The importance of Opposites

Edward D. Willink[1]

Willink Transformations Ltd, Reading, England,
ed_at_willink.me.uk

Lightning presentation at the 16th International Workshop in OCL and Textual Modeling, October 2, 2016, Saint-Malo, France.

1 The importance of Opposites - E.D.Willink

By itself, OCL is almost useless since it lacks models to query. Once embedded within a model provider, OCL is still of limited utility since a side-effect free language cannot modify anything. The QVTc and QVTr declarative languages extend OCL to support model transformation without undermining the side-effect free characteristics of OCL. UML navigations and OCL constraints are used to specify the relationships between input and output model elements. No model mutations occur within the definition of the model transformation, rather the necessary model mutations are relegated to an implementation detail to be orchestrated by a practical tool.

Model transformation rules relate potentially overlapping patterns of source and target elements. The ATL example[1] in Figure 1 shows the relationship between the `forwardList`, `forwardList.name`, `forwardList.headElement` source pattern and the `reverseList`, `reverseList.name`, `reverseList.headElement` target pattern. ATL supports the overlap between the `headElement` mapping and another rule (not shown) using an implicit and opaque `resolveTemp` capability.

```
rule list2list {
  from
    forwardList : ForwardList!DoublyLinkedList
  to
    reverseList : ReverseList!DoublyLinkedList (
      name <- forwardList.name,
      headElement <- forwardList.headElement -- resolveTemp
    )
}
```

Fig. 1. Example using ATL.

Modeling the overlaps is difficult, if not impossible, without introducing new objects to identify each pattern of source and target elements. QVTc (and QVTr) therefore introduce an additional trace model which comprises a trace class for each pattern, with trace properties to identify the role of each source and target class within the pattern. Each trace class instance therefore groups related source and target elements. Simple UML navigations enforce most of the required relationships. Additional OCL constraints enforce more complex relationships. Figure 2 shows the UML Instance Diagram variant that the Eclipse QVTd implementation uses for the example.

The left hand column shows the blue source pattern. The right hand column shows the target pattern. The 'copied' value is shared at the top of the middle

[1] The example is an excerpt from "Local Optimizations in Eclipse QVTc and QVTr using the Micro-Mapping Model of Computation", E.D.Willink, "Second International Workshop on Executable Modeling (EXE 016)", http://www.eclipse.org/mmt/qvt/docs/EXE2016/MicroMappings.pdf

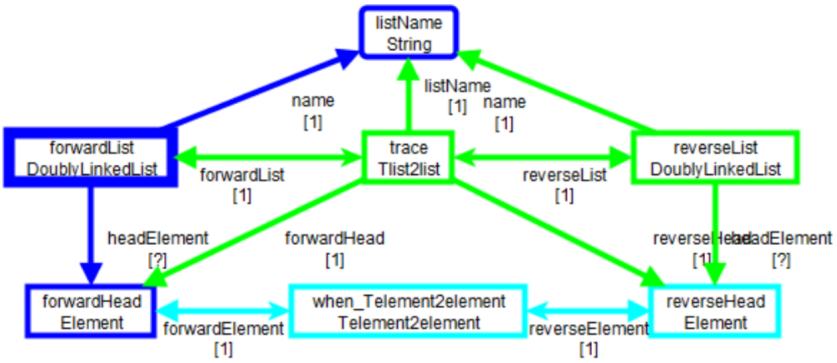

Fig. 2. QVTs-like exposition of QVTc mapping.

column. The additional two trace objects in the middle column identify the green creation of a match of the `list2list` rule using a `Tlist2list` instance and a cyan dependency on a `Telement2element` instance, which is a match of the `element2element` rule (not shown).

The transformation author defines the trace model explicitly in QVTc, or implicitly in QVTr. The trace model relates source and target models that are usually developed independently and so the relationships from trace model to source or target model are necessarily unidirectional. There is no navigable path from `forwardList` in the source model to `trace` in the middle model. This conflicts with the bidirectional navigability shown in Figure 2 that is necessary to use OCL navigation effectively.

Fortunately, OCL ignores the accidental navigability that may be a deliberate optimization of generated code or the unavoidable consequence of independent model development. In OCL, all object to object properties are navigable in both directions. Where the UML exposition is unidirectional, OCL automatically synthesizes an opposite using the name of the unnavigable target class allowing the use of `forwardList.Tlist2list`. If the forward name is ambiguous, the opposite name may be used to disambiguate: `forwardList.Tlist2list[forwardList]`.

The comprehensive opposite navigation capabilities of OCL therefore provide the foundation for the rigorous modeling of a side effect free QVTc or QVTr declarative transformation.