Information Sciences 329 (2016) 900–920

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Generic ontology of datatypes

Panče Panov [a,*], Larisa N. Soldatova [d], Sašo Džeroski [a,b,c]

[a] Department of Knowledge Technologies, Jožef Stefan Institute, Jamova cesta 39, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Jamova cesta 39, Ljubljana, Slovenia
[c] Centre of Excellence for Integrated Approaches in Chemistry and Biology of Proteins, Jamova cesta 39, Ljubljana, Slovenia
[d] Department of Computer Science, Brunel University London, Kingston Lane, Uxbridge, UB8 3PH, United Kingdom

ARTICLE INFO

Article history:
Received 15 April 2014
Revised 1 July 2015
Accepted 9 August 2015
Available online 13 August 2015

Keywords:
Data type
Data mining
Ontology
Knowledge representation

ABSTRACT

We present OntoDT, a generic ontology for the representation of scientific knowledge about datatypes. OntoDT defines basic entities, such as datatype, properties of datatypes, specifications, characterizing operations, and a datatype taxonomy. We demonstrate the utility of OntoDT on several use cases. OntoDT was used within an Ontology of core data mining entities for constructing taxonomies of datasets, data mining tasks, generalizations and data mining algorithms. Furthermore, we show how OntoDT can be used to annotate and query dataset repositories. We also show how OntoDT can improve the representation of datatypes in the BioXSD exchange format for basic bio-informatics types of data. The generic nature of OntoDT enables it to support a wide range of other applications, especially in combination with other domain specific ontologies: the construction of data mining workflows, annotation of software and algorithms, semantic annotation of scientific articles, etc. OntoDT is open source and is available at http://www.ontodt.com.

1. Introduction

Data processing is at the heart of science. Scientific research workflows rely heavily on datatype representations. Especially in data mining research it is impossible to efficiently (semi-) automatically connect parts of workflows, such as data preprocessing and data mining, perform analysis of the research results and communicate the research outputs, without machine processable representation of datatypes and their properties. There is a need for a standardized semantically-defined and machine amenable representation of scientific datatypes to support cross-domain applications. Unfortunately, the existing representations of datatypes do not fully address such a need.

In the literature, there exist different definitions of datatypes. In computer science, a datatype is usually defined as a "classification that identifies various types of data, such as boolean, integer, discrete and others, that determines the possible values for that type, operations on the values of the data, and the way the values of that type can be stored" [56]. Nell and Walker [8] discuss the difference between a data structure and datatype in the sense that "data structure refers to the study of data and how to represent data objects within a program; that is, the implementation of structured relationships" while a datatype defines "the properties of classes of objects in addition to how these objects might be represented in a program". Martin [36] also

* Corresponding author. Tel.: +386 1477 3307.
E-mail addresses: pance.panov@ijs.si (P. Panov), larisa.soldatova@brunel.ac.uk (L.N. Soldatova), saso.dzeroski@ijs.si (S. Džeroski).

http://dx.doi.org/10.1016/j.ins.2015.08.006

discusses the difference between data structures and datatypes and states that “depending on the point of view, a data object is characterized by its type (for the user) or by its structure (for the implementer)”.

In this paper, we present OntoDT, a generic ontology of datatypes. OntoDT defines the semantics, i.e., meaning of the key entities and represents the knowledge about datatypes in a machine friendly way. The OntoDT ontology is based on the latest revised version of the ISO/IEC 11404 standard for datatypes [23].

This paper is organized as follows. In Section 2, we present the background related to the development of the OntoDT ontology. In Section 3, we review and discuss the related work. Next, in Section 4, we present the ontology design principles and implementation, and in Section 5 we present the key OntoDT classes. In Section 6, we present the OntoDT datatype taxonomy. Finally, we present the ontology evaluation (Section 7), and three use cases of the ontology (Section 8). We conclude the paper with a discussion (Section 9) and a summary of contributions and points for further work (Section 10).

2. Background

The OntoDT development started within the frame of an ontology for data mining (OntoDM) [44]. The main idea of using a formalized description of datatypes for the domain of data mining was to characterize the types of data contained in a dataset, the applicability of a data mining task on data from a given datatype, and the applicability of a data mining algorithm on a dataset. Due to generality and reuse purposes, OntoDT has evolved to become an independent ontology.

The OntoDT ontology aims to address the need for a machine-friendly standard representation of general-purpose datatypes. It is based on the International Standard ISO/IEC 11404 for representing datatypes in computer systems [23]. The standard specifies the terminology and the semantics for a collection of data types commonly occurring in programming languages and software interfaces. The datatypes defined in the standard are general in nature and serve a wide variety of information processing applications. The standard specifies both primitive datatypes, being defined without a reference to other datatypes, and non-primitive datatypes, which are completely or partially defined in terms of other datatypes.

The ISO/IEC 11404 standard includes a list of 62 definitions of datatype related terms. It also specifies the conditions that have to be fulfilled by an information processing entity in order to conform to the standard directly or indirectly. The standard describes fundamental notions such as a definition of a datatype, a value space, datatype properties, a datatype generator, characterizing operations, etc. We extracted the key terms from the standard, organized these terms into a logically consistent IS-A hierarchy of ontological classes, defined their properties and relations to other entities, re-used suitable textual definitions from the standard, where possible, and added new ontological definitions, where necessary.

3. Related work

The problem of data typing is an important problem that has been addressed from different aspects and in different forms. For example, the research data alliance (RDA) [50], whose major goal is to speed up the international data-driven innovation and discovery by facilitating research data sharing and exchange, has identified that the problem of data typing is an important problem that deserves attention. For this purpose, the RDA formed a data type registy (DTR) working group [11] with the goal to: compile a set of use cases for datatype use and management, formulate a data model and expression for datatypes (prototype registry available at [10]), design a functional specification for type registries, and propose a federation strategy among multiple type registries.

Meek [37] discussed a proposal for a taxonomy of datatypes using as a base the first version of the ISO 11404 standard [22]. The taxonomy starts with a number of primitive datatypes that are then used to construct others. The proposed taxonomy is given only in the form of an overview and a discussion, without any formal representation.

The W3C XML Schema Definition Language (XSD) [67] is widely used for the recording of data on the semantic web, and it is also based on the ISO 11404 standard [23]. XSD supports simple, complex, and custom-defined datatypes. It is a simple and flexible language, but it is not based on a formal model and consequently many aspects are left to interpretation. XSD terms are not formally defined. For example, a definition of the term *attribute* (‘Defines an attribute’) is circular and does not explain how an attribute is different from e.g. *an element* [68].

XSD is flexible and does not strictly regulate the custom-defined datatypes; it also does not enforce the separation of data and its semantic meaning. A side effect of those features is an unnecessary proliferation of custom-defined datatypes. The issue is that different users may create different data models for the same data, and it may be hard to reconcile those models. For example, users can define datatypes such as *start of the project, beginning of the project, start date*. All these datatypes are of the same type *date datatype* and the data encoded with these custom datatypes have the same semantic meaning. A formal ontology can serve as a reference model and resolve such an issue.

The RDF data cube vocabulary is focusing on the publication of multidimensional data on the web [51]. It enables the exchange and sharing of statistical data encoded in a tabular form. The adopted cube model has a set of properties for the description of statistical datasets composed of observations. These include *dimensions* (e.g. time, age, sex), *attributes* (e.g. unit measure), and *measures* (values of observations). A dataset can have *reference metadata* (e.g. a SPARQL endpoint where it can be accessed, its publisher). This purpose-specific vocabulary is well defined and sufficient for recoding of statistical information. However, its strict statistic-oriented model prevents the extension of this vocabulary for other non-tabular datatypes.

An attractive feature of this vocabulary is a distinction between the semantic meaning of observations, the measurement units used, and the data structure specification. The *dimension* property links observations to other resources, i.e. Simple Knowledge

Organization System (SKOS) [57], which defines the semantics of the data. The *attribute* property is used to record information about the units. OntoDT adopts a similar approach for the defining semantic meaning of the data and also for modeling units. Unfortunately, the RDF Data Cube Vocabulary does not have a clear separation between operational information (e.g. if a data item is an estimate or an accurate measurement) and the specification of datatypes. A better approach would be to capture the information about data pre-processing, processing and post-processing separately and then link it to the data items (observations) [45].

Several ontologies that include data-related aspects have been developed in various domains, but they are typically too domain specific and do not have representations of arbitrary complex datatypes. Below we briefly discuss some of such ontologies.

The EMBRACE Data and Methods (EDAM) is an ontology of bioinformatics operations, types of data, formats, and topics [13,24,46]. The data branch of EDAM has the following key classes: *core data, identifier, parameter, report, search and retrieval*. These terms do not correspond to conventional datatypes and are rather labels for the capturing the semantic meaning of the data. There is also no clear distinction between data and knowledge items. The *core data* class has subclasses that correspond to what is conventionally considered as knowledge, e.g., *biological model, ontology, workflow, schema*. It also includes subclasses such as *data index, experimental measurement, structure* that are more relevant to datatypes, and are intended to be used for data annotation. One cannot apply data mining algorithms to the data of the type *experimental measurement* without specifying the datatype (e.g., numerical datatype).

Linked Models is a web resource for publishing RDF/OWL models of commonly used industry and government standards [33]. The work is motivated by the desire to use semantic web technologies for interoperability, information aggregation and validation of specifications created with UML and/or XML schema tools. Linked Models include the Dtype ontology that specifies datatypes required for dealing with OWL representations of data structures based on the XML schema.

Several ontologies have been produced for sensor observation services [53]. For example, the Semantic Sensor Network Ontology (SSN) describes sensors in terms of capabilities, measurement processes, observations and deployments [6,55]. SSN focuses mainly on tabular data, i.e. *Sensor Data Sheet*, and considers *Observation* as a social construct (a subclass of the class *Situation*). An ontology for ecological observational data (OBOE) defines the notion of *scientific observation* as a unifying concept for capturing the basic semantics of ecological data [34]. Observations are distinguished at the level of the entity (e.g., location, time), and the characteristics of an entity (e.g., height, name, color) are classified as data.

Many biomedical ontologies available at BioPortal include a class named *datatype* [4]. These include: the NanoParticle Ontology (NPO) [65], the Health Level Seven Reference Implementation Model [17], the Syndromic Surveillance Ontology [64], the Microarray and gene expression data ontology (MO) [38], the Phylogenetic ontology [47], and the National Cancer Institute Thesaurus [39]. Unfortunately, the representations and semantic meanings of the term *datatype* across these resources are not consistent. For example, MO defines *DataType* as “Primitive data types found in computing languages such as float, boolean, etc.” Image and Data Quality Assessment Ontology (IDQA) [21] defines *Data Type* as “Superclass for different type of data: data itself and images (TS)”. Such representations are very domain-specific and not always accurate.

Finally, ontologies designed to support data mining studies, e.g., the Data Mining OPtimization ontology (DMOP) [9,19,28] and Data Mining Workflow ontology (DMWF) [29], include representation of datatypes only on a basic level. Thus there is a limited number of formal representations of data types and these representations are not sufficiently generic to ensure cross-domain interoperability required by data mining research, and particularly mining of complex biomedical data. We have developed OntoDT to address the need for a consistent representation of datatypes across various domains.

4. Design and implementation

The design of the OntoDT ontology follows best practices in ontology engineering, such as the OBO Foundry principles, which are widely accepted in the biomedical domain [60]. These include ontology completeness, the absence of multiple inheritance, the absence of orphan classes, extensibility, the use of formally defined relations, the use of upper-level ontology, orthogonality with other ontologies, version management, a unique identifier space, and others. OntoDT is developed to be complementary to and integrated with state-of-the-art ontologies for representing scientific knowledge. This ensures interoperability with other resources and facilitates cross-domain reasoning.

We used the information artifact ontology (IAO) as the upper level ontology [20]. IAO has been designed to support the representation of information entities, and it is compliant with the basic formal ontology (BFO) [3] and the OBO relational ontology (RO) [52,61]. There are several other upper level ontologies, i.e. SUMO [40,62], DOLCE [12], but they are focused on modeling primarily real world entities. In contrast, OntoDT aims to represent information content entities, such as datatypes. Such entities have different properties compared to the real world objects. They do not occupy a physical space and they do not have physical dimensions. Therefore, we adopted the IAO framework for the representation of information content entities that stay in a relation of aboutness with the corresponding real world entities. This framework enables a formally defined representation of data (as information content entities) and its semantic meaning (what entity this data is about). In this way, we achieve a firm ontological foundation for the proposed representation of datatypes, operations on datatypes and datatype properties.

The use of an upper level ontology and a set of widely accepted formally defined relations eases the interoperability of OntoDT with other external ontologies, e.g., the Software Ontology [63], where possible. OntoDT re-uses existing ontological resources, such as Open Biomedical Ontologies [42]. For example, it reuses the OBI ontology [41], i.e., the class *OBI:0000658: data representational model.* Classes that are reused in OntoDT are imported following the Minimum Information to Reference an External

Table 1
OntoDT competency questions.

Q_n	Query
1	What is the set of characterizing operations for a datatype X?
2	What is the set of datatype qualities for a datatype X?
3	What is the value space for a datatype X?
4	What is the set of datatypes that have a datatype quality X?
5	What is the set of datatypes that have a characterizing operation X?
6	What is the set of datatypes that have a datatype quality X and characterizing operation Y?
7	What are the aggregated datatypes that have an aggregate generator property X?
8	What is the set of aggregate properties for an aggregate datatype X?
9	What are the field components for a tuple datatype X?
10	What is the base datatype for a set/bag/sequence datatype X?
11	What is the base datatype for an extended datatype X?
12	What is the subtype generator for an extended datatype X?
13	What is the set of extended datatypes that have datatype X as their base datatype?
14	What is the set of extended datatypes that are generated by a subtype generator X?

Ontology Term (MIREOT) principle [7]. The ontology is developed in the OWL ontology language using the Protégé tool [48]. OntoDT is open source and is available at http://www.ontodt.com.

For the design and evaluation of OntoDT, we followed a methodology proposed by Grüniger and Fox [16]. Their methodology proposes first to define the ontology's requirements in a form of informal questions (or queries). Next, the terminology of the ontology (its classes and relations) is specified using some first order logical language (e.g., description logics). The language must provide the necessary terminology to formally restate the informal competency questions. This allows us to formulate the competency questions as an entailment queries with respect to the axioms in the ontology. In this way, one can evaluate the ontology and claim that it is adequate.

The design of OntoDT has been governed by a set of competency questions. Examples of such questions is as follows: "What is the set of characterizing operations for a datatype X?" and "What is the set of datatypes that have a datatype quality X and characterizing operation Y?" (see Table 1 for a full list). In order to support the formulated competency questions, OntoDT includes information on datatypes, datatype properties, characterizing operations, datatype generators, properties of generators and other support specification entities.

5. The key OntoDT classes

In this section, we describe the key entities for the representation of datatypes. In addition, we discuss the most important representational issues identified in the process of modeling.

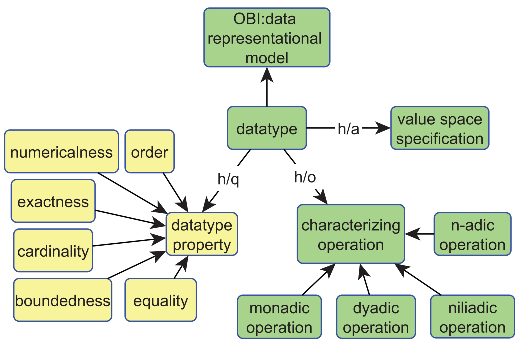
5.1. Datatype and value space

In the OntoDT ontology, the *datatype* class is modeled as a subclass of the *OBI: data representational model* class. It defines the type of data, with the set of distinct values that the data can take, the properties of those values, and the operations on those values. The *datatype* class is represented with the HAS-MEMBER relation to the *value space specification* class and the HAS-OPERATION relation to the *characterizing operation* class. In addition, OntoDT models *datatype properties* as subclasses of the *quality* class and connects them using the HAS-QUALITY relation. In Fig. 1a, we present the structure of the datatype class and in Fig. 1b the OWL Manchester syntax of the class definition.

The *value space specification* class is modeled in OntoDT as a subclass of the *OntoDM: specification entity* class. It specifies the collection of values for a given datatype. The value space of a given datatype can be defined in different ways: by enumerating the values; with axioms using a set of fundamental notions; as a subset of values defined in another value space with a given set of properties; or as a combination of arbitrary values from some other defined value space by specifying a construction procedure [23].

5.2. Characterizing operations

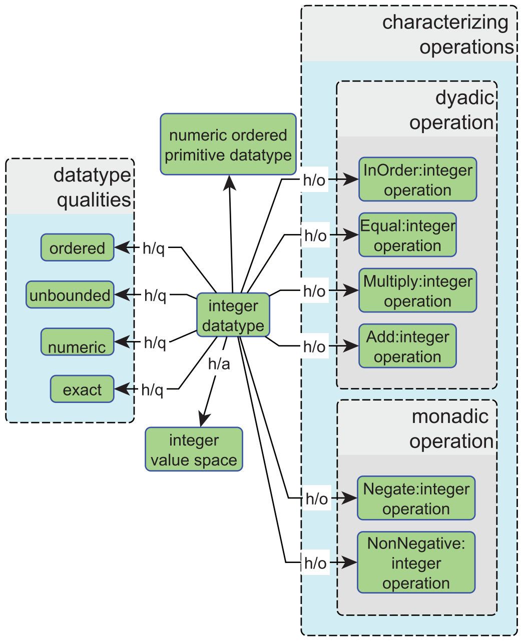
A *characterizing operation* is defined as *IAO: directive information entity* that specifies those operations on the datatype that distinguish it from other datatypes having identical value spaces. The characterizing operation of a datatype can be: *niliadic, monadic, dyadic* and *n-adic* (see Fig. 1a). A *niliadic operation* specifies an operation that yields values of a given datatype. A *monadic operation* specifies an operation that maps a value of a given datatype into a value of the given datatype, or into a value of the *boolean* datatype. A *dyadic operation* specifies an operation that maps a pair of values of a given datatype into a value of the given datatype, or into a value of the *boolean* datatype. An *n-adic operation specification* specifies an operation that maps an ordered n-tuple of values ($n > 2$), each of which is of a specific datatype, into values of a given datatype. Finally, all characterizing operation classes have defined subclasses, which represent datatype specific operations.



(a) The datatype class.

```
Class: datatype
    Annotations:
        definition "set of distinct values,
                    characterized by properties
                    of those values, and by
                    operations on those values",
        label "datatype",
        'definition source' "ISO/IEC 11404:2007(E)"
    SubClassOf:
        'data representational model',
        has_quality some 'datatype property',
        has_operation some 'characterizing
                             operation',
        has_attribute some 'value space'
```

(b) OWL Manchester syntax



```
Class: 'integer datatype'

    Annotations:
        definition "integer is the mathematical
                    datatype comprising the
                    exact integral values.",
        label "integer datatype"@en,
        'definition source' "ISO/IEC 11404:2007(E)"

    SubClassOf:
        'numeric ordered primitive datatype',
        has_quality some unbounded,
        has_quality some numeric,
        has_quality some exact,
        has_quality some ordered,
        has_operation some Equal:integer,
        has_operation some Add:integer,
        has_operation some Multiply:integer,
        has_operation some InOrder:interger,
        has_operation some Negate:integer,
        has_operation some NonNegative:integer,
        has_attribute some 'integer value space'

    DisjointWith:
        'real datatype',
        'rational datatype',
        'scaled datatype'
```

(c) Integer datatype class.

(d) OWL Manchester syntax

Fig. 1. Representation of datatypes in OntoDT. The rectangular boxes represent ontology classes. Unlabeled arrows represent IS-A relations, while labeled arrows have the following meaning: h/o represents HAS-OPERATION relation, h/q represents HAS-QUALITY, h/a represents HAS-ATTRIBUTE, and h/m represents HAS-MEMBER. Full lines denote existential relations.

5.3. Datatype properties

A *datatype property* is defined as a *quality* that specifies the intrinsic properties of the data units represented by the datatype, regardless of the properties of their representations in computer systems. Each datatype has a set of unique datatype properties. These include property classes such as: *order*, *numericalness*, *cardinality*, *exactness* ,*equality*, and *boundedness* (see Fig. 1a).

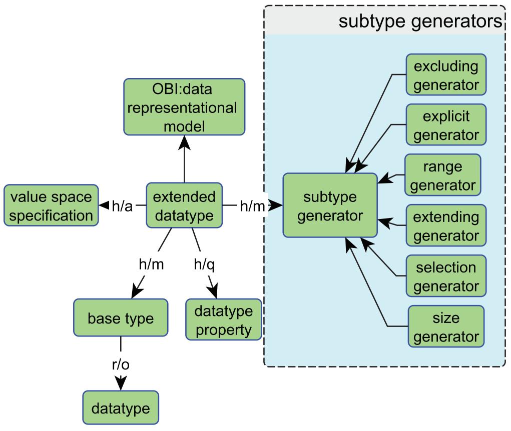
Order is a datatype property that denotes whether there exists an order relation defined on its value space. *Numericalness* denotes whether the values in the value space are quantities expressed in a mathematical numbering system. *Cardinality* denotes the notion of cardinality of the value space. *Exactness* denotes whether every value from the value space is distinguishable from every other value in the value space. Finally, *boundedness* is a property that denotes the boundaries of the value space.

All datatype property classes have defined subclasses. For example, the *boundedness* class has the following subclasses: *bounded* (*bounded below*, *bounded above*) and *unbounded* (*unbounded below*, *unbounded above*).[1]

5.4. Example of a datatype class: integer datatype

In Fig. 1c, we present the representation of the integer datatype in OntoDT and in Fig. 1d we present OWL Manchester syntax of the *integer datatype* class definition. The *integer datatype* is a subclass of the *numeric ordered primitive datatype* class and

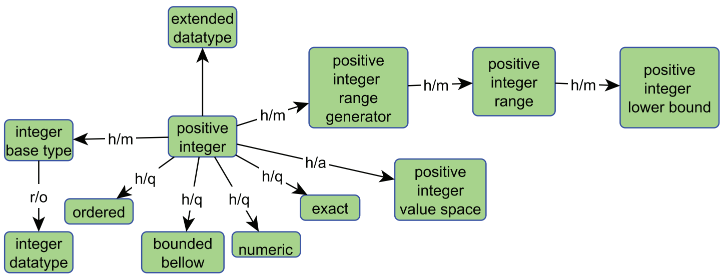
[1] See the OntoDT ontology for the definitions.



(a) The extended datatype class.

```
Class: 'extended datatype'

    Annotations:
        definition "datatype derived from another
                    datatype by restricting the
                    value space to a subset whilst
                    maintaining all characterising
                    operations.",
        comment "synonym - subtype",
        label "extended datatype",
        'definition source' "ISO/IEC 11404:2007(E)"

    SubClassOf:
        'data representational model',
        has_member some 'base type',
        has_member some 'subtype generator',
        has_quality some 'datatype property',
        has_attribute some 'value space'

Class: 'base type'

    Annotations:
        definition "base type denotes the role of a
                    datatype as a parametric datatype
                    on which a generator operates
                    to produce a new datatype."
        label "base type",

    SubClassOf:
        'datatype role',
        role_of some datatype,
        is_member_of some
            ('array datatype' or
             'sequence datatype' or
             'bag datatype' or
             'set datatype' or
             'extended datatype')
```

(b) OWL Manchester syntax.



(c) The positive integer extended datatype instance.

Fig. 2. Representation of extended datatype in OntoDT. The rectangular boxes represent ontology classes. Unlabeled arrows represent IS-A relations, while labeled arrows have the following meaning: r/o represents ROLE-OF relation, h/q represents HAS-QUALITY, h/a represents the HAS-ATTRIBUTE, and h/m represents HAS-MEMBER. Full lines denote existential relations.

represents a mathematical datatype, whose value space is composed of exact integral values. It is *ordered*, *unbounded*, and *exact*, and its values are *numeric*. Furthermore, the integer datatype is characterized by a set of monadic and dyadic operations. The monadic operations include the following operations: *Equal*, *InOrder*, *Add*, and *Multiply*; while the dyadic operations include the operations: *NonNegative* and *Negate*. Finally, we explicitly state that the integer datatype is disjoint with the other numeric ordered primitive datatypes (real datatype, rational datatype, and scaled datatype).

5.5. Extended datatype

In OntoDT, an *extended datatype* (named 'subtype' in the ISO standard) is defined as a *IAO: data representational model* that is derived from an existing datatype by restricting the value space to a subset of the base datatype, while maintaining all operations (see Fig. 2a and b). The base type denotes the role of a datatype as a parametric datatype on which a generator operates to produce a new datatype (see Fig. 2b). An extended datatype is defined by a *subtype generator* that represents the relationship between the value spaces of the base type and the extended datatype.

In OntoDT, we define the following classes of subtype generators: *range generator*, *selection generator*, *exclusion generator*, *size generator*, *extension generator*, and *explicit subtype generator*. Subtype generators can change the set of datatype properties valid for the base datatype, and this is the reason we do not represent them simply as subclasses of the datatype class. For example, applying the range generator to an unbound datatype will make it bounded.

Using these notions, we can represent an extended datatype of any previously defined type. For example, by using a range subtype generator we can place a new upper and/or lower bound on the value space of a chosen base datatype. The *positive integer* datatype is an extended datatype of the *integer datatype* obtained by limiting the value space with a lower bound of zero (see Fig. 2c).

5.6. Querying OntoDT

OntoDT can be queried using the Description Logic (DL) [1] query plug-in available in the Protégé software. For the purpose of querying, we classified the ontology using the HermiT 1.3.8 reasoner [18]. The DL queries have the form of class expressions. It is possible to run various queries that concern the datatypes, datatype components, properties and operations, e.g. "Find all subclasses of datatype that have as part aggregate generator"; "Find all characterizing operations of integer datatype"; "Find all datatype qualities of ordinal datatype"; "Find all datatypes that have non-numeric datatype quality and one dyadic operation". Below we analyze several queries in more detail.

Example 1. "Find all datatypes that have dyadic arithmetic operations."

```
has_operation some 'dyadic aritmetic operation'
```

The dyadic arithmetic operation class contains subclasses of arithmetic operations that require two operands. The query execution returns 5 datatypes that have such operations. These include: complex datatype, integer datatype, rational datatype, and scaled datatype.

Example 2. "Find all generated datatypes whose generators have a direct access property."

```
has_member some (has_quality some 'direct access property')
```

The direct access property is an access type property which determines how component values can be extracted from a given aggregate-values directly. It has two subclasses: index access and key access. The query execution returns 3 datatypes that have this property: array datatype, class datatype and record datatype.

Example 3. "Find all datatypes that have dyadic comparison operations and are bounded".

```
has_operation some 'dyadic comparison operation' and has_quality some bounded
```

The dyadic comparison operation class contains subclasses of comparison operations that require two operands. The bounded class is a datatype property that characterizes bounded datatypes and has two subclasses. The query execution returns 2 datatypes that have this property: enumerated datatype and ordinal datatype.

6. The OntoDT datatype taxonomy

In the OntoDT ontology, we define a taxonomy of datatypes (see Fig. 3). The top-level ontology classes include primitive datatypes, generated datatypes, and user defined datatypes. *Primitive datatypes* are defined by explicit specification and are independent of other datatypes. *Generated datatypes* are syntactically and semantically dependent on other datatypes, and are specified implicitly with *datatype generators*. *User defined datatypes* are defined by a datatype declaration and allow defining additional identifiers and refinements to both primitive and generated datatypes. At the lower levels, the datatypes are distinguished with respect to their datatype properties. In this section, we describe in more detail all three major classes of datatypes.

6.1. Primitive datatype

A *primitive datatype* IS-A datatype whose value space is defined either axiomatically or by enumeration [23]. All primitive datatypes are conceptually atomic and therefore are defined in terms of well defined abstract notions. According to the definition of a datatype, each primitive datatype class has a set of datatype properties, a set of characterizing operations, and a value space specification.

The ISO 11404 standard defines twelve primitive datatypes, some of which are defined as datatype families.[2] In the OntoDT ontology, we model all primitive datatypes from the standard as classes (see Fig. 3). The classes of primitive datatypes can be further instantiated by specifying additional parameters that are different for each class of primitive datatypes. For example, to define an instance of the *real datatype*, we additionally need to specify the *radix* and the *factor*, which taken together, describe the precision to which values of the datatype are distinguishable. Both *radix* and *factor* are represented as subclasses of the *value expression* class.

[2] The discrete (or state), enumerated, character, date-and-time, scaled, real, and complex datatypes are defined as datatype families, while the mathematical datatypes (boolean, ordinal, integer, and rational) and the void datatype are defined atomically.

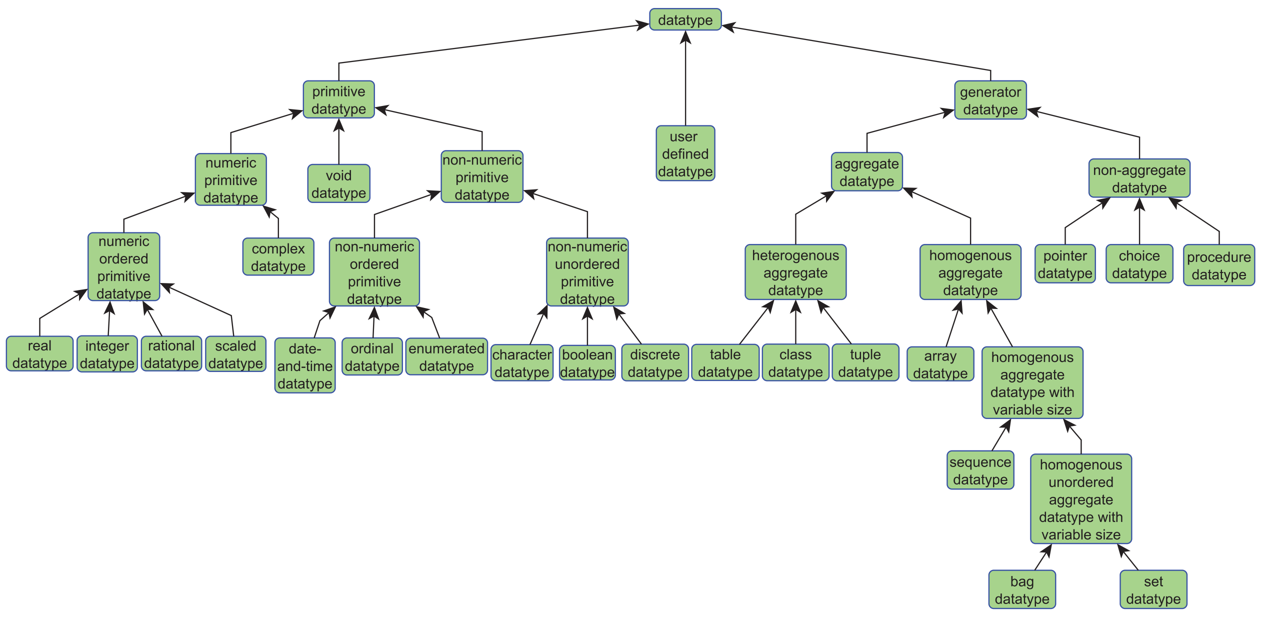


Fig. 3. The OntoDT datatype taxonomy. The rectangular boxes represent ontology classes. Unlabeled arrows represent IS-A relations.

Example. In Fig. 4, we present an example of representation of one subclass of primitive datatype – the *discrete datatype*. The discrete datatype has the following datatype properties: *unordered*, *non-numeric* and *exact* (see Fig. 4a). Additionally, this datatype has one characterizing operation (*equal*). The instances of the discrete datatype differ between each other in the *discrete-value-list specification*. It is *specification entity* that specifies the *discrete-value identifiers* for the datatype.

For example, if we want to describe, formally represent and reason about datasets, we would need to represent the datatypes describing the data examples contained in the datasets. If we take the well known Iris dataset[3] from the UCI repository [2], we can formally represent datatypes describing the Iris data examples by defining instances of OntoDT datatype classes.

Here, we show a representation of the datatype representing the Iris-class attribute, as an instance of the *discrete datatype* class (see Fig. 4a). It has as member a *discrete-value-list specification*, which includes three *discrete-value identifiers*: *'Iris Setosa'*, *'Iris Versicolour'*, and *'Iris Virginica'*. Finally, in Fig. 4b, we present the OWL Manchester syntax of the *Iris class datatype* instance.

6.2. Taxonomy of primitive datatypes

We propose a taxonomy of primitive datatypes,[4] with respect to the datatype properties (see Fig. 3). At the first level, with respect to the *numeric property*, we distinguish between *numeric primitive datatype* and *non-numeric primitive datatype*. In addition, we define the *void datatype* class as a primitive datatype representing an object whose presence is required, but carries no information.

On one hand, at the second level of the taxonomy with respect to the *order* property, we distinguish between *numeric ordered primitive datatype* and *complex datatype.*[5] *Numeric ordered primitive datatype* has four subclasses: *real datatype*, *scaled datatype*, *integer datatype*, and *rational datatype*.

On the other hand, we distinguish between *non-numeric ordered primitive datatype* and *non-numeric unordered primitive datatype*. *Non-numeric ordered primitive datatype* has three subclasses: *date-and-time datatype*, *enumerated datatype*, and *ordinal datatype*. *Non-numeric unordered primitive datatype* has three subclasses: *character datatype*, *discrete datatype*, and *boolean datatype*.

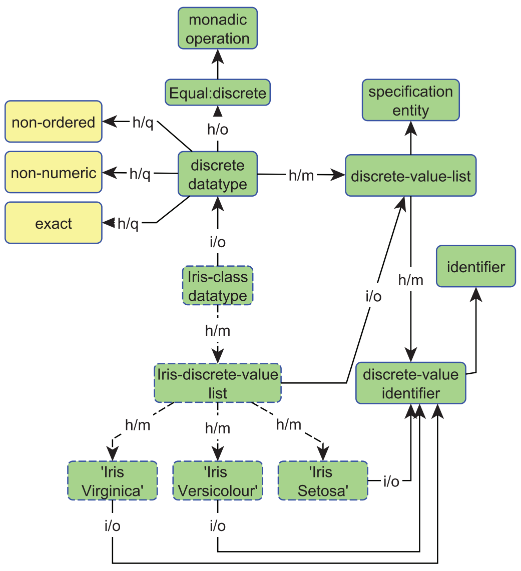
6.3. Generated datatype

A *generated datatype* is a *datatype* that is defined with a *datatype generator*. A *datatype generator* is an *IAO:directive information entity*, which specifies the conceptual operation on one or more datatypes which yields a datatype [23]. It specifies the criteria for the number and properties of datatypes to be operated upon. Next, it defines a construction procedure which creates a new value space from the value space of the element datatypes. Finally, it specifies the set of characterizing operations for the resulting datatype.

[3] Iris dataset: https://archive.ics.uci.edu/ml/datasets/Iris, accessed 7.12.2014.

[4] Definitions of all primitive datatype classes, presented in this section, are given in the ontology.

[5] A complex datatype represents complex numbers and is an unordered datatype.



(a) Discrete datatype class and instance.

```
Individual: 'Iris-class datatype'
    Types:
        'discrete datatype'
    Facts:
     has_quality  Exact,
     has_quality  Non-ordered,
     has_member  Iris-discrete-value-list,
     has_operation  Equal-discrete,
     has_quality  Non-numeric
Individual: Iris-discrete-value-list
    Types:
        discrete-value-list
    Facts:
     has_member  'Iris Versicolour',
     has_member  'Iris Virginica',
     has_member  'Iris Setosa'
Individual: 'Iris Versicolour'
    Types:
        'discrete-value identifier'
    Facts:
     has-value  "Iris Versicolour"^^Literal
Individual: 'Iris Virginica'
    Types:
        'discrete-value identifier'
    Facts:
     has-value  "Iris Virginica"^^Literal
Individual: 'Iris Setosa'
    Types:
        'discrete-value identifier'
    Facts:
     has-value  "Iris Setosa"^^Literal
```

(b) OWL Manchester syntax.

Fig. 4. Representation of the discrete datatype class and instance in OntoDT. The rectangular boxes represent ontology classes. The rectangular boxes with dashed lines represent instances. Unlabeled arrows represent IS-A relations, while labeled arrows have the following meaning: h/o represents HAS-OPERATION relation, h/q represents HAS-QUALITY, i/o represents INSTANCE-OF, and h/m represents HAS-MEMBER. Full lines denote existential relations. Dashed lines denote relations between instances.

Each of the datatypes from a collection of datatypes, to which the datatype generator is applied, is called a parametric datatype. An important characteristic of all datatype generators is that they can be applied to many different parametric datatypes. Parametric (or component) datatypes in OntoDT are modeled as roles of datatypes.

In general, we distinguish between two groups of generators: *aggregate generators* and *non-aggregate generators*. *Aggregate generators* generate datatypes whose values can be decomposed, while *non-aggregate generators* generate datatypes whose values are atomic. This leads to two main groups of generated datatypes: *aggregate datatypes* and *non-aggregate datatypes*.

6.3.1. Aggregate datatypes

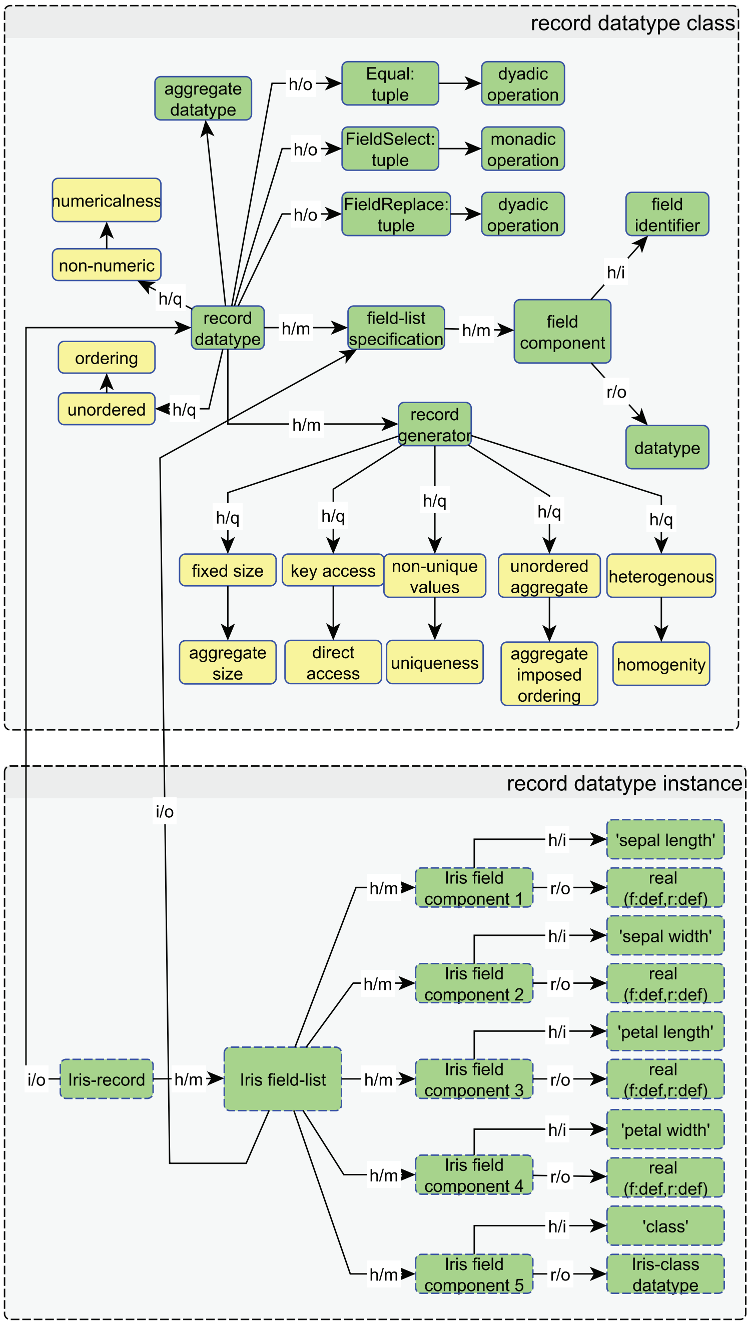
An *aggregate datatype* (or a structured datatype) is a *generated datatype*, each of whose values is made up of values of other datatypes (parametric or component datatypes) joined together by an *aggregate generator*. An *aggregate generator* is a *datatype generator* that specifies the algorithmic procedure applied to the value spaces of the component datatypes to yield the value space of the aggregate datatype, and a set of characterizing operations specific to the generator. The component values of an aggregate value are accessible through characterizing operations.

Subclasses of aggregate generator in OntoDT include: the *record generator* (or tuple generator), the *class generator*, the *set generator*, the *bag generator*, the *sequence generator*, the *array generator*, and the *table generator*. Every aggregate generator defines a separate aggregate datatype class[6]: *record datatype*, *class datatype*, *set datatype*, *bag datatype*, *sequence datatype*, *array datatype*, and *table datatype*.

Aggregate datatypes are distinguished by properties that describe the relationships among the component datatypes, the relation between each component and the aggregate, and the sets of characterizing operations. The aggregate specific properties are independent of the component datatype properties. They are defined as qualities of the *aggregate generator*. The aggregate specific properties include the following quality classes: *homogenity*, *aggregate size*, *uniqueness*, *aggregate-imposed identifier uniqueness*, *aggregate-imposed ordering*, *access type*, *recursiveness*, *structurness*, and *component mandatoriness*.

Example. In Fig. 5a, we present an example of the record datatype (also called a tuple datatype). The *record generator* specifies the procedure for generating the *record datatype* and has a set of *aggregate generator properties*. These include properties such

[6] See OntoDT for the definitions of all aggregate datatypes.



(a) Record datatype class and instance

```
Class: 'record datatype'
    Annotations:
        comment "synonim: tuple datatype",
        label "record (tuple) datatype",
    SubClassOf:
        'heterogenous aggregate datatype',
        has_member some 'record generator',
        has_member some field-list,
        has_operation some Equal:record,
        has_operation some FieldReplace:record,
        has_operation some FieldSelect:record,
        has_quality some exact,
        has_quality some non-ordered,
        has_quality some non-numeric
    DisjointWith:
        'class datatype', 'table datatype'
Class: 'record generator'
    Annotations:
        label "record generator"@en,
    SubClassOf:
        'heterogenous aggregate generator',
        has_quality some 'key access',
        has_quality some 'non-unique values',
        has_quality some 'one dimensional',
        has_quality some 'fixed size',
        has_quality some 'unordered aggregate'
    DisjointWith:
        'class generator', 'table datatype'
Class: field-list
    Annotations:
        label "field-list",
    SubClassOf:
        'list specification',
        has_member some 'field component'
Class: 'field component'
    Annotations:
        label "field component",
    SubClassOf:
        'datatype role',
        role_of some datatype,
        has_identifier some 'field identifier'
```

(b) OWL Manchester syntax.

Fig. 5. The record datatype in OntoDT. The rectangular boxes represent ontology classes. The rectangular boxes with dashed lines represent instances. Unlabeled arrows represent IS-A relations, while labeled arrows have the following meaning: h/o represents HAS-OPERATION relation, h/q represents HAS-QUALITY, i/o represents INSTANCE-OF, h/m represents HAS-MEMBER, h/i represents HAS-IDENTIFIER, and r/o represents ROLE-OF. Full lines denote existential relations.

as: *heterogenous*, *unordered aggregate*,*non-unique values*, *fixed size*, and *key access*. The values of the record datatype are heterogeneous aggregations of values of the component datatypes. Each aggregation has one value for each component datatype. The component datatypes are keyed by an identifier and are organized in a *field-list*. Each *field component* contains a unique identifier of the component and its datatype. Finally, in Fig. 5b, we present the OWL Manchester syntax of the *record datatype* class.

In Section 6.1, we presented an example of datatype representation of the class attribute of the Iris dataset, which is an instance of a primitive datatype. Here, we present an example of a record datatype instance, describing data examples from the Iris dataset (see Fig. 5a). The *Iris-record* instance inherits all the characterizing operations, and datatype qualities from the

Table 2
Statistical metrics for the OntoDT ontology.

Metrics	Value
Class count	417
Axioms count	3086
Disjoint classes axioms count	115
Equivalent classes axioms	27
SubClassOf axioms	761
Object properties count	14
Annotation axioms count	1606
DL expressivity	ALCHOIQ(D)

parent class. Additionally, the *Iris-tuple* datatype has a specification of the component datatypes. For example, the *Iris field-list* contains *Iris-field-component* instances for each component datatype. Each component specification includes an identifier (e.g., '*sepal length*') and denotes the datatype of the component (e.g., *real(f:def,r:def)*, where f:def and r:def represent the fraction and radix parameters needed to define an instance of a real datatype class). Finally, the class component is described by the Iris-class datatype instance discussed in Section 6.1.

6.3.2. Non-aggregate datatypes

A *non-aggregate datatype* is a *generated datatype* that is specified by a *non-aggregate generator*. Examples of non-aggregate datatypes include: the *choice datatype*, the *pointer datatype*, and the *procedure datatype*. A *choice datatype* is a *non-aggregate datatype*, each of whose values is a single value from any of a set of alternative datatypes. A *pointer datatype* is a *non-aggregate datatype*, each of whose values constitutes a means of reference to values of another datatype and are atomic. A *procedure datatype* is a *non-aggregate datatype*, each of whose values is an operation on values of other datatypes and is atomic.

6.4. Taxonomy of generated datatypes

We propose a taxonomy of generated datatypes (see Fig. 3), by using datatype properties and properties of aggregate generators.[7] At the first level, we distinguish between *non-aggregate datatypes* and *aggregate datatypes*. At the second level, if we focus only on the *aggregate datatypes*, with respect to the *homogenity property*, we distinguish between a *heterogenous aggregate datatype* and a *homogenous aggregate datatype*. *Heterogenous aggregate datatype* has three subclasses: a *tuple datatype*, a *class datatype*, and a *table datatype*.

At the third level, if we focus only on a *homogenous aggregate datatype*, with respect to the *size property*, we distinguish between a *homogenous aggregate datatype with variable size* and an *array datatype* (which has fixed size). At the next level, with respect to the *aggregate ordering property*, we distinguish between a *homogenous unordered aggregate datatype with variable size* and a *sequence datatype* (which is ordered). Finally, *homogenous unordered aggregate datatype with variable size* has two subclasses: a *bag datatype* and a *set datatype*.

6.5. User defined datatypes

A *user defined datatype* is a *datatype* that is defined by a type specification [23]. A *type specification* defines a new datatype that refers to an existing datatype or a datatype generator. This specification can be used to rename an existing datatype, to define a new datatype and to define a new datatype generator. It includes a *type identifier*, a *type-parameter list* and a *type definition*. Examples of defined datatypes in OntoDT include the *labeled graph* class and its two subclasses: a *tree datatype* and a *Directed Acyclic Graph (DAG) datatype*, a *natural number datatype*, a *modulo datatype*, a *bit datatype* and others.

7. Ontology evaluation

We assess the quality of OntoDT from three different aspects. We analyze a set of ontology metrics; assess how well the ontology meets a set of predefined design criteria and ontology best practices; and assess the ontology with respect to a set of competency questions.

A variety of ontology metrics is available for assessing ontologies [15]. We used statistical ontology metrics from the Protégé software [48] and the BioPortal web service [4], such as the number of classes, the number of axioms, the number of disjoint classes, the number of equivalent classes, the number of annotation axioms, and others. The values of these statistical ontology metrics for the OntoDT ontology are presented in Table 2. The Description Logics (DL) expressivity [1] of the ontology language defined by the ontology is ALCHOIQ(D).

[7] Definitions of all generated datatype classes are given in the ontology.

In the design and implementation phases of ontology development, we used a set of predefined ontology best practices and design criteria. After the ontology was constructed, we assessed it against these principles in order to see how the finalized ontology fits them. We concluded that OntoDT fits to the assessment criteria from the OBO Foundry and other commonly accepted ontology engineering criteria. The results of the evaluation are summarized in Tables A.1–A.4 of the Appendix A.

Following the methodology for the design and evaluation of ontologies proposed by Grüniger and Fox [16], we specified a set of competency questions in the design phase. We evaluated the ontology after the implementation phase against the competency questions as an entailment queries with respect to the axioms in the ontology and concluded that the ontology is adequate. In Section 5.6, we showed examples of formalized queries expressed as Description Logic class expressions.

8. Use cases

In this section, we present three use cases of the OntoDT ontology. The first use case is about the use of OntoDT as a mid-level ontology by the ontology of core data mining entities (OntoDM-core). In the second use case, we present how OntoDT is used for annotating and querying dataset repositories. Finally, in the third use case we discuss the use of OntoDT for representation and annotation of bio-informatics datatypes.

8.1. Use of OntoDT as a mid-level ontology by the OntoDM-core ontology

In data mining, the data used for analysis are organized in the form of a dataset. Every dataset consists of data examples. An individual data example has its own structure described with a datatype. The datatype describes the type of data contained in the data examples, with the set of distinct values they can take, their properties and operations. The task of data mining is to produce some type of a generalization from a given dataset (i.e., predictive model, set of patterns, clustering, probability distribution). A data mining task is solved by using a data mining algorithm, which is implemented as a computer program and when executed takes as input a dataset and gives as output a generalization. In this use case, we show how the OntoDT ontology can be used in context of describing the domain of data mining.

8.1.1. The OntoDM-core ontology

OntoDM-core is a domain ontology that defines the most essential data mining entities such as dataset, data mining task, generalizations, data mining algorithms, and constraints [45]. It provides a representational framework for the description of mining structured data, and in addition provides taxonomies of datasets, data mining tasks, generalizations, data mining algorithms and constraints, based on the type of data. For this purpose, OntoDM-core uses the OntoDT ontology as a mid-level[8] ontology for the representation of datatypes.

8.1.2. The role of OntoDT in OntoDM-core

The OntoDT ontology has a key role in OntoDM-core [45, see Fig. 2]. In OntoDM-core, the data is modeled using a data specification entity. It describes the datatype of the underlying data and is connected to the OntoDT datatype class via the IS-ABOUT relation. In this way, the OntoDM-core ontology exploits the OntoDT mechanism for representing arbitrary complex datatypes, in the context of representing the mining of structured data.

In the case of datasets, the datatype information is needed to specify the type of data contained in the dataset. For the case of data mining tasks (predictive modeling, pattern discovery, probability distribution estimation, clustering), the information about a datatype is needed to specify on which type of data the data mining task at hand is applicable. For the case of generalizations (patterns, models, probability distributions, clusterings), the information about a datatype is needed to specify on which type of data the generalization is defined. Finally, a datatype information is important in order to provide a recommendation of a set of applicable data mining algorithms given a specific dataset.

8.1.3. Data mining datatypes

For the data mining domain, the OntoDT ontology provides a set of data mining specific datatypes which are derived from the OntoDT basic datatypes by subclassing. These include: tuple of primitives (record of primitive components), set of discrete datatype, sequence of real (time series), labeled graph with boolean edges and discrete datatype nodes, tree of discrete datatype nodes, Directed Acyclic Graph (DAG) of discrete datatype nodes and others. Furthermore, OntoDT provides a flexibility to define an arbitrary datatype, that can later be used to define a specific data mining task, applicable only to that datatype. For example, the hierarchical classification task [58] can be only applied to data having a record of primitive components on the descriptive side and a labeled graph (or tree) with boolean edges and discrete datatype nodes on the target side. Consequently, a set of applicable algorithms on a dataset are all algorithms that can solve a data mining task on that dataset.

[8] A mid-level ontology serves as a bridge between more general entities defined in the upper level ontology and the low-level domain entities from the domain ontology.

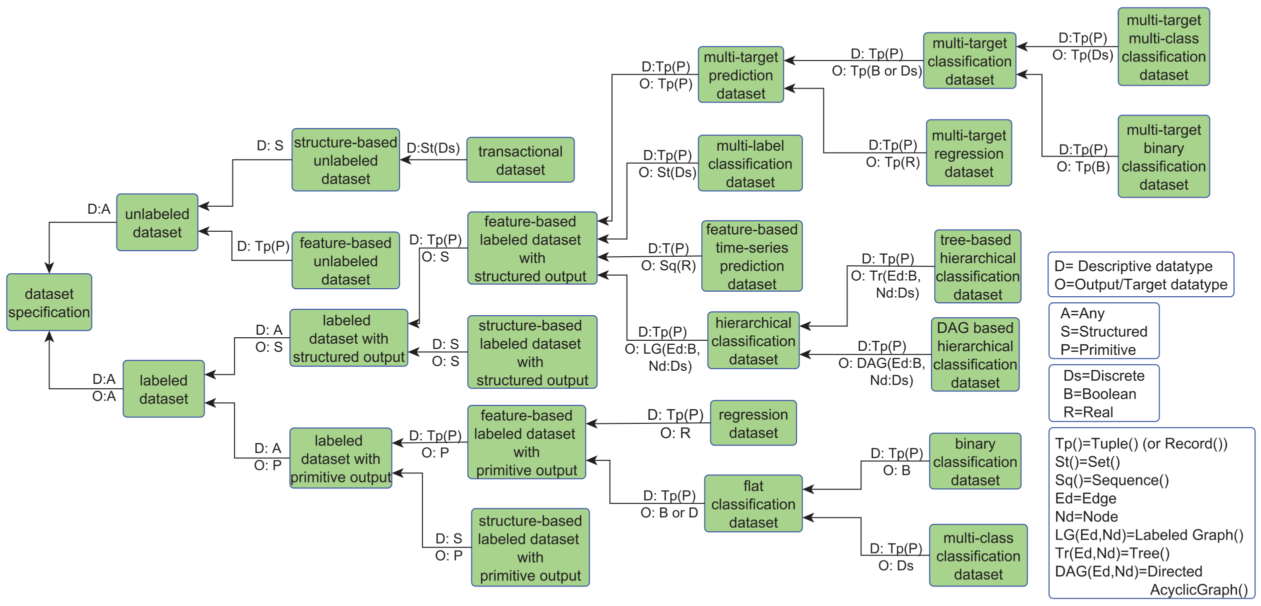

dataset specification
unlabeled dataset
labeled dataset
D:A
D:A O:A
structure-based unlabeled dataset
feature-based unlabeled dataset
transactional dataset
D: S
D: Tp(P)
D:St(Ds)
labeled dataset with structured output
labeled dataset with primitive output
D: A O: S
D: A O: P
feature-based labeled dataset with structured output
structure-based labeled dataset with structured output
feature-based labeled dataset with primitive output
structure-based labeled dataset with primitive output
D: Tp(P) O: S
D: S O: S
D: Tp(P) O: P
D: S O: P
multi-target prediction dataset
multi-label classification dataset
feature-based time-series prediction dataset
hierarchical classification dataset
D:Tp(P) O: Tp(P)
D:Tp(P) O: St(Ds)
D:T(P) O: Sq(R)
D:Tp(P) O: LG(Ed:B, Nd:Ds)
regression dataset
flat classification dataset
D: Tp(P) O: R
D: Tp(P) O: B or D
multi-target classification dataset
multi-target regression dataset
D: Tp(P) O: Tp(B or Ds)
D:Tp(P) O: Tp(R)
multi-target multi-class classification dataset
multi-target binary classification dataset
D:Tp(P) O: Tp(Ds)
D:Tp(P) O: Tp(B)
tree-based hierarchical classification dataset
DAG based hierarchical classification dataset
D: Tp(P) O: Tr(Ed:B, Nd:Ds)
D:Tp(P) O: DAG(Ed:B, Nd:Ds)
binary classification dataset
multi-class classification dataset
D: Tp(P) O: B
D: Tp(P) O: Ds
D= Descriptive datatype
O=Output/Target datatype
A=Any
S=Structured
P=Primitive
Ds=Discrete
B=Boolean
R=Real
Tp()=Tuple() (or Record())
St()=Set()
Sq()=Sequence()
Ed=Edge
Nd=Node
LG(Ed,Nd)=Labeled Graph()
Tr(Ed,Nd)=Tree()
DAG(Ed,Nd)=Directed AcyclicGraph()

Fig. 6. OntoDM-core dataset taxonomy obtained by using the OntoDT datatype taxonomy. The labels on the arrows denote the datatypes used to define the dataset types. The meaning of the labels is presented in the legend.

8.1.4. *Taxonomy of datasets*

In Fig. 6, we show the taxonomy of datasets from OntoDM-core that was obtained by using the OntoDT taxonomy of datatypes. At the first level of the taxonomy of datasets, we have the *unlabeled dataset* (a dataset that has only descriptive data and is usually used for clustering and pattern discovery tasks) and the *labeled dataset* (a dataset that has both descriptive and output/target data and is usually used for predictive modeling tasks).

At the second level of the unlabeled dataset taxonomy, we distinguish between a *feature-based unlabeled dataset* and a *structure-based unlabeled dataset*. A feature-based unlabeled dataset is a dataset that has record of primitives as the underlying descriptive datatype. A structure-based unlabeled dataset is a dataset that has some aggregate datatype (other than record of primitives) as the underlying descriptive datatype.

At the second level of the labeled dataset taxonomy, we distinguish between a *labeled dataset with primitive output* and a *labeled dataset with structured output*. The first type can have any primitive datatype as its output data specification, while the second type can have an aggregate datatype on the output side. Both dataset types can have an arbitrary datatype on the descriptive side.

If we focus only on labeled datasets, the taxonomy can be further extended based on the datatypes on the descriptive and output part of the data. The *labeled dataset with a primitive output* class is extended with two subclasses: *feature-based labeled dataset with primitive output* and *structure-based labeled dataset with primitive output*. The first subclass has a record of primitives on the descriptive side, while the second subclass can have any aggregate datatype (other than record of primitives) on the descriptive side. Feature-based labeled datasets with primitive output are further extended based on the type of primitive output. This includes the following subclasses: *regression dataset* (having a real datatype as output), *binary classification dataset* (having a boolean datatype as output), and *multi-class classification dataset* (having a discrete datatype as output).

In analogy, the *labeled dataset with structured output* class is extended with two subclasses: *feature-based labeled dataset with structured output* and *structure-based labeled dataset with structured output*. The first subclass has a record of primitives on the descriptive side, while the second subclass can have any aggregate datatype (other than record of primitives). Feature-based labeled datasets with structured output are further extended based on the type of structured output. This includes the following subclasses: *multi-target prediction dataset* (having as output datatype a record of primitives), *multi-label classification dataset* (having as output datatype a set of discrete), *feature-based time series prediction dataset* (having as output datatype a sequence of reals), and *hierarchical classification dataset* (having as output datatype a labeled graph with boolean edges and discrete nodes). Finally, a *multi-target prediction dataset* is further extended depending on the primitive datatypes that compose the record. This includes the following subclasses: *multi-target regression dataset*, *multi-target binary dataset*, and *multi-target multi-class dataset*.

To summarize, in this use case we showed how the OntoDT ontology is used by a domain ontology of data mining. Furthermore, OntoDT has a key role in data specification, which is then base for specifying datasets, data mining tasks, generalizations, and data mining algorithms. OntoDT supports the retrieval of suitable DM algorithms for a dataset with the specified datatype properties. Finally, we showed how the structure of the OntoDT taxonomy of datatypes can be used to produce a taxonomy of datasets.

8.2. Annotation and querying machine learning dataset repositories using OntoDT

There is a large number of datasets, used for different machine learning and data mining tasks, available on-line. The best known dataset repository for machine learning is the UCI Machine Learning Repository[9] [2]. It stores more than 300 datasets used for the empirical analysis of machine learning algorithms. The datasets in the repository are annotated with several descriptors, including: machine learning task (classification, regression, clustering, other), attribute type (categorical, numerical, mixed), and datatype (multivariate, univariate, sequential, time series, text, domain-theory, other). The descriptors used in this repository are not based on any taxonomy (or ontology) of machine learning tasks, nor taxonomy of datatypes, which limits their applicability and interoperability. For example, one cannot describe a multi-target prediction dataset, whose data examples have as target/class part a tuple of values instead of just one value, as is the case for a traditional predictive modeling dataset. Here, we show how OntoDT can be used for annotation of datasets with datatype information, and show how the annotations can be used to query dataset repositories.

8.2.1. Annotation scheme

We propose a scheme for annotating machine learning and data mining datasets with information about datatypes using classes from the OntoDM-core and the OntoDT ontology. From the OntoDM-core ontology, we use the class for representing datasets (*OntoDM-core: DM-dataset*) and the class that contains the specification of the dataset (*OntoDM-core: dataset specification*), which is connected to the DM-dataset class via the IS-ABOUT relation. A dataset specification includes information about the datatype of the data examples by using relations to the classes from OntoDT. The OntoDT taxonomy of datatypes was used to produce a taxonomy of datasets (see Section 8.1).

To show the benefit of using the OntoDT ontology, we annotated 193 instances of labeled datasets,[10] used in predictive modeling experiments by Kocev et al. [30, see Tables 3–5] and Madjarov et al. [35, see Table 1]. In the experiments, one dataset was used for several different learning tasks. We represented each variant of a dataset in a concrete learning setting as a dataset instance. All different variants of the same dataset were grouped under the same dataset class (as instances of that class). For example, the EDM dataset [27] has 16 continuous descriptive attributes and 2 continuous target attributes. This dataset was used for the tasks of multi-target regression, multi-target classification (using the discretized target attributes), and traditional regression and classification for both target attributes separately. We represent all 6 variants of this dataset as separate dataset instances of the EDM dataset class, as each dataset instance is characterized by a different datatype.

In Fig. 7, we present an example annotation of one dataset instance of the EDM dataset, which has continuous descriptive attributes and two continuous target attributes. Each labeled dataset instance is described by a *labeled dataset record datatype*, which is a subclass of the *record datatype* with the distinctive feature that it contains only two field components, one describing the datatype on the description side and one the datatype on the target side. In that sense, the *dset:EDM-MCT* dataset instance is described by the labeled dataset record datatype instance containing an instance of the record of real datatype in both descriptive and target field components.

8.2.2. Inferred ontology

After annotating the dataset instances with OntoDT terms, we imported the annotations in the ontology using the Populous tool [25]. Next, we performed reasoning using the HermiT reasoner version 1.3.8 [18] and produced an inferred ontology that was used for running queries about datasets and datatypes.[11] By using reasoning, some of the knowledge that was implicitly encoded in the ontology was made explicit. The transitivity of the IS-A relation is one example of the implicit knowledge built inside the ontology. This allows us to ask queries about datatypes that are higher in the taxonomy and the result would include all of its subclasses as well. For example, if we would query for datasets that have some *homogenous aggregate datatype* on the output/target side, by using the inferred ontology, we would get all datasets that contain target datatypes that are subclasses of the homogenous aggregate datatype class to the lowest levels as answer of our query. In this case, this would include the set datatype, the bag datatype, the sequence datatype, and the array datatype.

8.2.3. Querying annotated dataset repositories

We queried OntoDT by using the OWL2Query Protégé plug-in [43], which employs SPARQL-DL [59]. The OWL2Query plug-in is a conjunctive query, meta-query and a visualization engine that facilitates the creation of SPARQL queries using an intuitive graph based syntax and evaluates them by using an OWL-API compliant reasoner [31]. In addition, we also used the built-in SPARQL Protégé engine to run SPARQL queries.

For example, a query “Give me all labeled datasets that have a record datatype on the output/target side” can be encoded in SPARQL-DL as follows:

```
Q(datasetClass) : −PV(is-about, ?datasetSpecInstance, ?datasetInstance),
PV(has-member, ?datasetSpecInstance, ?datasetDtypeInstance),
```

[9] URL: http://archive.ics.uci.edu/ml/datasets.html, accessed 02.12.2014.
[10] The annotations of the datasets are available on the ontology web page.
[11] The inferred ontology is available on the ontology web page.

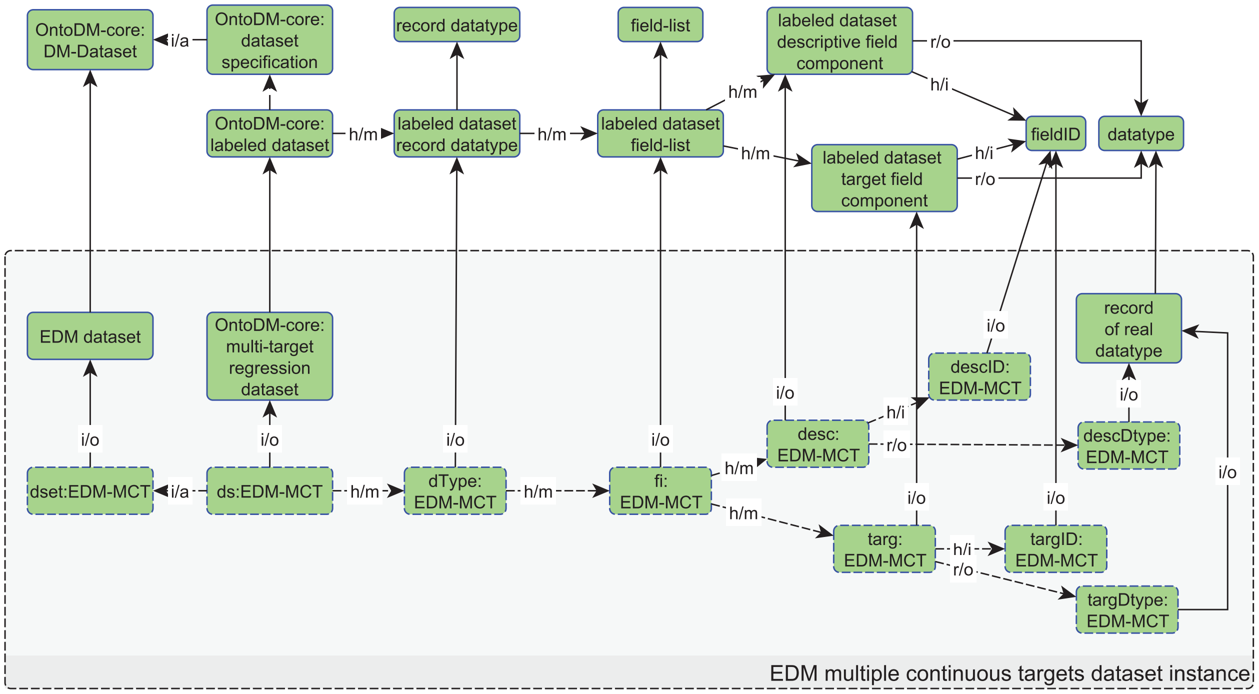


Fig. 7. Annotation of datasets using OntoDT. An example of a EDM dataset instance. The rectangular boxes represent ontology classes. The rectangular boxes with dashed lines represent instances. Unlabeled arrows represent IS-A relations, while labeled arrows have the following meaning: i/o represents the INSTANCE-OF relation, h/m represents HAS-MEMBER, h/i represents HAS-IDENTIFIER, i/a represents IS-ABOUT, and r/o represents ROLE-OF. Full lines denote existential relations. Dashed lines denote relation between instances.

```
PV(has-member, ?datasetDtypeInstance, ?fieldListInstance),
PV(role-of, ?fieldComponent, ?targetFieldDatatypeInstance),
PV(has-member, ?fieldListInstance, ?fieldComponent),
T('labeled dataset record datatype', ?datasetDtypeInstance),
T('field list', ?fieldListInstance),
T('labeled dataset target field component', ?fieldComponent),
T('record datatype', ?targetFieldDatatypeInstance),
T(?datasetClass, ?datasetInstance),
SCO(?datasetClass, 'DM-dataset').
```

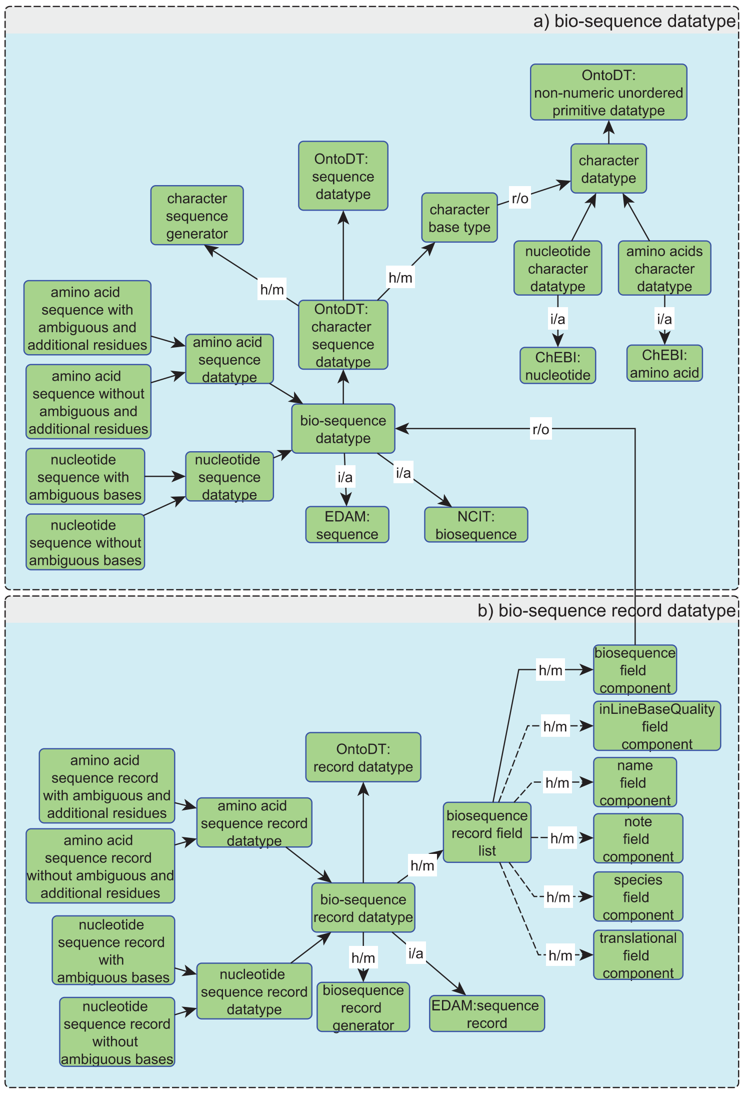
where PV is a property value query atom, T is a type query atom, and SCO is a subclass of query atom. The result of executing this query is a list of 18 dataset classes, which have instances with a record datatype on the output side. Without the OntoDT annotations, and the inferred ontology axioms, provided by the reasoner, it would be difficult to retrieve datatype information about labeled datasets from the repository.

This use case demonstrates that OntoDT provides logically consistent descriptors for annotating machine learning datasets and facilitates information retrieval about the datasets. This approach can be generalized and extended to annotate and query arbitrary datasets, learning tasks, and algorithms with datatype information (e.g., for semantical annotation in the OpenML platform[12] [66]). Furthermore, by using the ontology one can search the annotated datasets depending on the datatype. OntoDT can thus serve as a reference model for the consistent annotation of dataset repositories with datatype information.

8.3. The representation of bioinformatics datatypes

OntoDT is a generic ontology and it allows easy extensions to represent domain specific datatypes. This can be done by directly extending the OntoDT datatype taxonomy and defining the semantic meaning of the domain datatypes by linking them to the corresponding entities in domain ontologies. For example, we can define an *amino-acid sequence datatype* as a subclass of the *character sequence datatype* class (which is a sequence datatype having characters as its base type). Its semantic meaning can be

[12] OpenML is a collaboration platform through which scientists can share, organize and discuss machine learning experiments, data, and algorithms.


a) bio-sequence datatype
OntoDT: non-numeric unordered primitive datatype
character datatype
OntoDT: sequence datatype
character sequence generator
character base type
r/o
nucleotide character datatype
amino acids character datatype
h/m
h/m
amino acid sequence with ambiguous and additional residues
OntoDT: character sequence datatype
i/a
i/a
amino acid sequence datatype
ChEBI: nucleotide
ChEBI: amino acid
amino acid sequence without ambiguous and additional residues
bio-sequence datatype
r/o
nucleotide sequence with ambiguous bases
nucleotide sequence datatype
i/a
i/a
nucleotide sequence without ambiguous bases
EDAM: sequence
NCIT: biosequence
b) bio-sequence record datatype
biosequence field component
h/m
inLineBaseQuality field component
h/m
amino acid sequence record with ambiguous and additional residues
OntoDT: record datatype
name field component
h/m
amino acid sequence record datatype
biosequence record field list
h/m
note field component
amino acid sequence record without ambiguous and additional residues
h/m
species field component
h/m
bio-sequence record datatype
nucleotide sequence record with ambiguous bases
translational field component
h/m
h/m
i/a
nucleotide sequence record datatype
biosequence record generator
EDAM:sequence record
nucleotide sequence record without ambiguous bases

(a) Bio-sequence and bio-sequence record

```
Class: 'bio-sequence datatype'
    EquivalentTo:
        'nucleotide sequence datatype' or
        'amino acid sequence record'
    SubClassOf:
        'character sequence datatype',
        is-about some NCIT:biosequence,
        is-about some EDAM:sequence
Class: 'nucleotide sequence datatype'
    SubClassOf:
        'bio-sequence datatype',
        is-about some 'EDAM: nucleic acid sequence',
        is-about some 'NCIT: nucleotide sequence',
        has_member some 'nucleotide character base type'
Class: 'nucleotide character datatype'
    SubClassOf:
        'character datatype',
        is-about some 'ChEBI: nucleotide'
Class: 'amino acid sequence datatype'
    SubClassOf:
        'bio-sequence datatype',
        is-about some 'EDAM: protein sequence',
        is-about some 'NCIT: amino acid sequence',
        has_member some 'amino acids character base type'
Class: 'amino acids character datatype'
    SubClassOf:
        'character datatype',
        is-about some 'ChEBI: amino acid'
Class: 'bio-sequence record'
    EquivalentTo:
        'nucleotide sequence record' or
        'amino acid sequence record'
    SubClassOf:
        'record (tuple) datatype',
        is-about some 'EDAM:sequence record',
        has_member some 'bio-sequence record field-list'
Class: 'bio-sequence record field-list'
    SubClassOf:
        field-list,
        has_member some 'bio-sequence field component',
        has_member only 'species field component',
        has_member only 'name field component',
        has_member only 'translational-data field component'
Class: 'bio-sequence field component'
    SubClassOf:
        'sequence field component',
        has_identifier some 'field identifier',
        role_of some 'bio-sequence datatype'
```

(b) OWL Manchester syntax.

Fig. 8. Representation of bio-sequence and bio-sequence record datatypes from BioXSD in OntoDT. The rectangular boxes represent ontology classes. Unlabeled arrows represent IS-A relations, while labeled arrows have the following meaning: r/o represents ROLE-OF RELATION, i/a represents IS-ABOUT relation, and h/m represents HAS-MEMBER relation. Full lines denote existential relations, while dashed lines denote universal relation.

defined via the IS-ABOUT relation to the *amino acid sequence* entity provided by the National Cancer Institute Thesaurus [39]. In this way, OntoDT can be used for representation of bioinformatics datatypes.

Currently, BioXSD is used to define the basic bio-informatics types of data [26], but it inherits the limitations of XSD (see Section 2). BioXSD does not support arbitrary datatypes and it does not provide a clear framework for the representation of the semantic meaning of the data. We propose to enhance the representation of bioinformatics datatypes by exploiting the rigorous taxonomy of datatypes defined in OntoDT and the framework for the representation of semantic meanings adopted by OntoDT following the RDF data cube vocabulary [51]. OntoDT is fully interoperable with OBO bio-ontologies because it was developed by following the OBO Foundry recommendations (see Section 4) and therefore it fully supports the representation of the semantic meaning of the data by the corresponding entities defined in domain-specific bio-ontologies.

For example, the BioXSD datatype *sequence* represents a string of 1-letter coded nucleotides or amino-acids. A *sequence record* is a datatype containing a sequence, and optionally some metadata about the sequence (for the purpose of identification). The semantic meanings of the terms *sequence* and *nucleotide* are curtail for the capturing of the semantic meaning of the data of the datatype *sequence*. However, this datatype *sequence* is not explicitly linked to the classes *nucleotide* and *amino-acid* defined in the ChEBI ontology [5], which is recommended by OBO Foundry as a reference ontology.

In Fig. 8, we present the extension of OntoDT to represent the bio-sequence and bio-sequence record datatypes from BioXSD. We represent the *bio-sequence datatype* class as a subclass of the *character sequence datatype* class with the defined semantic meaning in the NCI Thesaurus [39] and EDAM ontology [13] (see Fig. 8a). In order to define the *nucleotide* and *amino acid sequences datatypes*, we define two subclasses of the *character datatype* class: *nucleotide character datatype* and *amino acid character datatype*. In order to define their semantic meaning, we explicitly link them to the *nucleotide* and *amino acid* classes from the ChEBI ontology [5]. Consequently, the *bio-sequence datatype* class has two subclasses: *nucleotide sequence datatype* and *amino acid sequence datatype*. Furthermore, both datatypes have two subclasses, depending on whether they include ambiguous bases (in the case of nucleotides) or ambiguous and additional residues (in the case of amino acids). For example, the *nucleotide sequence datatype* class has two subclasses: *nucleotide sequence with ambiguous bases* (relates to general nucleotide sequence in BioXSD) and *nucleotide sequence without ambiguous bases* (relates to nucleotide sequence in BioXSD).

We represent the *bio-sequence record datatype* class as a subclass of the *record datatype* class (see Fig. 8b). This datatype is defined by a *record generator* and the *bio-sequence-field-list*. As defined in BioXSD, the datatype contains a bio-sequence as a mandatory component and a set of metadata (such as name, note, species, translationalData, reference, inlineBaseQuality) as non-mandatory components. In OntoDT, we model the *bio-sequence field component* class is as a role of the bio-sequence datatype (defined previously).

Having in mind that we can have sequences of nucleotides and amino acids, bio-sequence record datatype has two subclasses: *nucleotide sequence record datatype* and *amino acid sequence record datatype*. Both datatypes have two subclasses, depending on whether they include ambiguous bases (in the case of nucleotides) or ambiguous and additional residues (in the case of amino acids). For example, *nucleotide sequence record datatype* has two instances, *nucleotide sequence record with ambiguous bases* (relates to general nucleotide sequence record in BioXSD) and *nucleotide sequence record without ambiguous bases* (relates to nucleotide sequence record in BioXSD). In a similar way, we can define other datatypes from BioXSD as subclasses or instances of the OntoDT datatypes.

This use case demonstrates that OntoDT provides logically consistent representation of bioinformatics datatypes from BioXSD and enables an accurate representation of the semantic meanings of the data of the specified datatypes. OntoDT has been designed as a generic and comprehensive ontology of datatypes and consequently any datatype from other resources can also be represented by OntoDT. We suggest that OntoDT can serve as a reference model for the consistent representation of datatypes used within biomedical domains and wider.

9. Discussion

In this section, we discuss several aspects of OntoDT that concern its integration with other ontological resources and the potential application areas and in e-science. First, we discuss the integration of OntoDT with other ontologies. Next, we focus on a discussion of how OntoDT can be used to enrich BioXSD annotations. Finally, we discuss the prospects of using OntoDT for web-services, cloud computing and laboratory automation.

Integration of OntoDT with other ontologies. OntoDT adopts a modular approach where not only the information about units of measurements but also other operational information and the semantic meaning of the underlying data is captured and maintained separately. Following best practices, OntoDT clearly separates the semantic meaning of the data from the data itself and its structure. Data processing becomes increasingly complex and accurate recording of how the data have been processed is vital for data analysis. A modular approach for the recording of information is flexible, extensible, and also reduces the complexity of the underlying representational model. The employed designing approach (see Section 4) facilitates a seamless integration of the relevant resources. For example, OntoDT can be easily linked with ontologies of quantities and units (e.g. [32,49]) and also with ontologies defining operations on the data, e.g. OntoDM-core [45].

Using OntoDT annotations in BioXSD. BioXSD uses a combined approach of a pure XML Schema annotated by a data-type ontology using Semantic Annotations for Web Services Description Language (WSDL) and XML Schema [54]. SAWSDL defines a set of extension attributes for the WSDL and XML Schema definition languages. Application of attributes allows the description of additional semantics by using references to conceptual semantic models, e.g., ontologies. BioXSD datatypes are annotated with terms from the EDAM ontology [24] using SAWSDL. In the same way, BioXSD datatypes can be annotated with OntoDT terms. By adding OntoDT annotations to BioXSD, we would allow the web services (such as MaxAlign, ProP, NetNES, BLAST and others) that use BioXSD as representational formalism, to utilize the information about datatypes, their properties and operations that can be performed on them, which is not available when using only EDAM ontology annotations.

For example, by annotating the datatype bio-sequence record from BioXSD with terms from the OntoDT ontology, the web services would have the information that bio-sequence record is in fact a record datatype that is heterogenous and has components (in this case sequence as mandatory, and other non-mandatory ones), its values are unordered, it has fixed size, and each component can be accessed by keying. In addition, web services can also get information about the possible operations on that datatype (in this case equality operation, field select operation and field replace operation). Finally, web services could also get information about the properties and operations on the component datatypes. For example, for the mandatory component bio-sequence, web services can get the information that it is a homogeneous ordered datatype with variable size where each element can be accessed by its position.

Service oriented architectures. With the increasing complexity of scientific workflows, where different web services and other computer programs have to exchange data, there is a need for a reference model of data and datatypes. An ontology of datatypes is a step towards such a model. The ontology could be used to enhance various IT solutions, for example the orchestration in service oriented architecture, cloud computing and laboratory automation. Orchestration describes the automated arrangement, coordination and management of complex computer systems and services [14]. In order to identify what software components or services are capable of performing a required task, the orchestration module needs to know the functionality of the software, it inputs and outputs, including datatypes it can operate with. The OntoDT ontology, we propose in this paper, could provide more structured information about the datatypes and consequently can improve the orchestration of software components and services.

Cloud computing. Another application area for OntoDT is cloud computing. Cloud computing is a network-based service that involves a large number of computers connected through a communication network, i.e., the Internet.[13] The popular models of cloud computing service include software as a service and infrastructure as a service. It is vital for the functioning of such services to specify the available resources, including what datatypes they can work with. Therefore, the sophisticated description of datatypes provided by OntoDT could contribute to the improvement of cloud computing. There are a number of open standards under development, with the aim of delivering interoperability and portability of cloud software.[14] OntoDT can contribute towards the representation of datatypes within those standards.

Laboratory automation. Finally, another potential application domain for OntoDT is laboratory automation. Laboratory automation typically comprises many different automated laboratory instruments, devices, software and methodologies to expedite the efficiency and effectiveness of scientific research in laboratories. Unfortunately, those laboratory instruments, devices and software often do not communicate with each other and with the users effectively. One of the reasons is that they operate with different and sometimes proprietary datatypes that other components of the laboratory workflow cannot input directly. A reference model of datatypes would contribute to the solution of this serious problem. OntoDT could thus be used to improve the communication between various pieces of equipment and software in a biological laboratory.

10. Conclusions

In this paper, we have presented OntoDT, a proposal for a generic ontology of datatypes. The ontology is based on the ISO/IEC 11404 standard for datatypes in computer systems. It defines the key entities for representation of datatypes, such as datatype, extended datatype, datatype properties, datatype characterizing operations, datatype value space. In addition, it defines also the support entities needed for representing specific datatypes. The ontology has been constructed by following best practices in ontology design so that it is complementary and can be easily integrated with other state-of-the-art ontologies for science. Finally, the ontology has been evaluated from several different aspects, such as ontology metrics, assessment in terms of design principles, and assessment in terms of competency questions.

The contributions of this paper are as follows. First, we propose a taxonomy of datatypes based on the properties of datatypes and their structure. Second, we show the suitability of OntoDT for querying about datatypes. Next, we demonstrate how OntoDT is crucial in defining the key entities in the domain of data mining such as data specification, data mining task, generalization and implicitly data mining algorithm, and we show how the OntoDT taxonomy of datatypes is used to produce a taxonomy of datasets. Finally, we demonstrate how OntoDT is used for annotation of dataset repositories and for representing bioinformatics datatypes.

We envision several dimensions of further development of OntoDT that would overcome the current limitations of the ontology. First, we want to further establish the connection with domain ontologies and represent domain dependent semantic datatypes for different domains (e.g., biology, ecology, economics) using the OntoDT ontology and the semantics of the domain entities from domain ontologies. Next, we want to populate the OntoDT ontologies with more complex datatypes such as text, audio, images, video, and to include their specific operations and properties. Finally, we would like to link the representation of data formats and datatypes by re-using the ontologies that deal with different data formats (e.g., EDAM) and integrating them with OntoDT.

Acknowledgments

We would like to acknowledge the support of the European Commission through the projects: MAESTRA – Learning from Massive, Incompletely annotated, and Structured data (grant number FP7-ICT-612944) and HBP – The Human Brain Project (grant number FP7-ICT-604102). We would also like to acknowledge the support of the Engineering and Physical Sciences Research Council UK (grant number EP/K030469/1).

Appendix A

See Tables A.1–A.4

[13] http://www.netlingo.com/word/cloud-computing.php, accessed 18.12.2014.
[14] http://www.infoworld.com/d/cloud-computing/openstack-foundation-launches-202694, accessed 18.12.2014.

Table A.1
Scope and structure assessment.

#	Principle	Assessment
1	Coverage	OntoDT provides a representation of datatypes commonly used in programming languages and software. It is based on the ISO/IEC 11404:2007 standard for datatypes.
2	Upper-ontology	OntoDT uses the classes from IAO, which has the BFO ontology as an upper-level ontology.
3	Relations	OntoDT uses relations defined in RO, IAO and OBI. The relations defined in IAO and OBI are candidates for inclusion into RO.
4	Ontology reuse	OntoDT reuses classes and relations from OBI and IAO.
5	Modularity	OntoDT is part of the OntoDM ontology, which contains also the OntoDM-core and the OntoDM-KDD subontologies. It can be used independently.
6	Use of disjoint classes	In OntoDT, we extensively use disjoint class axioms.
7	Use of single inheritance	In OntoDT, each class has only one superclass. This reduces the potential inconsistency and errors in reasoning processes.
8	IS-A completeness	All OntoDT classes are connected via the IS-A relation. There are no orphan classes.
9	Domains and ranges for relations	Imported relations from RO, IAO and OBI have defined ranges and domains.
10	Inverse relations	Most of the imported relations from RO, OBI, and IAO have defined inverse relations.
11	Orthogonality with other ontologies	OntoDT is orthogonal to other ontologies already lodged within OBO.
12	Instantability	More extensive population of the ontology with instances is planned for the future.

Table A.2
Naming and vocabulary assessment.

#	Principle	Assessment
1	Ontology language	OntoDT is expressed in the W3C standard Web Ontology Language OWL-DL.
2	Use of annotation properties	We reuse the OBI consortium defined meta-data (http://obi-ontology.org/page/OBI_Minimal_metadata) to provide additional semantic annotation of the classes and relations.
3	Label annotations	We use label annotations to provide human readable names of classes and relations in the ontology.
4	Ontology namespace	OntoDT has its own namespace http://www.ontodm.com/OntoDT#. The classes and relations that are imported from other ontologies have kept their source ontology namespace and ID.
5	Ontology term IDs	The IDs of the ontology terms include a combination of an ontology module ID and a multiple digit code. For the OntoDT we use `OntoDT_xxxxx`.
6	Multi-lingual capabilities	At this moment the OntoDT ontology does not provide multi-lingual capabilities.
7	Naming conventions	The ontology uses set of naming conventions provided by the OBO Foundry.
8	Referencing external classes	The external classes are referenced by using the MIREOT principle.

Table A.3
Documentation and collaboration assessment.

#	Principle	Assessment
1	Definitions	Most of the OntoDT classes have textual definitions that are taken from the ISO/IEC 11404. They are regularly updated and revised. The source of the definitions is properly referenced in the annotations.
2	Documentation	The ontology is documented on its dedicated web page.
3	Collaboration efforts	OntoDT still does not participate in any collaboration effort.

Table A.4
Availability, maintenance and use assessment.

#	Principle	Assessment
1	Use of reasoners	We use the HermiT reasoner to test the class and relations consistency and for producing the inferred ontology.
2	Openness and availability	OntoDT is open and is available in its web page http://www.ontodt.comand additionally at BioPortal (http://bioportal.bioontology.org/).
3	Versioning	For tracking the changes in the ontology we use the industry standard Subversion tool.
4	Users of the ontology	The ontology is reused by the OntoDM-core ontology.
5	Maintenance	The ontology has a dedicated person that cares about its maintenance.
6	Handling of obsolete classes	Deleted classes in the OntoDT class hierarchy are listed under the *obsolete* class, so that applications based on them can still use the terms. The domain terms that have been collected so far but are still not represented in the ontology are listed under the *non-curated* class in the OntoDT class hierarchy.

References

[1] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, New York, NY, USA, 2003.
[2] K. Bache, M. Lichman, 2013, UCI machine learning repository, URL: http://archive.ics.uci.edu/ml (accessed 8.12.14).
[3] Basic Formal Ontology (BFO) web page, 2014, URL: http://www.ifomis.org/bfo (accessed 31.03.14).
[4] BioPortal web page, 2014, URL: https://bioportal.bioontology.org (accessed 31.03.14).

[5] Chemical Entities of Biological Interest (ChEBI) web page, 2014, URL: http://www.ebi.ac.uk/chebi/ (accessed 08.12.14).
[6] M. Compton, P. Barnaghi, L. Bermudez, R. Garca-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W.D. Kelsey, D.L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor, The SSN ontology of the W3C semantic sensor network incubator group, Web Semant.: Sci., Serv. Agents World Wide Web 17 (2012) 25–32. http://dx.doi.org/10.1016/j.websem.2012.05.003.
[7] M. Courtot, F. Gibson, A.L. Lister, J. Malone, D. Schober, R.R. Brinkman, A. Ruttenberg, MIREOT: the minimum information to reference an external ontology term, Appl. Ontol. 6 (1) (2011) 23–33.
[8] N. Dale, H.M. Walker, A classification of data types, Comput. Sci. Edu. 3 (3) (1992) 223–232.
[9] Data Mining OPtimization Ontology (DMOP) web page, 2014, URL: http://www.e-lico.eu/DMOP.html (accessed 31.03.14).
[10] Data Type Registry (DTR) web page, 2015, URL: http://typeregistry.org/ (accessed 11.05.15).
[11] Data Type Registry work group web page, 2015, URL: https://rd-alliance.org/groups/data-type-registries-wg.html (accessed 11.05.15).
[12] Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) web page, 2014, URL: http://www.loa.istc.cnr.it/old/DOLCE.html (accessed 08.12.14).
[13] EDAM ontology web page, 2014, URL: http://edamontology.org (accessed 31.03.14).
[14] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
[15] J. Garcia, F.J. Garca-Penalvo, R. Theron, A survey on ontology metrics, Knowledge Management, Information Systems, E-Learning, and Sustainability Research, Communications in Computer and Information Science, Springer, Berlin Heidelberg, 2010, pp. 22–27.
[16] M. Grüninger, M. Fox, Methodology for the design and evaluation of ontologies, in: IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995.
[17] Health Level Seven Reference Implementation Model (HL7) web page, 2014, URL: http://www.hl7.org (accessed 31.03.14).
[18] HermiT OWL reasoner web page, 2014, URL: http://www.hermit-reasoner.com (accessed 31.03.14).
[19] M. Hilario, P. Nguyen, H. Do, A. Woznica, A. Kalousis, Ontology-based meta-mining of knowledge discovery workflows, in: N. Jankowski, W. Duch, K. Grabczewski (Eds.), Meta-Learning in Computational Intelligence, S, 2011, pp. 273–315.
[20] Information Artifact Ontology (IAO) web page, 2014, URL: http://code.google.com/p/information-artifact-ontology (accessed 31.03.14).
[21] Image and Data Quality Assessment Ontology (IDQA) web page, 2014, URL: https://bioportal.bioontology.org/ontologies/IDQA (accessed 31.03.14).
[22] ISO/IEC 11404:1996, 1996, Information technology – Programming languages, their environments and system software interfaces – Language-independent datatypes, URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=19346.
[23] ISO/IEC 11404:2007, 2007, Information technology – General-Purpose Datatypes (GPD), URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber =39479.
[24] J. Ison, M. Kalas, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer, P. Rice, EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats, Bioinformatics 29 (10) (2013) 1325–1332.
[25] S. Jupp, M. Horridge, L. Iannone, J. Klein, S. Owen, J. Schanstra, K. Wolstencroft, R. Stevens, Populous: a tool for building OWL ontologies from templates, BMC Bioinformatics 13 (Suppl 1) (2012) S5.
[26] M. Kalaš, P. Puntervoll, A. Joseph, E. Bartaševičiute, A. Topfer, P. Venkataraman, S. Pettifer, J.C. Bryne, J. Ison, C. Blanchet, K. Rapacki, I. Jonassen, BioXSD: the common data-exchange format for everyday bioinformatics web services, Bioinformatics 26 (18) (2010) i540–i546.
[27] A. Karalič, I. Bratko, First order regression, Mach. Learn. 26 (2––3) (1997) 147–176.
[28] C.M. Keet, A. awrynowicz, C. dAmato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens, M. Hilario, The Data Mining OPtimization Ontology, Web Semantics: Science, Services and Agents on the World Wide Web. doi: 10.1016/j.websem.2015.01.001.
[29] J.-U. Kietz, F. Serban, S. Fischer, A. Bernstein, Semantics inside! but let's not tell the data miners: intelligent support for data mining, in: V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab, A. Tordai (Eds.), The Semantic Web: Trends and Challenges, Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 706–720.
[30] D. Kocev, C. Vens, J. Struyf, S. Džeroski, Tree ensembles for predicting structured outputs, Pattern Recognit. 46 (3) (2013) 817–833.
[31] P. Kremen, B. Kostov, Expressive Owl queries: design, evaluation, visualization, Int. J. Semant. Web Inform. Syst. 8 (4) (2012) 57–79.
[32] Library for Quantity Kinds and Units: schema, based on QUDV model OMG SysML(TM), 2014, Version 1.2, URL: http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu (accessed 08.12.14) .
[33] Linked models web page, 2014, URL: http://linkedmodels.org (accessed 31.03.14).
[34] J. Madin, S. Bowers, M. Schildhauer, S. Krivov, D. Pennington, F. Villa, An ontology for describing and synthesizing ecological observation data, Ecol. Inform. 2 (3) (2007) 279–296.
[35] G. Madjarov, D. Kocev, D. Gjorgjevikj, S. Džeroski, An extensive experimental comparison of methods for multi-label learning, Pattern Recognit. 45 (9) (2012) 3084–3104.
[36] J.J. Martin, Data Types and Data Structures, Prentice Hall International, UK, 1986.
[37] B. Meek, A taxonomy of datatypes, ACM SIGPLAN Notic. 29 (9) (1994) 159–167.
[38] Microarray and Gene Expression Data ontology (MGED) web page, 2014, URL: http://mged.sourceforge.net/ontologies/MGEDontology.php (accessed on 31.03.14).
[39] National Cancer Institute Thesaurus (NCIT) web page, 2014, URL: http://ncit.nci.nih.gov (accessed 31.03.14).
[40] I. Niles, A. Pease, Towards a standard upper ontology, in: Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001, FOIS '01, ACM New York, NY, USA, 2001, pp. 2–9.
[41] Ontology of Biomedical Investigations (OBI) web page, 2014, URL: http://obi-ontology.org (accessed 31.03.14).
[42] Open Biomedical Ontologies (OBO), 2014, Foundry web page, URL: http://www.obofoundry.org (accessed 31.03.14).
[43] OWL2Query web page, 2014, URL: http://krizik.felk.cvut.cz/km/owl2query/index.html (accessed 31.03.14).
[44] P. Panov, L. Soldatova, S. Džeroski, Representing entities in the OntoDM data mining ontology, in: S. Džeroski, B. Goethals, P. Panov (Eds.), Inductive Databases and Constraint-Based Data Mining, Springer, New York, 2010, pp. 27–58.
[45] P. Panov, L. Soldatova, S. Džeroski, Ontology of core data mining entities, Data Mining Knowl. Discov. 28 (5–6) (2014) 1222–1265.
[46] S. Pettifer, J. Ison, M. Kalaš, D. Thorne, P. McDermott, I. Jonassen, A. Liaquat, J.M. Fernndez, J.M. Rodriguez, I. Partners, D.G. Pisano, C. Blanchet, M. Uludag, P. Rice, E. Bartaseviciute, K. Rapacki, M. Hekkelman, O. Sand, H. Stockinger, A.B. Clegg, E. Bongcam-Rudloff, J. Salzemann, V. Breton, T.K. Attwood, G. Cameron, G. Vriend, The EMBRACE web service collection, Nucleic Acids Res. 38 (Suppl 2) (2010) W683–W688.
[47] Phylogenetic ontology (PHYLONT) web page, 2014, URL: https://bioportal.bioontology.org/ontologies/PHYLONT (accessed 31.03.14).
[48] Protégé Software web page, 2014, URL: http://protege.stanford.edu (accessed 31.03.14).
[49] Quantities, Units, Dimensions and Data Types Ontologies (QUDT) web page, 2014, URL: http://qudt.org/(accessed 08.12.14).
[50] Research Data Aliance (RDA) web page, 2015, URL: http://rd-alliance.org/ (accessed 11.05.15).
[51] RDF Data Cube Vocabulary web page, 2014, URL: http://www.w3.org/TR/vocab-data-cube/ (accessed 08.12.14).
[52] Relational Ontology (RO) web page, 2014, URL: http://purl.org/obo/owl/OBO_REL (accessed 31.03.14).
[53] Review of Sensor and Observations Ontology web page, 2014, URL: http://www.w3.org/2005/Incubator/ssn/wiki/Review_of_Sensor_and_Observations _Ontologies(accessed 08.12.14).
[54] Semantic Annotations for Web Services Description Language (SAWSDL) web page, 2014, URL: http://www.w3.org/TR/sawsdl (accessed 31.03.14).
[55] Semantic Sensor Network Ontology (SSN) web page, 2014, URL: http://purl.oclc.org/NET/ssnx/ssn (accessed 08.12.14).
[56] C.A. Shaffer, A Practical Introduction to Data Structures and Algorithm Analysis, Prentice Hall, Upper Saddle River, NJ, 1997.
[57] SKOS Simple Knowledge Organization System web page, 2014, URL: http://www.w3.org/2004/02/skos/ (accessed 08.12.2014).
[58] C. Silla, A. Freitas, A survey of hierarchical classification across different application domains, Data Mining Knowl. Discov. 22 (1–2) (2011) 31–72.
[59] E. Sirin, B. Parsia, SPARQL-DL: SPARQL query for OWL-DL, in: Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7, 2007., no. 258 in CEUR Workshop Proceedings, CEUR-WS, 2007.

[60] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L.J. Goldberg, K. Eilbeck, A. Ireland, C.J. Mungall, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R.H. Scheuermann, N. Shah, P.L. Whetzel, S. Lewis, The OBO foundry: coordinated evolution of ontologies to support biomedical data integration, Nat. Biotechnol. 25 (11) (2007) 1251–1255.
[61] B. Smith, W. Ceusters, B. Klagges, J. Kohler, A. Kumar, J. Lomax, C. Mungall, F. Neuhaus, A.L. Rector, C. Rosse, Relations in biomedical ontologies, Genome Biol. 6 (2005) R46.
[62] Suggested Upper Merged Ontology (SUMO) web page, 2014, URL: http://www.adampease.org/OP/ (accessed 08.12.14).
[63] SWO ontology web page, 2014, URL: http://theswo.sourceforge.net (accessed 31.03.14).
[64] Syndromic Surveillance Ontology (SSO) web page, 2014, URL: http://surveillance.mcgill.ca/projects/sso (accessed 31.03.14).
[65] D.G. Thomas, R.V. Pappu, N.A. Baker, Nanoparticle ontology for cancer nanotechnology research, J. Biomed. Inform. 44 (1) (2011) 59–74.
[66] J. Vanschoren, J.N. van Rijn, B. Bischl, L. Torgo, OpenML: networked science in machine learning, SIGKDD Explorat. 15 (2) (2013) 49–60.
[67] W3C XML Schema Definition Language web page, 2014, URL: http://www.w3.org/TR/xmlschema11-1/ (accessed 08.12.14).
[68] XML Schema Reference – XSD Elements web page, 2014, URL: http://www.w3schools.com/schema/schema_elements_ref.asp (accessed 18.12.14.