subject to the object.

Even though ontologies are fundamentally a knowledge encoding mechanism, the information they contain is significantly more useful when harnessed to perform tasks. Using ontological information typically requires constructing an application with specific instructions on how to query a particular ontology, and how to interpret the information that is subsequently returned. Such applications are typically built for a particular use, and have a limited scope. Here we explore the idea of a framework that can extend an ontology with application functions called *behaviors* that bind to resource patterns. *Active ontologies* of this type can more easily be used in daily scientific research, since the contextually relevant functions they are enhanced with can support a wide range of scientific research endeavors.

5.1 BEHAVIORS

Behaviors are pieces of application functionality that are bound to ontological patterns and triggered when new instances of such patterns are detected, allowing contextual actions to be performed. Behaviors may be one of a number of built-in primitive actions provided by the Curio API that can be performed on resources within an ontology, or pluggable modules that provide functionality beyond simple resource management. A behavior is automatically

executed when the pattern it is mapped to is detected in an ontology after a triggering event, but a behavior may also be triggered manually by a user if desired. Triggering events are any which could cause the creation of an ontology pattern at the instance level that could correspond to one of the patterns bound to a behavior. Creating or modifying resources, properties, or relations are examples of triggering events. Moving resources from one location to another is also a triggering event. Removal of resources, properties, or relations is not a triggering event, since ontology patterns bound to behaviors are currently only inclusive of elements, not exclusive.

Patterns are composed of one or more resource types, and may also have constraints on a resource type's attributes, relations, parent resource types, or child resource types. By default, patterns apply to entire classes of resources, but they may be constrained to smaller groups of individuals, or even specific individuals through the use of constraints. This functions in much the same way as patterns present in SPARQL [78] queries, but also takes a great deal of influence from CSS's [79] style of matching elements within HTML.

Behaviors may be triggered by other behaviors when adding or removing information from an ontology, meaning that chain reactions may occur. This could lead to unforeseen benefits or difficulties. However, the ultimate intent is to better utilize the information present in ontologies and turn ontologies into more active participants in daily scientific research.

5.1.1 Behavior Description Language

The Behavior Description Language (BDL) has been created for the purpose of easily specifying the ontology patterns that behaviors bind to, and takes inspiration from the syntax of CSS, which binds styling and formatting parameters to elements in an HTML page. CSS was chosen as the

PATTERN	DESCRIPTION
`*`	any ontology class
`C`	a resource of type C
`C[foo]`	a C resource with a property or relation “foo”
`C[foo="bar"]`	a C resource whose “foo” property exactly equals “bar”
`C[foo^="bar"]`	a C resource whose “foo” property begins with “bar”
`C[foo$="bar"]`	a C resource whose “foo” property ends with “bar”
`C[foo*="bar"]`	a C resource whose “foo” property contains “bar”
`C[foo=D]`	a C resource whose “foo” relation is connected to a D resource
`C(D,E,F)`	a C resource with D, E, and F children
`C D`	a D resource that is a descendant of a C resource
`C > D`	a D resource that is a child of a C resource
`C ~ D`	a D resource that is a sibling of a C resource

Table 2: BDL selector patterns

inspiration for BDL because of its relative conciseness compared to XQuery [80] or SPARQL, and its wide adoption by the web community. CSS uses *selectors* to specify which HTML elements that a style will apply to. Selectors are essentially patterns that are tested against elements within a tree structure to determine whether they are a match. Selectors can be composed of a simple selector, or chains of simple selectors separated by *combinators*. Simple selectors can constrain elements on their type, classes, or attributes. Combinators can connect multiple simple selectors in order to allow constraints on an element’s familial relations within the hierarchy (i.e., an element’s ancestor or sibling relations). Normally, a syntax geared towards trees is not readily applicable to graph structures, since graphs generally do not possess a

hierarchy in which to base constraints on. However, Curio's approach, to start with a tree-based hierarchy of instance data and subsequently build a graph structure of properties and relations around it, means that CSS's use of selectors can be more readily adapted to this framework.

Selectors in BDL function much like selectors in CSS, and serve to constrain which elements are bound to behaviors. However, BDL's selector patterns are meant to constrain on graph patterns as well as on a hierarchy. BDL selectors may constrain on:

1. **CLASSES** – Constraints on a resource's ontological types
2. **ATTRIBUTES** – Constraints on attributes that are common to all resources (i.e. non-user created attributes such as name, description, uri, etc)
3. **PROPERTIES** – Constraints on ontological properties that have been created by a user
4. **RELATIONS** – Constraints on a resource's ontological relationships to other resources
5. **POSITION** – Constraints on ancestor/sibling relationships, as well as child resources

BDL's syntax attempts to conform to syntax precedents set by CSS whenever possible, albeit with additions to accommodate the ability to constrain on child resources and relations. However, BDL does not require certain types of selectors specified in CSS, since its purpose is for binding behaviors to ontology patterns. Examples of selector patterns in BDL are described in **Table 2**.

It is important to note that constraints on resource relations or resource children can be nested indefinitely. For example, in the pattern `C[foo=D]`, `D` may also have multiple property

or relation constraints. Multiple selectors are handled in the same way as CSS, as in `C[foo="bar"][baz="qux"]`. BDL is not yet as mature as CSS, and so more selectors may be required in the future. However, the currently listed selectors are able to support a wide range of use cases.

A wide selection of built-in functions are also provided for basic resource management purposes. These functions support creation, modification, and deletion of resources, properties, and relations. There are also functions for moving resources from one location to another and invoking plugins. Some simple examples of specifying behaviors are shown below.

```
Experiment>GeneList{
     extract(GeneExtractor);
}

Experiment>Gene{
     move-to("Experiments:Genes");
}
```

The first example demonstrates invoking an extractor named "GeneExtractor" on resources of type "GeneList", which are the children of resources of type "Experiment". The second example will invoke the *move-to* function to move the selected resources of type "Gene", which are also child resources of an "Experiment" resource, to the "Genes" resource located at the root of the space named "Experiments".

5.1.2 PLUGINS

Ontology behaviors are intended to be capable of extremely diverse tasks, and are required to be as flexible as researchers need to accomplish their work. Since the built-in operations for resource management are generic and not expected to be flexible enough to accommodate all of a researcher's goals, plugins may be constructed to perform domain-specific tasks. Plugins are software modules that provide enhanced capabilities for research in a specific domain. Obviously the needs of researchers in different fields vary considerably, so plugins are envisioned to perform fairly specific tasks. Multiple plugins may also be bundled together. Curio currently supports four major types of plugins: *Importers*, *Exporters*, *Extractors*, and *Viewers*.

Importers may be used as a major source of automatically harvested input data for Curio. Since they are written as pluggable Java modules, Importers can potentially interface with any external source of data, whether local or remote file systems, web services, databases, ontology stores, or others, and utilize the Curio API to modify the user's ontology by adding or removing resources, relations, or properties. This is useful for gathering data from heterogeneous sources so that they may be organized in the same place. This is intended to help mitigate the burden scientists currently face when viewing their research data, as it is typically spread out amongst multiple storage systems.

Exporters are the opposite of importers, in that they allow resources to be written to some output, or even transferred from Curio back to external storage systems. Like importers, they are pluggable Java modules, and can theoretically be programmed to interface with any external data

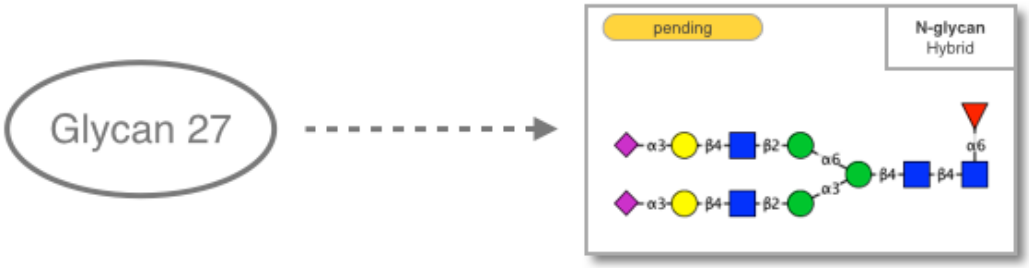

Figure 9: Visualizing a resource representing a glycan

storage system. However, exporters are not meant to modify the user's ontology, only write or transfer resources to another location or format.

Extractors work somewhat like importers in that they can also add and remove resources, relations and properties. However, they extract these elements from existing resources, such as imported XML documents, spreadsheets, or other raw scientific data. Despite the name, extractors can serve many different functions ranging from data analysis to scientific workflows, and effectively serve as processing modules within Curio. For example, an extractor may be written to convert existing resources to other formats suitable for analysis, while another extractor could be written to perform a scientific analysis, essentially extracting an output analysis from an input raw data file.

Viewers may also be constructed for visualization of ontology data. Contextually relevant, domain-specific visualizations are invaluable to scientists for analyzing their data in ways that are intuitive to them. Generic visualizations of ontologies (i.e., as nodes and edges) are typically of limited value, and do not afford scientists the ability to view data in the manner that they are accustomed, or in ways that make sense to them, causing data analysis to be more difficult than it needs to be. Custom visualizations, by contrast, can drastically improve the user experience, and greatly increase the usefulness of ontologies by allowing researchers to view and

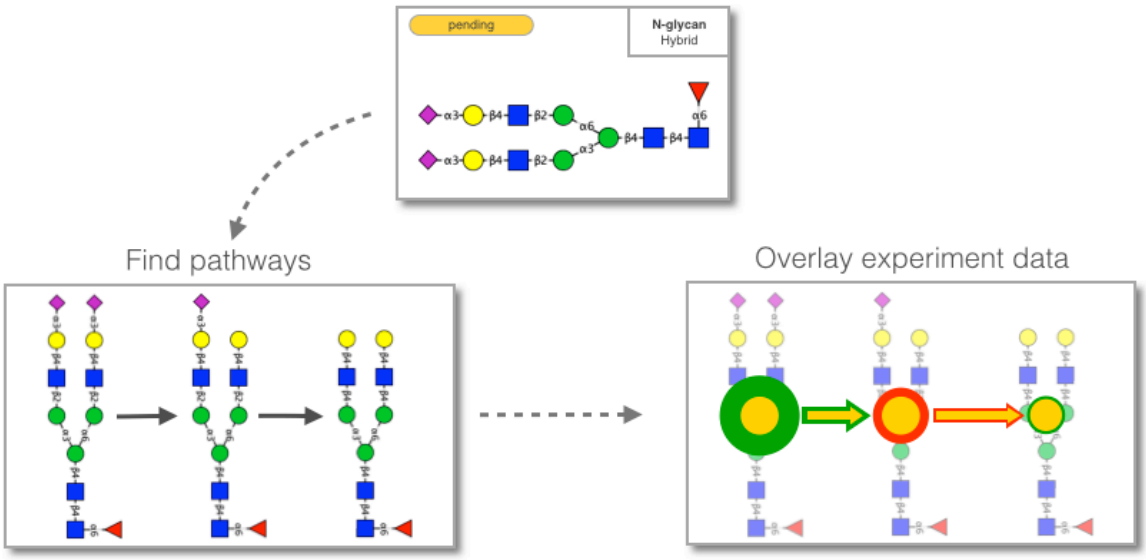

Figure 10: An example of contextual actions. The glycan resource visualized in Figure 9 could be bound to a behavior plugin that locates biosynthetic pathways in which it resides, and renders experiment data on top of the path for analysis purposes.

analyze data in representations that are accepted by their scientific communities. For example, in Figure 9, a resource representing a glycan in Curio could be rendered in CFG notation, a familiar format for glycobiologists. Furthermore, in Figure 10, this glycan visualization could be combined with a second plugin that allows scientists to extrapolate the glycan into various biosynthetic pathways, and could even combine such pathways with experiment data for analysis.

In general, behaviors give rise to interesting possibilities with respect to the role of ontologies. As mentioned previously, it is easy to envision behaviors granting an ontology the ability to fully interact with other sources of knowledge, not simply extract information from them. An ontology could conceivably self-populate its individuals, or dynamically create provenance information as new data is created or imported, verify its own data against other sources of data, or even rearrange its internal structure based on contextual triggers. Given a combination of these abilities, an ontology could even function as an agent, growing by

collecting data, and regulating its structure as it increases in size. The ontology could also automatically push updates or additional information to remote sources of data, if allowed. Scientists who create ontologies with Curio and import data from external sources would not simply be consumers of data, but also potential content creators that could push changes, along with additional information, back to these sources. Once behaviors are in place, many of these processes or workflows could be automated, significantly reducing the workload scientists face when working with research data.

5.2 Data Provenance

Provenance is a very important part of scientific data management systems. A good scientific data management system should be able to track the provenance of research data to ensure that the data is not misplaced or forgotten after the research on it has been published, both to ensure experiment reproducibility, and for possible use in future projects. Such a system should also be able to generate provenance where appropriate, so that a researcher does not have to manually document what has been done to an item of data. Methods of documenting provenance, and what provenance should be recorded should also be flexible. Current data management systems' procedures for recording provenance cannot typically be altered.

Indeed, data provenance is one of the most important uses of ontology behaviors. To illustrate how a behavior can assist in recording provenance, let us consider a scenario that could be employed in qRT-PCR experiments. The qRT-PCR workflow contains a data processing step that accepts a raw data file as input and generates a spreadsheet as output. By providing an extractor plugin to perform the data processing step and binding the extractor to the input file

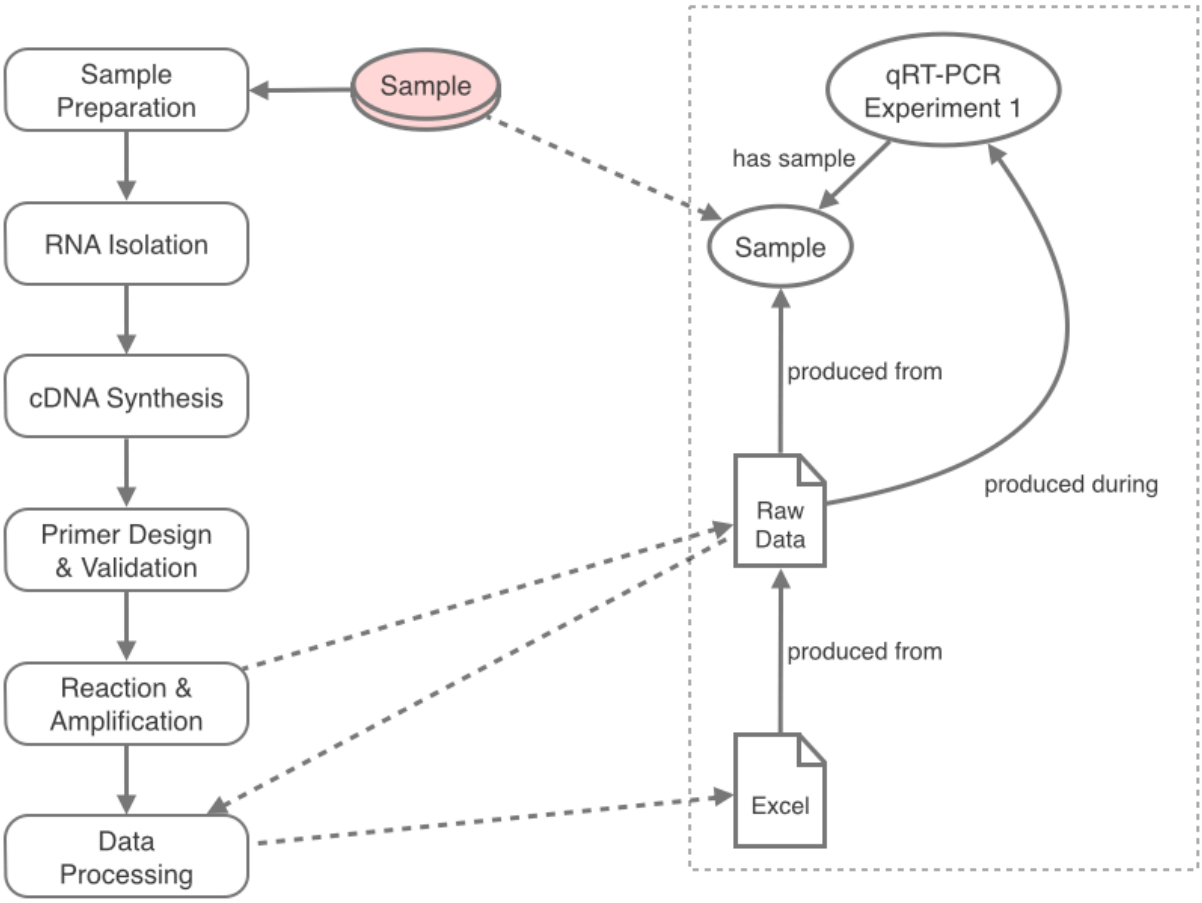

Figure 11: Data provenance scenario using the qRT-PCR workflow. Resources within Curio are displayed within the outlined area. Relationships between resources could be created automatically by binding behaviors at certain stages of the workflow.

with appropriate selectors, ontological relationships between the input data and output data could be generated automatically by the extractor, since it can utilize the Curio API. Moreover, the workflow itself could be represented as a resource (or series of resources) in Curio, so that a relationship between the input and output resources and the workflow (or processing step) could be created as well. If desired, metadata in the form of attributes could also be assigned to resources involved in the process. A simple example of qRT-PCR workflow resources enhanced with provenance information is shown in Figure 11. The aforementioned scenario could also fit

other workflows in numerous areas, including data analysis, data annotation, data curation, report generation, running simulations, and more.

Versioning is yet another possibility for provenance relations, as newer versions of files can contain relations to previous versions, while keeping records of how they were transformed, similar to a wiki. A data import plugin would be required to interface with workflow programs if automatic population is desired, however.

5.3 WORKFLOWS

Researchers are increasingly utilizing workflows for performing experiments, data analysis, or other tasks related to their research. Current scientific workflow systems provide capabilities geared specifically towards scientific research. Efforts in Astronomy, Bioinformatics, Physics, Chemistry, Botany, Climate Science, Computer Science, Neuroscience, and Ocean Science among others, are all benefitting from scientific workflows. Most existing scientific workflow systems allow for graphical creation of workflows with references to outside resources, such as web services, to perform the actual execution of their tasks.

Workflows can be created in Curio through the use of behaviors, enabling tasks to be automatically executed as certain conditions are met. This method of specifying a workflow, however, is envisioned to be more loosely structured than in traditional workflow systems. In the Curio model, resource patterns would have behaviors assigned to them, which could then trigger actions on other resources, causing chains of actions to occur. These behaviors would be specified in place of using a graphical workflow editor to create a flow-diagram specification of a workflow that is currently prevalent in workflow systems. Smaller sets of behaviors could also

easily be specified to handle commonplace tasks without a researcher even thinking of them as workflows. This method also allows the possibility of overlapping workflows, as multiple workflows' behaviors could interact with the same resources concurrently. Care would naturally have to be taken to ensure that two workflows' activities do not conflict when specifying multiple behaviors on the same resources.

CHAPTER 6

ORGANICALLY CONSTRUCTED ONTOLOGIES

Often in scientific data management systems, knowledge modeling is done first, usually by knowledge modeling experts or computer scientists. Even though there is often close collaboration with domain experts to construct domain models, disagreements between domain experts is a frequent occurrence, especially on how concepts should be modeled. In an ideal scenario, domain specialists should be able to construct their own knowledge models, either collaboratively or by themselves, since they have extensive background experience and know what is useful to model. If there are multiple models of the same domain, these same specialists could work together in a collaborative software environment to integrate models, if desired.

Static domain models often lack practicality for scientists who wish to use them in their efforts to better understand and analyze their data, since they also often need the flexibility to change the model as new data becomes available, or as their understanding of the domain changes. A domain model need not be pre-constructed and static, but may grow organically from inputting and linking scientific data. For many researchers, ontology development in these cases may be better performed from the perspective of the data (i.e. from the bottom-up). A data management system that allows a scientist to construct a model out of their research data would also be better suited to handle changes to the model when the need arises, since the model is not static and is meant to be readily adapted to new data.

One option for ontologies starting with imported data is to use a basic schema grounded in the types of data that have been imported. For instance, if GLYDE-II files have been imported, the type of the resource could be GLYDE-II, which would inherit from XML, which would inherit from Text Document, etc. If an extractor (mentioned in the previous chapter) is available that can read the contents of GLYDE-II files, a representation of a glycan could be extracted from the GLYDE-II file. Alternately, a reference to an existing representation in an external ontology could be created. The scientist could then enhance the GLYDE-II file by creating or referencing a class called Glycan, instantiating a new Glycan object, and linking that object to the file that it originated from. In this manner, source data (the GLYDE-II file) can reside in the same continuity as the resources that they describe (the Glycan resource).

6.1 Ontology Construction Methods

Ontologies, described in section 2.3, act as common vocabularies for scientific domains, and are typically constructed through consensus on what concepts mean. Designing ontologies can be controversial, with potential disagreements about what concepts are important, how they should be modeled, or even what they mean. Here, we mainly consider ontology design to be referring to designing ontology schemas, as opposed to schemas plus instances. Ontology design is often an iterative process, going through multiple revisions before a final ontology schema is produced. Even then, the ontology may be further revised as new discoveries are made, if the scope of the ontology is to be expanded, or if the ontology is to be used for different purposes than originally conceived.

Often ontology schemas are modeled before the designers even know precisely what instance data will need to be included, and are engineered in a top-down fashion. The top-down approach begins with the most general concepts of a domain, and then subdivides them into more specific concepts. The bottom-up approach, which is the opposite of the top-down approach, groups together the most specific concepts of a domain (usually at the instance level) to form more general or complex concepts. Methods to combine the two in hybridized approaches have also been proposed. Of the two approaches, top-down is by far the most popular, as, currently, there is no general-purpose algorithm that is capable of automatically creating an ontology schema from instance-level data. The Glycan Ontology (GlycO), which is used by many of the bioinformatics projects currently in development at the CCRC, was developed in a top-down fashion, and is intended to model the particulars of glycan chemistry and structure. However, bottom-up and hybrid approaches also continue to gain popularity with various applications [81-85]. Curio intends to allow researchers to gradually enhance their research data, which will function as the instance level of the ontology, with ontology classes, properties, and relations. While this is a bottom-up approach, it does not use an algorithm to automate construction of the ontology schema. Rather, it will let schemas evolve naturally out of the process of enhancing research data with metadata.

6.2 RESOURCES

Resources in Curio are representations of data intended to be capable of embodying a wide variety of things including objects in the real world, research data, experiments, workflows, documents, or anything else that a researcher might need. These resources may be organized

hierarchically using resources that act as containers, virtually identical to the organizational method present in a file system. Resources may be further organized with Spaces, which can be considered largely analogous to different volumes in a file system, and carry their own security considerations. Spaces can be used for organizational purposes, serving as a partition for resources, or they can be used for collaborative purposes, with multiple users having access to the same space and being able to edit the same resources.

Curio resources also share many commonalities with the definitions for resources outlined by the World Wide Web, and later the Semantic Web. The Semantic Web defines resources by encoding them in RDF and its derivative formats. Resources in RDF are entities referred to by any URI or Internationalized Resource Identifier (IRI). By this definition, anything with an identity can have a URI or IRI, and is potentially a resource. Thus, in RDF, statements can be made about anything that has an identity, and statements can even be made about other statements.

Curio is intended to serve not only as a meta-organizational layer over existing storage systems, but also as a storage system in its own right, and is capable of creating resources that only reside within its database. It supports the integration of resources residing in multiple systems, with no guarantee that those systems are otherwise interoperable. Curio is also capable of enhancing any resources it contains with new metadata. Metadata enhancements take the form of relations between resources, which may be thought of as labeled hyperlinks, or properties, which function as annotations. As with RDF, property and relation types may apply to all resource types, or may be constrained to only apply to a specific resource type. An example of enhanced resources is displayed in Figure 12.

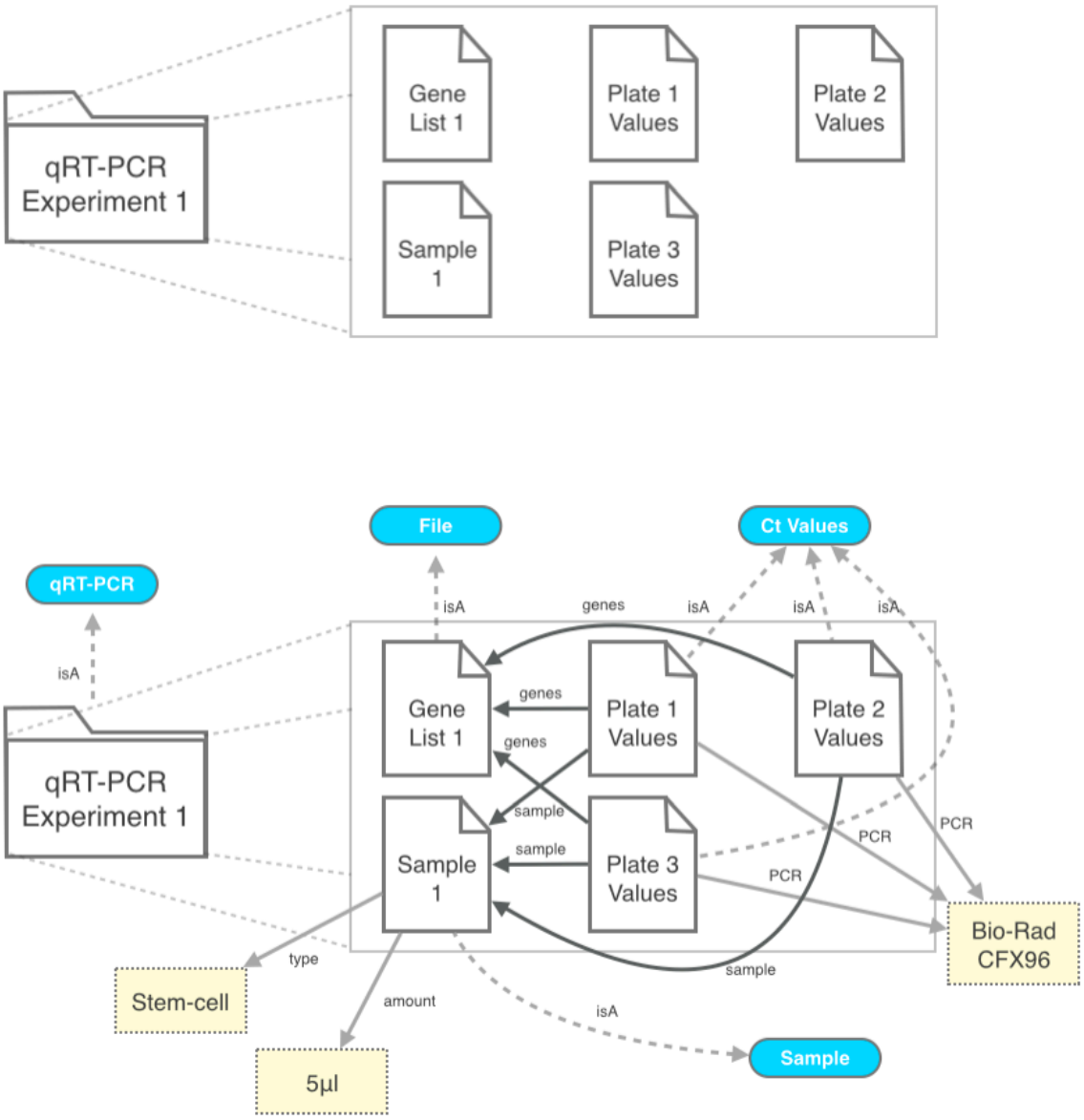

Figure 12: An example of resources before and after metadata enhancement. By default, resources may exist hierarchically like files, and they may be enhanced with properties and relations.

Scientists are able to reorganize resources irrespective of how they were organized in their place of origin. These organizational changes are not propagated back to the sources unless an exporter is used. For example, a scientist may import a group of resources from the file system on a computer and reorganize them inside Curio, but this does not mean that the changes

are reflected in the file system itself. Resources from different origins may be organized and interspersed within Curio regardless of where they were imported from. Information extracted from databases may exist alongside imported files, or instances imported from ontologies, or even resources imported from the web. Curio is not entirely dependent on importing resources, however, as it also supports the creation of new resources.

Resources may also be organized into a hierarchical structure at the instance level, similar to the way file systems organize files. This is accomplished with the use of resources that act as collections, also referred to as containers. Containers are considered specialized resources that can contain other resources and other containers. This allows for preserving the structure of data imported from inherently hierarchical sources such as file systems, and also provides a convenient means of creating further organization amongst resources in Curio even if a user does not wish to enhance their resources with metadata. Such relationships between resources can represent partonomy, or other types of hierarchical relations as needed. Furthermore, the semantics of containers may be mapped to external vocabularies such as SIOC or SKOS as needed when exporting data.

Much like in other ontology-based systems, resources in Curio may be assigned types, which are analogous to classes in RDFS or OWL. As with RDFS or OWL classes, resource types may have property and relationship types associated with them, constructing a schema. It is expected that these types will be defined after the import or creation of resources in Curio, not before. This will allow schemas to emerge from available resources, instead of forcing resources to fit a predefined schema. However, if a community has already agreed on a common vocabulary and desires to import a predefined schema, this is also possible.

6.3 Resource Aggregation

As discussed previously, data is encoded in many different formats, and is kept in many different types of storage systems. Scientific data is especially prone to information fragmentation owing to the number of ad-hoc file formats in existence, as well as the number of domain-specific data management systems being used. Research data may be found in file systems, databases, ontologies, LIMS systems, the World Wide Web, or the Linked Data cloud, among potentially many others. Aggregating data from so many sources, and extracting from so many different formats is a challenge unto itself. While Curio cannot provide a direct solution to this problem, it does provide a basis for solving this problem in the form of an API for developing importers that can be utilized to ingest data from any of these various information sources as Curio resources. Additionally, this API can be used to build extractors for creating resources by parsing structured and semi-structured data from different data formats.

As mentioned in the previous chapter, importers are plugins that serve as adapters for external sources of data, allowing resources and metadata to be created. These resources and metadata can be created in Curio to directly mirror the organization scheme used by the original source of the data. Importers may ingest the entire contents of a resource into the system, or may simply create a reference to an external resource, which is especially desirable if the content of a resource is considered too large to efficiently store locally. Resources ingested from different sources are treated equally, as Curio is agnostic to where a resource comes from. This allows researchers to freely combine resources imported from multiple sources in the same space, and relate or otherwise organize them as they see fit.

6.4 Resource Organization

The traditional file system organization paradigm (using files and folders in hierarchy) is familiar to most computer users, and the methods that people use when organizing their data in file systems have been studied in the past [86]. Every major desktop operating system (e.g. Windows, Mac OS, Linux) in current use is still designed around this form of organization. Even mobile operating systems utilize it behind their user interfaces, with some mobile operating systems employing aspects of it in the user interface as well. Hierarchical file systems are likely going to be around for quite some time yet, despite claims to the contrary [87]. RDF-based ontologies can also support hierarchical means of organizing knowledge, but this is typically only used at the schema level. To support the import of data from hierarchical systems, instances require the ability to be recursively grouped into containers. Importing data elements from a hierarchy into a flat structure is not sufficient, as there would be information loss when discarding their previous organizational structure.

Luckily, preserving organizational semantics when importing data from file systems into ontologies is not terribly problematic. There are already elements in basic RDF to support collections of resources, whether ordered (Seq) or unordered (Bag). Moreover, Semantic Web vocabularies like SKOS have support for collections of concepts, as well as other useful constructs for knowledge organization.

When importing data into Curio as resources, hierarchical relationships present in organizational systems that use hierarchies, such as file systems, are preserved between resources at the instance level. Then, should a scientist choose to enhance the resources further, an ontology schema can be built on top of them, complete with relations between resources, and

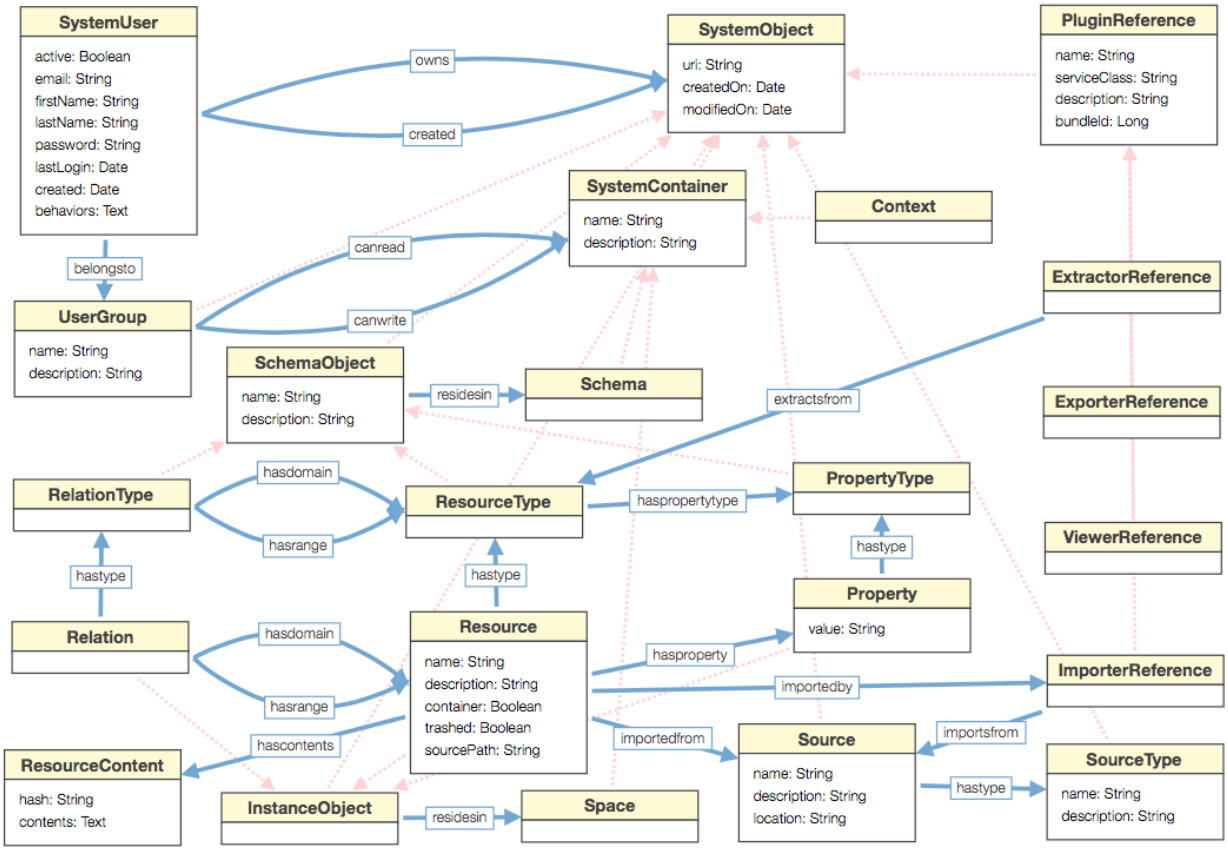

Figure 13: Curio's data model

other metadata. Scientists using such a system need only be aware of having the ability to organize and annotate their data in new ways, not that they are actually creating an ontology. Becoming proficient with knowledge-modeling should not be necessary for domain experts, who simply want enhanced capabilities for their data, like annotation or provenance. With this approach, the ontology grows naturally from the way scientists already organize and use their data. We also argue that schemas for data should not be rigid and difficult to alter, since the highly dynamic nature of many areas of scientific research demands a continually changing understanding of the way data is structured.

However, considering that many data models are created prior to organizing data, Curio does support preloading such models for use by individuals within a community. Often,

scientific communities may agree on standard concepts and wish to use these as a starting point for further data modeling, or data exchange. In such situations, the community can import its own ontological schema, whose concepts can be utilized by scientists in their own personal data models, and, in some cases, extended. Resource types created by individual scientists can also be shared, or mapped to by others, in the case that other scientists agree on terminology and want to integrate their data.

Curio's data model (Figure 13) is designed to reside above the schema layer of ontologies, and is intended to contain both the schema and instances of user-constructed ontologies. In effect, it is an extra level of abstraction and serves as a metamodel for data modeling. It will support access control via spaces, and keep track of the sources that data is imported from. The pre-construction of ontological schemas (discussed previously), as well the ability to specify ontology patterns and behaviors (described in section 5) that attach to these patterns will also be supported.

6.4.1 SPACES

Spaces are virtual containers that can be used in Curio for various purposes. Spaces may be employed for organization, but they can also facilitate collaboration, or control access to different collections of resources. This is similar to many existing organizational systems that also partition data for these purposes, such as file systems and database management systems. Access control is particularly important in governmental institutions such as the United States National Labs, which require tight controls on sensitive material. Unfortunately, this also places burdens on scientists who want to collaborate with others, as they must go through a tedious

process of gaining clearance for outside organizations to be given access to data within a protected network. Similar situations exist in most research labs with sensitive data in their networks. As expected, scientists welcome systems that allow easier collaboration under such tight restrictions. Spaces could help facilitate such interaction without compromising security.

Another interesting possibility that spaces provide is the opportunity to publish research to a community immediately. In this scenario, other scientists in the community could review research as it is made public, and make comments or criticisms to help the author improve it. Research materials could easily be linked to the published documents for other scientists to use in reproducing experiments or verifying results. This could result in a much more open scientific community that is less afraid to publish negative research results along with positive ones, since the barrier to publication is lessened.

6.4.2 CONTEXTS

Like spaces, *contexts* are available to assist scientists with organizing or controlling access to their resources. Contexts act as organizational elements, but they do not reside in the same continuum as classes or spaces, and do not function the same way. Whereas classes group resources by assigning a definition based on their characteristics, and typically exist in a hierarchy, contexts operate like tags or keywords for resources, describing them in a non-hierarchical manner.

Contexts can also support access control for resources, since resources linked to a particular context may be restricted to particular groups of users. This affords a greater degree of granularity when controlling access to resources, as contexts can be used within spaces to grant

or restrict access to groups of resources. Contexts may also be used in concert with user groups by plugins to determine the extent of functionality that is available to a user. For example, specific groups of users may need an extractor to only extract certain values, or there may be a need for a viewer to deny access to certain menu options.

For applications like the Qrator, which exist within a single space, queues of structures are modeled as container resources. Access to these queues must be restricted to specific user roles if we are to adequately model the original Qrator application. Since reviewers, curators, and administrators need different access rights within the application, simply granting access to the entire space will not work in this case. Contexts may be used instead to grant read or write access to specific queues. Moreover, viewers employed by the Qrator may use contexts to restrict available functions in menus, such as the decisions that are available to curators when reviewing structures.

CHAPTER 7

CURIO PROTOTYPE IMPLEMENTATION

A prototype implementation of Curio has been developed using many of the lessons learned from previous implementations of data curation, analysis, and management systems for glycobiology. This system attempts to be a truly flexible, yet customizable platform for constructing scientific research environments that allow resource management, workflow design, and data analysis capabilities in a modular architecture. Building upon previous work on projects in glycomics research, Curio aims to have an intuitive UI, more varied ways of organizing data in order to fit scientists' preferences, and the ability to add behaviors to ontological patterns in order to facilitate scientific workflows, data analysis, or other capabilities. The current architecture for Curio is described in section 7.2.

The persistence module used in Curio was refined in previous projects, such as the aforementioned GlycoVault and Qrator. Highlighted features of the persistence module include (Java) code generation of system objects, object factories, web services, and SQL schemas from UML-like models. The persistence module serves as a means of storing, retrieving, and searching for objects efficiently without requiring pre-constructed SQL queries. Possible future additions include support for graph databases, triple stores, file system storage methods, or hybrid systems that make use of multiple storage solutions in concert. SPARQL support is another possibility, especially if a preexisting triple store is used.

The ability to add behaviors to ontological patterns has also been implemented. A CSS-inspired scripting language syntax has been developed to allow easier construction of behaviors that are matched to graph patterns. A set of basic operations has been constructed that allows scientists to easily create small workflow-like processes to be performed when triggered. Furthermore, a pattern scanning mechanism has been developed to activate behaviors in the event that the ontology changes. Patterns for automatic execution of behaviors are stored in a trie-like structure, and executed when matching patterns are detected in the ontology.

Aggregating data from external sources is also provided. Curio can theoretically support the import of data from databases, ontologies (Linked Open Data), web services, and file systems (among others) through its plugin architecture. As discussed in Chapter 5, data import plugins can be used to acquire resources from various locations, and data extraction plugins can be used to create resources or metadata by parsing specific data formats. For example, when importing a GLYDE-II file from a web address, an importer for downloading web data will be used to acquire the file, where it can then be passed to a GLYDE-II extraction plugin for parsing. As mentioned previously, these two resources, the GLYDE-II file and the extracted data, can both exist in Curio which allows opportunities for the creation of provenance metadata, among other things.

Users can manage resources through a web-based user interface that heavily utilizes jQuery and Bootstrap functionality. The user interface provides a default file-system-inspired view of resources, but is also capable of rendering resources in domain-specific representations dictated by visualization plugins. Resources are divided up into spaces, which are displayed along the top of the screen. Visualizations for resources are also available if a visualization plugin exists and is available for a particular resource pattern.

7.1 CHALLENGES

As many application developers will attest, building flexible systems for handling research data is quite difficult. There are no known algorithms that can analyze unknown data types and adapt to them automatically without human intervention, or algorithms that can automatically infer schemas from instance data for all domains. Moreover, not many systems exist that do not require their users to adapt their data to fit a predefined method of organization. As stated previously, there is typically a tradeoff between usefulness in a specific domain, and the ability to support multiple domains (even if only to a limited degree). In building Curio, we intend to construct a framework that could support any domain, yet one which must be customized in order to be truly useful. While this is not as convenient as an algorithm that does most of the work for us, it can still provide a useful platform that supports a great deal of flexibility in data modeling, and interoperability between data in different domains. Such a platform can also eliminate much of the effort expended in constructing modules that are common among many web-based research applications, such as persistence, web services, and user interface.

There are also challenges in developing a user interface that can support many different kinds of visualization through customization, yet is still mostly familiar to users. To this end, we have decided to base the default visualization of resources around files and folders, as we believe that this offers the most common frame of reference for everyone who has dealt with computer-based information management. We also take inspiration from web browsers' address bars and website breadcrumbs to display the user's current location within the information space. However, visualization plugins must still be developed for specific applications in order to provide more useful ways of interacting with resources. Still, we believe that starting with

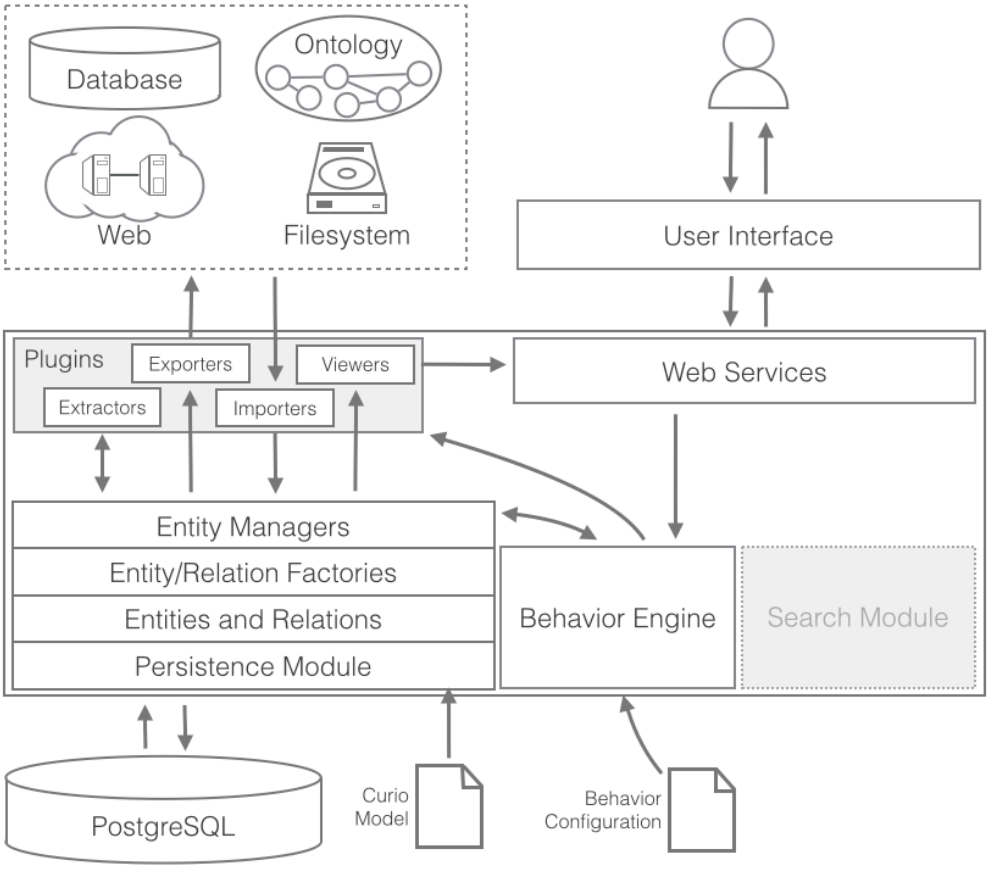

Figure 14: Curio's architecture

familiar visualization paradigms, such as files and folders, will make it easier for researchers to adjust to using the system, and will minimize the discomfort of transitioning into the Curio environment.

7.2 ARCHITECTURE

Curio incorporates software that was previously developed for other projects, as well as modules that had to be written specifically for it. The software that serves as Curio's persistence layer was used in other bioinformatics projects, but was developed for use in any application. While

this software's major role is to function as a persistence layer, it was also used to generate a significant amount of Curio's code, and can offer the services of a rapid prototyping system. In addition to the persistence software, a portion of the JavaScript in the user interface of Qrator was able to be adapted for use in Curio.

Curio also makes use of the OSGi (Open Services Gateway Initiative)[8] framework to allow the addition of domain specific plugins that define behaviors for data import, extraction, visualization, and analysis. These plugins are managed by Curio's plugin manager, and can be linked to resource patterns for automatic execution. Currently, installed plugins are globally available to all users. Apache Felix[9], which is an OSGi implementation, was used in the construction of Curio's plugin manager.

Overall, Curio has been constructed with a software stack much like any other web-based application. At the lowest levels, a persistence layer interacts with a PostgreSQL database. The persistence layer is responsible for creating, retrieving, updating, and deleting the objects in the layer above it, which are, in turn, interacted with through object and relation factories in the layer above them. Object managers form another layer of abstraction above the factories, and are used by a layer of web services that allow the JavaScript-based user interface and other external applications to interact with a Curio deployment. The user interface is built with a combination of jQuery[10], Bootstrap[11], and D3.js[12], and serves as an interactive environment for users to manage their resources.

[8] http://www.osgi.org/
[9] http://felix.apache.org/
[10] http://jquery.com/
[11] http://getbootstrap.com/
[12] http://www.d3js.org/

7.3 PERSISTENCE MODULE

The persistence module used in the construction of Curio was spun off as a separate project during the development of the aforementioned Qrator and GlycoVault projects. Frequent alterations in the data models of these applications prompted the development of a system that could allow such changes to be rapidly implemented without requiring tedious updates to every level of the software stack. It is often the case that modifications made to the data models of web-based applications that use a database necessitate updating database schemas, database queries, objects, object factories, and web service classes, depending on the extent of the changes. Such updates require a lot of development time, and if many adjustments are made to a data model during an application's development cycle, they can severely delay project completion dates.

The persistence module has undergone many design and implementation iterations, but currently serves as a code generation system as well as a dynamic persistence layer. It is capable of automatically generating large amounts of application code for Java projects, as well as storing and retrieving Java objects, all based on a project's data model. It dynamically generates queries to create, read, update, and delete objects (CRUD). Currently, only PostgreSQL databases are supported, but the intent is to eventually support other SQL databases, and even other types of data storage methods, such as NoSQL databases, RDF or OWL triple stores, or even flat-file based systems. There is also the possibility of utilizing multiple storage systems in tandem as a hybrid, with data being stored in whichever system would best accommodate it.

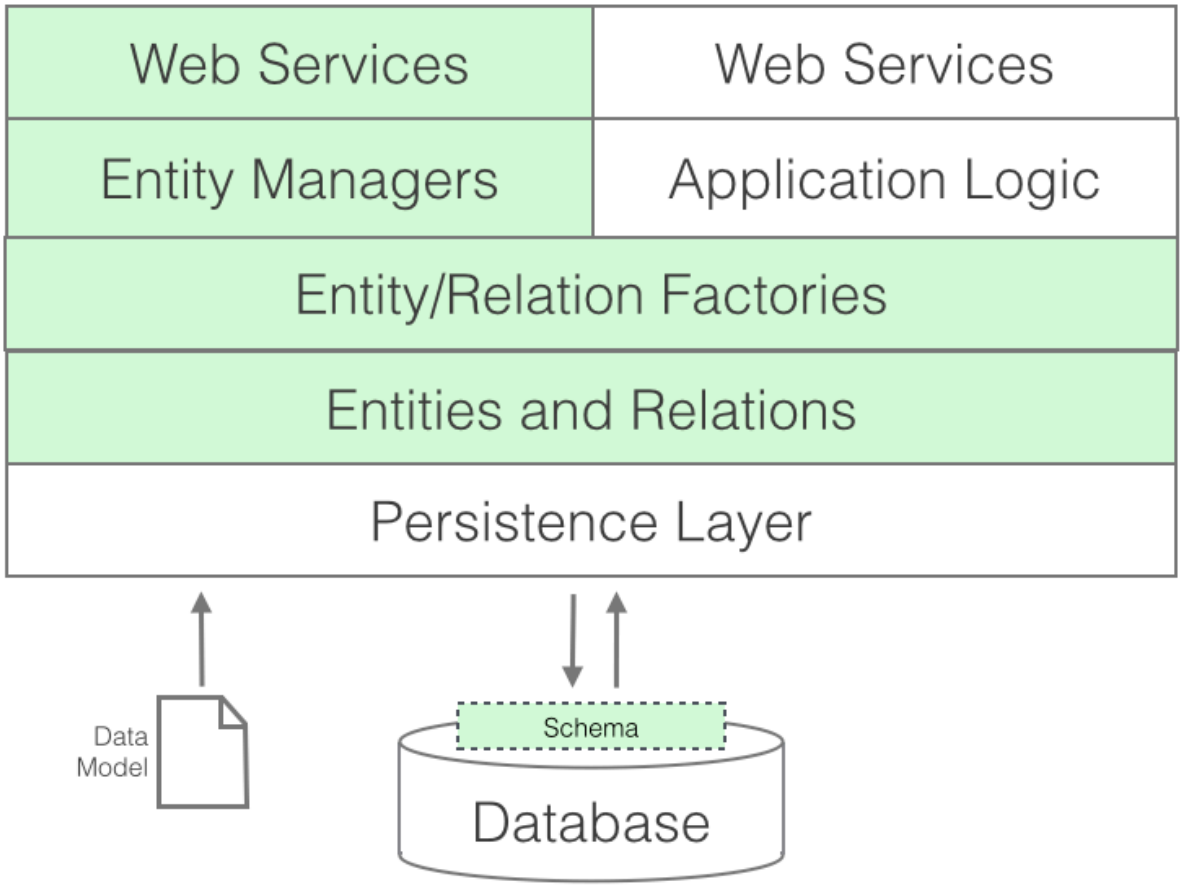

Figure 15: Persistence module's code generation capability for a standard software stack. Sections in green are capable of being automatically generated. Unshaded areas already exist or must still be manually coded. The persistence module takes a data model specification as input.

Use of this module has saved considerable time during the development and maturation of the data models used in Curio, Qrator, and GlycoVault. When designing any prototype application, changes to classes and their relationships are expected as new requirements are added. Usually, this requires manual corrections to every level of the software stack, with the severity of the corrections depending on the number and significance of the changes. However, the persistence module's code generation capability mitigates many of the necessary corrections, since it can automatically generate many classes required by an application. The code generation functionality has thus far been focused specifically on web-enabled applications, as it supports

TYPE	ATTRIBUTES	COMMENTS
Integer	name type default nullable unique size (*String only*) values (*Enumeration only*)	
Long		
Float		
Short		
Boolean		
Date		
Text		Treated as a text blob of varying size.
Object		Stored as a byte array.
String		Requires size attribute (in bytes). Stored as varchar.
Enumeration		Requires a list of values in the enumeration.

Table 3: Attributes of persistence module entities

the generation of web service classes. Currently, SQL schemas, entity classes, relationship classes, entity and relation factories, entity and relation "manager" classes (which act as an abstraction and convenience layer on top of factories), and even web service classes can be generated. Parts of a typical web application that can be generated are shown in Figure 15. In its dynamic persistence capacity, this module currently serves as the data storage and retrieval layer for the three aforementioned projects.

When using the persistence module, a data model must be provided as input. The model is encoded in JSON, and contains many attributes that are similar to UML models and SQL schemas. Within the model specification, there is a section for specifying entities, and another for specifying relations between entities in the model. An example of a model file is shown in Appendix A. The persistence module completely relies on the data model specification in order to generate code and read and write objects, and reads this file each time the persistence module starts. As the file is parsed, an internal data model is constructed with references to generated

interfaces, classes, and even getter and setter methods to facilitate dynamically retrieving attributes of object instances when necessary.

Entities in the model file are specified in a JSON map, with keys corresponding to the entity name, and values consisting of the entity's specification. Within the entity specification, there are multiple properties describing how the entity's class is to be generated when utilizing code generation. Among these are a properties indicating whether the entity is abstract, an array of the attributes that the entity should have, a comment describing the entity, the entity's database key, and whether the entity represents a user. The last property is useful when the application makes use of access controls that are optionally available. Attributes of entities are specified as JSON objects within a JSON array. Attributes can be one of ten different types, with specific properties corresponding to each type (Table 3).

After specifying a model, code generation may be performed by invoking methods within the persistence module's code generator. Code generation uses a series of Freemarker[13] templates to generate a variety of different types of classes. Templates may be customized for each project, and theoretically could generate non-Java classes as well, though this is untested. Individual methods within the generator class are provided for generating class files for objects, relations, web services, object and relation factories, object managers, and even database schemas. Object managers are classes that provide convenience methods for each type of object specified in the data model and invoke object factory methods to carry out many tasks. Methods in an object manager may allow easy retrieval of specific objects, lists of connected objects via relations, updates to objects, or the ability to create new objects or remove them.

[13] http://freemarker.org/

When using the persistence module as a dynamic persistence layer, configuration information must be updated in a Java properties file in order to connect to a database. Once the database URL, database driver, user name, and password are set, the persistence module may be used to programmatically alter the database's contents. Adding, removing, and modifying instances of objects may be performed at the factory layer, or with individual object manager classes. Querying for objects can likewise be performed in factories or object managers, and utilize filter classes to target specific objects for retrieval.

The persistence module's query package provides functionality for running *select*, *insert*, *update*, and *delete* queries. Insert and Delete classes have functions for programmatically inserting and deleting objects and relations, respectively, while the Update class contains methods for updating a preexisting object's attributes. Select queries are used for retrieving data, and include the option of traversing relations between objects. Select queries also allow the use of filters to constrain which objects are returned by the query. Select, insert, update, and delete queries may all be constructed from a QueryBuilder class.

Query filters are used to filter the results of a persistence module query based on the attributes of the objects being returned. Options for filtering attributes include comparisons for equals, greater/less than, greater/less than or equal to, within, not within, like, and unlike. Filters have the capability to group restraints so that they are evaluated together in the final query. Restraint groups can be separated by logical *AND* and *OR* conditions. Moreover, filters can also control the sorting order of the returned query results by providing the capability to sort ascending or descending.

7.4 Web Services

Curio provides a series of REST-based web services that support managing resources, behaviors, and plugins from the user interface, or from external applications that have the appropriate access credentials. Operations available for managing resources and their metadata include creating, updating, or deleting resources, properties, or relationships. There are also services for user authentication, user group management, and management of schemas, spaces, and contexts.

All web services in Curio have been defined by annotating methods according to the JSR-311 specification [88], also known as JAX-RS. JAX-RS allows web services to be exposed from Java methods using a set of Java annotations supported by interfaces and classes. JAX-RS supports creating services that handle GET or POST requests, and supports query parameters, form parameters, or parameters embedded in the URI itself.

While many web services for resource management can be generated by Curio's persistence module, the services it generates are only for basic operations like instantiating or updating objects and their relations, and involve at most two objects. Thus, the functionality of many of the service calls is outside the scope of what can presently be automatically generated, and so were programmed manually.

7.5 Plugins and Behaviors

Curio features a behavior manager that indexes and triggers behaviors bound to ontology patterns. Behaviors are indexed in a trie structure, which is an ordered tree structure where all descendants of a node share a common prefix with the node. While this type of structure is

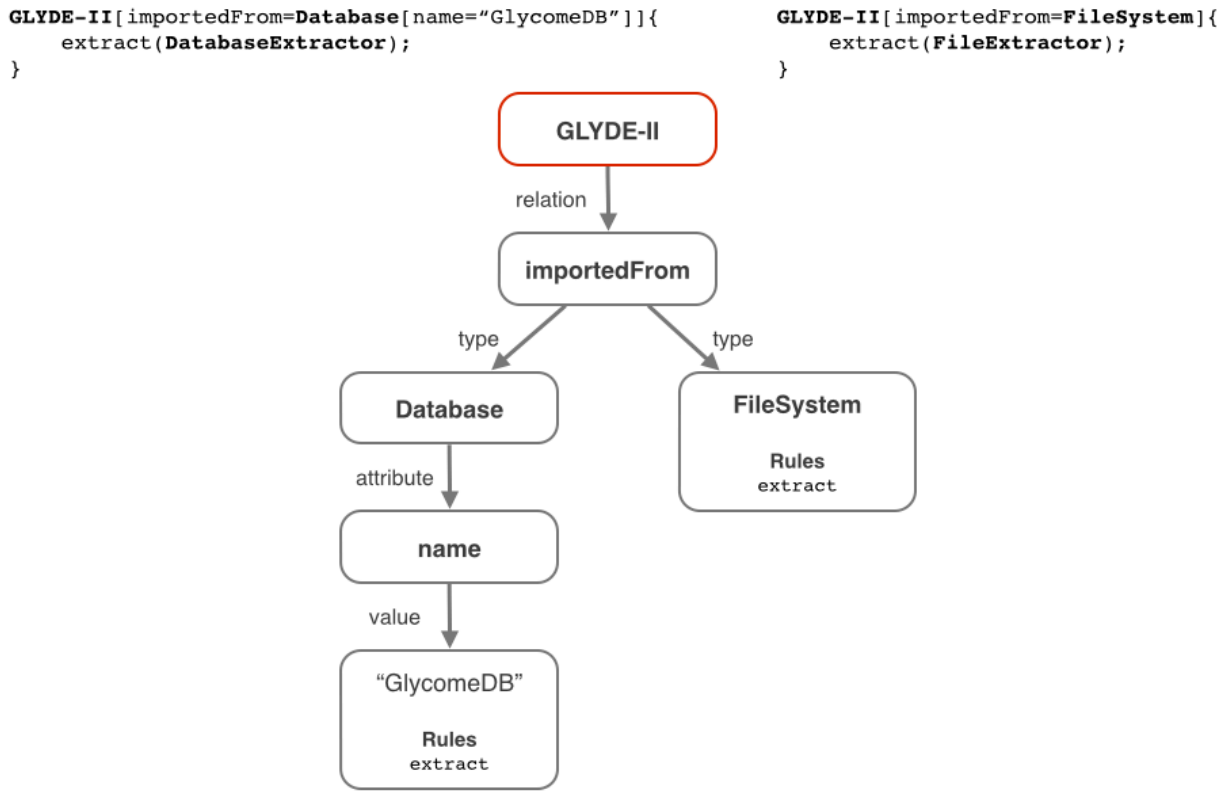

Figure 16: Behavior pattern indexing structure. The class being constrained is outlined in red.

usually associated with string matching, the behavior index contains nodes that represent matching criteria instead. This structure affords efficient storage of behavior patterns, along with efficient lookup speeds. Examples of patterns that have been translated into this structure are shown in Figure 16. Nodes in this structure may contain behaviors if they were the terminal nodes created when the ontology pattern was parsed. When a pattern is matched against the index, behaviors contained within the leaf node are executed, if there are any.

Curio also contains a plugin manager that accepts plugins uploaded by users, stores them, and executes them when explicitly requested, or when triggered by the behavior manager. The plugin manager is built around the OSGi framework, which also powers the plugin systems of popular IDEs like Netbeans and Eclipse. OSGi is a dynamic component system that allows users

to create bundles of software functionality that can be remotely installed, uninstalled, started, stopped, or updated, all without requiring a reboot of the environment. Plugins are where much of the customization of Curio occurs, as new visualization modules, processing modules, or IO modules can be added as needed. Apache Felix was chosen as the OSGi API implementation due to its lightweight design.

7.6 User Interface

Curio's user interface is web-based, and utilizes the web services mentioned in section 7.4. We were able to incorporate various JavaScript and CSS styling frameworks in its construction, which accelerated development considerably. jQuery was chosen to simplify construction of interface elements and various JavaScript widgets that were necessary, and also for its fully featured AJAX library. Bootstrap was incorporated for its simple and attractive styling elements, and useful pre-constructed JavaScript widgets. For heavier visualization, D3.js was chosen for the ease in which it transforms user-provided data into SVG diagrams.

jQuery is a popular library that mitigates many browser-specific JavaScript quirks, and generally makes modifying HTML with JavaScript easier via the use of *selectors*. Selectors allow quickly finding elements within an HTML page by their tag name, class, or id, among other possible constraints. Along with selectors, jQuery allows the ability to execute algorithms on selected elements easily. There is also an extensive AJAX library included in jQuery. Curio's user interface utilizes AJAX heavily, as it provides a more responsive user experience compared to form submissions and page reloading. AJAX allows the interface to incrementally

request updated information that it can load quietly in the background, instead of disrupting what the user is doing by refreshing the entire page.

Bootstrap is a front-end framework that contains various design elements, interface components, and JavaScript widgets for building user interfaces. Notable features of Bootstrap include responsive web design support, a collection of scalable icons supplied by Glyphicons[14], an extensive CSS stylesheet that provides a modern appearance to form fields, tables, headers, and others, and a collection of jQuery plugins.

D3.js, short for Data-Driven Documents, is another JavaScript library that can render datasets as complex, interactive diagrams. D3.js functions much like jQuery in many regards, and is able to use selectors for transforming elements on an HTML page. D3.js is also capable of binding elements within a dataset to SVG or HTML elements rendered in a webpage. Moreover, D3.js can specify transition functions on these elements to support animations. This combination of features allows complex and richly interactive diagrams and interface elements to be created. An earlier incarnation of D3.js, Protovis, gained a measure of acceptance with visualization practitioners and academics [89]. Thus, by including this library as a standard part of Curio's visualization capability, we hope to make the creation of custom data viewers easier for domain specialists.

The default method of browsing resources in Curio's user interface was designed to be very similar to many file-based operating systems' ways of interacting with files and folders. Dragging and dropping resources onto container resources (folders) moves those resources inside the container. Clicking on container resource icons displays its contents in a nested structure similar to the list views that are present in many modern operating systems. Double clicking on

[14] http://www.glyphicons.com

a resource will either navigate inside it if it is a container, or display the contents of the resource if it is not a container and its contents are stored locally. This is still in the early stages of development, and works best with text documents. Other formats could be supported in the future via plugins. Resources and containers may also be "trashed", which actually just flags the resources for removal and does not display them, except when the trash is viewed. Trashed resources may be restored or deleted at the user's discretion. There is also a location bar, much like a web browser's address bar, that displays the current space and path within the space. Other controls within the address bar are for navigating to the previous location, future search capability, refreshing the display, selecting viewer plugins, and showing or hiding the metadata panel.

One of the most useful aspects of Curio's interface is the ability to add and view resource metadata. The metadata panel is located to the right side of the resource viewer, and may be hidden if more space is needed when viewing resources using a plugin. Properties (called Annotations) and relations may be viewed in their respective tabs, and can be added via popups or from a contextual menu when right clicking on a resource. Attributes of currently selected resources are displayed above the metadata, including what resource types are assigned, who the author is, the path of the resource within its space, the location of the resource in its system of origin (if it was imported), and a description of the resource if one was provided.

Curio also features a series of menus at the top of the screen for managing spaces, plugins, behaviors, and user account information. Currently, these menus are still under construction. For example, while a user may switch between spaces from the spaces menu, there is still no way to create new spaces from the interface. Also pending are dialog boxes for plugin management, and behavior specification. Even though plugins are currently global, and may be

utilized by any user of the system, plugin management will be limited to a user's own uploaded plugins. Plugins will be uploaded within OSGi bundles, which may contain multiple plugins of different types. The behavior management panel will also likely have a graphical method of specifying ontology patterns for binding to behaviors. After a pattern is specified, selecting which behaviors to execute should be a simple process.

On the left side of the main resource view panel, there is currently a small panel used for collecting frequently used resources. This area can be considered analogous to a clipboard, or a favorites list, and is useful for collecting resources in order to move them to locations that may require a significant amount of further browsing, or for keeping track of certain key resources when creating relations with other resources. As the user interface becomes more refined, this panel may not need to be as prominent, or may not be needed altogether.

CHAPTER 8

EVALUATION PLAN

Work on converting existing applications to fit within the Curio framework continues. Curio is currently still in the proof-of-concept stage of implementation, and so we still lack a completely refactored end-to-end workflow. However, as implementation progresses we continue to assess if the applications already built for the glycomics domain will be capable of being refactored without losing functionality. Ontology behaviors are being constructed from existing code that duplicates the functionality of these applications. These behaviors mostly require extending the appropriate plugin class, and packaging the plugins as an OSGi bundle. The data from the applications can likewise easily be imported by Curio using importer plugins. Once the applications have been refactored, and their data more tightly integrated within the Curio framework, we expect that they will act more like pieces of a whole, and less like separate, distinct applications. Of course, further analysis of the refactoring process and what effects it has on the functionality of applications will be needed.

The bits of functionality from each application will also have the potential to be shared amongst resources from others. For example, this could allow data from one application to be rendered by another's visualization method, or the creation of significantly more useful metadata than one application would generate on its own.

8.1 QRATOR

With Qrator, we have mostly concentrated on determining whether it can be completely ported into Curio at all, since it contains a variety of very specific functions that would require replication. But, it does make an excellent case study, mainly owing to the fact that it does contain an assortment of functions. We are confident that we will be able to successfully replicate all of its functionality in Curio with a combination of built-in actions and plugins. In the process of formulating a plan for how these functions should be refactored, we realized that there are multiple ways to go about replicating its functionality. Thus, while Curio does support flexibility, which is one of its primary goals, careful attention must still be given to how behaviors are implemented.

As discussed in Section 4, Qrator's primary function is to curate glycan structures. Curation in this case is basically sorting out "good" structures from "bad" structures. Qrator does this by assigning structure classifications based on decisions made by curators as the structure moves through the curation workflow. Structures are assigned one of five statuses: *pending*, *reviewed*, *deferred*, *rejected*, or *approved*. These statuses can also be considered states within the curation workflow. When porting the structure of this workflow to Curio, containers have been established to model each status (or state) of the workflow, along with an additional *import* container. We then assigned these containers the resource type *Queue*. Structure resources may then be transferred between these queues via a combination of controls within a viewer plugin, and the built-in *moveto* behavior.

A viewer plugin was constructed for structure queues using existing elements of Qrator's user interface, borrowing most heavily from the CFG rendering code. The status menu was also

retained, to allow curators to transfer the structure to a different stage of the workflow. Contexts are used to ensure that users only have access to the controls that their curation role allows. Curation roles present in Qrator, *submitter*, *reviewer*, *curator*, and *admin*, were translated into contexts in order to restrict which queues and which interface controls are able to be accessed. The viewer plugin, since it is intended to visualize lists of structures, is bound to any container with the class *Queue*.

GLYDE-II files can be imported from either external databases or the local file system via import plugins, and parsed using our existing GLYDE-II parser embedded within an extractor plugin. We assign a resource type, *Glycan*, to the extracted files, making it easier to bind the *moveto* behavior to the *import* container for automatically transferring them to the *pending* container. The extracted files become the candidate structures, and contain a JSON representation of a glycan's structure, which is also currently used in the Qrator application. This makes it easier to render the structure in CFG notation, again as is done in Qrator.

An extractor plugin will need to be constructed for matching glycans against different canonical trees. The canonical trees, instead of being stored in a database, will take the form of resources whose contents are represented in the same JSON format that is used by the candidate files discussed previously. The plugin itself will repackage the matching algorithm that Qrator currently uses as an OSGi bundle, and will take the canonical tree resources as input. After matching a candidate structure against a canonical tree, a list of potential matches will be presented to a reviewer via the previously mentioned viewer plugin bound to the structure's parent queue. After choosing a match, a candidate structure could potentially have its contents replaced with the selected match's configuration data.

As structures move through the curation workflow, metadata about which user performed which curation step can be recorded automatically via instantiating a series of properties related to curation. Curio automatically records authorship data when a property, relation, type, or instance is created, so all that would be necessary is to define properties corresponding to each stage of curation that could be instantiated when a structure is moved between queues. In the future, this may not be necessary, since we are currently considering adding an additional layer of provenance to record when resources are moved, regardless of the context. Moreover, annotations could easily be added as properties to structures, much as they are in Qrator. References would need to be modeled as separate entities with properties of their own. Properties associated with references could include a link to the source database, the id of the structure in the source database, or potentially publication metadata if the reference is for a publication. While we are currently working to implement publication references in Qrator, it should be noted that this capability is not currently supported. However, if Qrator were ported to Curio, this capability would be easier to implement, as there would not be a need to code anything directly.

8.2 GLYCOVAULT

Most of GlycoVault's functionality is already subsumed by Curio. Thus, its data model could be converted into a schema for glycobiologists to use when organizing their experiments, and serving as a basis on which to share common concepts. Various glycomics applications could also use this schema to integrate their data more effectively by using common classifications, referencing common resources, or even to share common behaviors among resources. Within

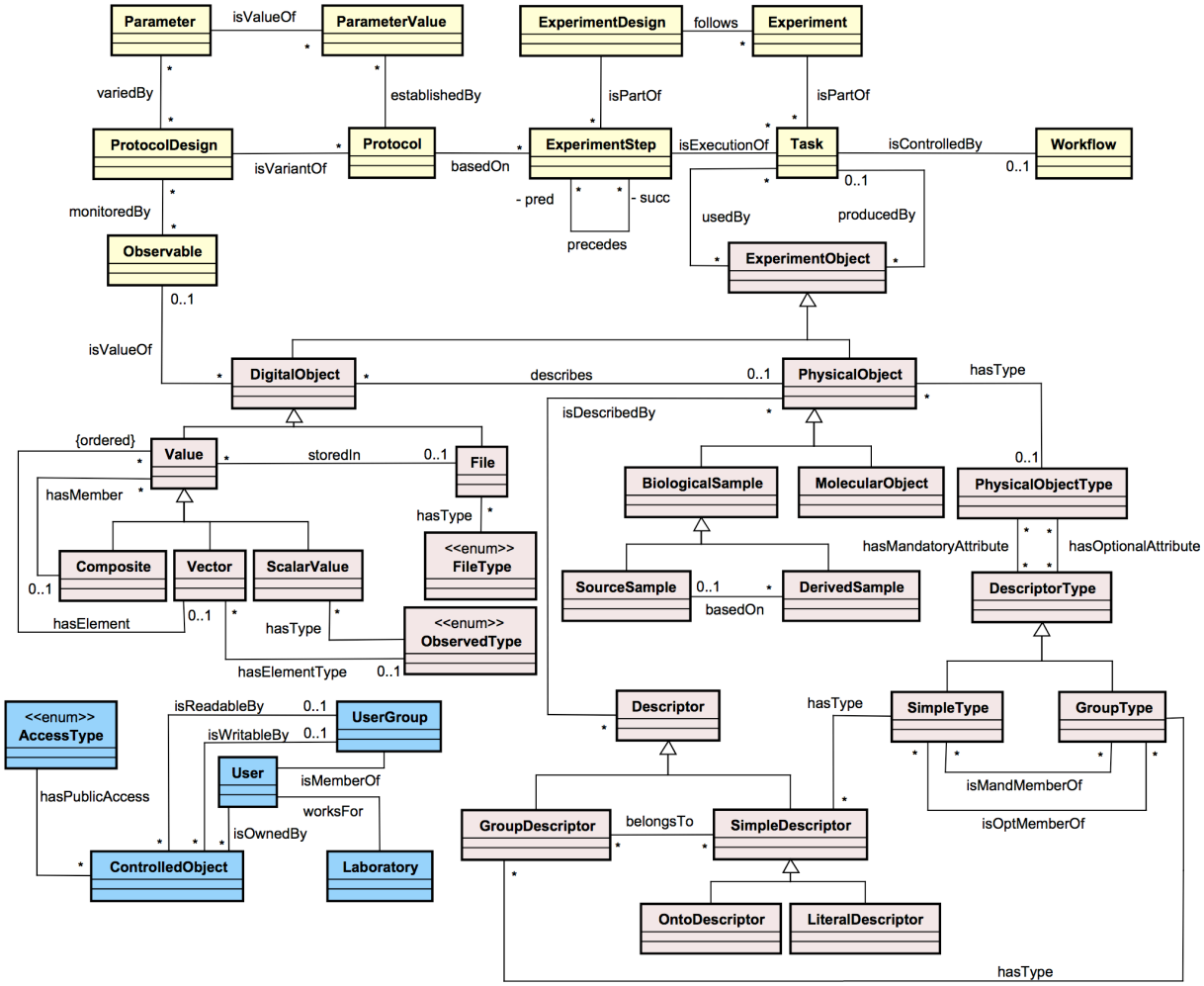

Figure 17: GlycoVault's data model, minus attributes for classes

Curio, however, scientists would not be limited to using GlycoVault's schema, and could pick and choose concepts from the schema to supplement their own data models, or as a starting point for building such models.

GlycoVault's schema was designed to provide a flexible means to design and store experiments associated with glycobiology. Thus, there is the notion of creating templates for experiment workflows and protocols, called "designs". Individual experiments may share common experiment designs, which are sequences of protocol designs arranged in directed acyclic graphs (DAGs). Protocol designs were incorporated because there is also frequently a

need to create variants of protocols with different parameters and observable data. There is also a large section of the schema that is devoted to recording experiment data and resources. This section can accommodate digital data, as well as references and metadata about objects in the physical world. The complete GlycoVault schema is shown in Figure 17. Though this schema was designed for flexibility in storing experiments and experiment data, changes were still made very frequently during the development process to accommodate unforeseen types of data or experiment models. In a system like Curio, this would not be a problem, since it supports easy redesign of data models.

GlycoVault also helps to illustrate the value of Curio's plugin system, since its modules for data import, export, and extraction are currently coded as web services. Modularization of these web services would simplify maintenance considerably, and would make it easier for individual organizations to customize their installations. Furthermore, additional IO plugins could be constructed as required to support future data formats, along with viewers for displaying experiment designs in a graphical manner. D3 has interactive examples of data arranged as a graph structure, which could easily be adapted to the DAG structure present in many glycomics experiment workflows.

8.3 Pathway Browser

The Pathway Browser was constructed to view biosynthetic pathways in conjunction with experiment data by taking ontological data about glycans and pathways from GlycO and ReactO, respectively, and overlaying them with experiment data from GlycoVault. Fortunately, when porting this application to Curio, integrating data from multiple sources becomes significantly

easier. Glycan data could just as easily come from within the port of the Qrator application, since it deals with structure curation and has a large collection of glycans already. Consequently, GlycO may not be needed at all for glycan information, or the GlycO schema and instances could be absorbed into Curio altogether. Experiment data could likewise be taken from the port of GlycoVault. In this scenario, the interaction of data from the two sources happens seamlessly, since they now can exist within the same system.

Porting the Pathway Browser's functionality should require far less data modeling and plugin implementation than Qrator, since it is mostly a visualization and analysis application and does not store information of its own. Likely, its entire functionality could be encompassed by implementing a viewer plugin that allows a user to select a set of glycans within Qrator and transform them into a pathway visualization, or to remotely query ReactO for pathway information if the metadata is not present in Curio. Experiment data from GlycoVault linked with the selected glycans and reactions could then be rendered and overlaid on the pathway diagram.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

We have presented Curio, a software framework that aims to provide scientists with an extensible platform for developing powerful, customizable ways of working with their research data. This framework can help researchers gain a clearer overall picture of their research by allowing them to aggregate and integrate data that may be spread out over multiple systems or data formats. Granting access to specific resources when collaborating on projects is also possible through the use of spaces and contexts, allowing groups of researchers to access common data that they can browse and edit. Data may be further enhanced by using behaviors bound to patterns in the data itself, enabling processes or workflows to be constructed for analysis, visualization, provenance, or other purposes. These two features, supporting complete flexibility when organizing and annotating research while allowing the natural construction of ontologies, and supporting the building of research processes via behaviors, can be extremely powerful tools for interacting with digital information, especially when combined.

We described a method of enhancing research data gradually with metadata, allowing a more organic approach of constructing personal data models that are internally represented as ontologies. The concept of resources was outlined, which are representations of data capable of embodying a wide variety of things, including objects in the real world, research data, experiments, workflows, or documents. Methods of using behavior plugins to support resource

aggregation were also described, as well as using Spaces and Contexts to control access to resources.

Gradually constructing ontologies from data using a gradual, organic approach allows researchers to enhance data in ways that matter to them, rather than being forced to adopt a predefined data model. Modeling scientific domains is still not an exact science, and there is often debate in scientific circles on what aspects of a domain are important to model, or how these aspects should be modeled. Arriving at a complete consensus is not always necessary, however, especially when research is time sensitive and an imperfect domain model is needed quickly that could easily be altered in the future. In these cases, scientists should be given the freedom to organize and model data in the way they prefer, without being forced to adopt a model that they disagree with, or one that does not make sense to them. Thus, Curio strives to support a great degree of flexibility with respect to data modeling, and allows scientists to add as much or as little metadata to their research as they want.

While flexible data modeling is powerful in its own right, the ability to bind behaviors to ontological patterns presents another interesting avenue for researchers to enhance their data. We defined behaviors as pieces of application functionality that are bound to ontological patterns, and triggered when new instances of such patterns are detected. These behaviors can take the form of predefined functions built into Curio, or user-provided plugins that offer custom functionality. To quickly specify such behaviors, we defined the Behavior Description Language, and described how it may be employed by users to define behaviors. Lastly, we showed how behaviors could be used to offer automatic provenance generation, or to model workflows directly within the Curio framework.

Typical usage of behaviors could encompass anything from fully-fledged scientific workflows to simple day-to-day tasks, but behaviors could potentially grant even more exotic capabilities. For example, a network of cooperative behaviors could act as an artificially intelligent agent. Such an agent could act as an automatic curator of sorts for the knowledge base it is bound to, importing new data, updating existing data, and even pushing data updates to other systems as requested.

This framework could also have many uses outside of scientific research. Personal data management, software development, electronic health records, education, and other fields could benefit from the customization that Curio offers. Applications may also exist in the rapidly evolving "Internet of Things". Devices connected to networks could be represented as resources and activated or controlled remotely via behavior plugins. Alternately, such devices could be part of a sensor web, and could actively feed new data into an ontology. The possibilities for customization are quite varied, as are the range of applications, and we plan to explore other uses for Curio in the future.

As development work on Curio continues, our list of future goals continues to expand. Perhaps most importantly, a customizable search module will be needed to support powerful methods of finding resources. This will likely necessitate the introduction of a new type of plugin, in addition to the four existing types. Other plugin types dedicated to data processing may also be introduced, especially with regard to supporting workflows. The user interface requires completion of unfinished sections and controls, and we will also add new functionality. The interface will also be subjected to usability testing, with lots of human feedback. The eventual goal is to have multiple complete, end-to-end workflows in place for demonstration purposes, and then to let scientists begin using the system in their research environments.

We hope that systems like Curio will gradually take the place of current scientific systems that rely on inflexible, difficult to customize research workflows. Scientists would benefit from fewer frustrations in organizing data, powerful integrated process specification, and more collaborative research. The trend towards a more interactive web has fostered a desire for more personalized virtual environments, and more collaboration, as shown with the rise of services like Facebook, or GitHub. Scientists should have the same capability to personalize their research environment and collaborate with other scientists. Giving researchers more powerful and flexible tools with which to do their work can help accelerate the future of scientific research.

REFERENCES

1. Berners-Lee, T., M. Fischetti, and F.B.-M.L. Dertouzos, *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. 2000: HarperInformation.

2. Ailamaki, A., V. Kantere, and D. Dash, *Managing scientific data.* Communications of the ACM, 2010. **53**(6): p. 68-78.

3. *GLYDE-II*. http://glycomics.ccrc.uga.edu/core4/informatics-glyde-ii.html.

4. Domenico, B., et al., *Thematic real-time environmental distributed data services (thredds): Incorporating interactive analysis tools into nsdl.* Journal of Digital Information, 2006. **2**(4).

5. Rew, R. and G. Davis, *NetCDF: an interface for scientific data access.* Computer Graphics and Applications, IEEE, 1990. **10**(4): p. 76-82.

6. Ludäscher, B., et al., *Scientific workflow management and the Kepler system.* Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1039-1065.

7. Belhajjame, K., et al. *Metadata management in the Taverna workflow system*. in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. 2008: IEEE.

8. Deelman, E., et al., *Pegasus: A framework for mapping complex scientific workflows onto distributed systems.* Scientific Programming, 2005. **13**(3): p. 219-237.

9. Shadbolt, N., T. Berners-Lee, and W. Hall, *The Semantic Web Revisited.* IEEE Intelligent Systems, 2006. **21**(3): p. 96-101.

10. Jones, W. *No knowledge but through information*. [personal information management (PIM), personal knowledge management (PKM)] 2010; http://firstmonday.org/ojs/index.php/fm/article/view/3062/2600.

11. Gruber, T.R., *Toward Principles for the Design of Ontologies Used for Knowledge Sharing.* International Journal Human-Computer Studies, 1995. **43**(5): p. 907-928.

12. Berners-Lee, T., J. Hendler, and O. Lassila, *The semantic web.* Scientific american, 2001. **284**(5): p. 28-37.

13. Berners-Lee, T. *Linked data-design issues*. 2006; http://www.w3.org/DesignIssues/LinkedData.html.

14. Cyganiak, R., D. Reynolds, and J. Tennison, *The RDF Data Cube Vocabulary, W3C Working Draft 05 April*. 2012, World Wide Web Consortium.

15. Miles, A. and S. Bechhofer, *SKOS simple knowledge organization system reference*. 2009, Technical report, World Wide Web Consortium.

16. Breslin, J.G., et al., *SIOC: an approach to connect web-based communities.* International Journal of Web Based Communities, 2006. **2**(2): p. 133-142.

17. Soldatova, L.N. and R.D. King, *An ontology of scientific experiments.* Journal of the Royal Society Interface, 2006. **3**(11): p. 795-803.

18. Pease, A., I. Niles, and J. Li. *The suggested upper merged ontology: A large ontology for the semantic web and its applications*. in *Working notes of the AAAI-2002 workshop on ontologies and the semantic web*. 2002.

19. Goble, C. *Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics*. in *Workshop on Data Derivation and Provenance, Chicago*. 2002.

20. Muniswamy-Reddy, K.-K. and M. Seltzer, *Provenance as first class cloud data.* ACM SIGOPS Operating Systems Review, 2010. **43**(4): p. 11-16.

21. Zhao, J., et al. *Annotating, linking and browsing provenance logs for e-Science*. in *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*. 2003: Citeseer.

22. Altintas, I., O. Barney, and E. Jaeger-Frank, *Provenance collection support in the kepler scientific workflow system*, in *Provenance and annotation of data*. 2006, Springer. p. 118-132.

23. Davidson, S.B., et al., *Provenance in Scientific Workflow Systems.* IEEE Data Eng. Bull., 2007. **30**(4): p. 44-50.

24. Davidson, S.B. and J. Freire. *Provenance and scientific workflows: challenges and opportunities*. in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008: ACM.

25. Simmhan, Y.L., B. Plale, and D. Gannon. *A framework for collecting provenance in data-centric scientific workflows*. in *Web Services, 2006. ICWS'06. International Conference on*. 2006: IEEE.

26. Simmhan, Y.L., B. Plale, and D. Gannon, *A survey of data provenance in e-science.* ACM Sigmod Record, 2005. **34**(3): p. 31-36.

27. Bose, R. and J. Frew, *Lineage retrieval for scientific data processing: a survey.* ACM Computing Surveys (CSUR), 2005. **37**(1): p. 1-28.

28. Cheney, J., L. Chiticariu, and W.-C. Tan, *Provenance in databases: Why, how, and where*. Vol. 1. 2009: Now Publishers Inc.

29. Glavic, B. and K.R. Dittrich. *Data Provenance: A Categorization of Existing Approaches*. in *BTW*. 2007: Citeseer.

30. Moreau, L., *The foundations for provenance on the web.* Foundations and Trends in Web Science, 2010. **2**(2–3): p. 99-241.

31. Eavenson, M., et al. *GlycoBrowser - A Tool for Contextual Visualization of Biological Data and Pathways Using Ontologies*. in *4th International Symposium on Bioinformatics Research and Applications (ISBRA 2008)*. 2008. Atlanta, Georgia.

32. Jones, W.P., *Keeping Found Things Found: The Study and Practice of Personal Information Management*. 2008: Morgan Kaufmann Publishers.

33. Frand, J.L. and C.G. Hixson, *Personal knowledge management: Who? What? Why? When? Where? How?* 1999, UCLA Anderson School of Management.

34. Pauleen, D., *Personal knowledge management: Putting the" person" back into the knowledge equation.* Online Information Review, 2009. **33**(2): p. 221-224.

35. Bedford, D.A., *Enabling personal knowledge management with collaborative and semantic technologies.* Bulletin of the American Society for Information Science and Technology, 2012. **38**(2): p. 32-39.

36. Razmerita, L., K. Kirchner, and F. Sudzina, *Personal knowledge management: The role of Web 2.0 tools for managing knowledge at individual and organisational levels.* Online Information Review, 2009. **33**(6): p. 1021-1039.

37. Levy, M., *WEB 2.0 implications on knowledge management.* Journal of knowledge management, 2009. **13**(1): p. 120-134.

38. Franklin, M., A. Halevy, and D. Maier, *From databases to dataspaces: a new abstraction for information management.* ACM Sigmod Record, 2005. **34**(4): p. 27-33.

39. Bush, V., *As we may think.* Atlantic Monthly, 1945. **176**(1): p. 101-108.

40. Davies, S., J. Velez-Morales, and R. King, *Building the Memex Sixty Years Later: Trends and Directions in Personal Knowledge Bases*. 2005, University of Colorado.

41. Nelson, T.H. *As we will think*. in *From Memex to hypertext*. 1991: Academic Press Professional, Inc.

42. Sauermann, L., A. Bernardi, and A. Dengel. *Overview and Outlook on the Semantic Desktop*. in *Semantic Desktop Workshop*. 2005.

43. *MindRaider*. http://mindraider.sourceforge.net.

44. *TheBrain*. http://www.thebrain.com.

45. Sauermann, L., et al., *Semantic desktop 2.0: The gnowsis experience*, in *The Semantic Web-ISWC 2006*. 2006, Springer. p. 887-900.

46. Bechhofer, S., et al., *Research objects: Towards exchange and reuse of digital knowledge.* The Future of the Web for Collaborative Science, 2010.

47. McLelland, A., *LIMS: A Laboratory Toy or a Critical IT Component?* LIMS/Letter, 1998. **4**(2).

48. Gibbon, G.A., *A brief history of LIMS.* Laboratory Automation & Information Management, 1996. **32**(1): p. 1-5.

49. Joyce, J.R. *Industry Insights: Examining the Risks, Benefits and Trade-offs of Today's LIMS*. 2010; http://www.scientificcomputing.com/articles-In-Industry-Insights-Examining-the-Risks-Benefits-and-Trade-offs-of-Today-LIMS-033110.aspx.

50. John, D., M. Banaszczyk, and G. Weatherhead, *The Multidisciplinary Electronic Laboratory Notebook: Pipe Dream or Proven Success?* American Laboratory, 2011. **43**(11): p. 16.

51. *eCat*. http://www.researchspace.com/electronic-lab-notebook/index.html.

52. *InfoGrid*. http://infogrid.org/.

53. *Fedora Repository*. http://fedorarepository.org.

54. Payette, S. and C. Lagoze, *Flexible and extensible digital object and repository architecture (FEDORA)*, in *Research and Advanced Technology for Digital Libraries*. 1998, Springer. p. 41-59.

55. Kahn, R. and R. Wilensky, *A framework for distributed digital object services.* International Journal on Digital Libraries, 2006. **6**(2): p. 115-123.

56. Eavenson, M., et al., *Qrator: A web-based curation tool for glycan structures.* Glycobiology, 2015. **25**(1): p. 66-73.

57. Ranzinger, R., et al., *GlycomeDB – integration of open-access carbohydrate structure databases.* BMC Bioinformatics, 2008. **9**(384).

58. Thomas, C.J., A.P. Sheth, and W.S. York. *Modular Ontology Design Using Canonical Building Blocks in the Biochemistry Domain*. in *International Conference on Formal Ontology in Information Systems (FOIS)*. 2006.

59. Packer, N.H., et al., *Frontiers in glycomics: Bioinformatics and biomarkers in disease An NIH White Paper prepared from discussions by the focus groups at a workshop on the NIH campus, Bethesda MD (September 11-13, 2006).* PROTEOMICS, 2008. **8**(1): p. 8-20.

60. Laine, R.A., *A calculation of all possible oligosaccharide isomers both branched and linear yields 1.05 x 10(12) structures for a reducing hexasaccharide: the Isomer Barrier to development of single-method saccharide sequencing or synthesis systems.* Glycobiology, 1994. **4**(6): p. 759-67.

61. Werz, D.B., et al., *Exploring the structural diversity of mammalian carbohydrates ("glycospace") by statistical databank analysis.* ACS Chem Biol, 2007. **2**(10): p. 685-91.

62. Herget, S., et al., *Statistical analysis of the Bacterial Carbohydrate Structure Data Base (BCSDB): characteristics and diversity of bacterial carbohydrates in comparison with mammalian glycans.* BMC Struct Biol, 2008. **8**: p. 35.

63. Herget, S., et al., *Introduction to Carbohydrate Structure and Diversity.* Bioinformatics for Glycobiology and Glycomics: An Introduction, 2009: p. 21-47.

64. Varki, A., et al., *Essentials of Glycobiology*. 2009, Cold Spring Harbor NY.

65. Ranzinger, R. and W.S. York. *Glyco-Bioinformatics today (August 2011) – Solutions and Problems*. in *2nd Beilstein Symposium on Glyco-Bioinformatics, Cracking the Sugar Code by Navigating the Glycospace*. 2012. Potsdam, Germany.

66. Doubet, S., et al., *The Complex Carbohydrate Structure Database.* Trends in Biochemical Science, 1989. **14**(12): p. 475-477.

67. Lütteke, T., et al., *GLYCOSCIENCES.de: an Internet portal to support glycomics and glycobiology research.* Glycobiology, 2006. **16**(5): p. 71R-81R.

68. Raman, R., et al., *Advancing glycomics: Implementation strategies at the Consortium for Functional Glycomics.* Glycobiology, 2006. **16**(5): p. 82R-90R.

69. Hashimoto, K., et al., *KEGG as a glycome informatics resource.* Glycobiology, 2006. **16**(5): p. 63R-70R.

70. Toukach, P.V., *Bacterial Carbohydrate Structure Database 3: Principles and Realization.* J. Chem. Inf. Model., 2011. **51**(1): p. 159-170.

71. Cooper, C.A., et al., *GlycoSuiteDB: a new curated relational database of glycoprotein glycan structures and their biological sources.* Nucleic Acids Research, 2001. **29**(1): p. 332-335.

72. Egorova, K.S. and P.V. Toukach, *Critical analysis of CCSD data quality.* J Chem Inf Model, 2012. **52**(11): p. 2812-4.

73. Takahashi, N. and K. Kato, *GALAXY (Glycoanalysis by the Three Axes of MS and Chromatography): a Web Application that Assists Structural Analyses of N-Glycans.* Trends in Glycoscience and Glycotechnology, 2003. **15**(84): p. 235-251.

74. Berry, E.Z., *Bioinformatics and Database Tools for Glycans*. 2004, Massachusetts Institute of Technology.

75. Pico, A.R., et al., *WikiPathways: Pathway Editing for the People.* PLoS Biol, 2008. **6**(7): p. e184.

76. Damerell, D., et al., *The GlycanBuilder and GlycoWorkbench glycoinformatics tools: updates and new developments.* Biol Chem, 2012. **393**(11): p. 1357-62.

77. Ranzinger, R., et al., *Glycome-DB.org: a portal for querying across the digital world of carbohydrate sequences.* Glycobiology, 2009. **19**(12): p. 1563-1567.

78. *SPARQL 1.1 Overview*. 2013; http://www.w3.org/TR/sparql11-overview/.

79. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. 2011; http://www.w3.org/TR/CSS2.

80. *XQuery 3.0: An XML Query Language*. 2014; http://www.w3.org/TR/xquery-30/.

81. López-Pellicer, F., et al., *Using a hybrid approach for the development of an ontology in the hydrographical domain.* 2008.

82. Janowicz, K., *Observation‐Driven Geo‐Ontology Engineering.* Transactions in GIS, 2012. **16**(3): p. 351-374.

83. Ra, M., et al. *The Mixed Ontology Building Methodology Using Database Information*. in *Proceedings of the International MultiConference of Engineers and Computer Scientists*. 2012.

84. Khan, L. and F. Luo. *Ontology construction for information selection*. in *Tools with Artificial Intelligence, 2002.(ICTAI 2002). Proceedings. 14th IEEE International Conference on*. 2002: IEEE.

85. Klischewski, R. *Top down or bottom up? How to Establish a Common Ground for Semantic Interoperability within e-Government Communities*. in *E-Government: Modelling Norms and Concepts as Key Issues, Proceedings of 1st International Workshop on E-Government at ICAIL*. 2003: Citeseer.

86. Barreau, D. and B.A. Nardi, *Finding and reminding: file organization from the desktop.* ACM SigChi Bulletin, 1995. **27**(3): p. 39-43.

87. Seltzer, M. and N. Murphy. *Hierarchical File Systems Are Dead*. in *HotOS*. 2009.

88. Hadley, M. and P. Sandoz, *JAX-RS: Java™ API for RESTful Web Services.* Java Specification Request (JSR), 2008. **311**.

89. Bostock, M. and J. Heer, *Protovis: A graphical toolkit for visualization.* Visualization and Computer Graphics, IEEE Transactions on, 2009. **15**(6): p. 1121-1128.

APPENDIX A

PERSISTENCE MODULE DATA MODEL EXAMPLE

```
{
    "entities": {
        "SystemObject": {
            "abstract": true,
            "attributes": [
                {
                    "name": "uri",
                    "size": 1024,
                    "type": "String",
                    "nullable": true
                },
                {
                    "name": "createdOn",
                    "default": "NOW()",
                    "type": "Date"
                },
                {
                    "name": "modifiedOn",
                    "default": "NOW()",
                    "type": "Date"
                }
            ],
            "comment": "An object in the system.",
            "key": "sid"
        },
        "SystemUser": {
            "attributes": [
                {
                    "name": "active",
                    "type": "Boolean",
                    "default": "true"
                },
                {
                    "name": "email",
                    "size": 256,
                    "type": "String",
                    "unique": true
                },
                {
                    "name": "firstName",
                    "size": 256,
                    "type": "String"
                },
                {
                    "name": "lastName",
                    "size": 256,
                    "type": "String"
                },
                {
                    "name": "password",
```

```
            "size": 256,
            "type": "String"
        },
        {
            "name": "lastLogin",
            "default": "NOW()",
            "type": "Date"
        },
        {
            "name": "created",
            "default": "NOW()",
            "type": "Date"
        },
        {
            "name": "behaviors",
            "type": "Text",
            "nullable": true
        }
    ],
    "comment": "A user of the system.",
    "key": "sid",
    "user": true
},
"UserGroup": {
    "comment": "A group of users.",
    "key": "sid",
    "parent": "SystemObject",
    "attributes": [
        {
            "name": "name",
            "size": 256,
            "type": "String"
        },
        {
            "name": "description",
            "default": "No description provided.",
            "size": 1024,
            "type": "String"
        }
    ],
    "unique": [
        [
            "name",
            "systemuserowns"
        ]
    ]
},
"SystemContainer": {
    "abstract": true,
    "comment": "A container for system objects.",
    "key": "sid",
    "parent": "SystemObject",
    "attributes": [
        {
            "name": "name",
            "size": 256,
            "type": "String"
```

```
                },
                {
                    "name": "description",
                    "default": "No description provided.",
                    "size": 1024,
                    "type": "String"
                }
            ]
        },
        "SchemaObject": {
            "abstract": true,
            "comment": "A parent class for resource types, property types,
and relation types.",
            "key": "sid",
            "parent": "SystemObject",
            "attributes": [
                {
                    "name": "name",
                    "size": 256,
                    "type": "String"
                },
                {
                    "name": "description",
                    "default": "No description provided.",
                    "size": 1024,
                    "type": "String"
                }
            ]
        },
        "InstanceObject": {
            "abstract": true,
            "comment": "A parent class for resources, properties, and
relations.  InstanceObjects are instances of SchemaObjects.",
            "key": "sid",
            "parent": "SystemObject"
        },
        "Context": {
            "comment": "A context reference for resources, properties, and
relations.  Useful for faceted browsing. (tag? view?)",
            "key": "sid",
            "parent": "SystemContainer",
            "unique": [
                [
                    "name",
                    "systemuserowns"
                ]
            ]
        },
        "Space": {
            "comment": "A container for resources, properties, and relations.
Spaces can be used to organize data, or allow access to certain Resources for
collaborative purposes.",
            "key": "sid",
            "parent": "SystemContainer",
            "unique": [
                [
                    "name",
```

```
                    "systemuserowns"
                ]
            ]
        },
        "Schema": {
            "comment": "A container for types of resources, properties, and
relations.",
            "key": "sid",
            "parent": "SystemContainer",
            "unique": [
                [
                    "name"
                ]
            ]
        },
        "Resource": {
            "comment": "A resource.",
            "key": "sid",
            "parent": "InstanceObject",
            "attributes": [
                {
                    "name": "name",
                    "size": 256,
                    "type": "String"
                },
                {
                    "name": "description",
                    "default": "No description provided.",
                    "size": 1024,
                    "type": "String"
                },
                {
                    "name": "container",
                    "type": "Boolean"
                },
                {
                    "name": "trashed",
                    "type": "Boolean",
                    "default": false
                },
                {
                    "name": "sourcePath",
                    "type": "String",
                    "size": 1024,
                    "nullable": true
                }
            ],
            "unique": [
                [
                    "name",
                    "hasparentresource"
                ],
                [
                    "sourcePath",
                    "systemuserowns"
                ]
            ]
```

```
},
"ResourceContent": {
    "comment": "A resource's contents.",
    "key": "sid",
    "attributes": [
        {
            "name": "hash",
            "type": "String",
            "size": 128,
            "unique": true
        },
        {
            "name": "contents",
            "type": "Text"
        }
    ]
},
"Property": {
    "comment": "A property of a resource.",
    "key": "sid",
    "parent": "InstanceObject",
    "attributes": [
        {
            "name": "value",
            "type": "String",
            "size": 1024
        }
    ],
    "unique": [
        [
            "value",
            "resourcehasproperty",
            "hastypepropertytype"
        ]
    ]
},
"Relation": {
    "comment": "A relation connecting two resources.",
    "key": "sid",
    "parent": "InstanceObject"
},
"Source": {
    "comment": "An external source of resources.",
    "key": "sid",
    "parent": "SystemObject",
    "attributes": [
        {
            "name": "name",
            "size": 256,
            "type": "String"
        },
        {
            "name": "description",
            "default": "No description provided.",
            "size": 1024,
            "type": "String"
        },
```

```
        {
            "name": "location",
            "size": 1024,
            "type": "String"
        }
    ]
},
"ResourceType": {
    "comment": "A resource type.",
    "key": "sid",
    "parent": "SchemaObject",
    "unique": [
        [
            "name",
            "residesinschema"
        ]
    ]
},
"PropertyType": {
    "comment": "A property type.",
    "key": "sid",
    "parent": "SchemaObject",
    "unique": [
        [
            "name",
            "resourcetypehaspropertytype",
            "residesinschema"
        ]
    ]
},
"RelationType": {
    "comment": "A relation type.",
    "key": "sid",
    "parent": "SchemaObject",
    "unique": [
        [
            "name",
            "hasdomainresourcetype",
            "hasrangeresourcetype",
            "residesinschema"
        ]
    ]
},
"SourceType": {
    "comment": "A type of external source.",
    "key": "sid",
    "parent": "SystemObject",
    "attributes": [
        {
            "name": "name",
            "size": 256,
            "type": "String"
        },
        {
            "name": "description",
            "default": "No description provided.",
            "size": 1024,
```

```
                    "type": "String"
                }
            ],
            "unique": [
                [
                    "name"
                ]
            ]
        },
        "PluginReference": {
            "abstract": true,
            "comment": "A reference to a module that extends system
capabilities.",
            "key": "sid",
            "parent": "SystemObject",
            "attributes": [
                {
                    "name": "name",
                    "size": 256,
                    "type": "String",
                    "unique": true
                },
                {
                    "name": "serviceClass",
                    "size": 256,
                    "type": "String",
                    "unique": true
                },
                {
                    "name": "description",
                    "default": "No description provided.",
                    "size": 1024,
                    "type": "String"
                },
                {
                    "name": "bundleId",
                    "type": "Long"
                }
            ]
        },
        "ImporterReference": {
            "comment": "A reference to a plugin that can import resources
from an external source.",
            "key": "sid",
            "parent": "PluginReference"
        },
        "ExtractorReference": {
            "comment": "A reference to a plugin that can extract additional
resources from the contents of a resource.",
            "key": "sid",
            "parent": "PluginReference"
        },
        "ExporterReference": {
            "comment": "A reference to a plugin that can export resources to
a data representation format.",
            "key": "sid",
            "parent": "PluginReference"
```

```
        },
        "ViewerReference": {
            "comment": "A reference to a plugin that can visualize resources
in a contextually relevant manner.",
            "key": "sid",
            "parent": "PluginReference"
        }
    },
    "relations": [
        {
            "cardinality": "OneToMany",
            "from": "SystemUser",
            "fromRole": "Owner",
            "name": "Owns",
            "nullable": false,
            "creator": true,
            "to": "SystemObject",
            "toRole": "Owned"
        },
        {
            "cardinality": "OneToMany",
            "from": "SystemUser",
            "fromRole": "Creator",
            "name": "Created",
            "nullable": false,
            "creator": true,
            "to": "SystemObject",
            "toRole": "Creation"
        },
        {
            "cardinality": "ManyToOne",
            "from": "InstanceObject",
            "fromRole": "Resident",
            "name": "ResidesIn",
            "nullable": true,
            "to": "Space"
        },
        {
            "cardinality": "ManyToOne",
            "from": "SchemaObject",
            "fromRole": "Resident",
            "name": "ResidesIn",
            "nullable": true,
            "to": "Schema"
        },
        {
            "cardinality": "ManyToMany",
            "from": "SystemUser",
            "name": "BelongsTo",
            "nullable": true,
            "to": "UserGroup"
        },
        {
            "cardinality": "ManyToMany",
            "from": "UserGroup",
            "name": "CanRead",
            "to": "SystemContainer",
```

```
    "toRole": "Container"
},
{
    "cardinality": "ManyToMany",
    "from": "UserGroup",
    "name": "CanWrite",
    "to": "SystemContainer",
    "toRole": "Container"
},
{
    "cardinality": "ManyToOne",
    "from": "Resource",
    "fromRole": "Child",
    "name": "HasParent",
    "nullable": true,
    "to": "Resource",
    "toRole": "Parent"
},
{
    "cardinality": "ManyToOne",
    "from": "Resource",
    "fromRole": "Target",
    "name": "ExtractedFrom",
    "nullable": true,
    "to": "Resource",
    "toRole": "ExtractionSource"
},
{
    "cardinality": "ManyToOne",
    "from": "ResourceType",
    "fromRole": "Child",
    "name": "HasParent",
    "nullable": true,
    "to": "ResourceType",
    "toRole": "Parent"
},
{
    "cardinality": "ManyToOne",
    "from": "ExtractorReference",
    "name": "ExtractsFrom",
    "nullable": true,
    "to": "ResourceType"
},
{
    "cardinality": "ManyToOne",
    "from": "ImporterReference",
    "name": "ImportsFrom",
    "nullable": true,
    "to": "Source"
},
{
    "cardinality": "ManyToOne",
    "from": "Resource",
    "name": "ImportedBy",
    "nullable": true,
    "to": "ImporterReference"
},
```

```
{
    "cardinality": "ManyToOne",
    "from": "PropertyType",
    "fromRole": "Child",
    "name": "HasParent",
    "nullable": true,
    "to": "PropertyType",
    "toRole": "Parent"
},
{
    "cardinality": "ManyToOne",
    "from": "RelationType",
    "fromRole": "Child",
    "name": "HasParent",
    "nullable": true,
    "to": "RelationType",
    "toRole": "Parent"
},
{
    "cardinality": "ManyToOne",
    "from": "RelationType",
    "name": "HasDomain",
    "nullable": true,
    "to": "ResourceType",
    "toRole": "Domain"
},
{
    "cardinality": "ManyToOne",
    "from": "RelationType",
    "name": "HasRange",
    "nullable": true,
    "to": "ResourceType",
    "toRole": "Range"
},
{
    "cardinality": "ManyToMany",
    "from": "Resource",
    "name": "HasType",
    "to": "ResourceType"
},
{
    "cardinality": "ManyToOne",
    "from": "Resource",
    "name": "HasContents",
    "nullable": true,
    "to": "ResourceContent"
},
{
    "cardinality": "OneToMany",
    "from": "ResourceType",
    "name": "HasPropertyType",
    "nullable": true,
    "to": "PropertyType"
},
{
    "cardinality": "OneToMany",
    "from": "Resource",
```

```
            "name": "HasProperty",
            "nullable": false,
            "to": "Property"
        },
        {
            "cardinality": "ManyToOne",
            "from": "Property",
            "name": "HasType",
            "nullable": false,
            "to": "PropertyType"
        },
        {
            "cardinality": "ManyToOne",
            "from": "Relation",
            "name": "HasType",
            "nullable": false,
            "to": "RelationType"
        },
        {
            "cardinality": "ManyToOne",
            "from": "Relation",
            "name": "HasDomain",
            "nullable": false,
            "to": "Resource",
            "toRole": "Domain"
        },
        {
            "cardinality": "ManyToOne",
            "from": "Relation",
            "name": "HasRange",
            "nullable": false,
            "to": "Resource",
            "toRole": "Range"
        },
        {
            "cardinality": "ManyToOne",
            "from": "Source",
            "name": "HasType",
            "nullable": false,
            "to": "SourceType"
        },
        {
            "cardinality": "ManyToOne",
            "from": "Resource",
            "name": "ImportedFrom",
            "nullable": true,
            "to": "Source"
        }
    ],
    "entity-implementation-package": "edu.uga.cs.curio.obj.entity.impl",
    "entity-interface-package": "edu.uga.cs.curio.obj.entity",
    "relation-implementation-package": "edu.uga.cs.curio.obj.relation.impl",
    "relation-interface-package": "edu.uga.cs.curio.obj.relation",
    "service-package": "edu.uga.cs.curio.web.generated",
    "manager-package": "edu.uga.cs.curio.manage.generated",
    "session-manager": "edu.uga.cs.curio.access.SystemSessionManager",
    "author": ""
```

```
}
```