# Lightweight Ontology-Based Tools for Managing Observational Data

Shawn Bowers[1], Riley Englin[1], Carlos Fonseca[1], Paul Jewell[1], Lauren Joplin[1], Patrick Mosca[1], Tyler Pacheco[1], Jacob Troxel[1], Tyler Weeks[1]

[1]Deptartment of Computer Science, Gonzaga University, Spokane, WA, USA

**Abstract.** We describe recent ontology and annotation editing capabilities to a specialized data management system for observational data. The system supports observations and measurements explicitly, allowing users to upload observational data sets as well as semantically describe and query data sets using formal OWL-DL ontologies. Recent extensions allow users to extend observational ontologies with domain-specific terms as well as provide detailed semantic annotations using a "markdown"-based approach. In addition, we describe a new implementation of the system using standard semantic web technologies for managing OWL-DL ontologies and RDF triples. Our approach supports a wide variety of observational data, and is especially targeted at helping scientists manage heterogeneous biodiversity and ecological data by allowing access to data through a common and generic observations and measurements data model.

## 1 Introduction

Performing an ecological analysis to study phenomena across geographic, temporal, or biological scales typically requires access to a variety of existing (already collected) observational data sets. A major challenge when performing such an analysis is understanding and reconciling the structural and semantic differences among data sets. In particular, data sets often differ in the number of attributes, the names of similar attributes, the relationships implied between attributes, and the coding conventions used for representing information within data sets. These differences not only make discovering relevant data difficult, but also requires researchers to spend considerable time interpreting and integrating potential data sets for use within any particular analysis. We aim to help address these challenges by providing a suite of lightweight, ontology-based tools that allow researchers to semantically describe, access, and analyze heterogeneous data sets (either their own, or those collected for use in research studies). In this paper, we describe tools that have recently been developed within the ObsDB system [5], which provides data management support built on top of a generic ontology model for formally representing observations and measurements [6]. Within ObsDB, data sets are viewed as semantically described collections of observations. In particular, when data is registered with ObsDB, it is converted automatically into the appropriate observational structure (and represented within the current version of ObsDB as an RDF graph). This approach allows otherwise hard to manage, heterogeneous table structures to be viewed and accessed uniformly as collections of observations and measurements.

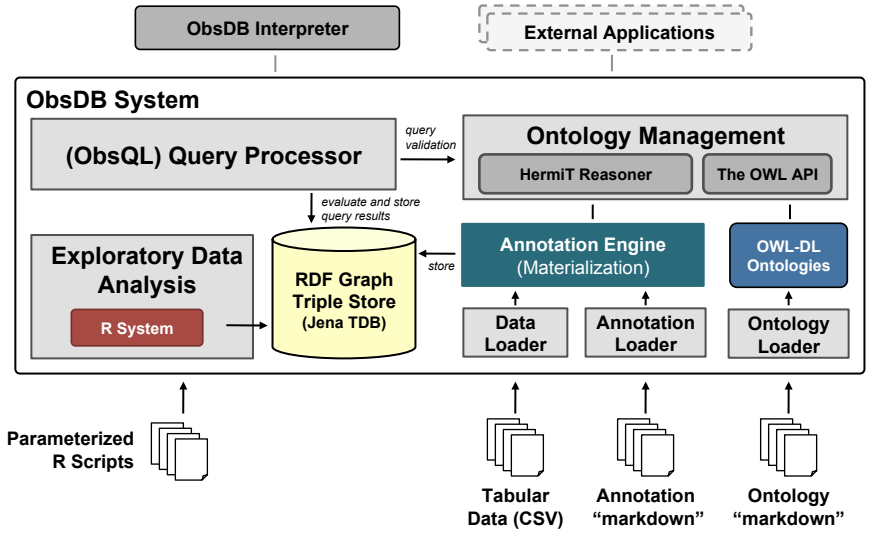


**Fig. 1.** Overview of the ObsDB system architecture.

The tools we have developed leverage a "markdown"-based approach for defining ontologies, for specifying data-level semantic annotations (from which data is "shredded" into the observational model), and for expressing data discovery queries. The "markdown" is then converted automatically within ObsDB (see Figure 1) into corresponding descriptions in OWL-DL (for representing ontologies), RDF (for storing observation and measurement instances), and SPARQL (for executing discovery queries and filtering collections). ObsDB also allows observations and measurements from multiple data sets to be merged into "virtual" observation collections (RDF graphs), which can be saved and further accessed and queried. Ontologies within ObsDB allow observations and measurements to carry rich semantic descriptions, including the types of entities observed, the characteristics of entities measured, the context in which observations were made, and detailed measurement standards (units) for facilitating unit conversion. ObsDB also supports an expressive query language for selecting data sets and filtering data by observation and measurement types. For instance, users can specify queries to find all data sets that contain specific measurements of entities (e.g., diameter and height measurements of trees), relationships and constraints among entities (e.g., the length of branches on trees of a minimum height and at specific elevations), and the use of desired measurement standards (e.g., in meters). Similar queries can also be expressed to obtain all observations (either within or across collections) matching such criteria, where unit conversions are automatically applied as needed. Through integration with the R system [1], analytical scripts can also be called from within ObsDB to perform a variety of exploratory analyses over observation collections.

[1] `http://www.r-project.org`

In this paper, we extend our prior work on ObsDB [5,8] by describing recent extensions to the system, focusing in particular on new support for ontology modeling, annotation, and querying, and its implementation over underlying semantic web technologies. We demonstrate our "markdown"-based approach using examples drawn from real-world ecological data, and also describe our ongoing and future work on further extending ObsDB with the goal of helping researchers more effectively manage heterogeneous observational data.

Figure 1 shows the main architectural components of ObsDB. The ObsDB system is implemented in Java and can be used from within other (external) applications (via API calls) or by using the ObsDB interpreter. ObsDB manages user loaded data set files, semantic annotation files, and ontology files. Ontologies are converted to OWL-DL files by ObsDB and are stored and managed using the OWL API[2]. Annotation files can be applied to data sets to produce a "materialized" set of RDF triples (an RDF Graph). All RDF data is stored within ObsDB using the Jena triple store[3] technology. Users can query RDF Graphs using the ObsDB query processor, which converts high-level queries expressed in ObsQL (the query language of ObsDB) into corresponding SPARQL queries. As part of the query evaluation process, ObsDB uses the HermiT OWL-DL reasoner for query expansion (which is also used to verify semantic annotations are semantically consistent). Finally, R scripts can be defined and registered with ObsDB to perform statistical and analytical operations over RDF Graphs stored within ObsDB.

The rest of this paper describes these features in more detail. Section 2 describes the underlying observations ontology employed by ObsDB and its newly supported ontology "markdown" approach. Section 3 describes the new semantic annotation approach employed by ObsDB. Section 4 briefly describes ObsQL and its new implementation in ObsDB. Finally, Section 5 concludes by describing related work and our future directions for ObsDB.

## 2 Ontology Creation and Management

The ObsDB system is built on a recent version of the Extensible Observation Ontology (OBOE) [6][4]. The OBOE model is implemented in OWL-DL and is compatible with the O&M ISO standard developed by the Open Geospatial Consortium (OGC) [1]. Figure 2 shows the top-level classes and properties supported by OBOE (the primary "OBOE core" classes). An *observation* is made of an *entity* (e.g., biological organisms, geographic locations, or environmental features, etc.) and primarily serves to group a set of measurements together to form a single observation event. A *measurement* assigns a value to a *characteristic* of the observed entity (e.g., the height of a tree), where a value is represented through a special class (similar to the notion of value partitions in [15]). Measurements also include *standards* (e.g., units) for relating values across measurements, and can specify additional information including collection protocols,

[2] `http://owlapi.sourceforge.net/`

[3] `http://jena.apache.org/documentation/tdb/`

[4] See `https://code.ecoinformatics.org/code/semtools/trunk/dev/oboe/oboe.1.1rc1/oboe-core.owl`

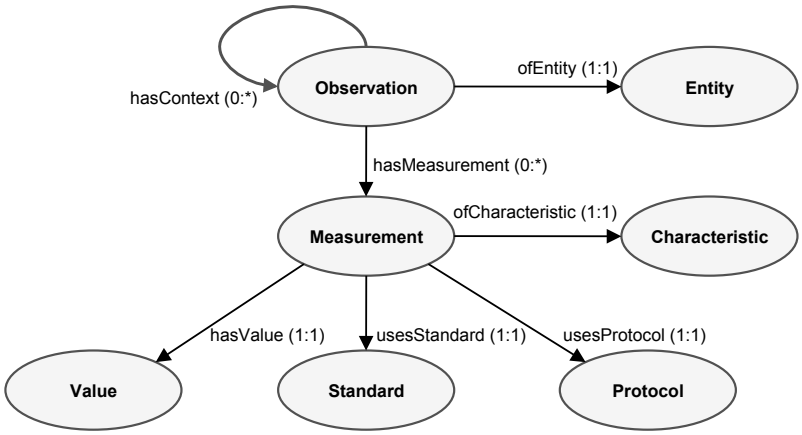


**Fig. 2.** Main concepts and properties in the ObsDB observations ontology (OBOE).

methods, precision, and accuracy (not all of which are shown in Figure 2). An observation (event) can occur within the *context* of zero or more other observations, e.g., an observation of a tree specimen may have been made within a specific geographic location, and the geographic location provides important information for interpreting and comparing tree measurements. In this case, by establishing a context relationship between the tree and location observations, the measured values of the location are assumed to be constant with respect to the measurements of the tree. Context forms a *transitive* relationship among observations. A key feature of the model is its ability for users to assert properties of entities (as measurement characteristics or contextual relationships) without requiring these properties to be interpreted as inherently (or always) true of the entity. Depending on the context an entity was observed in, its properties may take on different values. For instance, the diameter of a tree changes over time, and the diameter value often depends on the protocol used to obtain the measurement. The observation and measurement structure of Figure 2 allows RDF-style assertions about entities while allowing for properties to be contextualized (i.e., the same entity can have different values for a characteristic under different contexts), which is a crucial feature for modeling scientific data [6]. The primary differences between O&M and OBOE are that (1) OBOE was designed to explicitly be represented in OWL-DL; and (2) OBOE treats an observation (event) as a collection of measurements, allowing observations to be defined within a context hierarchy (which implicitly applies to an observation's associated measurements) as opposed to O&M which requires each measurement's context to be stated explicitly.

Figure 3 shows the main classes and properties defined in OBOE for representing measurement standards, including units of measure. Every measurement unit is associated with a measurement characteristic (e.g., length, mass, time, area, volume, etc.) and the set of units are divided into four subclasses. A *base unit* represents a unit that cannot be naturally divided into smaller units. Examples include meter, gram, second, kelvin, and so on. A *prefixed unit* applies a prefix (represented as a literal value) to a

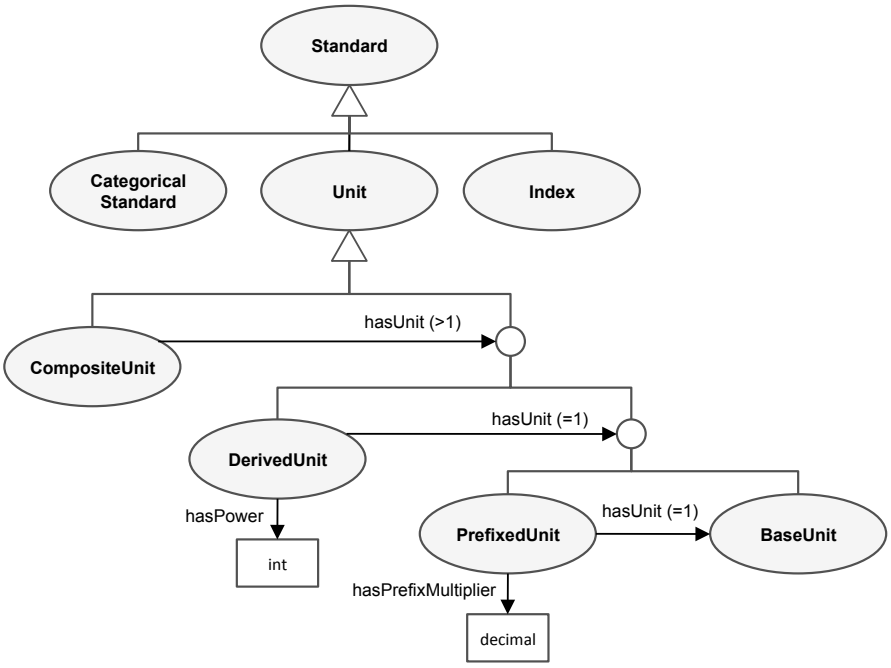


**Fig. 3.** The basic class hierarchy for describing measurement units (OBOE).

base unit. Examples include kilogram (with base unit gram and multiplier 1000) and centimeter (with base unit meter and multiplier 0.01). A *derived unit* assigns a power (other than 1) to either a prefixed or base unit. Examples include meter squared ($m^2$), hertz ($s^-1$), and microliter ($mm^3$). A *composite unit* combines 2 or more derived, prefixed, or base units. Examples include meter per second (which is composed of meter and a derived per second unit, i.e., $m \times s^{-1}$) as well as "dimensionless" units like gram per gram ($g/g$) in which retaining the original units is often needed for interpreting and integrating data.

Users of ObsDB can create their own ontologies that extend OBOE with domain-specific terms when annotating data. ObsDB supports a lightweight "markdown" syntax for describing terms, which was developed specifically to support OBOE modeling constructs and common modeling patterns. The advantage of having a lightweight syntax is that it allows non-expert users in OWL-DL to create and edit terms as needed, without needing to learn DL syntax (required, e.g., in Protege[5] or if editing OWL-DL/RDF syntax directly). Ontologies expressed in the lightweight syntax are automatically converted to the corresponding OWL-DL representation by ObsDB (see Figure 1). The lightweight syntax is based on YAML[6] and provides support for the following tasks:

(i). Importing domain-specific OBOE ontologies for extension;

[5] `http://protege.stanford.edu/`
[6] `http://www.yaml.org/`

(ii). Creating entity, characteristic, and protocol hierarchies;

(iii). Defining characteristic qualifiers (e.g., to specify an "average" length characteristic where "average" denotes the qualifier);

(iv). Creating base, prefixed, derived, and composite units;

(v). Specifying unit conversions; and

(vi). Defining categorical measurement standards.

For instance, the following example establishes a simple class hierarchy using the ObsDB lightweight ontology syntax:

```
!Entity
   name: "Organism"
   childClass:
      !Entity
      name: "Tree"
      # basic types of trees
      childClass:
         !Entity
         name: "DominantTree"
         comment: "A tree that extends above surrounding ...
      childClass:
         !Entity
         name: "OvertoppedTree"
         comment: "A tree that cannot sufficiently extend its crown ...
         equivalentClass: "SuppressedTree"
      ...
      # different species of trees
      childClass:
         !Entity
         name: "DouglasFir"
         equivalentClass: "Pseudotsuga_menziesii"
         comment: "See Garrison et al., 1972"
      ...
```

In this example, a tree class is defined as a subclass of a generic organism class. The tree class is also defined with three subclasses (dominant, overtopped, and Douglas fir). Each class has a name and an optional comment. In addition, equivalent classes (i.e., synonyms) can be specified as in the case of an overtopped tree (defined as being equivalent to a suppressed tree) and with Douglas fir (where in addition to the common name the taxonomic name is also given). The following defines an example physical characteristic.

```
import char: "http://code.ecoinformatics.org/.../oboe-characteristics.owl"
!PhysicalCharacteristic
   name: "DiameterAtBreastHeight"
   parentClass: "char:Diameter"
```

Here diameter at breast height (DBH) is defined as a subclass of the diameter class, which is imported from another ontology (as given by the `import` statement). The following example defines a simple base unit, composite unit, and unit conversion.

```
!BaseUnit
   name: "Meter"
   characteristic: "oboe:Length"
!CompositeUnit
   name: "MeterPerSecond"
   characteristic: "oboe:Speed"
   allUnits:
      - "Meter"
      - !Derived
```

```
            baseUnit: "Second"
            power: -1
   !UnitConversion
      name: "FootToMeter"
      source: "Foot"
      target: "Meter"
      multiplier: 0.3048
      offset: 0
```

In this example, the composite unit is defined over the base unit meter and a derived unit defined "on the fly" (i.e., without providing a specific name to the unit). Finally, the following illustrates a simple categorical standard definition.

```
!CategoricalStandard
   name: "TreeGrowthVigorStandard"
   comment: "Standard values for good, fair, and poor tree growth vigor"
   values: "TreeGrowthVigorValue" {"good_tree_growth_vigor",
              "fair_tree_growth_vigor", "poor_tree_growth_vigor"}
```

In this case, we are defining a value partition (as in [15]) consisting of three values representing good, fair, and poor tree growth.

After starting the ObsDB interpreter, users can load ontology files using the `load onto` command. When loading an ontology file, a namespace prefix and URI is also assigned to the ontology for use within ObsDB. For instance, the following shows the result of starting ObsDB and loading the "ont1.yml" ontology file:

```
ObsDB v1.0
Type 'help' for a list of commands. Type 'quit' to quit ObsDB.
> import onto 'ont1.yml' as 'ont1' using 'http://obsdb.org/ont1'
Ontology created
Ontology loaded
```

In this case, we are assigning the ontology the namespace prefix "ont1" and the URI "http://obsdb.org/ont1". Once loaded, the ontology can be accessed via the namespace. Ontologies can also be updated using the ObsDB `update onto` command, which allows ontologies to be modified without having to remove (or `drop`) an ontology and then load the updated version.

In general, we have found that using a lightweight approach such as this has a number of benefits for specifying OBOE extensions and for annotating data. In particular, the approach allows new ontology terms and entire ontologies to be quickly and easily created by simply opening and editing a text file, and the high-level syntax supports otherwise complex description-logic definitions without requiring users to be experts in description logic (which is often the case in Protege's OWL-DL editor). The latter is especially an issue in ontologies like OBOE that leverage description logic constraints that must be maintained, e.g., via modeling patterns such as value partitions and various class and property restrictions. Once a user loads an ontology into ObsDB, the system automatically performs syntax and semantic validation (e.g., checking for inconsistencies). Together with the lightweight text-based syntax, this allows for rapid editing, loading, and validation of OBOE ontology extensions.

## 3 Observational Data Sets and Semantic Annotations

Semantic annotations in ObsDB define how to translate a tabular data set into a corresponding collection of semantically relevant observations and measurements. Semantic

| STAND | PLOT | TAG | SPP | YEAR | DBH | CANCLASS | VIGOR |
|---|---|---|---|---|---|---|---|
| B388 | 1 | 3319 | PSME | 1999 | 22.5 | C | 1 |
| B388 | 1 | 3320 | PSME | 1999 | 16 | I | 1 |
| B388 | 2 | 3336 | PSME | 1999 | 33 | D | 1 |
| B388 | 2 | 3339 | CACH | 1999 | 5.8 | S | 1 |
| B646 | 1 | 1817 | PSME | 1999 | 22 | C | 1 |
| B646 | 1 | 1815 | CACH | 1999 | 5.7 | I | 1 |
| B684 | 2 | 2207 | ALRU | 1999 | 19.9 | C | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... |

**Fig. 4.** Example data set consisting of tree (allometry) observations and measurements.

annotations are defined using *semantic templates* [8] that specify the observation and measurement types (and their various relationships) for the data set. The observations and measurements given by each template are automatically filled in (to create observation and measurement instances) based on user-defined mappings from data set attributes to measurement types. For instance, consider the example data set in Figure 4. This data set[7] consists of eight attributes and approximately three thousand rows of data (only six of the rows are shown in the figure). The first two attributes specify contextual information concerning the stand and plot where the tree was observed. We can annotate these attributes using the annotation "markdown" syntax supported by ObsDB as follows. For instance, the attribute denoting the stand is annotated by the semantic template:

```
import ont1: 'http://obsdb.org/ont1'
observation 'StandObs':
   entity: 'ont1:Stand'
   measurement:
      characteristic: 'obs:Name'
      value: '$STAND'
   entityKey: '$STAND'
```

which creates an observation individual of a stand entity for each unique value of the STAND attribute in the data set (thus, for stand B388, B646, and B684 in Figure 4). The measurement in this case is simply the name of the stand, which is taken directly from the attribute values. The entityKey field of the template specifies that each unique value should generate a new observation (as opposed to each row, regardless of the STAND value, generating a new observation). In this example, we also import a domain-specific ontology and assign a namespace prefix to be used to refer to corresponding classes within the annotation file. Similar to the stand template, the following template can be used to annotate the plot information in the data set.

```
observation 'PlotObs':
   entity: 'ont1:Plot'
   measurement:
      characteristic: 'obs:Name'
      value: '$PLOT'
   context: 'StandObs'q
   entityKey: '$PLOT' within 'StandObs'
```

[7] Based on one of the many data sets available on the H.J. Andrews Experimental Forest LTER site (`http://andrewsforest.oregonstate.edu/`).

Here the plot is nested within the stand, and so each plot observation has as context the corresponding stand observation. In this data set, the names of plots across stands are not unique (e.g., stand B288 contains a plot 1 as does stand B646). This information is denoted using the "within" keyword. Although not included here, additional measurements can also be added to the template, e.g., the area of the plot (which in this case could be specified as a constant value assuming all plots are of the same area). The year attribute would be annotated similarly to the stand as follows.

```
observation 'YearObs':
   entity: 'ont1:TimePeriod'
   measurement:
      characteristic: 'ont1:Year'
      value: '$YEAR'
   entityKey: '$YEAR'
```

The remaining attributes would be annotated via a single tree observation template:

```
observation 'TreeObs':
   entity: match '$CANCLASS' with
      'D' => 'ont1:DominantTree'
      'S' => 'ont1:SuppressedTree'
      ...
   entity: match '$SPP' with
      'PSME' => 'ont1:Pseudotsuga_menziesii'
      'CACH' => 'ont1:Castanopsis_chrysophylla'
      'ALRU' => 'ont1:Alnus_rubra'
      ...
   measurement:
      characteristic: 'obs:Name'
      value: '$TAG'
   measurement:
      characteristic: 'ont1:DiameterAtBreastHeight'
      standard: 'ont1:Meter'
      value: '$DBH'
   measurement:
      characteristic: 'ont1:Vigor'
      standard: 'ont1:TreeGrowthVigorStandard'
      value: match '$VIGOR' with
         '1' => individual: 'ont1:good_tree_growth_vigor'
         '2' => individual: 'ont1:fair_tree_growth_vigor'
         '3' => individual: 'ont1:poor_tree_growth_vigor'
   context: 'PlotObs', 'YearObs'
   entityKey: '$TAG'
```

In this example, the entity is defined by two separate attributes: the CANCLASS attribute specifies the canopy class the tree belongs to; and the SPP attribute specifies the tree species. In both cases, attribute values are mapped to ontology classes (i.e., a value such as "PSME" denotes a specific species type). Each tree observation also has three associated measurements: the (tag) name of the tree, the diameter at breast height (measured in meters); and the growth vigor. For the tree's vigor, each value in the dataset is mapped to a specific OWL-DL individual value (as opposed to a class) that is defined in the corresponding vigor standard of the ontology example of Section 2. Finally, each tree observation is made within the context of its corresponding plot and the year in which the observation was made.

Data sets and annotations are loaded independently in ObsDB. The `import table` command is used to register a CSV file denoting a data set with ObsDB. For example, the following command can be used to load the example table of Figure 4.

```
> import table 'table1.csv' as 'table1'
File copied to /data/tables/ directory.
File loaded.
```

Annotations are loaded into ObsDB using the `import annotation` command:

```
> import annotation 'annot1.txt' as 'annot1'
File copied to /data/annotations/ directory.
File loaded.
```

Once loaded, annotations can be applied to tables to generate an RDF graph of corresponding observation and measurement instances using the ObsDB `apply` command:

```
> apply 'annot1' to 'table1' as 'coll1' using 'http://obsdb.org/coll1'
***Generating Data From Files***
Annotation: data/annotations/annot1.oal
Table: data/tables/table1.csv
Output Triples: data/graphs/coll1.ttl
```

Here "coll1" is used to name the resulting named RDF graph, which is given the corresponding URI. ObsDB performs the following steps when applying an annotation to a table: (1) it verifies the annotation and data file are both syntactically correct; (2) it verifies the imported ontologies exist and that they have been loaded into ObsDB; (3) it generates the corresponding observations and measurements (i.e., by *materializing* the semantic annotation templates); and (4) it uses the HermiT OWL-DL reasoner to add inferred axioms (based on ontology definitions and constraints) to the RDF Graph, and ensures the resulting graph is consistent. Adding inferred axioms is performed to support query expansion within ObsDB. The resulting RDF Graph is stored within ObsDB and can then be further accessed and queried. [8] describes an earlier version of the materialization algorithm used by ObsDB. The approach used in the current version of ObsDB extends this work by supporting more complex value matching annotation primitives (as shown above, e.g., with the canopy class and vigor measurements) as well as by materializing data sets to named RDF Graphs as opposed to an underlying relational database representation.

## 4 Data Discovery and Analysis

Once data sets are semantically annotated and converted into their corresponding RDF graphs, they can be accessed directly from within ObsDB. There are three main ways to access data sets: (1) using the `find` command to issue data discovery queries to locate observation collections (RDF graphs of observations and measurements); (2) using the `query` command to select observations within or across data sets, the results of which can be viewed or used to create new observation collections that subset existing collections or combine multiple existing collections; or (3) using the `exec` command to apply statistical and analytical functions to query results (using built-in aggregation operators or by calling external R scripts).

Each of the above ways to access observation collections are expressed in ObsDB using the high-level query language ObsQL [6]. ObsQL queries are similar in spirit to XPath queries for XML in that ObsQL is designed to provide a simple syntax for expressing common data discovery and subsetting operations. For example, the following ObsQL find expression can be used to locate all observation collections that contain observations of trees:

```
> find ont1:Tree []
Matching Graphs
---------------
coll1
```

This example returns the set of matching observation collections (i.e., RDF graphs), which in this case consists of the "coll1" example of Section 3. All ObsQL find and query expressions are rewritten by ObsDB into corresponding SPARQL queries. For instance, the above ObsQL expression is converted by ObsDB into the SPARQL ASK query:

```
PREFIX obs: <https://code.ecoinformatics.org/.../oboe-core.owl#>
PREFIX char: <https://code.ecoinformatics.org/.../oboe-characteristics.owl#>
PREFIX ont1: <http://obsdb.org/ont1#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
...
ASK {?temporaryObservationVariable0 obs:ofEntity ?temporaryVariable0 .
?temporaryVariable0 rdf:type ont1:Tree .}
```

As another example, we can refine the search above to look for only those collections that have a specific type of tree with a specific type of measurement:

```
> find ont1:DouglasFir [ont1:DiameterAtBreastHeight]
Matching Graphs
---------------
coll1
```

ObsQL also supports context constraints using the -> operator. For example, the following expressions locates collections that contain observations of Douglas Fir trees made within named plots:

```
> find ont1:DouglasFir [ont1:DiameterAtBreastHeight] -> ont1:Plot [obs:Name]
Matching Graphs
---------------
coll1
```

While the above expression is relatively straightforward to express in ObsQL, the corresponding SPARQL query is considerably more verbose:

```
...
ASK {?tempObsVar0 obs:ofEntity ?tempVar0 .
     ?tempVar0 rdf:type ont1:DouglasFir .
     ?tempObsVar0 obs:hasMeasurement ?tempMeasVar0 .
     ?tempMeasVar0 obs:ofCharacteristic ?tempPlaceholder0 .
     ?tempPlaceholder0 rdf:type ont1:DiameterAtBreastHeight .
     ?tempMeasVar0 obs:hasValue ?tempPlaceholder1 .
     ?tempPlaceholder1 obs:hasCode ?tempVar1Code .
     ?tempObsVar1 obs:ofEntity ?tempVar2 .
     ?tempVar2 rdf:type ont1:Plot .
     ?tempObsVar1 obs:hasMeasurement ?tempMeasVar1 .
     ?tempMeasVar1 obs:ofCharacteristic ?tempPlaceholder2 .
     ?tempPlaceholder2 rdf:type obs:Name .
     ?tempMeasVar1 obs:hasValue ?tempPlaceholder3 .
     ?tempPlaceholder3 obs:hasCode ?tempVar3Code .
     ?tempObsVar0 obs:hasContext ?tempObsVar1 .}
```

We note that while ObsQL expressions are generally more concise and easier to specify than their corresponding SPARQL queries (in part, because ObsQL is tailored specifically to supporting queries over OBOE models), only a subset of SPARQL can be expressed in ObsQL. In addition to the above example, it is also possible to specify multiple contexts for an observation, for example:

```
> find ont1:IntermediateTree [] -> (ont1:Stand [], ont1:TimePeriod [])
Matching Graphs
---------------
coll1
```

finds all collections with an observation of the given tree type within the context of both a stand and a time period. Note that here the stand is an indirect context for the corresponding tree since the tree has a plot as context, and the plot has the stand as context. It is also possible to query for observations and measurements within a collection. For example, the following query returns the diameter values of all codominant trees within coll1:

```
> query ont1:CodominantTree [ont1:DiameterAtBreastHeight $d] in coll1
--------------------------------
| temporaryVariable0 | d       |
================================
| :ID1004            | "10.3"  |
| :ID1017            | "13.1"  |
| :ID10286           | "48.9"  |
| :ID10299           | "46"    |
...
```

Here, $\$d$ is a "place holder" variable for specifying output values. Note that although not shown here, multiple place holder variables can be given per query. Also, removing the "in" clause above will result in ObsDB querying all collections for matching observations.

Basic computations can also be performed on data when using ObsQL. For instance, when querying it is possible to select a specific unit from which ObsDB will apply appropriate unit conversions. For example, in this query:

```
> query ont1:CodominantTree [ont1:DiameterAtBreastHeight $d ont1:Foot] in coll1
-----------------------------------------------
| temporaryVariable0 | d                      |
===============================================
| :ID1004            | "33.79265091863517"    |
| :ID1017            | "42.979002624671914"   |
| :ID10286           | "160.43307086614172"   |
| :ID10299           | "150.91863517060366"   |
...
```

ObsDB uses the foot-to-meter unit conversion of Section 2 to convert the diameters in coll1 from meters to feet (since conversions are invertible). As another example, ObsDB can also perform statistical summaries of query results. For instance, the following query computes for each plot the average Douglas Fir tree diameter (in meters):

```
> exec avg $d by $p in coll1 where
     ont1:DouglasFir [ont1:DiameterAtBreastHeight $d ont1:Meter]
     -> ont1:Plot $p
-----------------------
| ?p       | mean     |
=======================
| :ID908   | 13.4     |
| :ID10203 | 59.51429 |
| :ID7     | 24.6     |
| :ID10837 | 155.4143 |
...
```

ObsDB supports the standard aggregate operations supported by SPARQL including average, mean, count, max, min, median, range, and standard deviation. Finally, custom R scripts can be used from within ObsDB. Each R script must contain a comment header denoting the name of the operation (for use within an `exec` command) and the variables that will be passed into the script (the script inputs). As a simple example, the following commented R script can be used to draw a basic histogram from within ObsDB.

```
#name: hist
#argument: $x A vector of x-axis variables
x=$x
hist(x)
```

Once defined, this script can be called from within ObsDB as follows.

```
> exec hist $d in coll1 where
    ont1:CodominantTree [ont1:DiameterAtBreastHeight $d > 75 ont1:Meter]
```

The result of this command generates the histogram shown in Figure 5, showing the distribution of codominant tree diameters greater than 75 $m$ in the underlying data set. This example also demonstrates an ObsQL query that performs a logical comparison on measurement values.

In general, ObsQL is designed to provide scientists with the ability to search for relevant data sets based on domain-specific ontology classes as well as perform basic exploratory analyses through the subselection of relevant observations and measurements of data sets, by applying aggregate operations, and by applying simple R analysis scripts. Although not shown in the examples above, ObsDB also allows the results of queries to be stored in new observation collections using the `as` keyword. This provides a basic form of data integration, in which observations from multiple data sets can be combined into a single collection, without having to perform similar operations directly on structurally heterogeneous tabular data sets.

## 5 Related Work and Future Directions

This paper described extensions to our prior work on semantic annotation and providing access to observational data through the OBOE model [6]. An early version of ObsDB was presented in [5], which did not directly support ontology editing and semantic annotations. Instead, the approach assumed that data was already "materialized" into RDF triples, which could then be loaded into the system. Similarly, ontologies were assumed to be defined outside of the system and accessible through resolvable URIs. The early version of ObsDB also employed a relational database system for storing and querying observational data, which lead to a number of performance issues as well as limiting its interoperability with other semantic web technologies. In [8] we described an approach for supporting semantic annotation templates. This work also relied on observations and measurements being stored using a relational database system. [8] also gives a formal specification of annotations along with alternative implementation strategies (in the spirit of classical data integration view-based approaches, e.g., [12,11]). Taken together, we extend our prior work by providing: an ontology editing "markdown" language designed specifically for OBOE; additional annotation constructs (for value mappings);

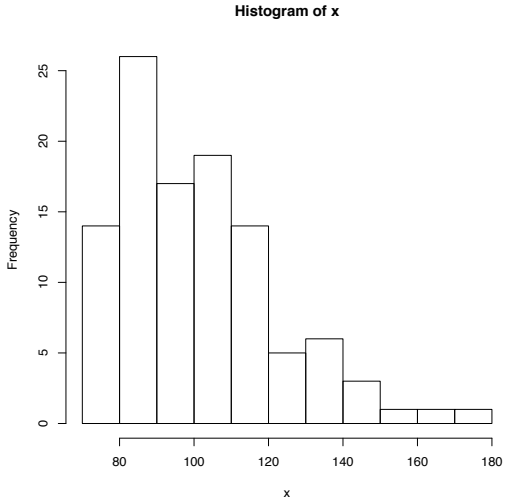


**Fig. 5.** Example histogram produced via an R script run in ObsDB.

a new ObsDB implementation over a popular RDF triple store system (Jena TDB); automated unit conversion; and extended query capabilities (e.g., value comparisons, grouping operations, and a wide range of aggregate operations).

A number of systems have recently been developed that leverage observation-based conceptual and ontology models. Some examples include [13] which describes and compares four implementations of the OGC's sensor observations service [2] for accessing and querying real-time sensor data (and which relies on O&M for representing observations), [10] which describes a system for managing resources based on hydrological observations (focusing on supporting large-scale observatory networks), and [16] which describes an implementation of a water quality portal that integrates a number of data sets via observational ontologies. ObsDB largely differs from these approaches by providing a personal data management system (as opposed to targeted applications over an observational model), with support for ontology-based data annotation, ontology editing, and various forms of exploratory query support. Annotations have been studied in various forms in the database literature (e.g., [9,4]), and as mentioned previously, our approach is similar to the more general use of views in data integration. Finally, the need for uniform mechanisms to describe observational data has led to many proposals for observational data models and ontologies (e.g., [14,3,7]). ObsDB is largely complementary to these efforts by providing a framework for managing observational data according to a generic observational model (based on OBOE) that supports the use of domain-specific ontologies, and a high-level query language for discovering and accessing observations (within and across datasets).

Our ongoing and future work on ObsDB is focused on further extending support for exploratory analysis of observational data via the R system. We are also interested in developing tools within ObsDB to support comparing data sets based on their annotations. For instance, given two observation collections, we would like to determine

how closely they "match" in terms of observation and measurement types and to automatically create mappings and transformations to unify the collections into a single integrated annotation template (for further analysis). We are also interested in developing support in R to access observations stored in ObsDB, e.g., to be able to programmatically load one or more observation collections through R calls to perform more sophisticated analyses (within an R script).

## Acknowledgements

This work supported in part through NSF grants IIS-1118088, DBI-0743429, and DBI-0753144.

## References

1. OGC: Observations and measurements encoding standard (O&M): `http://www.opengeospatial.org/standards/om`
2. OGC: Sensor observation service (SOS): `http://www.opengeospatial.org/standards/sos`
3. Semantic Web for Earth and Environmental Terminology (SWEET), `http://sweet.jpl.nasa.gov/sweet/`
4. An, Y., Mylopoulos, J., Borgida, A.: Building semantic mappings from databases to ontologies. In: AAAI (2006)
5. Bowers, S., Kudo, J., Cao, H., Schildhauer, M.P.: Obsdb: A system for uniformly storing and querying heterogeneous observational data. In: eScience. pp. 261–268 (2010)
6. Bowers, S., Madin, J.S., Schildhauer, M.P.: A conceptual modeling framework for expressing observational data semantics. In: ER. pp. 41–54 (2008)
7. C. Mungall, *et al.*: Integrating phenotype ontologies across multiple species. Genome Biology 11(R2) (2010)
8. Cao, H., Bowers, S., Schildhauer, M.P.: Approaches for semantically annotating and discovering scientific observational data. In: International Confere on Database and Expert Systems Applications (DEXA). pp. 526–541 (2011)
9. Geerts, F., Kementsietsidis, A., Milano, D.: Mondrian: Annotating and querying databases through colors and blocks. In: ICDE. p. 82 (2006)
10. Horsburgh, J.S., Tarboton, D.G., Maidment, D.R., Zaslavsky, I.: Components of an environmental observatory information system. Computers & Geosciences 37(2), 207–218 (2011)
11. Kolaitis, P.G.: Schema mappings, data exchange, and metadata management. In: PODS (2005)
12. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS. pp. 233–246 (2002)
13. McFerren, G., Hohls, D., Fleming, G.: Evaluating sensor observation service implementations. In: IEEE International Geoscience & Remote Sensing Symposium (IGARSS). pp. 363–366 (2009)
14. P. Fox, *et al.*: Ontology-supported scientific data frameworks: The virtual solar-terrestrial observatory experience. Computers & Geosciences 35(4), 724–738 (2009)
15. Rector, A.L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns. In: EKAW. pp. 63–81 (2004)
16. Wang, P., Fu, L., Patton, E.W., McGuinness, D.L., Dein, F.J., Bristol, R.S.: Towards semantically-enabled exploration and analysis of environmental ecosystems. In: eScience. pp. 1–8 (2012)