Dangling cross-references

Core semantic question

Do we allow cross-references that point out from the scope?

Formally: Does `MyClass.myCrossReference(x, y);` imply `in_scope(y)`?

> Here `in_scope(y)` denotes y being in the scope (e.g. member of the ResourceSet, or subtree). It should be equivalent to `EObject(y)`, but distinguished here for clarity.

There are two possible ways to answer this question, leading to different semantics and implementation tasks.

Remark: `MyClass.myCrossReference(x, y);`and `MyClass(x);` both already imply `in_scope(x)`, and it should stay that way. The only question is about `in_scope(y)`.

Option 0 (unspecified) - don't care

This is the current solution. It is the cheapest, but users seem (rightly) confused that the semantics permit unspecified behaviour in the corner case of dangling edges.

Normalization. Also note that in a query composed of `MyClass.myCrossReference(x, y);`and `MyTargetClass(y);` the second constraint can be eliminated (if it is the exact value type of the reference).

Option 1 (no) - dangling references permitted

In this case, the base index must publish a single base relation:

- `EReferencePossiblyDangling<MyClass.myCrossReference>(x, y)`.

And this is what Pattern constraints translate to. Thus there is no change required in either the base index or (hopefully) in the matchers.

Normalization. Take a query composed of `MyClass.myCrossReference(x, y);`and `MyTargetClass(y);` the second constraint can **NOT** be eliminated, so PConstraint inference logic needs to be updated accordingly. Therefore power users might want to improve performance by omitting type constraints on y - we might want to avoid this side effect. How to avoid it?

- Turn query parameter type constraints into check-only ‘instanceof’ checks (or introduce new syntax for this?) - those can still be eliminated, and still offer type inference support e.g. for the generator.

Option 2 (yes) - dangling references rejected

In that case, some module must filter reference targets according to their residence in the scope, and (if necessary) incrementally update this filtered result. That responsibility can be put on either side of the IQRC interface.

Option 2a - dangling references rejected by query backend

The base index publishes two separate IInputKeys.

- One for an unfiltered EReference base relation `EReferencePossiblyDangling<MyClass.myCrossReference>(x, y)`.
- The second one is the scope membership: `in_scope(y)`. This actually already exists as `EClassTransitiveInstancesKey<EObject>(y)`, just labeled differently here for clarity.

According to IQMC, there is no impliciation relationship between these two tables at `y`, though `EReferencePossiblyDangling<MyClass.myCrossReference>(x, y)` of course still implies `EClassTransitiveInstancesKey<MyClass>(x)` and therefore `in_scope(x)`.

As there is no implication guaranteed by the base index, it is the responsibility of the query backend to compute a join for these two base relations in order to obtain the match set of the query constraint `MyClass.myCrossReference(x, y)`, which would therefore now translate into two PConstraints.

Normalization. Note that performance-wise this join can be possibly avoided if `in_scope(y)` is subsumed by any other pattern constraint (i.e. anything else dclensures that it is in scope).

Also note that in a query composed of `MyClass.myCrossReference(x, y);` and `MyTargetClass(y);` the second constraint can be eliminated (if it is the exact value type of the reference) only if

A. we ensure `in_scope(y)` in one way or the other (e.g. may be subsumed by a reference constraint that has y as source), and also
B. we upgrade the PConstraint-level inference mechanism to handle such complicated cases.

Also note that if we add this advanced type inference capability, then the second PConstraint associated with the language-level pattern constraint shall be

`EClassTransitiveInstancesKey<MyTargetClass>(y)`, so that this will not be a reason to unnecessarily index all EObjects.

Option 2b - dangling references rejected by base index

In this case, the base index must publish a single base relation:

- `EReferenceNonDangling<MyClass.myCrossReference>(x, y)`.

According to IQMC, there is an impliciation relationship: `EReferenceNonDangling<MyClass.myCrossReference>(x, y)`implies `EClassTransitiveInstancesKey<MyTargetClass>(y)` and therefore `in_scope(y)` (and of course the same for `x`, as always).

This way, the Pattern->PSystem mapping remains simple, and the query backends as well. The downside is that the base index must maintain and index the results of a join (in essence, contain a Rete join node).

Normalization. As for query normalization,in a query composed of `MyClass.myCrossReference(x, y);`and `MyTargetClass(y);` the second constraint can be eliminated (if it is the exact value type of the reference) just like in the current solution.

Summary

Option	0	1	2a	2b
Rejected on language-level	unspecified	no	yes	yes
Responsible for computation of rejection	Type inference, sometimes, accidentally	-	Query backend	Base Index (Engine Context)
Base Index has to maintain in_scope(x)	Not necessary	Not necessary	Yes	Yes, unless subsumed in all cases
`MyTargetClass(y)` subsumed	Yes :)	No :(	Yes… but in_scope(y) must still be checked by query backed, though	Yes… but no performance gain: in_scope(y) must still be checked

			sometimes it can be subsumed as well; Also, subsumption requires smarter PSystem-level inference than what we have now	internally by the backend, and cannot be subsumed
Components to modify	Users :)	Small change to EMF metacontext	EMF meta context, PBody inferrer, PSystem type inference logic (impact on query backends)	Base index only (though complex, e.g. includes join!)
Estimated development cost	0	Epsilon	Couple of days	Couple of days
Estimated performance impact	0	Extra join often	Extra join, hopefully often eliminated (with advanced type inference)	Extra join always
Estimated user experience	Confusion, it seems	Power users might want to improve performance by omitting type constraints on y - we might want to avoid this, but my solution leads to confusion once again.	These 2 options are indistinguishable to the average user	