Relationship Navigation

Leon Starr

December 21, 2021

mint.sm-meta.action.tn.2

Version 0.1.0

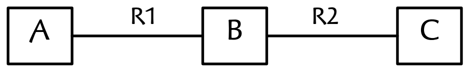
Change Log

Version	Date	Changes	Modified by
0.1.0	2021-12-21	Created initial version	Leon Starr
			Leon Starr

Relationship Navigation

Relationship navigation refers to the specification of a traversal path on the class model of some subject matter domain.

Here we have three classes {A, B, C} and two associations {R1, R2}:



Multiplicity, conditionality and attributes are omitted since they are not immediately relevant for our purposes of traversal specification.

We start with an instance set typed by class A. In other words we have a set of instance references to a subset, not necessarily a proper subset, of class A's instance population. We can think of these existing an instance flow originating from A. In a textual action language this will most likely be represented as a variable of type 'Class A instance reference'.

In Scrall we could initialize this variable as follows:

```
Aset ..= A(*) // Get all instances of A
```

To avoid referring to any attributes, we just go ahead and select all of the instances currently in A's population as our selection criteria doesn't matter for this particular study.

Now we would like to initialize another instance variable with the set of instances references typed by the B class related to all of the instances (if any) referenced in our Aset variable obtained by traversing R1. We can do this in Scrall like so:

```
Bset ..= Aset/R1  // Assign instance references across R1
```

We can see that there is no ambiguity in the expression since traversal from A across R1 has class B as its only possible destination.

In fact, we can go a step further to get all related C instances via both R1 and R1 like so:

```
Cset ..= Aset/R1/R2
```

Again, there is no ambiguity in this specification. If you hop across R1 from a set of A instance references to a set of B instance references and from there across R2 you must land on the C class and hence a set of C instance references. Note that if any hop yields an empty set, all subsequent hops will yield empty sets.

We can refer to each traversal across an association as a *hop*.

We can define a *path* as a sequence of hops from an originating instance set.

Each action language may use a different syntax for specifying the semantics of path and hop. For consistency, we will use Scrall for this exploration, but our focus will always be on identifying the semantics that can be supported by any action language syntax.

The question we wish to explore in this technical note is:

What is the minimum amount of information necessary to specify an unambiguous path across a sequence of relationships on a class model?

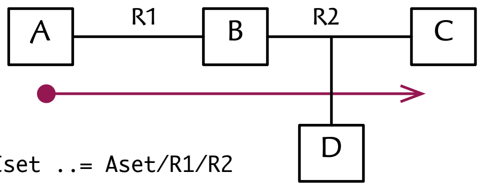
For association relationships that are not formalized by association classes where each association connects two distinct classes, i.e non-reflexive associations, we can see that a sequence of relationship names, R1, R2, etc. following an initial instance set suffices.

But what minimal set of information must we supply when we traverse a mix of generalization relationships, association classes, reflexive relationships and so forth? Let's find out!

Cases

1 Straight Hop

Each hop in the path is specified by the relationship name (rnum)



```
Cset ..= Aset/R1/R2

Cset ..= Aset/R1/R2/C
```

A straight hop may be performed on any non-reflexive association whether or not it is formalized by an association class.

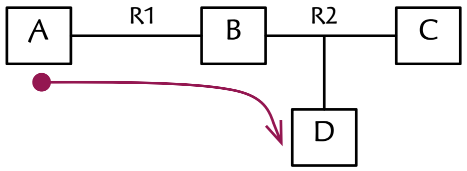
There are two ways to establish that a hop is a straight hop and not a To Association Class hop.

a. Specify only the association name (if the association class is not specified, it is assumed to be a straight hop)

b. Explicitly specify the destination class, perspective or both

The minimum straight hop specification, however, is simply the relationship name (rnum)

2 To Association Class Hop



```
Dset ..= Aset/R1/R2/D
```

In Scrall, you need to specify the destination association class name

When navigating to an association (reflexive or non-reflexive) formalized by an association class, there is more than one choice of destination. In this example, a traversal on R2 could lead to either C or D.

Semantically we can disambiguate by indicating that the R2 traversal is a 'To Association Class Hop'. Since an association can never be formalized by more than one association class, this hop classification suffices.

In your action language you can express this semantic with two choices:

a. Use notation to indicate the type of hop, for example:

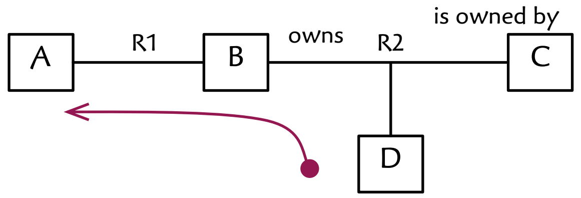
```
Aset/R1|-R2
```

b. Just specify the destination class (Scrall does this)

```
Aset/R1/R2/D
```

Cases

3 From Asymmetric Association Class Hop



When navigating from an asymmetric (two distinct perspective) association formalized by an association class, there is more than one choice of destination starting from the association class and working outward.

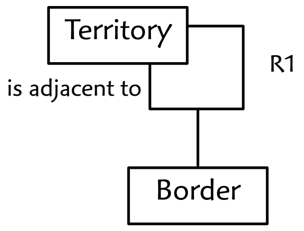
Starting from the association class it is necessary to specify either the destination class or perspective (verb phrase).

```
Aset ..= Dset/R2/B/R1

Aset ..= Dset/R2/owns/R1
```

Here are two ways you can do it in Scrall

4 From Symmetric Association Class Hop



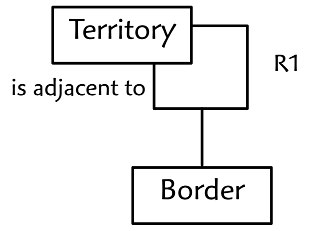
A symmetric association features only one perspective. In the Territory adjacency example we say that a Territory is adjacent to another Territory. Saying that Brazil is adjacent to Venezuela is the same as saying that Venezuela is adjacent to Brazil.

When an association class formalizes a symmetric reflexive association, there is only one way to hop from the association class and it is not necessary to specify a destination class or perspective.

```
territories ..= borders/R1
```

Cases

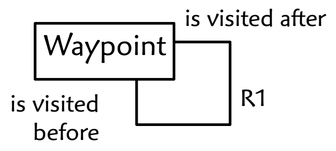
5 **Symmetric Hop**



A symmetric hop is similar to a straight hop except that the traversal leads back to the originating class. If only the relationship number is specified, this is assumed and no further information needs to be specified.

```
bordered territories ..= territories/R1
```

6 **Asymmetric Hop**



There are two directions of traversal when hopping an asymmetric reflexive association. Given an instance of Waypoint we can look for the previous visit or the next visit. In addition to the relationship number a perspective must be specified.

```
next waypoint .= current waypoint/R1/is visited after
```

With any Circular Hop (Symmetric, Asymmetric or Ordinal) there is the possibility of repeatedly hopping until one or more instances are found that meet some condition.

For example, you might want to keep hopping through connected Waypoints to find the one that is furthest rather than the one that is nearest.

```
furthest waypoint .= current waypoint/R1/is visited after/~|
```

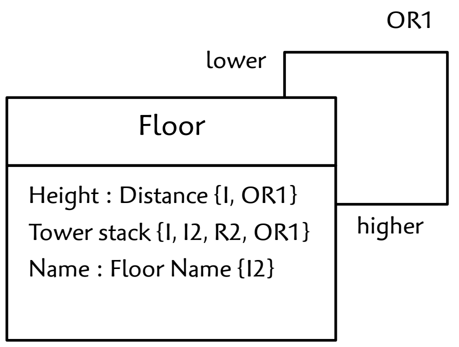
Or perhaps all Waypoints up ahead:

```
upcoming waypoints ..= current waypoint/R1/is visited after/~*
```

The above examples are illustrated with Scrall syntax, but the semantic is that a circular hop additionally specifies [first | last | all]

Cases

7 Ordinal Hop



An ordinal hop is just like an Asymmetric hop since it also has two perspectives.

OR1 is formalized by the Height and Tower stack attributes. Increasing values of Height are unique within a Tower Stack (separate vertical unit of a Building)

Instead of using referential attributes, an ordinal relationship uses a relational operation that selects all floors higher than a given floor and then chooses the lowest value in that subset effectively gives us the next highest floor.

Traversal in one direction or the other requires the relationship name and either the next ascending or descending value as indicated by the perspective name.

```
next highest floor .= my floor/OR1/higher
```

Tower stacks 1 and 2

Cases

8 Superclass Hop

9 Subclass Hop

Summary

Now we are in a position to model the key entities necessary to specify traversal across the class model.

Note that we do not specify any filtering of instances or selections beyond simple navigation. Selection and filtering is a distinct problem that can be combined with navigation.

A Path defines a sequence of Hops along contiguous Relationships on a Class Model.

A single Hop leads from an Instance Flow typed by some source Class across a single Relationship and yields another Instance Flow typed by some destination Class.

An Association Hop traverses an Association and a Generalization Hop traverses a Generalization.