An Investigation into OWL for Concrete Syntax Specification using UML Notations

Anitta Thomas[1], Aurona J. Gerber[2,3], and Alta van der Merwe[2]

[1] School of Computing, University of South Africa
The Science Campus, Florida Park, South Africa
[2] Department of Informatics, University of Pretoria
Pretoria, South Africa
[3] Center for Artificial Intelligence Research (CAIR), CSIR Meraka
Pretoria, South Africa
thomaa@unisa.ac.za,
{aurona.gerber,alta.vdm}@up.ac.za

Abstract. The Web Ontology Language OWL is a prominent ontology language for specifying ontologies. Although OWL ontologies are well-used for representing and reasoning about knowledge in various domains, they are sparsely studied for visual language specification. The work in this paper, therefore, explores OWL for visual language specification by specifying the concrete syntax of selected UML class diagram notations in an ontology. The selected diagram notations are specified as spatial configurations of primitive elements and qualitative base spatial relationships of Region Connection Calculus-8 (RCC-8). Furthermore, the automated reasoning features of ontology reasoners are investigated to verify the completeness and the correctness of the specification. The verification results indicate that the given specification needs to be revised to support applications to draw the selected notations. The value of such a specification in supporting a semantic diagram interpretation application is demonstrated using the automated instance classification feature of ontology reasoners.

Keywords: UML Notations, Concrete Syntax Specification, RCC-8, OWL, Ontology, Ontology Reasoner

1 Introduction

In Computing, diagrams are studied as elements of visual languages and an aspect studied about visual languages is its specification. In general, a visual language specification aims to capture the syntax and semantics of the relevant language [22]. Moreover a syntax specification of a visual language can focus on two different aspects; the abstract syntax and the concrete syntax, where the concrete syntax deals with the visual representation of the concepts in the abstract syntax [1, 23]. The term concrete syntax is used throughout this paper even though such syntax can also be referred to as layout syntax [9], visual syntax [31] or token syntax [16]. Although a clear distinction between the abstract and

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-42333-3_15

the concrete syntax or the syntax and the semantics is not applicable to all visual languages [22], in general, a visual language specification deals with one or more of these three aspects; abstract syntax, concrete syntax and semantics.

To specify a visual language symbolically, a specification technique is used. There are numerous specification techniques used for visual language specifications. Two such specification techniques are graph grammars [9, 19] and Description Logics (DLs) [10, 11]. There are many factors that influence the choice of a specification technique. Adequate expressiveness [21], computational efficiency [21], existing knowledge and expertise [22] and availability of supporting tools are a few such factors.

OWL is the World Wide Web Consortium (W3C) semantic web language for authoring ontologies [28]. OWL ontologies represent explicit domain knowledge to semantically enrich the Web. Such ontologies are widely used for knowledge representation in numerous disciplines such as biology, medicine, geography, astronomy and agriculture [24]. OWL is a machine readable form of an expressive DL and thus OWL ontologies make use of ontology reasoners [24]. An area where OWL ontologies is under-explored is the field of visual language specification. It is worthwhile to explore OWL for visual language specifications because such specifications can be utilized within the context of semantic web applications. Moreover, there exists numerous tools to support OWL ontology development, which can be leveraged for developing visual language specifications.

There are many aspects to explore about OWL as a specification technique. This paper explores two such aspects: how can the concrete syntax of diagram notations of a visual language be specified in an OWL ontology? What are the inherent features of OWL and OWL ontology reasoners that can be exploited to support visual language specifications and applications? To answer these questions, this work uses selected notations of a set of UML class diagram constructs for concrete syntax specification, verification and application. Although UML notations are only used as sample notations in this paper, they were chosen because UML is widely-known [18] and UML, in general, does not have a formal concrete syntax specification [26].

In this work, the concrete syntax specification defined in an ontology (hereafter referred to as a concrete syntax ontology) is envisaged to support technical applications that can reason about the visual structure of diagrams. An application to assist visually impaired users in deciphering a raster or a vector image of an UML diagram is an example of such an application. It should be noted that the proposed concrete syntax ontology is not for reasoning about the models represented in UML diagrams (as in [3, 18]) and thus, this work is not concerned about model translations from UML to OWL (as in [7, 33]). This study only uses UML notations to explore OWL as a specification technique and thus no comparison between OWL and other existing specification techniques is made in this work.

This work contributes to the current knowledge of visual languages in three different ways: it explores OWL as a concrete syntax specification technique, it explores OWL reasoner features to verify concrete syntax specification and to

support diagram interpretation applications and it provides a formal encoding of selected UML class diagram notations. The former two aspects make generic contribution to the area of visual language specification.

This paper is structured into eight sections. Section 2 is a background section that includes brief background information on OWL, selected UML class diagram notations, and RCC-8. Section 3 presents a brief discussion of related work. In section 4, the selected UML class diagram notations are modeled as spatial configurations of primitive elements and spatial relationships of RCC-8 along with their respective encoding in the OWL ontology. Criteria for completeness and correctness to verify the concrete syntax specification are discussed in section 5. In section 6 the concrete syntax given in section 4 is evaluated for completeness and correctness using the automated reasoning features of ontology reasoners, where applicable. In section 7 a conceptual application of the concrete syntax ontology within a diagram interpretation context is presented. Section 8 concludes with a summary and the intended value of the contributions.

2 Background

2.1 OWL

An OWL ontology models a domain using classes, properties, instances and data values [15]. A class represents a set of objects, a property describes a possible relationship between objects, instances describe the objects itself and a data value links an instance to a specific data type [14]. OWL provides a rich set of constructs such as union, intersection and negation to describe classes and characteristics such as transitivity, symmetry and reflexivity to describe properties. Due to the compositional nature of OWL, complex classes can be described using other classes in the ontology [14].

An ontology reasoner is a key aspect in working with ontologies. An ontology reasoner is used to check the correctness as well as to infer new knowledge based on what is described in the ontology. In other words, it helps in detecting inconsistencies as well as in maintaining the class hierarchies by inference based on the explicitly stated information in the ontology. The automated reasoning capabilities of an ontology reasoner are important in maintaining correct ontologies [14].

2.2 Selected UML Class Diagram Notations

UML is managed by the standards consortium Object Management Group and it has a specification, the current version being UML 2.4.1 [26, 27]. UML can be used to model both structural and behavioral features of an application [27] and it can be used in the design, analysis, implementation and documentation phases of software applications [26, 27]. Among the numerous diagrams of UML to support structural and behavioral modeling, a class diagram is used to represent a structural model of an application which typically consists of classes and the relationships among classes, using both graphical and textual notations [27].

UML class diagram constructs are specified in the Classes package of the UML 2.4.1 specification [27]. This package includes fifty six constructs that can be used to represent object oriented models using class, package and object diagrams. The set of UML constructs that are considered in this work are *Class*, *Interface*, *Association*, *Aggregation*, *Composition*, *Dependency*, *Generalization*, *Realization* and *InterfaceRealization*. Figure 1 lists these nine UML constructs and their ten notations used in this paper. Based on these selected notations all constructs except two (*Association* and *Interface*) have one notation each. Further details of these constructs including their semantics can be found in [27].

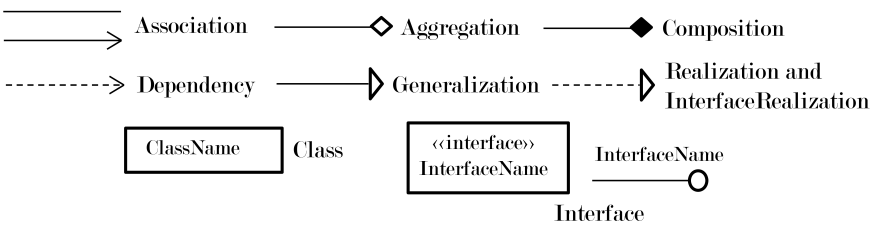

Fig. 1. Selected UML class diagram notations [27]

The constructs and notations in Figure 1 were selected because they are identified as typical constructs and notations in UML class diagrams (pages 147 to 150 in [27]). The notations depicted in Figure 1 are not the only notations for the selected nine constructs and similarly, these nine constructs are not the only constructs with notations that can be used in a class diagram. However to limit the scope of this paper a set of UML notations is selected as in Figure 1.

2.3 RCC-8

RCC-8 represents space qualitatively and it excludes numerical representation and computation of space. As one of the most commonly used calculus for qualitative spatial representation and reasoning, RCC-8 is used in areas such as Geographic Information System, engineering design, robotic navigation, biology, qualitative document structure recognition and visual language specification [4].

RCC-8 contains a set of eight base spatial relationships. These spatial relationships are disconnected (DC), externally connected (EC), partially overlapping (PO), equals to (EQ), tangential proper part of (TPP), non-tangential proper part of ($NTPP$), inverse tangential proper part of (TPP^{-1}) and inverse non-tangential proper part of ($NTPP^{-1}$) [29]. These eight relationships are mutually exclusive and exhaustive meaning that two regions in a given space satisfy exactly one of these spatial relationships [5]. Depictions of these eight spatial relationships are given in Figure 2.

OWL has been extensively studied for both spatial representation and reasoning using RCC-8 [2, 12, 20, 30]. Realizing complete RCC-8 spatial reasoning

in OWL is not straightforward due to some inherent limitations of OWL [12, 13, 20]. An example of such a limitation is the lack of support for complex role inclusion axioms required to encode the entries of the composition table of RCC-8 [12]. On the other hand, there have been successful encoding of RCC-8 reasoning in OWL using Semantic Web Rule Language [2, 20].

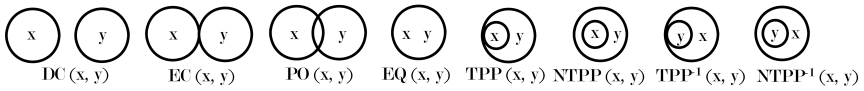

Fig. 2. Depictions of the eight spatial relations in RCC-8 [29]

3 Related Work

Visual language specification techniques can be broadly classified into grammar-based, logic-based and algebraic-based formalisms [22]. Since OWL is a logic-based ontology language, it can be categorized as a specification technique in the logical formalism. Within the logical formalism, DLs were explored for visual syntax specification [22]. For example, a general DL formalism has been previously used to specify the concrete syntax of entity-relationship (ER) diagram constructs and syntactical constructs of Pictorial Janus [22]. The concrete syntax specification of ER diagrams was then used in DL systems CLASSIC and LOOM to automate diagram reasoning to realize a syntax-directed diagram editor that can validate diagrams [11]. The concrete syntax of Pictorial Janus was used to formalize its semantics, which was also used to realize a diagram editor that verifies the semantics of diagrams of Pictorial Janus [10]. Although DL provide the logical foundations for OWL [24], the use of OWL ontologies as visual language specifications is still under-explored. As a W3C standard, OWL is the knowledge representation language to realize semantic web [28] and due to the standardization, it has extensive tool support. Thus authoring a visual language specification in OWL can make use of the existing OWL tools and such a specification is desired in potential semantic web applications.

OWL ontologies are used to represent knowledge about visual concepts such as shapes and graphical concepts. Two examples of such generic shapes ontologies are discussed in [25, 32]. Such generic ontologies do not in general encode concrete syntax of visual languages.

RCC-8 has been previously used for visual language specification [4]. RCC-8 was used to model the syntax of the visual programming language Pictorial Janus. The syntax of Pictorial Janus is presented pictorially and thus RCC-8 was used to model the spatial configurations of various syntactical constructs of this programming language. The specification technique used for specifying the visual syntax of Pictorial Janus is many-sorted logic [8]. The study in [8]

indicates that RCC-8 can indeed be used for modeling the concrete syntax of diagrammatic notations.

4 Concrete Syntax Specification of UML Notations

In this section the use of OWL to model the concrete syntax of UML class diagram notations is explored. The selected UML notations are modeled using primitive elements and spatial relationships as in [8, 10, 11, 31]. The details of the primitive elements and the spatial relationships for the selected UML notations and how they are specified in the OWL ontology are discussed in the next subsections 4.1 and 4.2. The specification of the UML notations is given in section 4.3.

4.1 Modeling and Encoding Primitive Elements

The selected UML notations are composed of nine primitive elements namely arrow, circle, filled diamond, unfilled diamond, line, dotted line, rectangle, string and triangle. These nine primitive elements are depicted in Figure 3.

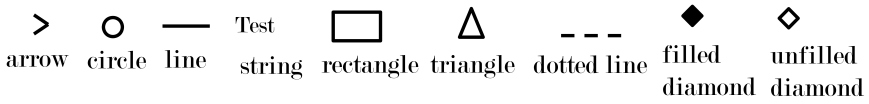

Fig. 3. Primitive elements of the selected UML notations

These nine primitive elements are specified as nine OWL primitive classes in the ontology. The class names representing these primitive elements are *Arrow*, *Circle*, *DiamondFilled*, *DiamondUnfilled*, *Line*, *DottedLine*, *Rectangle*, *String* and *Triangle*. These nine classes are added as subclasses of an OWL class *Primitives*. The modeling approach used to model the primitive elements can be classified as the non-attributed representation of graphical objects as stated in the visual language literature [6, 21].

4.2 Modeling and Encoding Spatial Relationships

Eight spatial relationships in RCC-8 are added as object properties in the OWL ontology. The spatial relationships DC, EC, PO, EQ, TPP, $NTPP$, TPP^{-1}, $NTPP^{-1}$ are encoded in the concrete syntax ontology as object properties *isDisconnectedTo*, *isExternallyConnectedTo*, *isPartiallyOverlappingTo*, *isEqualTo*, *isTangentialProperPartOf*, *isNonTangentialProperPartOf*, *hasTangentialProperPartOf* and *hasNonTangentialProperPartOf* respectively. The modeling approach used to describe the spatial relationships can be seen as the connection-based representation [6].

In this work RCC8 is primarily used for spatial representation. Even though realizing spatial reasoning in RCC-8 is not the primary focus of this work, some semantics of RCC-8 spatial relationships is captured using OWL object property characteristics mentioned in section 2.1. Specifically this work uses the symmetric object property characteristic to encode the fact that *isDisconnectedTo*, *isExternallyConnectedTo*, *isEqualTo* and *isPartiallyOverlappingTo* are symmetrical, and *hasTangentialProperPartOf* and *hasNonTangentialProperPartOf* are inverse of *isTangentialProperPartOf* and *isNonTangentialProperPartOf* respectively [2].

4.3 Specification of UML Class Diagram Notations

The nine UML constructs are modeled as nine OWL classes namely *UMLClass*, *Interface*, *Association*, *Aggregation*, *Composition*, *Dependency*, *Generalization*, *Realization* and *InterfaceRealization* in the ontology. The concrete syntax of the selected notations is then specified in the respective OWL class definitions. These nine OWL classes are included as subclasses of *UMLConstructs*, a sibling class of *Primitives* (mentioned in section 4.1).

Class A *Class* is represented using a rectangle and a string inside the rectangle where the string indicates the class name. The spatial configuration of the selected notation of *Class* can then be modeled using rectangle, string and RCC-8. This spatial configuration is specified in the ontology as:

```
Class: UMLClass
EquivalentTo: Rectangle and
(hasNonTangentialProperPartOf exactly 1 String)
SubClassOf: UMLConstructs
```

Interface One notation of *Interface* uses a rectangle and two strings inside the rectangle and the second one uses a line, a circle and a string. Thus the spatial configurations of *Interface* can be modeled using rectangle, line, circle, string and RCC-8, which is specified in the ontology as:

```
Class: Interface
EquivalentTo: Rectangle and
(hasNonTangentialProperPartOf exactly 2 String),
Line and (isDisconnectedTo some String)
and (isExternallyConnectedTo some Circle)
SubClassOf: UMLConstructs
```

Association An *Association* is a link between two *Class*es, which is represented using a line and an arrow, or a line. The spatial configurations of *Association* can thus be modeled using a line, an arrow and two classes as:

```
Class: Association
EquivalentTo: Line and (isExternallyConnectedTo some UMLClass)
and (isExternallyConnectedTo some (Arrow
and (isExternallyConnectedTo some UMLClass))),
```

```
Line and (isExternallyConnectedTo min 2 UMLClass)
SubClassOf: UMLConstructs
```

Aggregation An *Aggregation* connects two *Class*es using a line and an unfilled diamond. The spatial configuration of *Association* is thus specified as:

```
Class: Aggregation
EquivalentTo: Line and (isExternallyConnectedTo some UMLClass)
and (isExternallyConnectedTo some (DiamondUnfilled
and (isExternallyConnectedTo some UMLClass)))
SubClassOf: UMLConstructs
```

Composition A *Composition* is a relationship between two *Class*es represented using a line and a filled diamond. The spatial configuration of *Composition* is thus specified as:

```
Class: Composition
EquivalentTo: Line and (isExternallyConnectedTo some UMLClass)
and (isExternallyConnectedTo some (DiamondFilled
and (isExternallyConnectedTo some UMLClass)))
SubClassOf: UMLConstructs
```

Dependency A *Dependency* is indicated using a dotted line and an arrow between two *Class*es. Therefore the spatial configuration of *Dependency* is specified as:

```
Class: Dependency
EquivalentTo: LineDotted and (isExternallyConnectedTo
some UMLClass) and (isExternallyConnectedTo
some (Arrow and (isExternallyConnectedTo some UMLClass)))
SubClassOf: UMLConstructs
```

Generalization A *Generalization* is indicated using a line and a triangle between two *Class*es. Therefore the spatial configuration of *Generalization* is specified as:

```
Class: Generalization
EquivalentTo: Line and (isExternallyConnectedTo some UMLClass)
and (isExternallyConnectedTo some (Triangle
and (isExternallyConnectedTo some UMLClass)))
SubClassOf: UMLConstructs
```

Realization A *Realization* is indicated between two *Class*es using a dotted line and a triangle. Therefore the spatial configuration of *Realization* is specified as:

```
Class: Realization
EquivalentTo: LineDotted and (isExternallyConnectedTo
some UMLClass) and (isExternallyConnectedTo
some (Triangle and (isExternallyConnectedTo some UMLClass)))
SubClassOf: UMLConstructs
```

InterfaceRealization An *InterfaceRealization* is indicated between a *Class* and an *Interface* using a dotted line and a triangle. Therefore the spatial configuration of *InterfaceRealization* is specified as:

```
Class: InterfaceRealization
EquivalentTo: LineDotted and (isExternallyConnectedTo
some UMLClass) and (isExternallyConnectedTo
some (Triangle and (isExternallyConnectedTo some Interface)))
SubClassOf: UMLConstructs
```

5 Verification Criteria for Concrete Syntax Specification

In this section criteria to verify the concrete syntax specification given in section 4.3 are discussed. The notations depicted in Figure 1 are distinct, which means that a notation should only map to one UML construct. However the mapping of a UML construct to a unique notation is only valid for seven of these constructs as *Interface* and *Association* have two notations each. Even though the notations for *InterfaceRealization* and *Realization* are the same, the former UML construct is a link between an *Interface* and a *Class* but the latter connects two *Class*es, thus resulting in two distinct spatial configurations. The following two general criteria are identified to verify the given concrete syntax specification in section 4.3:

Completeness: A specification is complete if each UML construct has an encoded concrete syntax in the concrete syntax ontology. This definition of completeness is adapted from [1].

Correctness: A specification is correct if an encoded spatial configuration maps exactly to one UML construct represented in the concrete syntax ontology. This definition of correctness is also adapted from [1]. Another correctness criterion is to check whether the encoded spatial configurations in the ontology always lead to acceptable notations. This criterion evaluates whether the spatial configurations of the notations have been modeled correctly.

6 Evaluation of the Concrete Syntax Specification

In this section the use of the automated reasoning features of ontology reasoners is explored to verify the concrete syntax specification given in section 4.3 according to the criteria given in section 5. The concrete syntax specification given in section 4.3 is complete based on the fact that the OWL classes representing UML constructs are defined classes. i.e. these nine OWL classes have definitions that encode the concrete syntax of the relevant notations.

To ensure that the concrete syntax specification is correct based on the first criterion for correctness, the automated reasoning feature of the ontology reasoner can be utilized. Specifically the OWL class feature *disjointness* can be used to ensure that each notation is encoded distinctly, which ensures that a notation maps uniquely to a UML construct. When two OWL classes are disjoint

the sets of objects represented by these classes are disjoint. Class disjointness is not a reasoner service but an OWL class descriptor that needs to be explicitly stated for the classes in an ontology [15]. If the reasoner, however, infers that two classes cannot be disjoint based on the explicit and inferred knowledge, then it will be highlighted as a contradiction in the ontology.

To demonstrate that class disjointness ensure unique spatial configuration for each of the OWL classes presented in section 4.3, two separate cases are considered. The first case is for the OWL classes (examples: *UMLClass* and *Interface*) that are defined as specializations of the same OWL class (example: *Rectangle*) representing a primitive element. The second case is for OWL classes that are defined as specializations of different OWL classes (examples: *UMLClass* and *Association*) representing different primitive elements (examples: *Rectangle* and *Line*).

Case 1: Consider three OWL classes $C1$, $C2$ and P in the ontology where $P \sqsubseteq Primitives$, $C1 \sqsubseteq UMLConstructs$, $C2 \sqsubseteq UMLConstructs$. $C1$ and $C2$ are two different classes representing two separate UML constructs in the ontology in section 4.3. If *C1* and *C2* are defined as specializations of P, then $C1 \sqsubseteq P$ and $C2 \sqsubseteq P$. Since $C1$ and $C2$ are defined using P, they are expressed as $C1 \equiv P \sqcap (exp1)$ and $C2 \equiv P \sqcap (exp2)$. For $C1$ and $C2$ to be disjoint, the sets of objects represented by $P \sqcap (exp1)$ and $P \sqcap (exp2)$ do not have objects in common. Since both $C1$ and $C2$ are defined as specializations of P, the expressions $exp1$ and $exp2$ must represent distinct spatial configurations.

Case 2: Consider four OWL classes $C1$, $C2$, $P1$ and $P2$ in the ontology where $P1 \sqsubseteq Primitives$, $P2 \sqsubseteq Primitives$, $C1 \sqsubseteq UMLConstructs$, $C2 \sqsubseteq UMLConstructs$. $C1$ and $C2$ are two different classes representing two separate UML constructs in the ontology in section 4.3. If *C1* and *C2* are defined as specializations of $P1$ and $P2$ respectively, then $C1 \sqsubseteq P1$ and $C2 \sqsubseteq P2$. Since $C1$ and $C2$ are specializations of $P1$ and $P2$ respectively, they are expressed as $C1 \equiv P1 \sqcap (exp1)$ and $C2 \equiv P2 \sqcap (exp2)$. For $C1$ and $C2$ to be disjoint, the sets of objects represented by $P1 \sqcap (exp1)$ and $P2 \sqcap (exp2)$ do not have objects in common. If $P1$ and $P2$ are disjoint, then $C1$ and $C2$ represent distinct spatial configurations provided that $C1$ and $C2$ cannot be defined as specializations of $P2$ and $P1$ respectively.

To check the correctness of the concrete syntax specification, the eight spatial relationships were defined distinct from one another, eight of the subclasses of the OWL class *Primitives* were marked *disjoint* from one another and the nine OWL subclasses of *UMLConstructs* representing the nine UML constructs were also marked *disjoint* from one another. Moreover, selected OWL classes defined under *UMLConstructs* and *Primitives* were also made disjoint from one another. For instance, the OWL class *UMLClass* is made disjoint from every subclass of *Primitives* except the OWL class *Rectangle*, as *UMLClass* is defined as a *Rectangle*, which means that an instance of *Rectangle* can be an instance of *UMLClass* as well. Invoking the reasoner after including the disjoint class descriptor for the OWL classes as stated above does not indicate any contradiction in the concrete syntax ontology in section 4.3. In the absence of a contradiction the encoded

spatial configurations in the concrete syntax ontology given in section 4.3 must be distinct.

The second criterion of correctness can be checked by counter-examples of notations that satisfy the spatial configuration specified in the ontology but differs from the intended notation. For example, Figure 4 demonstrates an incorrect notation of *Interface*, which satisfy the spatial configuration specified for this construct in the concrete syntax ontology. However the notation in Figure 4 cannot be considered as an interface as the string representing the interface name is not placed 'near' to the line and circle.

Fig. 4. Incorrect notation for *Interface*

The fact that the visual syntax specification in section 4.3 does not satisfy one of the criteria discussed in section 5 indicates that the specification is not suitable to be used in technical applications designed to validate and draw UML notations. This unsuitability is because of the fact that the given specification is not precise enough. To be useful in such applications, the current specification of UML notations needs to be revised. For example, to specify that the string that represents the interface name should be placed 'near' the circle and line requires a spatial relationship that cannot be expressed using RCC-8 [22]. However, as demonstrated in the next section, the given concrete syntax ontology can be used in an application that interprets a *valid* UML class diagram by mapping lower level graphical data to higher-level semantic UML concepts. A *valid* class diagram in this context refers to a diagram that uses only the notations depicted in Figure 1.

7 Concrete Syntax Ontology to Support Diagram Interpretation

Automated diagram interpretation can be seen as a form of image interpretation [11]. The process of extracting higher-level semantic concepts from low-level graphical data in image interpretation [17] is also applicable in an automated diagram interpretation application. The concrete syntax ontology presented in this paper can be used for inferring UML constructs using the automated reasoning feature, *instance checking*, of the ontology reasoners provided that a class diagram is described using the primitive elements and the spatial relationships [22] given in sections 4.1 and 4.2.

Instance checking is an automated reasoner service [15], which makes use of the explicitly stated and inferred information about the classes and instances in an ontology to determine whether an instance belongs to a class. a is inferred to

be an instance of class A if a satisfies the class descriptions of A [15]. Instance checking can be utilized in a diagram interpretation application where the UML constructs need to be interpreted from a valid class diagram.

Consider the UML class diagram in Figure 5(a), which depicts four *Class*es (A, B, C, D), one *Generalization* (between A and B), one *Aggregation* (between B and C) and one *Association* (between C and D). To utilize instance checking to interpret the class diagram in Figure 5(a), fourteen instances, which represent fourteen primitive elements, along with their OWL object properties, representing the spatial relationships were encoded in the ontology. Given below is a sample entry in the ontology for one of these fourteen instances, the string A in Figure 5(a), which is encoded as an instance named a of OWL class *String*. For ease of reference Figure 5(b) annotates the UML class diagram in Figure 5(a) with unique identifiers for the primitive elements used in the ontology.

```
Individual: a
Types: String
Facts: isDisconnectedTo  lineAB, isDisconnectedTo  lineDC,
isDisconnectedTo  rectC, isDisconnectedTo  d,
isDisconnectedTo  diamond, isDisconnectedTo  lineBC,
isDisconnectedTo  c, isDisconnectedTo  rectB,
isDisconnectedTo  triangle, isEqualTo  a, isDisconnectedTo  arrow,
isDisconnectedTo  b, isDisconnectedTo  rectD
isNonTangentialProperPartOf  rectA
```

Not all primitive elements have to be encoded in such detail (as listed above for *String* instance a) as the reasoner can infer many spatial relationships between instances. For example, based on the description of instance a of the OWL primitive class *String*, the reasoner will infer that *lineAB*, *lineDC*, *rectC*, *d*, *diamond*, *lineBC*, *c*, *rectB*, *triangle*, *arrow*, *b* and *rectD* have *isDisconnectedTo* object relationship with a because *isDisconnectedTo* is described to be symmetric in the concrete syntax ontology (section 4.2). Currently the encoding of instances (representing primitives) is done manually, however, ideally it should be done using a software tool.

When the reasoner is invoked with these fourteen instances in the concrete syntax ontology, it correctly identifies four instances of *UMLClass*, one instance of *Association*, one instance of *Aggregation* and one instance of *Generalization*. The correct interpretation of the UML constructs for the class diagram in Figure 5(a) indicates that the concrete syntax ontology is a feasible technique to support technical applications to realize diagram interpretation.

Admittedly, the class diagram given in Figure 5(a) is rather simple using only a subset of notations described in section 2.2. To deal with complex class diagrams, more UML constructs and notations have to be incorporated in the OWL ontology.

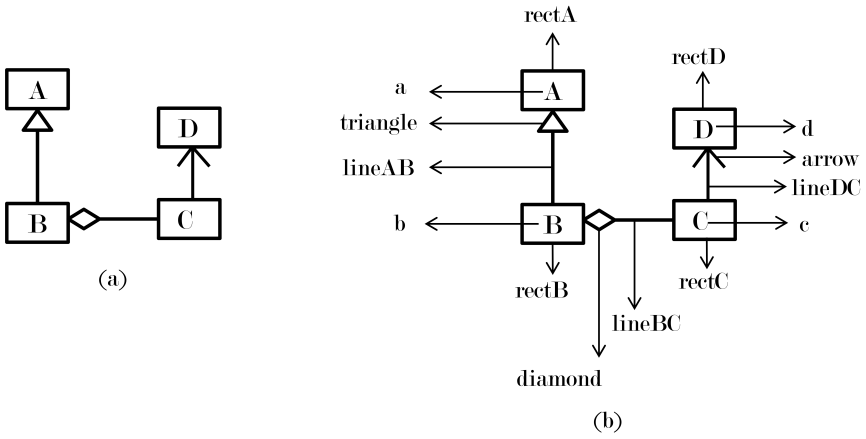

Fig. 5. (a) A sample UML class diagram (b) UML class diagram in (a) annotated with identifiers used in the concrete syntax ontology

8 Conclusion

In this work, the concrete syntax of selected UML notations were modeled using the base spatial relationships of RCC-8. The spatial configurations of the UML notations were then specified in an ontology, where an OWL class is defined for each UML construct. Furthermore criteria for verifying the completeness and correctness of the visual syntax specification were presented. This work also demonstrates how the automated reasoning features of the ontology reasoners can be leveraged to verify a concrete syntax specification. The verification results indicate that the given concrete syntax specification is not precise enough to be used in all possible diagram processing applications. Nevertheless, how a concrete syntax ontology can be used within a diagram interpretation application is discussed.

Given our results, it can be concluded that an OWL ontology is a feasible technique for concrete syntax specification provided that the spatial configurations are modeled using classes, properties, objects and data types as required by OWL. An advantage of using an OWL ontology is the use of ontology reasoners to verify the syntax specification and to support technical applications for diagram interpretation. Using OWL is of value because it allows the reuse of mature software artifacts including OWL ontology editors and reasoners for concrete syntax specification. Using OWL as a specification technique may also lead to the use of visual language specifications in semantic web applications.

Although this work focused on a limited number of notations, it is also of value to the general field of visual language specification. The process of modeling diagram notations, specifying the concrete syntax in an OWL ontology and utilizing the ontology reasoners as demonstrated in this work can be applied to another visual language as well.

References

1. Baar, T.: Correctly Defined Concrete Syntax for Visual Modeling Languages. In: Proceedings of the 9^{th} International Conference on Model Driven Engineering Languages and Systems. pp. 111–125. MoDELS'06, Springer-Verlag, Berlin, Heidelberg (2006)
2. Batsakis, S., Petrakis, E.G.: SOWL: A Framework for Handling Spatio-temporal Information in OWL 2.0. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) Rule-Based Reasoning, Programming, and Applications, Lecture Notes in Computer Science, vol. 6826, pp. 242–249. Springer Berlin Heidelberg (2011)
3. Berardi, D., Cali, A., Calvanese, D., Giacomo, G.D.: Reasoning on UML Class Diagrams. Artificial Intelligence 168, 70–118 (2005)
4. Cohn, A.G., Renz, J.: Qualitative Spatial Representation and Reasoning. In: Frank van Harmelen, V.L., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 551–596. Elsevier (2007)
5. Cohn, A., Bennett, B., Gooday, J., Gotts, N.: Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. GeoInformatica 1(3), 275–316 (1997)
6. Costagliola, G., Delucia, A., Orefice, S., Tortora, G.: A Framework of Syntactic Models for the Implementation of Visual Languages. In: Proceedings of IEEE Symposium on Visual Languages. pp. 58–65. IEEE (1997)
7. Gasevic, D., Djuric, D., Devedzic, V., Damjanovi, V.: Converting UML to OWL Ontologies. In: Proceedings of the 13^{th} International World Wide Web Conference on Alternate Track Papers & Posters. pp. 488–489. WWW Alt. '04, ACM, New York, NY, USA (2004)
8. Gooday, J., Cohn, A.: Visual Language Syntax and Semantics: A Spatial Logic Approach. In: Proc. AVI'96 Workshop on Theory of Visual Languages (1996)
9. Grunske, L., Winter, K., Yatapanage, N.: Defining the Abstract Syntax of Visual Languages with Advanced Graph Grammars: A Case Study based on Behavior Trees. Journal of Visual Languages & Computing 19(3), 343 – 379 (2008)
10. Haarslev, V.: Formal Semantics of Visual Languages using Spatial Reasoning. In: Proceedings of the 11^{th} IEEE International Symposium on Visual Languages. pp. 156–163 (Sep 1995)
11. Haarslev, V.: Using Description Logic for Reasoning about Diagrammatical Notations. In: L. Padgham (Ed.) Proceedings of the International Workshop on Description Logics. pp. 124–128 (1996)
12. Hogenboom, F., Borgman, B., Frasincar, F., Kaymak, U.: Spatial Knowledge Representation on the Semantic Web. In: IEEE Fourth International Conference on Semantic Computing (ICSC). pp. 252–259. IEEE (2010)
13. Hogenboom, F., Frasincar, F., Kaymak, U.: A Review of Approaches for Representing RCC8 in OWL. In: Proceedings 25^{th} ACM Symposium on Applied Computing (SAC'10). pp. 1444–1445. ACM (2010)
14. Horridge, M., Drummond, N., Jupp, S., Moulton, G., Stevens, R.: A Practical Guide to Building OWL Ontologies using Protégé 4 and CO-ODE Tools. University of Manchester (2009)
15. Horrocks, I., Parsia, B., Sattler, U.: OWL 2 Web Ontology Language: Direct Semantics (Second Edition). World Wide Web Consortium (2012)
16. Howse, J., Molina, F., Shin, S.J., Taylor, J.: Diagrammatic Representation and Inference: Second International Conference, Diagrams 2002 Callaway Gardens, GA, USA, April 18–20, 2002 Proceedings, chap. On Diagram Tokens and Types, pp. 146–160. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)

17. Hudelot, C.: Towards a Cognitive Platform for Semantic Image Interpretation; Application to the Recognition of Biological Organisms. Ph.D. thesis, University of Nice - Sophia Antipolis (2005)
18. Khan, A.H., Porres, I.: Consistency of UML Class, Object and Statechart Diagrams using Ontology Reasoners. Journal of Visual Languages & Computing 26, 42 – 65 (2015)
19. de Lara, J., Vangheluwe, H.: Defining Visual Notations and their Manipulation through Meta-Modelling and Graph Transformation. Journal of Visual Languages & Computing 15(34), 309 – 330 (2004)
20. Marc-Zwecker, S., de Bertrand de Beuvron, F., Zanni-Merk, C., Ber, F.L.: Qualitative Spatial Reasoning in RCC8 with OWL and SWRL. In: Proceedings of the Knowledge Engineering and Ontology Development Conference (KEOD 2013) (2013)
21. Marriott, K., Meyer, B.: On the Classification of Visual Languages by Grammar Hierarchies. Journal of Visual Languages & Computing 8(4), 375 – 402 (1997)
22. Marriott, K., Meyer, B., Wittenburg, K.B.: In: Marriott, K., Meyer, B. (eds.) Visual Language Theory, chap. A Survey of Visual Language Specification and Recognition, pp. 5–85. Springer-Verlag New York, Inc., New York, NY, USA (1998)
23. Minas, M.: Syntax Definition with Graphs. Electron. Notes Theor. Comput. Sci. 148(1), 19–40 (Feb 2006)
24. Motik, B., Cuenca Grau, B., Sattler, U.: Structured Objects in OWL: Representation and Reasoning. In: Proceedings of the 17^{th} International Conference on World Wide Web. pp. 555–564. WWW '08, ACM, New York, NY, USA (2008)
25. Niknam, M., Kemke, C.: Modeling Shapes and Graphics Concepts in an Ontology. In: Janna Hastings, Oliver Kutz, M.B., Borgo, S. (eds.) Proceedings of the First Interdisciplinary Workshop on SHAPES (2011)
26. Object Management Group: Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure (apr 2012)
27. Object Management Group: Information technology - Object Management Group Unified Modeling Language (OMG UML), Superstructure (apr 2012)
28. OWL 2 Web Ontology Language Document Overview (Second Edition). URL : http://www.w3.org/TR/owl2-overview/, accessed December 2, 2015
29. Renz, J.: Qualitative Spatial Reasoning with Topological Information. Springer-Verlag, Berlin, Heidelberg (2002)
30. Stocker, M., Sirin, E.: PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In: Proceedings of OWL: Experiences and Directions 2009 (OWLED 2009) (2009)
31. Thomas, A., Gerber, A.J., van der Merwe, A.: Visual Syntax of UML Class and Package Diagram Constructs as an Ontology. In: Fred, A., Dietz, J., Aveiro, D., Liu, K., Filipe, J. (eds.) Proceedings of the 7^{th} International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015). vol. 2, pp. 17–28. SCITEPRESS (2015)
32. Vasilakis, G., Garcia-Rojas, A., Papaleo, L., Catalano, C.E., Robbiano, F., Spagnuolo, M., Vavalis, M., Pitikakis, M.: A Common Ontology for Multi-Dimensional Shapes. In: Proceedings of the International Workshop on Multimedia Annotation and Retrieval enabled by Shared Ontologies. pp. 31–43 (2007)
33. Zedlitz, J., Jorke, J., Luttenberger, N.: From UML to OWL 2. In: Lukose, D., Ahmad, A., Suliman, A. (eds.) Knowledge Technology, Communications in Computer and Information Science, vol. 295, pp. 154–163. Springer Berlin Heidelberg (2012)