October 24th 2011, DSM'11@ SPLASH

Advancing Generic Metamodels

Henning Berg
hennb@ifi.uio.no
Birger Møller-Pedersen (presenter)
birger@ifi.uio.no
Stein Krogdahl
steink@ifi.uio.no

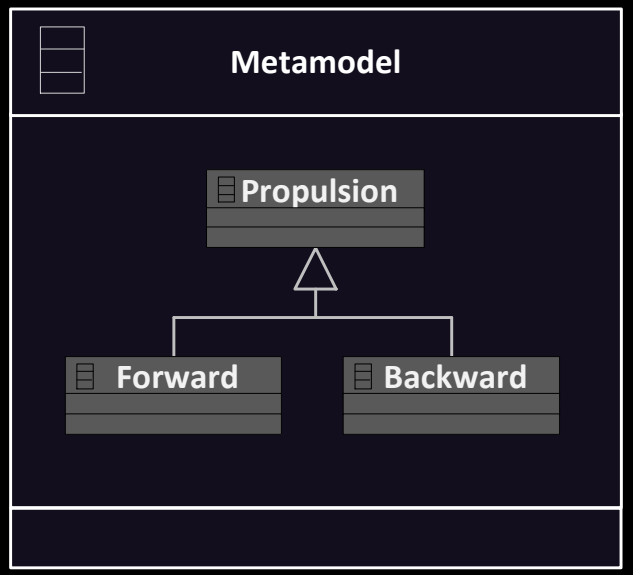
Department of
Informatics

UNIVERSITY
OF OSLO

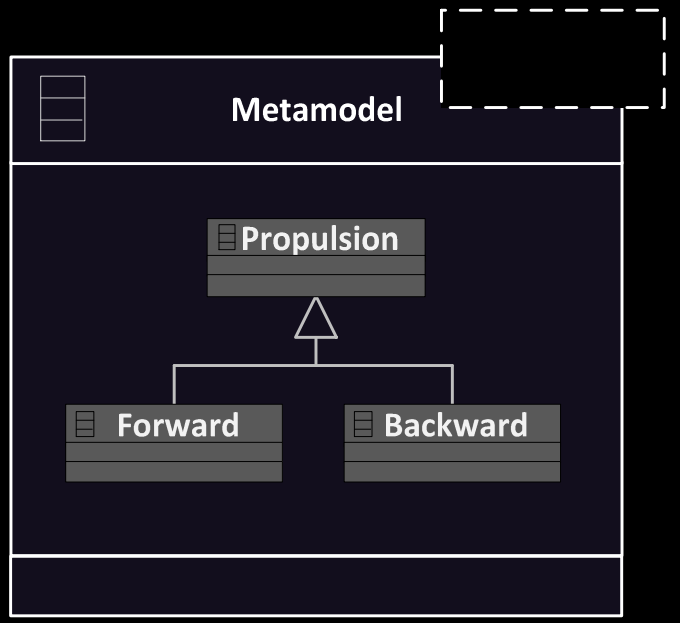
Domain-Specific Languages
are
difficult to customise

Are simple packages
too simple?

1) What if metaclasses were defined in an enclosing class?



2) What can be achieved by using type parameters on the enclosing class?



1) Common profiling attributes

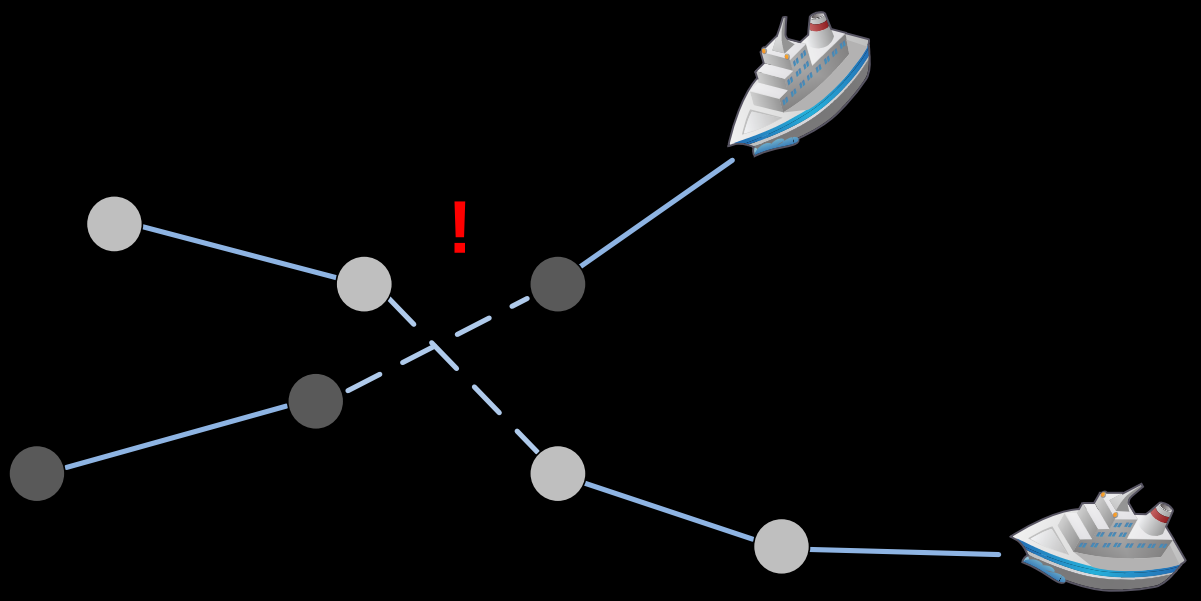
```
class Metamodel
{
    x : Integer;
    y : Real;

    class Metaclass1 {
      … x …
      … y …
    }
    class Metaclass2 {
      … y …
     }
    class Metaclass3 { … }
}
```

2) Type parameters on the enclosing class

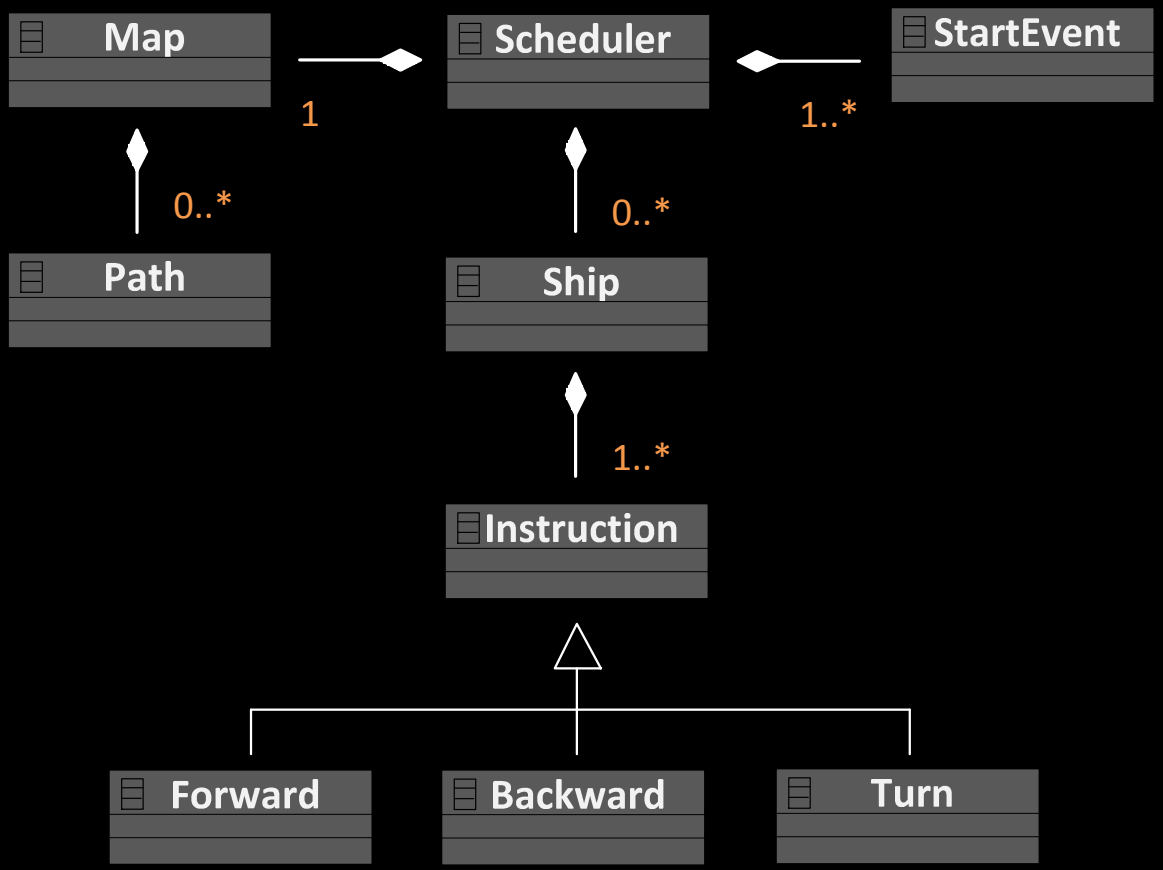
```
class B1 { … }
class B2 { … }

class GenericMetamodel<T1 : B1, T2 : B2>
{
    z : T1[0..*]

    class Metaclass1 {
        … T1 …
    }
    class Metaclass2 {
        … T1 …
        … T2 … }
    class Metaclass3 { … }
}
```

Example:

scheduling of
ship routes



Language: ShipScheduling

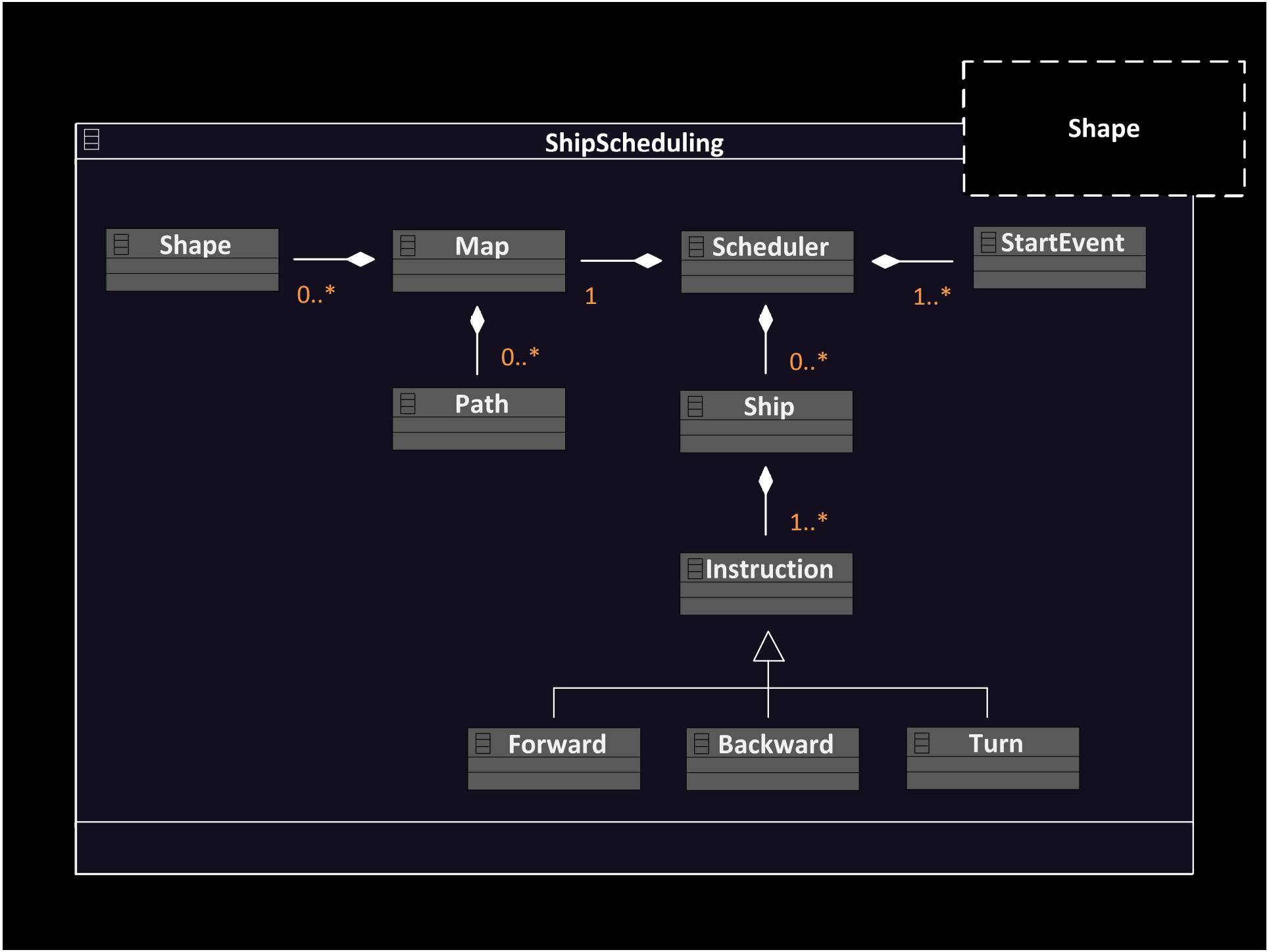


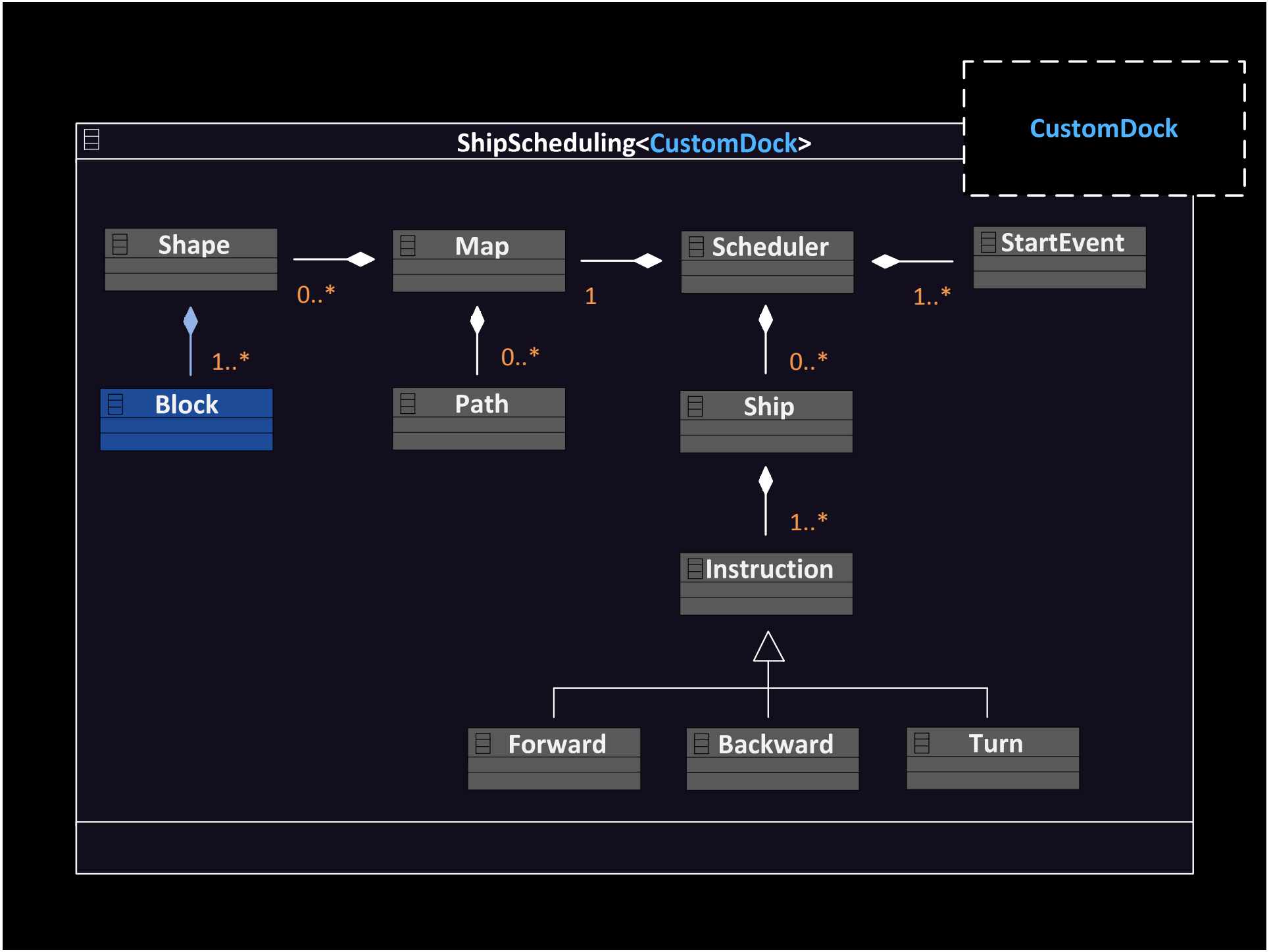
```
class ShipScheduling
{
    class Schedule { ... }
    class Ship { ... }
    class Map { ... }
    class StartEvent{ ... }
    class Path{ ... }
    class Instruction{ ... }
    class Forward inherits Instruction{ ... }
    class Backwards inherits Instruction{... }
    class Turn inherits Instruction { ... }
}
```

```
pathLength : Real[1..1]
maxSpeed : Real[1..1]
calculateSpeed( type : ShipType ) : Real is do ... end
shipsOnMap[0..*]
updateMap() is do
    ... pathLength ...
    ... calculateSpeed( type ) ...
end
verify() is do
    ... pathLength ...
end
```

```
abstract class Shape { … }

class ShipScheduling<S : Shape>
{
    class Scheduler { … }
    class Ship {
        instructions : Instruction[0..*]
        calculate() is do
            …
        end }
    class Map {
        shapes : S[0..*]
    }
}
```

```
require ”ShipScheduling.kmt”

class SimpleDock inherits Shape { size : Real }
class CustomDock inherits Shape { blocks : Block[1..*] }
class Block  {
    size : Real
    orientation : Real
    x : Integer
    y : Integer }


var s1 : ShipScheduling<SimpleDock>

var s2 : ShipScheduling<CustomDock>
```


Shape
ShipScheduling
Shape
Map
Scheduler
StartEvent
0..*
1
1..*
0..*
0..*
Path
Ship
1..*
Instruction
Forward
Backward
Turn


ShipScheduling<CustomDock>
CustomDock
Shape
Map
Scheduler
StartEvent
0..*
1
1..*
1..*
0..*
0..*
Block
Path
Ship
1..*
Instruction
Forward
Backward
Turn

virtual classes

```
class ShipScheduling<S : Shape>
{
    class Scheduler { … }
    virtual class Ship { … }
    virtual class Map { … S[0..*] }
}

class ShipSchedulingVariant<S::Shape>
    inherits ShipScheduling<S>
{
    class SpecialShip inherits Ship {
        type : Type
        calculate() is do … end
    }
    enumeration Type{ cruise; ferry; cargo; }
}
```

Attributes of enclosing class

allow profiling

Parameterisation of enclosing class

provides means for

customisation

Specialisation of enclosing class

and virtual classes

support metamodel

extension and variation