Getting It Right on the Dot

Bran Selic, Malina Software Corp. (CANADA) and Simula Research Laboratory (NORWAY)

April 18, 2013

Software developers familiar with UML may have run across models in which a mysterious black dot notation appears at the end of some associations, as shown below:

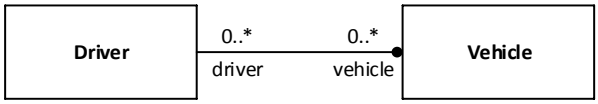

Figure 1. The mysterious dot notation

What exactly does this signify? This may come as a surprise to some experienced UML hands, but, it actually has the same meaning as the following UML 1 diagram:

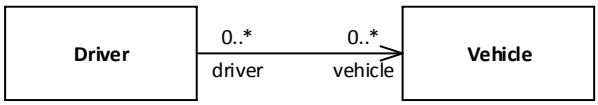

Figure 2. The equivalent UML 1 diagram

Even more surprising, perhaps, is that *the diagram in Figure 2, has a different meaning in UML 2 than it had in UML 1*. In other words, *the arrow at the end of an association has been replaced by the dot and, the meaning of the arrow notation has changed in part*. (This change occurred following the initial release of UML 2 (2.0), which is why it is rarely mentioned in UML 2 textbooks.)

It is perfectly reasonable to argue that it is not advisable to switch the meaning of a widely-used notation in mid-stream. But, it was necessary since the arrow notation in UML actually represented two distinct and unrelated concepts: *navigability* and *end ownership*.

Let us examine these two concepts to see how they differ and why it was necessary to distinguish them.

End Ownership

The term “end ownership” refers to the question of whether or not the end of an association is owned by the association or by an associated classifier. While this may at first seem a very esoteric issue that is of interest to only theoreticians, it actually has major significance to UML users. Specifically, it has to do with whether the associated classes involved in an association are affected by the association. If the end of an association is owned by a class, it means that the definition of that class has a feature that is typed by the class at the opposite end. Thus, in the example in Figure 1, the class Driver will have a feature (e.g., a pointer or a reference) to the class Vehicle, since the black dot at the “far” end of the association means that the end is “owned” by the class Driver. Conversely, since there is no black dot at the end opposite to the Vehicle class, it means that the Vehicle class does not have a feature that points to an instance of class Driver.

Note that this is primarily how the arrow notation was interpreted in UML 1. (Unfortunately, it was also used to represent navigability, which is a different concept, as explained below.)

To a programmer familiar with an object-oriented language, it may seem obvious that an associated class will invariable require some kind of feature that is typed by the class at the opposite end of that association; e.g.:

```
class Driver {
    . . .
    Vehicle vehicle [0..*];
    . . .
    }
```

Figure 3. The specification of the class Driver from Figure 1 in an OO language

However, there are other ways of realizing associations that do not require such features. A prototypical example of that can be found in databases, where the association between classes (or data types) is realized through an intervening table, e.g.:

Driver	Vehicle
Mickey Duck	Edsel (VIN #123456), BMW (VIN #1313)
. . .	. . .

In this case, the definitions of the classes Driver and Vehicle are unaffected by the fact that they are related (i.e., associated) through a table. The rows in such a table represent links that are represented by the association between the two classes.

Given the general nature of UML, it was necessary to support both kinds of associations, and the mechanism for doing this is through the concept of end ownership. Thus, if an association end is owned by a class, it means that there is a feature typed by the class at the opposite end. This is indicated by the black dot at the opposite end; i.e., in our example, the association end labeled vehicle is :owned" by the Driver class, which has a feature Driver::vehicle. (Note that it is possible to mix the two modes in a single association, as is the case in Figure 1.)

Unfortunately, the dot notation was introduced in UML 2.1, after many of the popular UML 2 textbooks were published. As a result, it is less familiar to UML users. (To make matters worse, the original versions of the UML 2 standard itself did not use that notation because there were no tools available that supported that notation when the standard was being drafted. Even the most recent issue of the standard, UML 2.5, still has some diagrams left over that use the arrowhead to represent end ownership.)

Navigability

The most important thing to note here about the concept of navigability as it was envisaged in UML is that *it is a run-time concept*[1]. That is, it serves to identify whether or not it is possible to *efficiently* navigate from an instance of one class to an instance of an associated class at run time. In practical programming language terms, this usually (but not necessarily!) means that an instance of one class has a pointer or a reference to associated instance(s) of the class at the opposite end. It is because of this common way of implementing efficient run-time navigation that the concept of navigability was conflated with the concept of end ownership in UML 1. However, there are ways of implementing efficient run-time navigation that do not require the use of pointers, such as tables or various compiler-generated mechanisms. This means that the UML 1 assumption that navigability implies the use of pointers that are represented by features of an associated class is not always valid.

Note that a major issue with the concept of navigability is that there is no precise interpretation of the meaning of the term "efficiently". Consequently, the concept of navigability in UML is rather vague and has lost much of its value for modeling. In a sense, the concept has been deprecated in UML 2 and even more so in later revisions, such as UML 2.5, where much effort was put into making the UML specification less ambiguous and more precise.

Summary

The upshot of all this is that the arrowheads one sees at association ends in most UML class diagrams should really be replaced by those little black dots. And, given the ambiguous and fuzzy nature of the navigability concept, there is little real need for using those arrowheads in models.

[1] See the discussion on the concepts of "navigability" and "navigation efficiency" in J. Rumbaugh et al., "The Unified Modeling Language Reference Manual – Second Edition", Addison-Wesley, 2005.