Visualizing and Analyzing Discrete Sets with a UML and OCL Software Design Tool

Short Paper - Software Demonstration

Martin Gogolla and Khanh-Hoang Doan

Computer Science Department, University of Bremen, Bremen, Germany
{gogolla|doankh}@informatik.uni-bremen.de

Abstract. This contribution discusses the visualization of discrete sets. With diagrams, we realize filtering set elements with particular properties, study set-theoretic operations, and exhibit set elements and their internal relationships. These techniques allow software developers to explore crucial set properties diagramatically.

1 Introduction

Modelling languages like UML (Unified Modeling Language) [5, 2], which includes the OCL (Object Constraint Language) [7, 1], allow developers to represent discrete sets of objects with so-called object diagrams. In UML tools, one can construct and manipulate such object diagrams and with this the underlying object sets. In the context of our tool USE (Uml-based Specification Environment)[1], this paper studies set visualization and set property analysis by (a) OCL queries in order to represent interesting subsets of a given set, (b) graphical representation of set theoretic operations like union or intersection, and (c) associations for representing crucial relationships between set elements.

The rest of this contribution is structured as follows. Section 2 studies OCL query selection. Section 3 focuses on set-theoretic operations. Section 4 treats internal relationships in sets determined by derived associations. All presented options are implemented in USE. Section 5 discusses very few related approaches. Section 6 ends the paper with concluding remarks and future work.

2 Set Visualization through OCL Queries

Figure 1 presents a set of integers in form of 16 `Int` objects where each object has an object identity (in the top of the object rectangle) and a unique attribute `val` showing the represented integer. The integers are randomly chosen from the interval `0..255` and are placed randomly in the object diagram. Such a situation, where an unknown set has to be traversed, occurs frequently in software development. Systematic placement of elements will be discussed later. The purpose of the following queries and selections is to explore this integer set and its properties with diagrams, however considering the basic layout as fixed.

[1] https://sourceforge.net/projects/useocl/

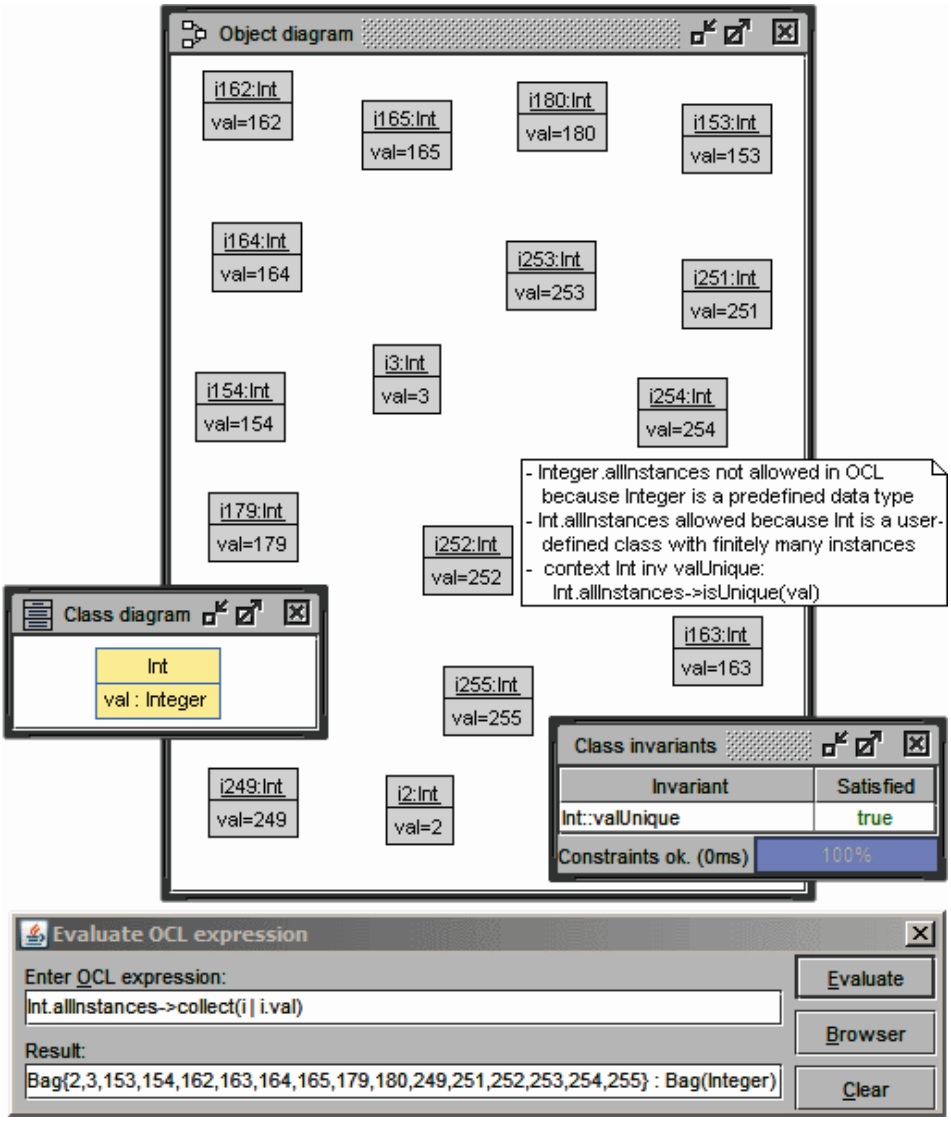

Fig. 1. Example set of integers represented with objects.

Figure 2 explains how subsets of the underlying basic set can be investigated with OCL queries and how the result can again be presented diagrammatically. The last OCL query selects those integer values which possess neighbors (predecessor and successor) in the underlying set. The result is presented in the object diagram through the dark gray objects. In order to understand the result here in the contribution better, we have indicated in the figure with light gray objects the non-selected part as well (only dark gray objects are displayed in the tool). The first two OCL queries in the right part of Fig. 2 show other OCL options, namely simple retrievals for integers with enumerations (`Set{2, 4, 8, 16, 32, 64, 128}`) or for prime numbers. Figure 2 demonstrates that in our approach sets can be selected and presented visually.

3 Set Visualization of Set-Theoretic Operations

Figure 3 shows how set-theoretic operations on the underlying set and its diagrammatic representation are achieved interactively. The first OCL query selects integers divisible by 2, and the second OCL query identifies integers divisible

Fig. 2. Subset selection through an OCL query and its visualization.

Fig. 3. Example for visual representation of set-theoretic operation union.

by 3. The union of these two sets is graphically constructed by (a) clearing the object diagram (pushing button Hide All in the first OCL window), (b) showing the integers divisible by 2 (pushing button Evaluate with option Show activated), and (c) adding the integers divisible by 3 (pushing button Evaluate also with option Show activated in the second OCL window). The result of the diagrammatically achieved result can be checked against the third OCL expression that computes the union by means of a logical disjunction. Analogously, other set-theoretic operations like intersection or difference can be performed (by activating other options like Crop or Hide).

4 Set Visualization through Derived Links

Figure 4 pictures how an internal structure of a set can be used for its diagrammatic arrangement, and a systematic object placement can be achieved. The employed internal structure of the set is in this case automatically determined by a computed, derived association between the set elements. For the previously used example integer set, the derived association identifies the neighbors

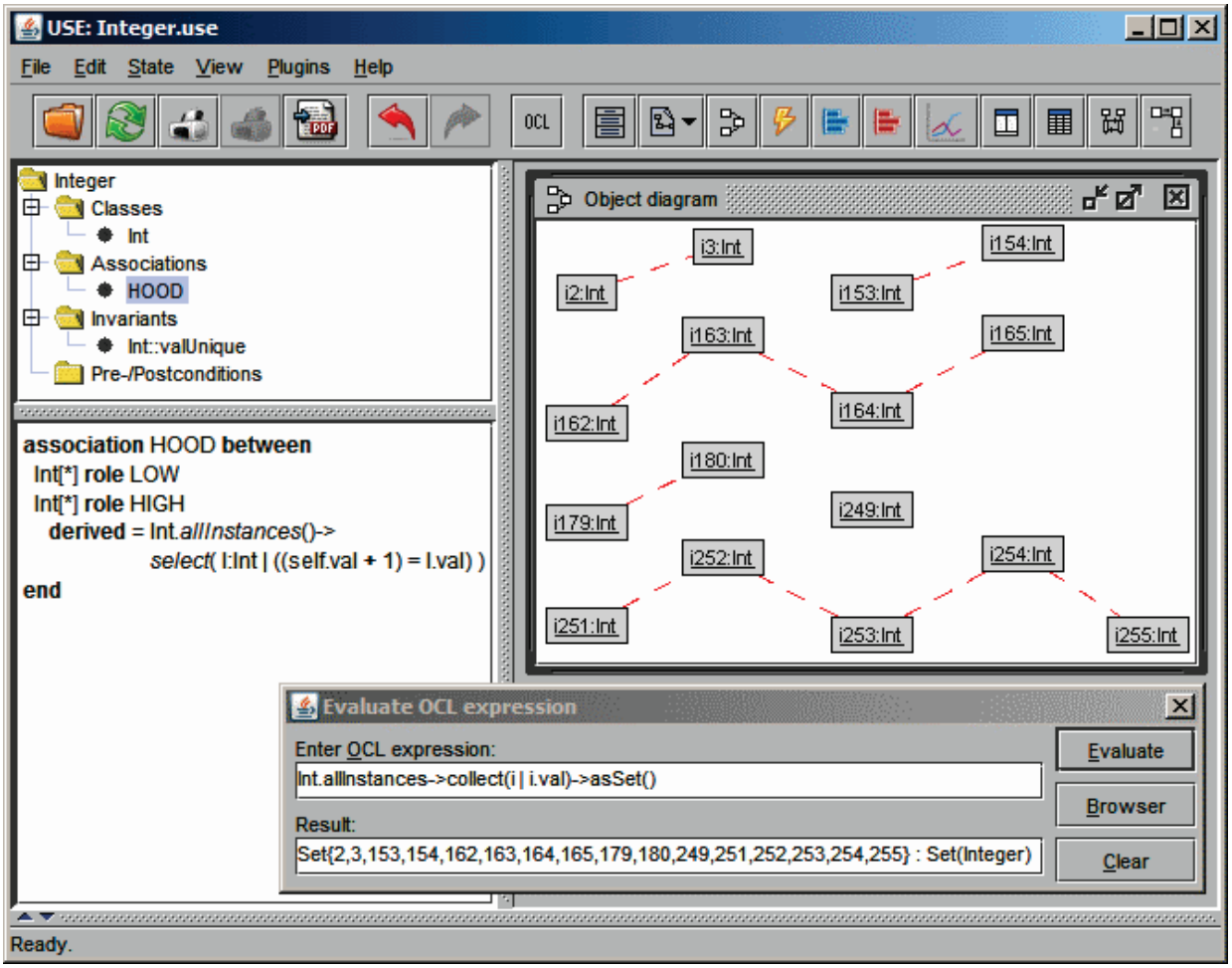

Fig. 4. Example for internal set element structure represented with derived links.

of a given integer, i.e., its predecessor and successor. Of course, other derived associations could be defined. This derived association divides the previously unconnected elements into components. In the example, there are six components.

Each single component constitutes a connected subset. Unconnected components have disjoint predecessors and successors.

Up to now we have considered integer sets. Similar techniques as discussed above can be applied however to sets of strings. Figure 5 is based on the titles of the papers at the Diagrams 2016 conference. It shows a weighted word cloud of the 28 words appearing in at least two different titles. In our view such a word cloud is also a diagrammatic representation of a set of strings. However, we will explore an alternative representation revealing more information.

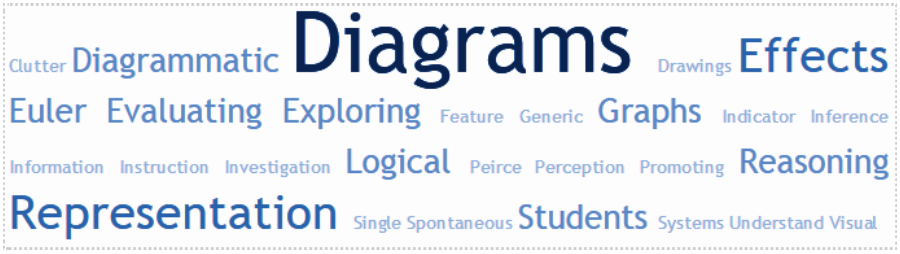

Fig. 5. Example for representing a word set (Diagrams 2016) with a word cloud.

Analogously to establishing a structure in a set of integers through its predecessors or successors, one can introduce an internal structure on a string set, e.g., through the common use of the word in different titles. Figure 6 shows the 28 words (each word appearing in at least two different titles) where a connection in form of a derived link is established, if the two words have a title in common. These links now classify the word set basically into four different components: (1) the top left Spontaneous-Instruction component, (2) the isolated Visual component, (3) the top right Perception-Reasoning component, and (4) the bottom Drawings-Understand component. This internal structure reveals more information than the word cloud in that it shows that certain words occur together in groups of titles. This structure indicates strong and weak connections in the word set: within one component, words have a strong connection, whereas words from different components have a weaker connection. These components could be shown also as a linear or an Euler diagram (see Fig. 7[2]).

To understand the shown derived links, the right bottom part of Fig. 6 shows how the Perception-Clutter-Diagrams triangle from the upper right is built. Here, all words occur in a single title. The shown word triangle (object subdiagram) does not show the other three titles to which each single word is also connected.

The technical definition for the derived, reflexive association on class `Word` looks as follows. A link indicates that the two words have a title in common. The link goes from a lexicographically lower string (role `fst`) to a lexicographically higher string (role `lst`).

```
association WordWord between
  Word [0..*] role fst -- first
  Word [0..*] role lst -- last
    derived = self.t.w->select(e | self.W<e.W)->asSet()
end
```

[2] https://www.cs.kent.ac.uk/people/staff/pjr/linear/, http://www.eulerdiagrams.org/inductivecircles.html

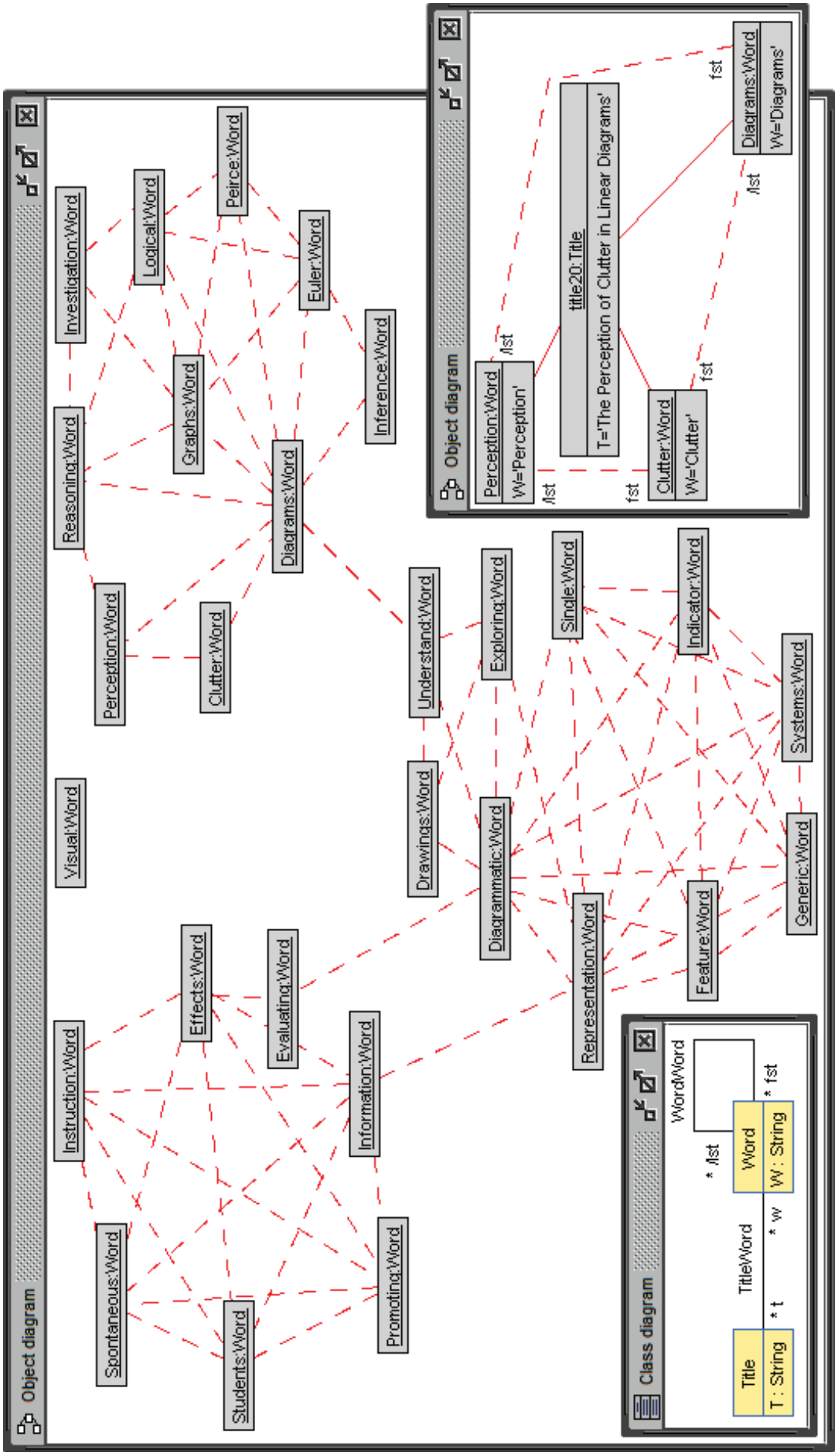

Fig. 6. Example for representing a set of words with derived links.

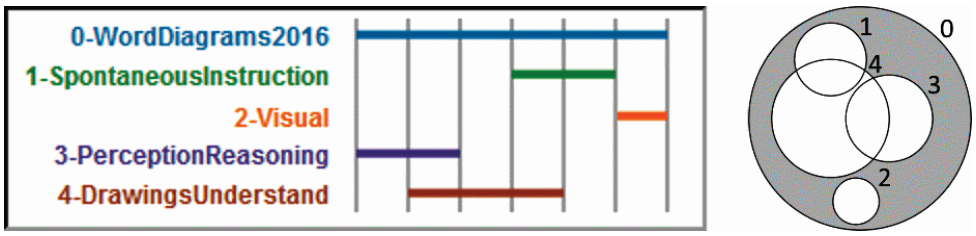

Fig. 7. Representing the word set components as a linear and as an Euler diagram.

5 Related Work

Due to space limitations we only mention very few approaches. In previous own work [3] we have discussed different options in our tool with emphasis on meta-models. Here we focus on set representations. The work in [4] discusses linear diagrams that could be derived from our objects diagrams with components. The work in [6] also applies UML diagrams focusing on ontology representation.

6 Conclusion

The problem discussed in this contribution has been to offer good diagrammatic representations for discrete, finite sets. We have shown how to use a software design tool to represent finite sets with UML object diagrams. Future work includes finding a general way to determine for a given set crucial internal relationships, i.e., associations, that can be applied to guide the process leading from the set to a diagram representing the set and displaying with the diagram layout meaningful relationships between set elements. Furthermore, finding a systematic way to go from the component representation to linear and Euler diagrams seems to be promising for string sets. Last but not least, larger case studies and examples should check the applicability and usefulness of the proposed techniques.

References

1. Gogolla, M.: Object Constraint Language. In Liu, L., Öszu, M.T., eds.: Encyclopedia of Database Systems. Springer, Berlin (2009) 1927–1929
2. Gogolla, M.: Unified Modeling Language. In Liu, L., Öszu, M.T., eds.: Encyclopedia of Database Systems. Springer, Berlin (2009) 3232–3239
3. Gogolla, M., Hamann, L., Xu, J., Zhang, J.: Exploring (Meta-)Model Snapshots by Combining Visual and Textual Techniques. In Gadducci, F., Mariani, L., eds.: Proc. WS GTVMT'2011, ECEASST 41 (2011)
4. Rodgers, P.J., Stapleton, G., Chapman, P.: Visualizing Sets with Linear Diagrams. ACM Trans. Comput.-Hum. Interact. **22**(6) (2015) 27:1–27:39
5. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language 2.0 Reference Manual. Addison-Wesley, Reading (2003)
6. Thomas, A., Gerber, A., van der Merwe, A.: An Investigation into OWL for Concrete Syntax Specification using UML Notations. In Jamnik, M., Uesaka, Y., Schwartz, S.E., eds.: Proc. 9th Int. Conf. Diagrams 2016, Springer, LNCS 9781 (2016) 197–211
7. Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (2003) 2nd Edition.