(Tried to capture the intent please ignore any property name typos)

At first shapes and constraints were different things. A shape had targets and could define constraints with special properties like sh:property

Shacl v0.1 (mostly IBM resource shapes + sparql)

Shape (no subclasses/ superclasses)

- Defines targets
- Defines constraints with the following properties
    - sh:property (a sh:PropertyConstraint *subClassOf sh:Constraint*)
    - sh:sparql (a sh:SparqlConstraint *subClassOf sh:Constraint*)

V0.2 introduction of node constraints

Shape (no subclasses)

- Defines targets
- Defines constraints with the following properties
    - sh:property (a sh:PropertyConstraint *subClassOf sh:Constraint*)
    - sh:node (a sh:NodeConstraint *subClassOf sh:Constraint*)
    - sh:sparql (a sh:SparqlConstraint *subClassOf sh:Constraint*)

The constraint parameter sh:shape was pointing to NodeConstraints

Comment: Up to this point the language was regular we had different properties for different types of constraints and everything was fine (Except that it was more verbose)

V0.3 merge of node constraints with shapes

Shape (=== sh:NodeConstraint *subClassOf sh:Constraint*) # now shape is both a shape and a node constraint

- Defines targets
- Defines constraints with the following properties
    - sh:property (a sh:PropertyConstraint *subClassOf sh:Constraint*)
    - sh:sparql (a sh:SparqlConstraint *subClassOf sh:Constraint*)
    - ~~sh:node (a sh:NodeConstraint *subClassOf sh:Constraint*)~~
        - Removed sh:node from the language
        - A shape can directly attach constraint parameters link sh:nodeKind that are interpreted like NodeConstraints
        - A shape can optionally use sh:shape to group constraints e.g.

ex:A a sh:Shape ;
  sh:nodeKind sh:IRI .
Is the same as
ex:A a sh:Shape ;
  sh:shape [sh:nodeKind sh:IRI]

We reuse sh:shape for NodeConstraints(or shapes) but we can also define node constraint directly in the shapes through sh:shape.

Comment: From now on the language is irregular and terminology is very confusing wrt constraints and shapes, multiple ways to define node constraints in shapes

V0.4 latest change

Broken down to steps to track changes

V0.4.1 move sh:sparql to components

sh:Shape (*subClassOf sh:Constraint*)
- Defines targets
- Defines constraints
  - sh:property (a sh:PropertyConstraint *subClassOf sh:Constraint*)
  - ~~sh:sparql (a sh:SparqlConstraint *subClassOf sh:Constraint*)~~
    - sh:sparql is now a simple constraint component parameter and not part of the language
    - Modularization of the language / minimal change for the end user
  - Can directly attach constraint parameters link sh:nodeKind that are interpreted like NodeConstraints

V0.4.2 move sh:property to components

sh:Shape (*subClassOf sh:Constraint*)
- Defines targets
- Defines constraints directly inline but only NodeConstraint (or shapes from now on
  - ~~sh:property (a sh:PropertyConstraint *subClassOf sh:Constraint*)~~
    - sh:property removed from the core language but re-introduced as a constraint component that points to a sh:PropertyConstraint
    - Modularization of the language / no change for the end user

V0.4.3 renaming

Sh:shape is renamed to sh:NodeShape
sh:propertyConstraint is renamed to sh:PropertyShape
Sh:constraint is renamed to sh:Shape
Comment: at this point we have sh:NodeShape as what it was sh:Shape before

So now sh:Shape is the superclass of sh:nodeShape and sh:PropertyShape, but only sh:nodeShape can define targets

sh:NodeShape (*subClassOf sh:Shape*)
- Defines targets
- Defines constraints inline

irregularities:
- Sh:shape, sh:or, ... still only links to NodeShapes
- Only NodeShapes can define targets (counterintuitive)
- (both fixed in v0.4.4)

V0.4.4 generalizing targets and values of sh:and,or,shape...

sh:Shape (and both sh:nodeShape & sh:PropertyShape)
- Defines targets
- Defines constraints inbline

We still have two subclasses sh:NodeShape / sh:PropertyShape but sh:shape, sh:or, … can now link to Shapes (in general), no restrictions anymore

sh:PropertyShape is not needed anywhere else besides the sh:property and this can already be defined with sh:shape.

Comment: terminology & language is improved

Irregularities:
- Property shapes can be defined by both sh:shape and sh:property -> multiple ways to do the same thing
- No equivalent way to define only NodeShapes
- No need to keep the hierarchy
    - Spec: “sh:PropertyShape is the class of property shapes and should be declared as a type for shapes that are IRIs. However, the presence of any rdf:type triple does not determine whether a node is treated as a property shape or not.”

- Spec: “sh:NodeShape is the class of node shapes and should be declared as a type for shapes that are IRIs. However, the presence of any rdf:type triple does not determine whether a node is treated as a node shape or not.”
- This means that the types and the hierarchy is not needed. We could mark a node shape as sh:PropertyShape and the other way around and SHACL would still work so these subclasses can be, besides redundant, confusing for the user.

V0.5 proposed change

Proposal A: Remove sh:property as redundant
Proposal B: Have only sh:Shape and no subclasses. If a shape has sh:path, then the value nodes are determined by the path. We can keep the terms “node shape” and “property shape” for convenience
The behavior of SHACL is exactly what we have now without the hierarchy complexity. There is available text to reuse so if this is accepted it will pose no delay for CR (in case this is a concern)