Visualizing OWL 2 Using Diagrams

Gem Stapleton
University of Brighton, UK
Email: g.e.stapleton@brighton.ac.uk

Michael Compton
Unafilliated

John Howse
University of Brighton, UK
Email: john.howse@brighton.ac.uk

***Abstract*—Diagrams can be an effective means of communicating complex ideas and can aid ontology engineering. Indeed, domain experts often do not have the expertise required to understand or create the complex logical statements of an ontology in description logic (DL). This paper presents a visualisation method, *concept diagrams*, geared toward expressing assertions and class expression axioms alongside providing support for literals, datatypes and data properties. *Property diagrams* are introduced, targeted at object property and data property expression axioms. We demonstrate that concept diagrams and property diagrams provide a large coverage of OWL 2 axioms and are, thus, closely aligned in expressive power.**

I. Introduction

Building ontologies is a challenging task. Tools, such as Protégé [1] and WebProtégé [2], allow the creation of textual representations of knowledge domains. However, when people develop conceptual structures, such as models of knowledge which will lead to an ontology, they often start by sketching [3]. This indicates that symbolic and textual notations do not fulfil all modelling requirements of ontology engineers, and that visual means of communicating are perhaps more accessible. Concept diagrams [4] were developed to fulfil this accessibility need.

While concept diagrams allow the definition of ontologies in a readily accessible form, they are more expressive than OWL 2 [5] when defining axioms that involve individuals, classes and object properties. In addition, concept diagrams do not provide direct support for object property expression axioms or any axioms involving literals, datatypes, or data properties. This paper addresses these issues by presenting a fragment of concept diagrams that is a close match to OWL 2 and contains syntax for literals, datatypes and data properties. The paper also introduces *property diagrams* which are designed for defining object and data property expression axioms. The concept diagram fragment and property diagram notation combine to form a diagrammatic representation that has a formal semantics and a translation into OWL 2. The main contributions of this paper are as follows:

- Identification of a fragment of concept diagrams that closely aligns with OWL 2 assertions and class expression axioms. (Sections IV and V.)
- Development of *property diagrams*, aligning with OWL 2 property expression axioms. (Section VI.)
- Development of extensions to include support for literals, datatypes and data properties. (Sections IV, V and VI.)

We now give an example to illustrate some of the contributions of this paper, illustrating potential benefits of concept diagrams over OWL 2. Fig. 1 represents one named individual, `i`, five named classes, $\mathtt{C}_1$ to $\mathtt{C}_5$, and two object properties, $\mathtt{op}_1$ and $\mathtt{op}_2$, expressing four OWL 2 axioms:

1) an assertion axiom, visualized by a labelled dot inside a curve: `ClassAssertion(`$\mathtt{C}_2$` i)`.
2) two simple subsumption-style class expression axioms, visualized by curve inclusion: `SubClassOf(`$\mathtt{C}_2$ $\mathtt{C}_1$`)` and `SubClassOf(`$\mathtt{C}_5$ $\mathtt{C}_4$`)`.
3) a disjoint-classes-style class expression axiom, visualized by curve disjointness: `DisjointClasses(`$\mathtt{C}_1$ $\mathtt{C}_3$ $\mathtt{C}_4$`)`.
4) a complex subsumption-style class expression axiom, visualized using a chain of arrows: `SubClassOf(`$\mathtt{C}_1$ `AllValuesFrom(`$\mathtt{op}_1$ `(ObjectIntersectionOf(`$\mathtt{C}_3$ `AllValuesFrom(`$\mathtt{op}_2$ $\mathtt{C}_4$`))))).`

`SubClassOf(`$\mathtt{C}_1$ `AllValuesFrom(`$\mathtt{op}_1$ $\mathtt{C}_3$`))` can be inferred from the OWL axioms; this can be read directly from the concept diagram: the arrow sourced on $\mathtt{C}_1$, labelled $\mathtt{op}_1$, targets an anonymous subset of $\mathtt{C}_3$, thus expressing that, from $\mathtt{C}_1$, under the relation $\mathtt{op}_1$, we can only 'reach' things inside $\mathtt{C}_3$. Other information is also made explicit in the concept diagram, such as the disjointness of classes $\mathtt{C}_2$ and $\mathtt{C}_5$, but again this needs to be inferred from the OWL axioms above. Statements which can be read directly from the diagram but must be inferred from the OWL are examples of *free rides* [6] and *observational advantages* [7], which are believed to be a reason why diagrams are often more accessible representations of information than symbolic or textual forms.

In terms of scalability, concept diagrams are able to express large numbers of OWL axioms in a single diagram; [8] includes a concept diagram which is equivalent to 772 OWL axioms. In general, a large number OWL axioms can be expressed in a single diagram, and any given ontology could be represented by many diagrams. This implies that, as a base case, concept diagrams are at least as scalable as OWL and arguably more so.

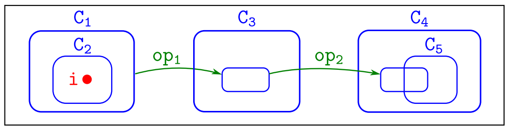

Fig. 1. A concept diagram.

Empirical research suggests that concept diagrams are an accessible notation for ontology engineering, relative to OWL and DL [9], [10]. The inaccessibility of traditional ontology engineering languages is recognised by Rector et al. [11]: "Understanding the logical meaning of any description logic or similar formalism is difficult for most people, and OWL-DL is no exception."

This insight is backed up by Warren et al. who conducted an empirical study of the most commonly used OWL constructs, including class subsumption, disjointness and equivalence alongside All Values From and other types of simple property restrictions [12]. These studies revealed that "despite training, users are prone to certain misconceptions" and "one-third of [participants] commented on the value of drawing a diagram ... In the context of [description logics], diagrams offer a strategy to overcome misconceptions and generally support reasoning." Warren et al. also believed that existing visual techniques promote the visualization of ontology structures rather than the cognitively more challenging features of description logics, with concept diagrams singled out as the exception; other techniques include SOVA [13], VOWL [14], OntoVis [15], OWLViz [16] and an adapted form of existential graphs [17]; most of these are not able to visualize a significant proportion of OWL 2 and are often based on node-link diagrams [18].

Paper Outline: We start by giving a brief overview of OWL 2 (§ II). Next (§ III) background on concept diagrams is presented, followed by (§ IV-A) restrictions and new additions to closely match concept diagrams with OWL 2. The relationship to OWL 2 is then investigated (§ V) followed by the introduction of property diagrams (§ VI).

II. Overview of OWL 2

The basic syntactic building blocks of OWL 2 ontologies are individuals and literals, named classes and datatypes, object properties and data properties. These are used to build more complex expressions, often defining anonymous classes (i.e. sets of elements without a specified name) from which axioms are defined. OWL 2 axioms are primarily defined via subsumption relations between classes, datatypes, object properties and data properties, as well as through making assertions about individuals and literals.

OWL expressions define axioms over two non-empty universal sets: the *object domain* and the *data domain*. *Individuals* and *literals* represent elements of the object domain and data domain respectively. *Classes* and *datatypes* represent subsets of the object domain and data domain respectively. *Object properties* represent binary relations on the object domain whereas *data properties* represent binary relations from the object domain to the data domain.

In what follows, we present details of the OWL syntax and semantics following [5]; we omit a description of the semantics where this should be self-evident and do not included details on expressions involving constraining facets or keys as they cannot be expressed by concept diagrams.

A. Building Sets and Binary Relations

In order to define a rich variety of axioms, OWL requires syntax to build complex expressions from basic building blocks. *Object property expressions*, together with data properties, represent binary relations. *Class expressions* and *data ranges* represent sets.

Object Property Expressions: Any object property, typically denoted by `OP`, is an object property expression, often denoted by `OPE`. The inverse of an object property is also an object property expression: `ObjectInverseOf(OP)`. For example, given the object property `isParentOf`, `ObjectInverseOf(isParentOf)` is an object property expression.

Data Ranges: All datatypes, denoted `DT`, are data ranges, denoted `DR`. OWL 2 has five ways to form data ranges from datatypes, one of which is formed using constraining facets and, so, is omitted here: `DataIntersectionOf(`$\mathtt{DR}_1$ `...` $\mathtt{DR}_\mathtt{n}$`)` (i.e. the intersection of `n` data ranges), `DataUnionOf(`$\mathtt{DR}_1$ `...` $\mathtt{DR}_\mathtt{n}$`)`, `DataComplementOf(DR)`, and `DataOneOf(`$\mathtt{lt}_1$ `...` $\mathtt{lt}_\mathtt{n}$`)` (this is the set containing literals $\mathtt{lt}_1$ to $\mathtt{lt}_\mathtt{n}$). Given the datatypes `Integer` and `String` we can form, for example, `DataUnionOf(Integer String)` and `DataComplementOf(Integer)`. We may also wish to form a set of specific integers, such as `DataOneOf(1 2 3 4 5)`, or strings, say `DataOneOf(Male Female)`.

Class Expressions: All classes, often denoted by `C`, are class expressions, denoted `CE`. OWL 2 has a large range of constructors that can be used to form complex class expressions. The most straightforward of these are: `ObjectIntersectionOf(`$\mathtt{CE}_1$ `...` $\mathtt{CE}_\mathtt{n}$`)` (i.e. the intersection of `n` class expressions), `ObjectUnionOf(`$\mathtt{CE}_1$ `...` $\mathtt{CE}_\mathtt{n}$`)`, `ObjectComplementOf(CE)`, and `ObjectOneOf(`$\mathtt{a}_1$ `...` $\mathtt{a}_\mathtt{n}$`)` (this is the set containing individuals $\mathtt{a}_1$ to $\mathtt{a}_\mathtt{n}$). For example, given the classes `Mammal`, `EggLayer` and `Platypus`, we can form `ObjectIntersectionOf(Mammal EggLayer)`, which will subsume `Platypus`, `ObjectUnionOf(Mammal EggLayer)` and `ObjectComplementOf(Mammal)`. The class expression `ObjectHasSelf(OPE)` is the set of all object domain elements that are related to themselves under `OPE` (i.e. the elements that give rise to the smallest reflexive subset of `OPE`).

A further 18 styles of class expression can be formed, nine of which involve object properties and either class expressions or individuals. `ObjectSomeValuesFrom(OPE CE)` is the set of all object domain elements that are related to some element of `CE` under `OPE`. `ObjectAllValuesFrom(OPE CE)` is the set of all object domain elements that are related to only elements of `CE` under `OPE`. For example, given the object property `isParentOf`, we may wish to form the set of parents of mammals (i.e. all of the things that are the parent of at least one mammal): `ObjectSomeValuesFrom(isParentOf Mammal)`. The set `ObjectAllValuesFrom(isParentOf`

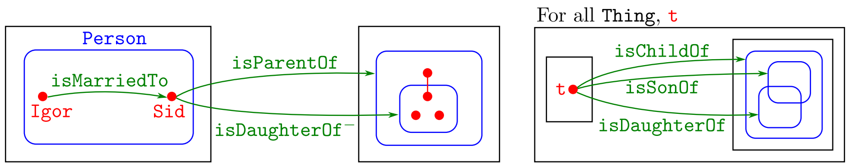

Fig. 2. Two concept diagrams for expressing OWL 2 axioms.

Mammals) comprises the individuals that are the parent of only mammals; this includes those individuals that are not a parent of anything.

ObjectHasValue(OPE a) is the set of all object domain elements that are related to individual a under OPE. So, ObjectHasValue(isParentOf Tim) is the set of Tim's parents. ObjectMinCardinality(n OPE) and ObjectMinCardinality(n OPE CE) are the sets of all object domain elements that are related to at least n elements under OPE and, in the latter case, these n elements are in CE. Four more axioms are similar, replacing Min with either Max or Exact. For example, ObjectExactCardinality(2 isParentOf Mammal) is the set of things that are parents of exactly two mammals. The last nine class expressions are similar to the nine just given, where Data, lt (a literal), DR (a data range) and DPE (a data property) are substituted for Object, a (an individual), CE (a class expression) and OPE (an object property expression), respectively.

B. Axioms

Axioms are formed from object property expressions, class expressions, and data ranges.

Class Expression Axioms SubClassOf(CE_1 CE_2) and EquivalentClasses(CE_1 CE_2) are used to assert a subset or an equality relationship between classes. DisjointClasses(CE_1 ... CE_n) asserts that the classes CE_1 to CE_n are pairwise disjoint. DisjointUnion(C CE_1 ... CE_n) expresses that C is the union of CE_1 ... CE_n which, in turn, are pairwise disjoint.

For example, we can express that all platypuses are egg-laying mammals and that all insects have at least two legs: SubClassOf(Platypus ObjectIntersectionOf(Mammal EggLayer)) and SubClassOf(Insect ObjectMinCardinality(2 hasLeg Leg)).

Object Property Expression Axioms There are 14 axioms of this kind; for space reasons we give details on seven, referring the reader to [5] for details on those omitted, which cover constraints such as being reflexive and transitive. Firstly, OWL 2 can assert that two object properties are in a subsumption relationship, equivalent, or disjoint: SubObjectProperty(OPE_1) OPE_2, EquivalentObjectProperty(OPE_1) OPE_2, and DisjointObjectProperty(OPE_1) OPE_2. To illustrate, SubObjectProperty(hasWing hasLimb) expresses that the object property hasWing is a sub-property of hasLimb.

Domain and range information for properties can be specified using ObjectPropertyDomain(OPE CE) and ObjectPropertyRange(OPE CE). Properties can also be functional, meaning that each element in the object domain is related to at most one element in the object domain, FunctionalObjectProperty(OPE). The seventh object property that we cover is the most complex: SubObjectProperty(ObjectPropertyChain(OPE_1 ... OPE_n) OPE). This object property expression axiom asserts that if OPE_1 relates y_0 to y_1, ..., and OPE_n relates y_{n-1} to y_n then OPE relates y_0 to y_n. In other words, OPE is a superset of the composition of OPE_0 to OPE_n.

Data Property Expression Axioms and Datatype Definitions Briefly, the first six object property expression axioms correspond to similar data property expression axioms, substituting Data for Object. Since data properties are relations between the object domain and the data domain, there are no axioms relating to constraints such as reflexivity. OWL 2 can also define datatypes using data ranges: DatatypeDefinition(DT DR).

Assertions SameIndividual(a_1 ... a_n) and DifferentIndividual(a_1 ... a_n) assert that the individuals a_1 up to a_n) are the same or, respectively, distinct. ClassAssertion(CE a) expresses that a is an element of CE. For example, ClassAssertion(Platypus Tom) express that the individual called Tom is a platypus.

ObjectPropertyAssertion(OPE a_1 a_2) (resp. NegativeObjectPropertyAssertion(OPE a_1 a_2)) asserts that a_1 is (resp. not) related to a_2 under OPE. To illustrate, ObjectPropertyAssertion(isParentOf Tom Finn) expresses that Tom is the parent of Finn. DataPropertyAssertion(DPE a_1 lt) and NegativeDataPropertyAssertion(DPE a_1 lt) are similarly defined. So, DataPropertyAssertion(hasAge Tom 3) expresses that Tom is aged 3.

III. Concept Diagrams

A brief introduction to the core syntax and semantics of concept diagrams is given here via examples. Individuals are denoted by labelled dots (or, more generally, trees), and the location of a dot indicates the set in which the individual lies. Closed curves represent sets, ie. classes in OWL. *Properties* (i.e. binary relations) are represented by arrows. Individuals, classes and properties can be named or anonymous.

Suppose that we wish to axiomatize the following: Igor and Sid are two different people; Igor is married to only Sid; Sid is a parent of at least three things, two of which

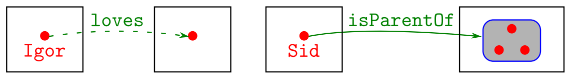

Fig. 3. Dashed arrows and shading.

are Sid's daughters; and the property `isChildOf` subsumes `isDaughterOf` and `isSonOf`. This information is captured by the two concept diagrams in Fig. 2. The lefthand diagram expresses the first three statements and includes similar syntax to Fig. 1. Here, though, we see the use of two *boundary rectangles* rather than just one. Semantically, spatial relationships only carry meaning within a common boundary rectangle and each such rectangle represents the top class, `Thing`. Therefore, whilst this diagram expresses that Sid and Igor are different people, it provides no information about whether Sid's children (identified using `isParentof`) are different from Igor, for instance. Likewise, the diagram does not assert that the three children are not people, which it would do if this configuration of curves was contained by a single rectangle. The use of multiple boundary rectangles is useful when one does not wish to assert distinctness of individuals or disjointness/subsumption relationships between classes.

Focussing on the (solid) arrows, their targets represent the set of things, or the single thing, to which the source is related under the named object property. So, the target of the `isMarriedTo` arrow represents the single thing, i.e. Sid, to which Igor is married. Likewise, the target of `isParentOf` is the set of all Sid's children. The arrow labelled `isDaughterOf`$^-$, denoting the inverse of `isDaughterOf`, builds the set of things that are Sid's daughters[1]. In general, solid arrows can also be sourced on curves (see Fig. 1): a solid arrow sourced on curve $\mathtt{C}_1$ with label `op` and target $\mathtt{C}_2$ expresses that, between them, the things in $\mathtt{C}_1$ are related to all and only the things in $\mathtt{C}_2$ under `op`.

The righthand diagram in Fig. 2 includes explicit quantification, in the form of 'For all `Thing`, `t`' written outside of the box. Here, the curve targeted by `isChildOf` encloses the target of `isSonOf`, expressing 'the set of things to which `t` is related under `isChildOf` include all of the things to which `t` is related under `isSonOf`'; informally, anything which is the son of `t` is also a child of `t`. Since `t` is any individual, we have expressed `SubObjectPropertyOf(isSonOf isChildOf)`. Similarly, `SubObjectPropertyOf(isDaughterOf isChildOf)` is also expressed, but the diagram tells us nothing about the relationship between `isSonOf` and `isDaughterOf`; to assert their disjointness, one would simply ensure that their targeted curves do not overlap.

Concept diagrams use *dashed* arrows as well as using solid arrows to represent property restrictions. Dashed arrows are used when we do not wish to express complete information about the set of things to which the source is related. For example, we may wish to express that Igor loves somebody, without identifying the set of things Igor loves: the left of Fig. 3 represents the loved anonymous individual by an unlabelled dot. The arrow connects diagrammatic syntax placed in different boxes to ensure that we have not asserted Igor loves someone different from himself. In general, dashed arrows can also be sourced on curves: a dashed arrow sourced on curve $\mathtt{C}_1$ with label `op` and target $\mathtt{C}_2$ expresses that, between them, the things in $\mathtt{C}_1$ are related to all the things in $\mathtt{C}_2$ under `op`, and *possibly* some other things. The other diagram in Fig. 3 illustrates the use of shading: in a shaded region, all things are represented by dots. So, this diagram asserts that Sid is the parent of exactly three things. Fig. 4 asserts an object property chain axiom: `SubObjectPropertyOf( ObjectPropertyChain(isParentOf isParentOf) isGrandparentOf)` using second-order quantification (although this axiom is also expressible in first-order logic) and a logical operator, implication.

The syntax of concept diagrams (not all of which has been illustrated in full here) is summarised as follows. Dots or trees visualize named or anonymous individuals, which can be joined by $=$ to assert equality. Closed curves visualize named or anonymous classes. Boundary rectangles enclose the dots, trees and closed curves. Arrows, which are dashed or solid, visualize restrictions on named or anonymous properties or their inverses; these may be sourced and targeted on boundary rectangles, dots or trees, and closed curves only. Shading is placed in regions to represent upper bounds on cardinality. Standard logical operators – negation, conjunction, disjunction, implication and bi-implication – form compound expressions. Quantification, universal or existential, is used over anonymous individuals, classes and properties. A full formalization is in [4].

IV. Targeting Concept Diagrams Towards OWL 2

As we have just seen, concept diagrams are second-order and they make explicit use of logical operators. Despite their high expressiveness, though, they fail to support literals, datatypes and data properties. The purpose of this section is to identify a fragment of concept diagrams that closely aligns with OWL 2.

A. *Supporting Literals, Datatypes and Data Properties*

We start by noting that OWL 2 cannot make assertions about the relationships (i.e. subsumption, disjointness and equivalence) between datatypes (or, generally, data ranges) and classes. Thus, we extend concept diagrams to permit the use of datatypes as labels for curves provided they do not appear within a boundary rectangle containing classes or individuals. Literals are represented similarly to individuals, but must be placed inside boundary rectangles containing datatypes. We also allow the use of arrows labelled by data properties provided the source (resp. target) is within a boundary rectangle containing classes or individuals (resp. datatypes or literals). An example is given in Fig. 5, which expresses:

- `ClassAssertion(ObjectIntersectionOf(`$\mathtt{C}_1$ $\mathtt{C}_2$`) i)`,

[1]A more elegant modelling solution would introduce a new property, say `hasDaughter`, defined to be the inverse of `isDaughterOf`.

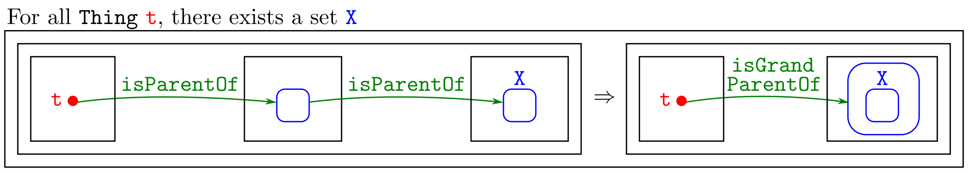

Fig. 4. Second-order quantification and connectives between diagrams.

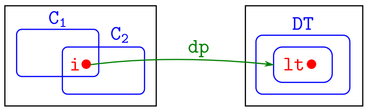

Fig. 5. Datatypes and data properties.

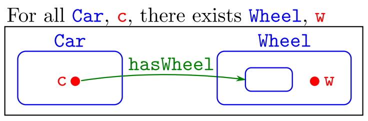

Fig. 6. First-order quantification.

- `SubClassOf(ObjectOneOf(i) DataAllValuesFrom(dp DT))`, and
- `DataPropertyAssertion(dp i lt)`.

B. A Fragment of Concept Diagrams for OWL 2

Concept diagrams, prior to the inclusion of literals, datatypes and data properties, are expressively equivalent to a well-defined fragment of dyadic second-order logic [4]. In particular, they can freely quantify over sets and binary relations (as well as elements), going well beyond the capabilities of OWL 2. Even the first-order quantification aspect leads to over-expressivity issues. For instance, the diagram in Fig. 6 involves alternating quantifiers, expressing that for every car there is a wheel that is not a wheel of the car, which is not expressible in OWL 2; however, the similar statement 'there is a wheel that is not the wheel of any car' is OWL 2 expressible. The main point, though, is that the first-order fragment of concept diagrams allows arbitrary alternations of quantifiers, leading to high expressivity beyond the limits of OWL 2.

The use of logical operators also allows the creation of diagrams that are inexpressible in OWL 2. For instance, Fig. 7 expresses that if there is exactly one car then there are exactly four wheels, using ⇒. The

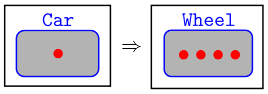

Fig. 7. Using connectives in a non-OWL-like fashion.

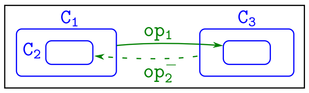

Fig. 8. A diagram with no explicit quantification and no connectives.

lefthand diagram expresses `EquivalentClasses(Car ObjectOneOf(_:x`$_1$`))`. The righthand diagram expresses `EquivalentClasses(Wheel ObjectOneOf(_:x`$_2$` _:x`$_3$` _:x`$_4$` _:x`$_5$`))` and `DifferentIndividuals(_:x`$_2$` _:x`$_3$` _:x`$_4$` _:x`$_5$`)`. Whilst the concept diagram uses ⇒, OWL 2 does not have free use of logical operators and cannot use them in the standard (inductive) way. This means that we cannot simply join the above two OWL axioms with ⇒ to form the expression made by the diagram. Given the discussion so far, we therefore place the following restrictions on the syntax of concept diagrams: prohibit the use of quantifiers, both first-order and second-order, and prohibit the use of logical operators. Of particular note is that recent empirical evidence established that the use of explicit quantification in concept diagrams is detrimental to human cognition and suggested that the use of operators could be cognitively difficult [19]. Therefore, these restrictions are not only useful for achieving good alignment with OWL 2, but also yield a more usable diagrammatic notation.

Of course, placing such restrictions on the syntax is (deliberately) designed to reduce the expressive power of concept diagrams. However, these restrictions mean some important, commonly occurring, axioms can no longer be expressed or must now be expressed in cumbersome ways. Fig. 8 illustrates some of the consequences. It expresses the following: `SubClassOf(C`$_2$` C`$_1$`)`, `DisjointClasses(C`$_1$` C`$_3$`)`, `SubclassOf(C`$_1$` ObjectAllValuesFrom(op`$_1$` C`$_3$`))`, and `SubclassOf(C`$_2$` ObjectSomeValuesFrom(op`$_2$` C`$_3$`))`. The first three axioms are directly expressed, but the SomeValuesFrom restriction requires the use of inverse: the dashed arrow explicitly says 'the things in C$_2$ are each related to, by something in C$_3$, under op$_2^{-}$'. This is equivalent to saying 'the things in C$_2$ are each related to something in C$_3$, under op$_2$', the required OWL 2 axiom. Moreover, without quantifiers, concept diagrams *can no longer* make more general cardinality restrictions, such as `SubclassOf(C`$_4$` ObjectMinCardinality(2 op C`$_5$`))`; the lefthand diagram in Fig. 9 shows how quantification can be used to express this axiom.

Our solution is to introduce new arrow annotations to allow cardinality restrictions, of which some values from axioms are a special case, to be expressed without the use of inverse, as shown on the right of Fig. 9. Here, the arrow annotation, ≥ 2, asserts that *each* element of C$_4$ is related, under op, to *at least two* elements of the target set which, in this case, is a subset

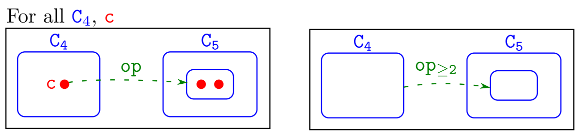

Fig. 9. Two ways of expressing `SubclassOf(C`$_4$ `ObjectMinCardinality(2 op C`$_5$`))`.

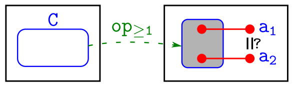

Fig. 10. `SubClassOf(C ObjectSomeValuesFrom(op ObjectOneOf(a`$_1$ `a`$_2$`)))`.

of `C`$_5$, as required. In summary, we augment the syntax of concept diagrams with arrow annotations – in particular $\leq n$, $= n$, and $\geq n$ – to support the expression of OWL 2 axioms.

Consider now the removal of logical operators. Diagrams are well-known for effectively making conjunctive statements and, through the use of spatial relations, can effectively convey negative assertions as well. However, they are often less well suited towards making disjunctive statements. In the case of concept diagrams, this has implications when we do not wish to assert the distinctness of individuals. For example, suppose we wish to assert `SubClassOf(C ObjectSomeValuesFrom(op ObjectOneOf(a`$_1$ `a`$_2$`)))`. Since, in concept diagrams, distinct dots (naturally) represent distinct individuals, with the original syntax we can only make this assertion using a disjunction of diagrams, accounting for the different possibilities concerning the equality or distinctness of `a`$_1$ and `a`$_2$; using disjunction in this way was not ideal anyway. By contrast OWL 2 adopts the standard approach of symbolic notations: the distinctness of individuals must be explicitly asserted. We therefore introduce further new syntax to concept diagrams to allow the possibility for the distinctness or equality of two individuals in the same boundary rectangle to be unspecified. Fig. 10 shows an example.

C. Summary: New Syntax for Concept Diagrams

The adapted version of concept diagrams, geared towards OWL 2, has the following syntax, with the additions shown in italics. Dots or trees visualize named or anonymous individuals *or literals*, which can be joined by $=$ to assert equality *which is augmented with ? if equality or distinctness is unknown*. Closed curves visualize named or anonymous classes *or datatypes*. Within a boundary rectangle, only (a) individuals, classes and object properties or their inverses, or (b) *literals and datatypes* can be visualized. Each boundary rectangle is either a *class rectangle* or a *data rectangle*, as determined by its contents in the obvious way. Arrows, which are dashed or solid, visualize restrictions on named object properties, their inverses, *or data properties*. Arrows labelled by an object property or its inverse must be sourced and targeted on a class rectangle or syntax within such a rectangle. *Arrows labelled by a data property must be sourced (resp. targeted) on a class (resp. data) rectangle, or syntax within such a rectangle. Arrows can, optionally, be annotated with cardinality constraints of the form $\leq n$, $= n$, or $\geq n$.* Shading can be placed in regions.

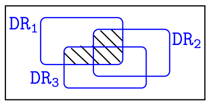

Fig. 11. Data ranges.

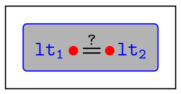

Fig. 12. Constructing data ranges from literals.

V. Relationship to OWL 2

We systematically consider the OWL 2 syntax, as given in [5], comparing its capabilities with concept diagrams as proposed in section IV-C. First we establish how to visualize OWL 2 expressions, then we consider the visualization of OWL 2 axioms. However, building explicit representations of, say, class expressions, is often not necessary when visualizing OWL 2 axioms using concept diagrams. This is important to note since, in what follows, some of the diagrammatic constructions for class expressions are cumbersome whereas the representation of axioms involving them can be considerably more succinct; we will return to this point during our discussions below. We start by observing that we have access to `ObjectInverseOf` through the use of $^-$. Further, we have not currently considered support for constraining facets or keys. Object and data property expression axioms are not well-supported by concept diagrams without access to quantifiers, so we focus on these in section VI.

A. Data Ranges

OWL 2 has four different constructors for data ranges, excluding one that employs constraining facets: `DataIntersectionOf(DR`$_1$ `... DR`$_n$`)`, `DataUnionOf(DR`$_1$ `... DR`$_n$`)`, `DataComplementOf(DR)` and `DataOneOf(lt`$_1$ `... lt`$_n$`)`. These are all comprehensively supported by concept diagrams. In the first three cases, one can construct a concept diagram containing the datatypes used to build any given data range, `DR`. Then there is a region in the diagram corresponding to `DR`. In the fourth case, a diagram is built containing the literals inside an unlabelled, shaded curve. Fig. 11 builds `DataIntersectionOf(DR`$_1$ `DataUnionOf(DR`$_2$ `DR`$_3$`))`; the lines highlight the required region (a convention we adopt to indicate the region of interest). Fig. 12 shows `DataOneOf(lt`$_1$ `lt`$_2$`)`.

B. Class Expressions

Many OWL 2 class expressions can be visualized by regions in concept diagrams, sometimes in much the same way as for data ranges. As these OWL 2 expressions are built inductively, given an expression that includes class expression `CE`, we determine whether any given expression can be visualized under the assumption that `CE` can be visualized. Assuming we can build a diagram containing regions for `CE`$_1$ ... `CE`$_n$, then we can also build

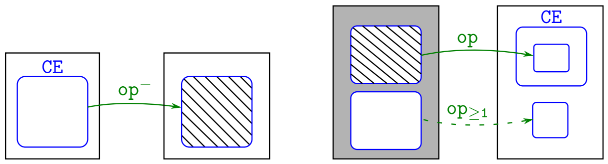

Fig. 13. `ObjectSomeValuesFrom(op CE)` (LHS) and `ObjectAllValuesFrom(op CE)`.

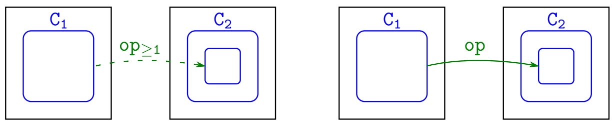

Fig. 14. Visualizing `SubClassOf(C`$_1$ `ObjectSomeValuesFrom(op C`$_2$`))` (LHS) and `SubClassOf(C`$_1$ `ObjectAllValuesFrom(op C`$_2$`))`.

diagrams that visualize `ObjectIntersectionOf(CE`$_1$ `... CE`$_n$`)`, `ObjectUnionOf(CE`$_1$ `... CE`$_n$`)`, and `ObjectComplementOf(CE)` similar to the illustration for data ranges in Fig. 11. Likewise, `ObjectOneOf(a`$_1$ `... a`$_n$`)` can be visualized similarly to Fig. 12.

The statements `ObjectSomeValuesFrom(op CE)` and `ObjectAllValuesFrom(op CE)` have constructions given in Fig. 13, where `CE` is a class, which are not particularly effective from the perspective of cognition. However, these are classic examples of where the construction is typically not required when creating OWL 2 axioms; see Fig. 14. Constructions for the remaining class axioms, with the exception of `ObjectHasSelf(OPE)` which is not supported, are similar to these examples and are omitted for space reasons. In summary, we can build concept diagrams to represent class expressions provided they do not involve `ObjectHasSelf(OPE)` or `DatatypeRestriction(DT F`$_1$ `lt`$_1$ `... F`$_n$ `lt`$_n$`)`.

C. Class Expression Axioms

There are only four kinds of class expression axioms: `SubClassOf(CE`$_1$ `CE`$_2$`)`, `EquivalentClass(CE`$_1$ `... CE`$_n$`)`, `DisjointClasses(CE`$_1$ `... CE`$_n$`)` and `DisjointUnion(CE`$_1$ `CE`$_n$`)`. If we wish merely to establish whether there exists a concept diagram for any given OWL 2 axiom, we can build directly on the insights revealed in the previous subsection: provided we can build diagrams representing the class expressions in question, the axiom can be visualized. Fig. 14 illustrates subclass axioms and Fig. 15 gives examples for the other styles – `EquivalentClasses(C`$_1$ `C`$_2$`)`, `DisjointClasses(SomeValuesFrom(op ObjectHasValue(a)) C)` and `DisjointUnion(C`$_1$ `ObjectSomeValuesFrom(op`$_1$ `C`$_2$`) ObjectSomeValuesFrom(op`$_2$ `C`$_3$`))` – where the axioms are expressed by building diagrammatic components for the class expressions involved; as previously indicated, these need not be the most effective diagrams for these axioms, but are merely shown to illustrate that the axioms can indeed be visualized.

D. Datatype Definitions

Datatype definitions can only take one form in OWL 2: `DatatypeDefinition(DT DR)`. Trivially, these can all be expressed using concept diagrams, provided the data range involved does not incorporate a constraining facet. An example for `DatatypeDefinition(DT DataUnionOf(DR`$_1$ `DR`$_2$`))` is given in Fig. 17.

E. Assertions

Similarly to datatype definitions, all OWL 2 assertions can be expressed by concept diagrams provided any class expression involved can be built diagrammatically. This observation means that six of the seven assertions can always be visualized: `SameIndividual(a`$_1$ `... a`$_n$`)`, `DifferentIndividuals(a`$_1$ `... a`$_n$`)`, `ObjectPropertyAssertion(op a`$_1$ `a`$_2$`)`, `NegativeObjectPropertyAssertion(op a`$_1$ `a`$_2$`)`, `DataProapertyAssertion(dp a lt)` and, lastly, `NegativeDataPropertyAssertion(dp a lt)`. Examples for each of these, along with the seventh style of assertion, `ClassAssertion(CE a)`, are in Fig. 18.

F. Summary

Concept diagrams, as adapted in this paper, now have a close alignment with OWL 2. They can build all data ranges that do not involve constraining facets and all class expressions that do not involve `ObjectHasSelf` or `DataTypeRestiction` expressions. Following from this, concept diagrams are capable of defining any class expression axiom, any datatype definition, and any assertion that does not involve one of the aforementioned constructions.

VI. Object and Data Property Expression Axioms

Whilst concept diagrams are now capable of defining a multitude of OWL 2 axioms, they cannot define most property and data expression axioms. *Property diagrams* are now introduced to express these axioms. They exploit a limited form of universal quantification through the use of a single variable, t. Two examples are given in Fig. 16. The lefthand diagram expresses[2] `SubObjectPropertyOf(op`$_2$ `op`$_1$`)` and `DisjointObjectProperties(op`$_1$ `op`$_3$`)`; it is also possible to see that `DisjointObjectProperties(op`$_2$ `op`$_3$`)` holds directly from the diagram. The righthand diagram expresses `ObjectPropertyRange(op`$_1$ `ObjectIntersectionOf(C`$_1$ `C`$_2$`))`.

In general, property diagrams adopt much of the basic syntax of concept diagrams, and are now informally defined. Property diagrams always include a boundary rectangle that contains the variable t, which is the source of arrows, and

[2]Formally, the diagram expresses that for each t, the set of things to which t is related under `op`$_2$ is subsumed by the similarly defined set under `op`$_1$, giving the `SubObjectPropertyOf` axiom. Likewise, a similar reading of the property diagram gives the `DisjointObjectProperties` axiom.

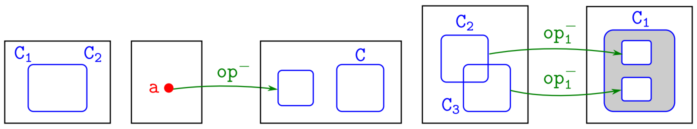

Fig. 15. Class expression axioms.

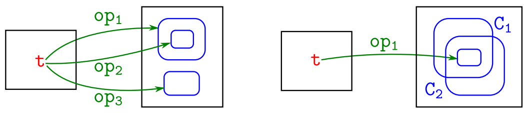

Fig. 16. Property hierarchies, disjointness and ranges.

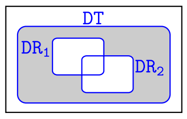

Fig. 17. Datatypes.

nothing else; this instance of t is called the *initial* t. Other boundary rectangles can include curves which are labelled or not; within any given rectangle, the labels used must be either classes or datatypes but not both. Each boundary rectangle is either a *class rectangle* or a *data rectangle*, as determined by its contents in the obvious way. Class rectangles may include at most one instance of t. Each arrow must have its source and target in different boundary rectangles. No chain of arrows can form a cycle; essentially, this means that the arrows induce a partial order on the boundary rectangles, with the rectangle containing the initial t being the least element. Arrows labelled with a data property must be sourced on the initial t. Arrows can, optionally, be annotated with ≤ 1 allowing functional axioms to be visualized.

Fig. 19 shows how to define all of the object property expression axioms. From these illustrations, it should be clear that all such OWL 2 axioms, except for domain and range, can be expressed by property diagrams. In the case of domain and range, the class expression involved needs to be formed from either named classes or anonymous classes that are not built using properties. Data property expression axioms can be similarly specified using property diagrams.

VII. Conclusion

In recognition of the effective nature of diagrams for conveying information, this paper has closely aligned concept diagrams with OWL 2, making them suitable for defining assertions and class expression axioms. They have also been extended to provide support for literals, datatypes and data properties. Recognising the importance of being able to define property expression axioms, property diagrams have been introduced for this purpose. Both concept and property diagrams are, to a considerable extent, well-matched and exhibit free-rides which are considered important features of cognitively effective diagrammatic notations. By comparing them to OWL and SOVA – an alternative ontology visualization method based on the commonly used node-link diagrams – we posit that concept and property diagrams are likely to be a relatively effective means of visualizing OWL axioms.

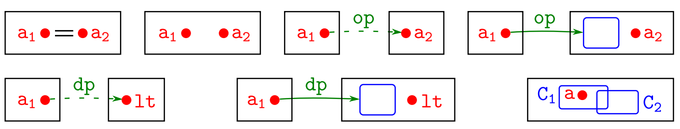

Fig. 18. Diagrams for each type of OWL 2 assertion axiom.

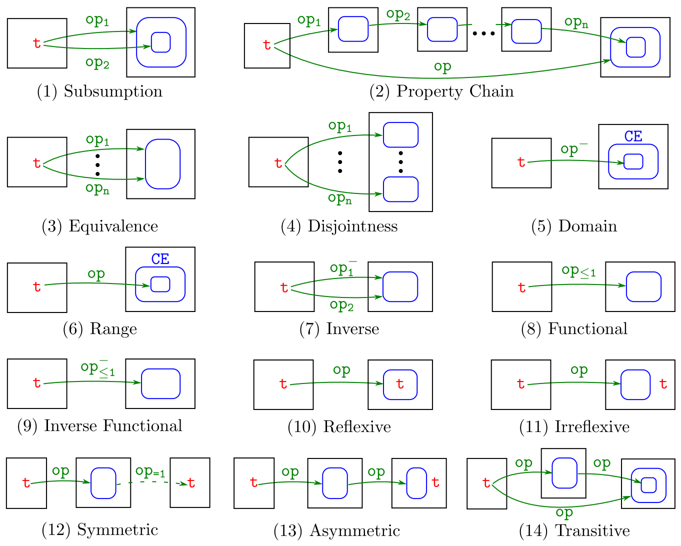

Fig. 19. Visualizing object property expression axioms.

An important aspect of making concept and property diagrams a practical ontology engineering tool is the provision of tool support. Such a tool should provide a number of features, including automated conversion from OWL to diagrams and automated conversion from diagrams to OWL, allowing the use of both notations. The former is particularly challenging, as it includes the requirement to automatically lay out concept and property diagrams. It will be important to empirically understand which layout features yield cognitively effective diagrams (e.g. avoiding crossings between arrows) and to subsequently incorporate suitable heuristics into computationally efficient layout algorithms.

Acknowledgement This research was partially funded by a Leverhulme Trust Research Project Grant (RPG-2016-082) for the project entitled Accessible Reasoning with Diagrams.

References

[1] "The Protégé web site," http://protege.stanford.edu/, 2017, accessed March 2017.

[2] "The webProtégé web site," https://protegewiki.stanford.edu/wiki/WebProtege, 2017, accessed March 2017.

[3] J. Howse, G. Stapleton, K. Taylor, and P. Chapman, "Visualizing ontologies: A case study," in *International Semantic Web Conference*. Springer, 2011, pp. 257–272.

[4] G. Stapleton, J. Howse, P. Chapman, A. Delaney, J. Burton, and I. Oliver, "Formalizing Concept Diagrams," in *Visual Languages and Computing*. Knowledge Systems Institute, 2013, pp. 182–187.

[5] "The OWL 2 Web Ontology Language," https://www.w3.org/TR/owl2-direct-semantics/, 2016, accessed Nov. 2016.

[6] A. Shimojima, *Semantic Properties of Diagrams and Their Cognitive Potentials*. Stanford, CA: CSLI Publications, 2015.

[7] G. Stapleton, M. Jamnik, and A. Shimojima, "Effective representation of information: Generalizing free rides," in *Diagrams 2016*. Springer, 2016, pp. 296–299.

[8] G. Stapleton, J. Howse, A. Bonnington, and J. Burton, "A vision for diagrammatic ontology engineering," in *VISUAL 2014*. CEUR, 2014, pp. 1–13.

[9] E. Alharbi, J. Howse, G. Stapleton, A. Hamie, and A. Touloumis, "Visual logics help people: An evaluation of diagrammatic, textual and symbolic notations," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2017.

[10] T. Hou, P. Chapman, and A. Blake, "Antipattern comprehension: an empirical evaluation." in *International Conference on Formal Ontology in Information Systems*. IOS Press, 2016, pp. 211–224.

[11] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe, "OWL Pizzas: Practical Experience of Teaching OWL-DL," in *Engineering Knowledge in the Age of the Semantic Web*, ser. LNCS, E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins, Eds. Springer, 2004, vol. 3257, ch. 5, pp. 63–81.

[12] P. Warren, P. Mulholland, T. Collins, and E. Motta, "The Usability of Description Logics," in *The Semantic Web: Trends and Challenges*, ser. LNCS, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab, and A. Tordai, Eds. Springer, 2014, vol. 8465, pp. 550–564.

[13] "Simple ontology visualization API," http://protegewiki.stanford.edu/wiki/SOVA#Download, 2016, accessed December 2016.

[14] S. Lohmann, V. Link, E. Marbach, and S. Negru, "WebVOWL: Web-based visualization of ontologies," in *Proceedings of EKAW 2014 Satellite Events*, ser. LNAI, vol. 8982. Springer, 2015, pp. 154–158.

[15] M. Sintek, "OntoViz," https://protegewiki.stanford.edu/wiki/OntoViz, accessed March 2017.

[16] M. Horridge, "OWLViz," https://protegewiki.stanford.edu/wiki/OWLViz, accessed March 2017.

[17] F. Dau and P. Eklund, "A diagrammatic reasoning system for the description logic $\mathcal{ACL}$," *Journal of Visual Languages and Computing*, vol. 19, no. 5, pp. 539–573, 2008.

[18] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou, "Ontology visualization methods a survey," *ACM Comput. Surv.*, vol. 39, no. 4, pp. 10+, Nov. 2007.

[19] E. Alharbi, J. Howse, G. Stapleton, and A. Hamie, "Evaluating diagrammatic patterns for ontology engineering," in *International Conference on the Theory and Application of Diagrams*. Springer, 2016, pp. 51–66.