

Aluno: Thiago Lessa Aramaki

Projeto de modelagem de DSL

Como proposta de desenvolvimento de uma DSL o domínio escolhido foi o de operação de oleodutos. O presente trabalho é uma prova de conceito de como poderia ser uma DSL voltada para o domínio mencionado. Diversos são os cenários de análises de processos de oleodutos. Muitas dessas análises de processo não requerem simulações hidráulicas sofisticadas. Basicamente, existem elementos que são comuns a todas as análises de processo que envolvem simulações hidráulicas em oleodutos, ou seja, que envolvem o transporte de fluidos na fase líquida. Está fora deste escopo do presente trabalho a modelagem de gasodutos, dutos que transportam fluidos na fase gasosa.

Também não foram contemplados a modelagem de equipamentos tais como bombas, válvulas de controle, e válvulas de um modo geral.

A análise transiente também foi excluída dessa primeira versão. Os cenários possíveis a serem modelados são cenários que envolvam o regime permanente.

Em um primeiro momento, a parte térmica será desprezada, ou seja, o sistema será considerado isotérmico para efeitos de simplificação.

O objetivo final é modelar dois cenários bem comuns. O gradiente hidráulico com o duto em operação e o gradiente hidráulico com o duto parado. Na figura abaixo, tem-se um exemplo de gradiente hidráulico com duto em operação, curva vermelha, transportando três fluidos distintos, diesel S500, diesel S10 e GLNA. Cada produto está representado por uma cor diferente. Variações das propriedades do fluido e geometria do duto podem impactar na curva do gradiente hidráulico. A curva em verde representa o perfil de elevação ao longo do duto.



Figure 1:

Figura 1 - Exemplo de gradiente hidráulico em operação

Uma outra situação ocorre quando o duto está parado. Na figura abaixo o duto encontra-se em repouso e pressurizado. A linha vermelha mostra um instante em que o duto está parado com um determinado perfil de temperatura. Após um determinado período de tempo, a temperatura esfria por meio de troca de calor com o solo, o que resulta em um novo perfil de temperatura que resulta em um novo gradiente hidráulico tal como o mostrado na figura abaixo.

Ambas as situações, operação e repouso, mostram uma situação conhecida como operação com coluna cheia. Ou seja, para cada nó do duto, a pressão interna é superior a pressão de vapor do produto, o que garante que o duto esteja totalmente na fase líquida.



Figura 2 - Exemplo de gradiente hidráulico com o duto parado e pressurizado

Pode acontecer do duto em um determinado momento estar com a “coluna quebrada”, ou seja, quer dizer que em algum ponto a pressão interna no duto ficou abaixo da pressão de vapor e o produto nessa região não está na fase vapor.

Os gráficos mostrados acima foram desenvolvidos em uma linguagem de uso geral, VB.Net. Pretende-se obter um MVP, produto mínimo viável, da funcionalidade de cálculo do gradiente hidráulico por meio de uma DSL.

A figura 1 mostra o modelo PIM desenvolvido. Existem três enumerados. O **FlowDirection** é utilizado pela classe **Station** para indicar se a estação está enviando ou recebendo, ou seja, o sentido do fluxo que passa pela estação. Pensando em projetos futuros será necessário que o modelo entenda a divisão de fluxo, ou seja, é possível que uma estação esteja recebendo enquanto que parte do fluxo segue a jusante para outro duto.

O **PipelineState** indica se o duto está parado ou em operação. Isso é importante para as análises que serão realizadas posteriormente pelo modelo.

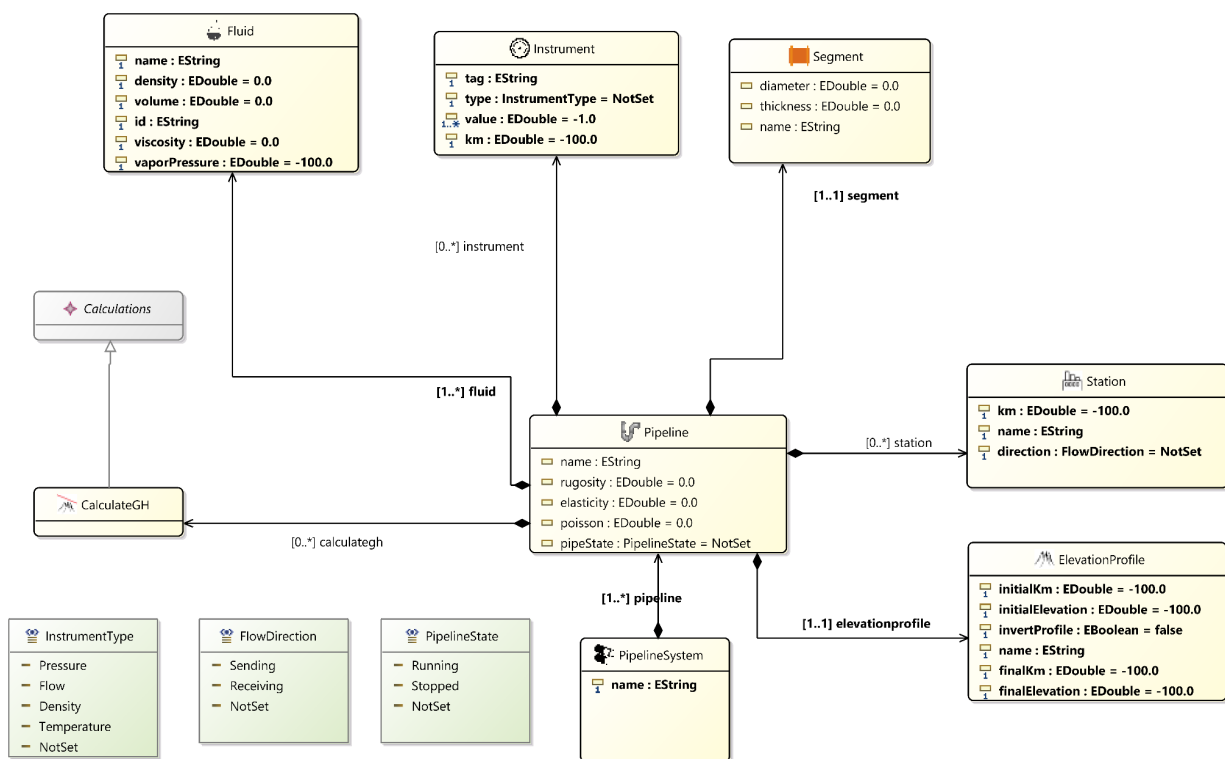


Figura 3 - Modelo independente de plataforma (PIM) da DSL proposta

O **InstrumentType** indica o tipo de instrumento que um determinado medidor representa. No domínio de oleodutos, os principais são: pressão, vazão, densidade e temperatura.

O **PipelineSystem** representa um sistema que pode ser composto por vários **Pipelines**. Um **pipelineSystem** somente existe, se houver pelo menos um **Pipeline** em seu sistema. Seu único atributo é o nome que identifica o mesmo.

O **Pipeline**, como pode ser observado, é o elemento central do modelo. Este elemento possui propriedades do duto, tais como nome do duto, rugosidade em mm, elasticidade em kgf/cm², coeficiente de poisson que é um número adimensional e depende do material do qual o duto é constituído, e o **pipeState** que indica o estado do duto, parado ou em repouso. Os atributos contidos valem para todos os segmentos que pertencem a este pipeline. Nesse modelo somente é permitido a existência de um segmento por Pipeline. Cada segmento contém informações do nome do segmento, diâmetro externo e espessura da parede do duto. Se houver em um mesmo duto variação de espessura, deve-se criar um outro objeto Pipeline, composto por um segmento que representa essa outra espessura.

Cada objeto **Pipeline** deve ter associado um único perfil de elevação, **ElevationProfile**. Este último contém dados como o km inicial, o km final, a cota inicial, cota final a elevação, nome que identifica o perfil e um atributo booleano que indica se o perfil de elevação deve ser invertido ou não. Dependendo da análise, o duto pode ser observado no fluxo ou refluxo. Nesse caso pode ser necessário inverter o perfil de elevação para que o usuário visualize os dados da melhor forma possível. Essa inversão é definida pelo atributo **invertProfile**. Essa funcionalidade não foi implementada na presente versão.

O **Pipeline** ainda pode conter 0 ou mais estações. Normalmente, somente encontram-se estações nas extremidades dos oleodutos. Em alguns dutos existem estações de bombeio intermediárias que também poderiam ser modeladas. Os atributos desse objeto são, o nome que identifica a estação, a localização da estação ao longo do duto, km, e o sentido do fluxo da estação.

O **Pipeline** pode ter 0 ou mais instrumentos. Pode existir um segmento que se conecta a outros segmentos e que não possui nenhuma instrumentação associada. Um caso comum é ter 4 em cada ponta do segmento, mas isso pode variar. Os instrumentos apresentam os seguintes atributos: tag que representa um identificador único do instrumento, a localização do instrumento ao longo do duto, km, o tipo do instrumento que pode ser um medidor de vazão, pressão, densidade, ou temperatura.

Finalmente, o **Pipeline** deve ter um ou mais fluidos. O fluido é representado pelo objeto **Fluid** que possui as propriedades básicas dos fluidos, massa específica (density), nome que identifica o fluido, um **id** que representa um código único que pode identificar um produto. Esse campo foi pensado para funcionalidades futuras e não está contemplado nessa versão. A

viscosidade cinemática, e pressão de vapor que é a pressão mínima na qual o fluido em questão permanece na fase líquida.

Os instrumentos representados pelo estereótipo **Instrument** possuem três propriedades. O tag que é o código pelo qual o instrumento é identificado, o valor que é o resultado de medição, e o tipo que foi descrito anteriormente por meio do enumerado **InstrumentType**.

Por fim, existe uma categoria de elementos que refere-se a funcionalidades do sistema. Nessa primeira versão apenas uma funcionalidade será implementada que é o cálculo do GH. Futuramente outras funcionalidades podem ser implementadas. Todas as funcionalidades devem herdar da classe Calculations.

Um exemplo de modelo foi criado para testar o PIM. O sistema foi chamado de OSBRA. Em seguida foi criado um pipeline chamado PLN2PIRp que possui um segmento denominado PLN2PIRs. Possui um perfil de elevação (ElevationProfile), três instrumentos, dois de pressão e um de vazão, três estações e dois fluidos, um diesel S500 e outro de gasolina (GLNA). A funcionalidade pretendida é o cálculo do gradiente hidráulico, CalculateGH.

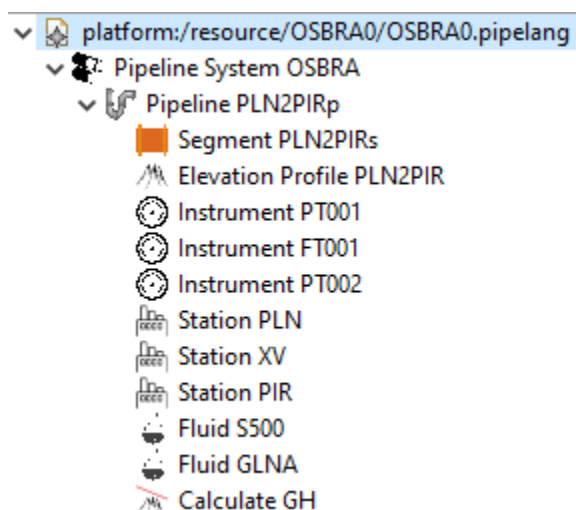


Figura 4 - Modelo de exemplo eCore do PIM

```
<?xml version="1.0" encoding="UTF-8"?>
<PipeLang:PipelineSystem xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:PipeLang="http://www.example.org/PipeLang" name="OSBRA">
  <pipeline name="PLN2PIRp" rugosity="0.045" elasticity="1899731.0">
```

```

    poisson="0.3">
      <segment diameter="20.0" thickness="0.25" name="PLN2PIRs"/>
      <elevationprofile initialKm="0.0" initialElevation="700.0"
name="PLN2PIR" finalKm="99.0" finalElevation="854.0"/>
      <instrument tag="PT001" type="Pressure" km="0.0">
        <value>72.0</value>
      </instrument>
      <instrument tag="FT001" type="Flow" km="0.0">
        <value>900.0</value>
      </instrument>
      <instrument tag="PT002" type="Pressure" km="99.0">
        <value>7.0</value>
      </instrument>
      <station km="0.0" name="PLN" direction="Sending"/>
      <station km="45.0" name="XV" direction="Sending"/>
      <station km="95.0" name="PIR" direction="Receiving"/>
      <fluid name="S500" density="840.0" volume="10000.0" id="2702200001"
viscosity="345.0" vaporPressure="0.0"/>
      <fluid name="GLNA" density="720.0" volume="-1.0" id="2702200001"
viscosity="1.0" vaporPressure="0.0"/>
      <calculategh/>
    </pipeline>
  </PipeLang:PipelineSystem>

```

Figura 5 - Modelo de exemplo em XML

As associações impostas pelo diagrama são respeitadas. Quando se tenta criar um objeto filho a partir do PipelineSystem, apenas o pipeline pode ser criado

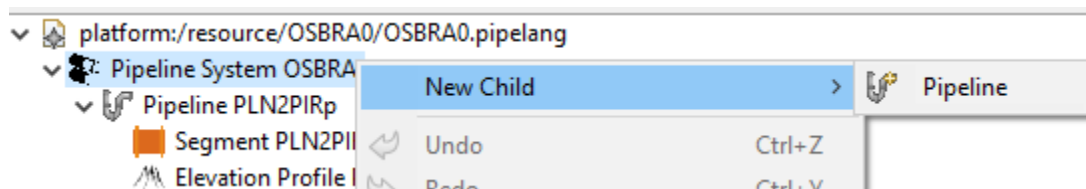


Figura 6 - Exemplo de criação de objetos a partir do PipelineSystem.

A partir do objeto pipeline, duas possibilidades são permitidas, a criação de objetos filhos e a criação de um outro objeto pipeline no mesmo nível. No caso de objetos filhos, apenas um perfil e um segmento podem ser criados, como no exemplo já foram criados, o menu desabilita a opção de selecioná-los.

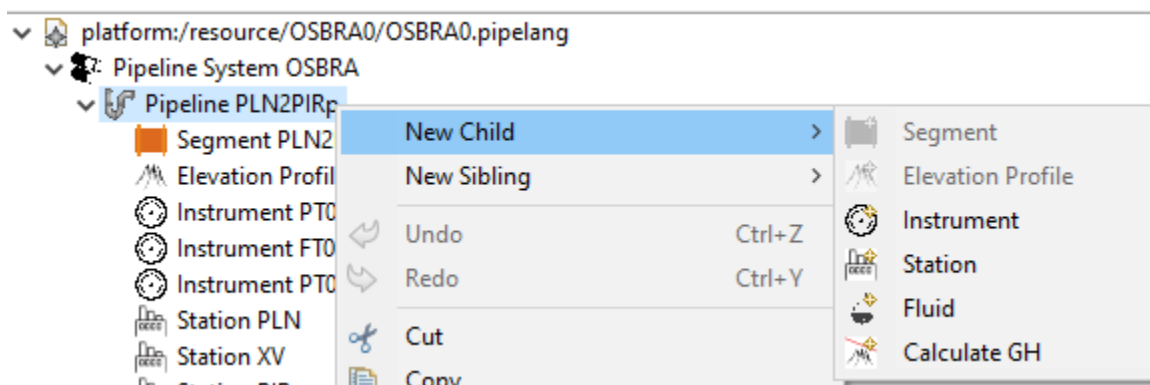


Figura 7 - Exemplo de criação de objetos a partir do Pipeline

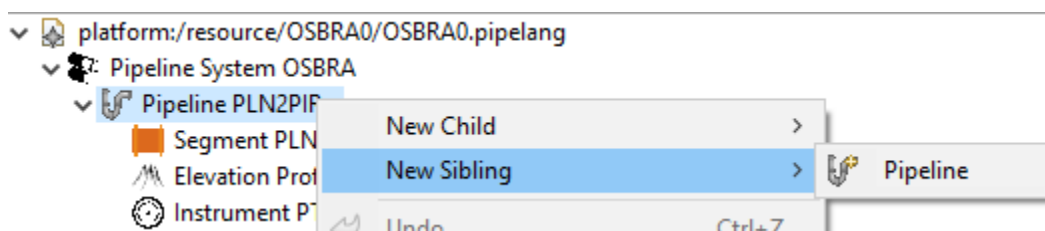


Figura 8 - Apenas a possibilidade de criar um objeto Pipeline de mesmo nível hierárquico.

Em seguida este modelo deve ser transformado para um modelo PSM por meio de um projeto ATL. Para facilitar o entendimento, a Figura 8 mostra o diagrama do modelo PSM.

Uma transformação importante refere-se a criação de mais de um segmento quando ocorre a mudança de fluido. Muitos cálculos são dependentes das propriedades do fluido e da geometria do duto. Então espera-se que com essa segmentação por fluido melhore a performance do código gerado e talvez até uma facilidade de paralelização do código. A associação entre o Pipeline e o Fluid agora é de um para um, pois no PSM só pode ter um fluido por segmento, diferentemente do que foi feito para o PIM cuja associação permite que um único pipeline possa ter mais de um fluido.

O restante das associações permanece igual.

Outra diferença é que a funcionalidade ficou mais detalhada, possuindo agora mais duas classes que são necessárias para o cálculo do gradiente hidráulico. A classe Reynolds é responsável pelo cálculo do número de Reynolds e pode ser observada na equação (1). Nessa

equação, ρ é a massa específica, v a velocidade do fluido, D , o diâmetro interno e μ a viscosidade cinemática.

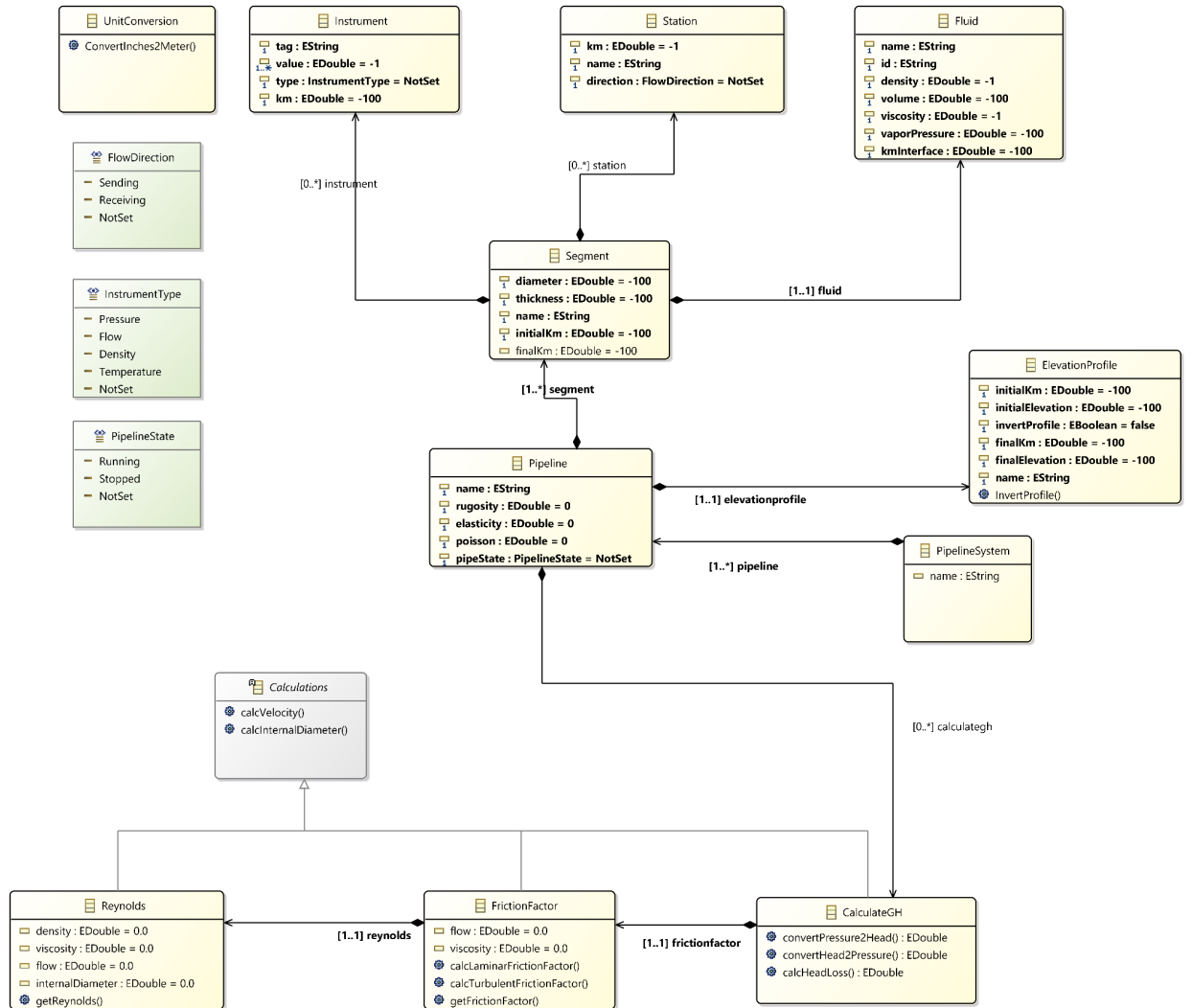


Figura 9 - diagrama do modelo PSM

$$Re = \frac{\rho \cdot v \cdot D}{\mu} \quad (1)$$

O fator de fricção utiliza a equação de Muller e depende do número de Reynolds. Se o número de Reynolds for menor ou igual a 1400, então diz-se que o fluido está no regime laminar e o fator de atrito é dado pela equação (2).

Se o Reynolds estiver entre 1400 e 2750 consideramos o regime em transição. Nesse caso estamos utilizando um valor fixo como mostrado na equação (2).

$$f = \frac{64}{Re} \quad (2)$$

$$f = 0.0457 \quad (3)$$

$$f = 0.25 \cdot \log_{10} \left(\frac{k}{1000} \cdot \frac{1}{3,7 \cdot D_{int}} + 5,74 \cdot Re^{-0,9} \right)^{-2} \quad (4)$$

Finalmente, para o regime turbulento utiliza-se a equação 3.

O cálculo do gradiente hidráulico está sendo realizado somente com a condição de contorno do lado expedidor. Em versões futuras as condições de contorno podem variar, por exemplo, realizar o cálculo a partir do recebimento, utilizando portanto, a pressão e vazão do recebimento. Ou ainda utilizar condições de ambos os lados.

Inicialmente, converte-se a pressão inicial no head inicial dado pela equação (5), onde, P é a pressão interna ou manométrica inicial, “Z” é a elevação, e “d” a massa específica em g/cm³. De forma análoga a conversão do Head para pressão pode ser observado na equação (6)

$$H = \frac{10 \cdot P}{d} + Z \quad (5)$$

$$P = \frac{d \cdot (H - Z)}{10} \quad (6)$$

Uma outra equação importante é o cálculo da perda de carga pela equação de Darcy-Weisbach, mostrada na equação (7)

$$h_f = f \cdot \frac{L}{D} \cdot \frac{v^2}{2g} \quad (7)$$

O cálculo do gradiente hidráulico consiste na subtração da perda de carga do head inicial. Para head final calculado, converte-se o mesmo para pressão e calcula-se novamente o head inicial do próximo segmento, considerando a densidade do próximo segmento.

As transformações necessárias para M2M para levar o modelo PIM ao PSM foram realizadas por meio de um projeto ATL que pode ser visualizado abaixo.

```
-- @path PipeLang=/PipeLang/model/PipeLang.ecore
-- @path PipeSim=/PipeSim3/model/PipeSim3.ecore

module PipeLangPipeSim;
create OUT: PipeSim from IN: PipeLang;

helper def : idSeq : Integer = 0;
helper def : numFluids : Integer = PipeLang!Fluid.allInstances().size();

helper def: comprimento(kmInicial: Real, kmFinal: Real): Real =
    (kmFinal - kmInicial) * 1000;

helper def: diametroInterno(diametroExterno: Real, espessura: Real): Real =
    (diametroExterno - 2 * espessura) * 0.0254;

helper def: area(diametroExterno: Real, espessura: Real): Real =
    180.toRadians() * thisModule.diametroInterno(diametroExterno,
    espessura) * thisModule.diametroInterno(diametroExterno, espessura) / 4;

helper def: calcVolume(diametroExterno: Real, espessura: Real, kmInicial:
Real, kmFinal:
    Real): Real =
    thisModule.area(diametroExterno, espessura) *
    thisModule.comprimento(kmInicial,
        kmFinal);

helper def: totalVolume(pipe: PipeLang!Pipeline): Real =
    thisModule.calcVolume(pipe.segment.diameter,
    pipe.segment.thickness, pipe.elevationprofile.initialKm ,
    pipe.elevationprofile.finalKm);
```

```

helper context PipeLang!Fluid def:calcKMInterface() : Real =
  if thisModule.numFluids=1 then
    thisModule.totalVolume(self.refImmediateComposite())
  else
    if thisModule.numFluids>1 then
      PipeLang!Fluid.allInstances().asSequence().subSequence(1,
PipeLang!Fluid.allInstances().asSequence().indexOf(self))->iterate(e; soma:
Real = 0 |
        if e.volume <> -1.0 then
          (soma +
e.volume)/thisModule.totalVolume(self.refImmediateComposite()) *
self.refImmediateComposite().elevationprofile.finalKm
        else

self.refImmediateComposite().elevationprofile.finalKm
        endif
      )
    else
      self.refImmediateComposite().elevationprofile.finalKm
    endif
  endif;

helper context PipeLang!Fluid def: fluidVolume(): Real =
  if thisModule.numFluids=1 then
    thisModule.totalVolume(self.refImmediateComposite())
  else
    if thisModule.numFluids>1 then
      PipeLang!Fluid.allInstances().asSequence().subSequence(1,
PipeLang!Fluid.allInstances().asSequence().indexOf(self))->iterate(e; soma:
Real = 0 |
        if e.volume <> -1.0 then
          soma + e.volume
        else

thisModule.totalVolume(self.refImmediateComposite()) - soma
        endif
      )
    else
      self.volume
    endif
  endif;

```

```

helper context PipeLang!Fluid def: getInitialKm(): Real =
  let index : Integer =
    PipeLang!Fluid.allInstances().asSequence().indexOf(self) in
    if index = 1 then
      self.refImmediateComposite().elevationprofile.initialKm

    else

PipeSim!Fluid.allInstances().asSequence().at(index-1).kmInterface
endif;

helper def: getStation(firstKm:Real, lastKm:Real):
Sequence(PipeLang!Station) =
  PipeLang!Station.allInstances()->iterate(st; conjunto:
Sequence(PipeLang!Station) = Sequence{} |
  if st.km >= firstKm and st.km <= lastKm then
    conjunto->append(st)
  else
    conjunto
  endif
);

helper def: getInstrument(firstKm:Real, lastKm:Real):
Sequence(PipeLang!Instrument) =
  PipeLang!Instrument.allInstances()->iterate(inst; conjunto:
Sequence(PipeLang!Instrument) = Sequence{} |
  if inst.km >= firstKm and inst.km <= lastKm then
    conjunto->append(inst)
  else
    conjunto
  endif
);

helper def: getSegments(pipe:PipeLang!Pipeline) : Sequence(PipeSim!Segment)
=
  let firstKm : Real = pipe.elevationprofile.initialKm in
  let lastKm : Real = pipe.elevationprofile.finalKm in
  PipeSim!Segment.allInstances()->iterate(seg; conjunto:
Sequence(PipeLang!Segment) = Sequence{} |
  if seg.initialKm >= firstKm and seg.finalKm <= lastKm then
    conjunto->append(seg)
  else
    conjunto
  endif
);

helper def: getCalculateGH() : Sequence(PipeSim!CalculateGH) =

```

```

        PipeLang!CalculateGH.allInstances()->iterate(gh; conjunto:
Sequence(PipeLang!CalculateGH) = Sequence{} |
        conjunto->append(gh)

);

helper def: println(enu: OclAny): OclAny =
    enu.debug();
-----

rule PipelineSystem2PipelineSystem {
    from
        s: PipeLang!PipelineSystem
    to
        t: PipeSim!PipelineSystem (
            name <- s.name,
            pipeline <- s.pipeline
        )
}
-- fazer o mapeamento das classes que são iguais
rule Station2Station {
    from
        s: PipeLang!Station
    to
        t: PipeSim!Station (
            name <- s.name,
            km <- s.km,
            direction <- s.direction
        )
}

rule Instrument2Instrument {
    from
        s: PipeLang!Instrument
    to
        t: PipeSim!Instrument (
            tag <- s.tag,
            value <- s.value,
            type <- s.type,
            km <- s.km
        )
}

```

```

rule Fluid2Fluid{
  from
    flu: PipeLang!Fluid
  to
    t1: PipeSim!Fluid (
      name <- flu.name,
      density <- flu.density,
      id <- flu.id,
      viscosity <- flu.viscosity,
      vaporPressure <- flu.vaporPressure,
      volume <- flu.fluidVolume().round(),
      kmInterface <- flu.calcKMInterface().round()
    ),
    seg:PipeSim!Segment (
      name <- flu.refImmediateComposite().name + '_' +
thisModule.idSeq,
      diameter <- flu.refImmediateComposite().segment.diameter,
      thickness <-
flu.refImmediateComposite().segment.thickness,
      initialKm <- flu.getInitialKm(),
      finalKm <- flu.calcKMInterface().round(),
      fluid <- flu,
      station <- thisModule.getStation(seg.initialKm,
seg.finalKm),
      instrument <- thisModule.getInstrument(seg.initialKm,
seg.finalKm)
    )
    do {
      thisModule.idSeq <- thisModule.idSeq + 1 ;

      thisModule.println(thisModule.idSeq);
    }
}

rule calculateGH {
  from
    s: PipeLang!CalculateGH
  to
    t : PipeSim!CalculateGH (
      )
}

```

```

abstract rule Pipeline2Pipeline {
  from
    s: PipeLang!Pipeline
  to
    t: PipeSim!Pipeline (
      name <- s.name,
      rugosity <- s.rugosity,
      elasticity <- s.elasticity,
      poisson <- s.poisson,
      pipeState <- s.pipeState,
      elevationprofile <- s.elevationprofile
    )
}

rule PipelineSegments extends Pipeline2Pipeline{
  from
    s: PipeLang!Pipeline
  to
    t : PipeSim!Pipeline (
      segment <- thisModule.getSegments(s),
      calculategh <- s.calculategh
    )
}

rule Elevation2Elevation {
  from
    s: PipeLang!ElevationProfile
  to
    t: PipeSim!ElevationProfile (
      name <- s.name,
      initialKm <- s.initialKm,
      initialElevation <- s.initialElevation,
      finalKm <- s.finalKm,
      finalElevation <- s.finalElevation,
      invertProfile <- s.invertProfile
    )
}

```

Algumas transformações mostradas acima, são destacadas abaixo. Para facilitar o uso pelo usuário final do PipeLang, DSL do PIM, foi criado um artifício para o cálculo do volume durante a transformação.

O helper chamado `totalVolume` realiza o cálculo do volume total do Pipeline. Uma função que foi definida para facilitar a criação do modelo no PIM foi definir que ao entrar com o volume igual a -1 no volume do fluido, esse valor passa a receber o volume total do duto caso

exista apenas um fluido. Caso exista mais de um fluido, todos os fluidos podem receber o valor de volume e o último fluido pode ficar com o valor de -1. Nesse caso a transformação calcula o volume remanescente para o usuário.

Também é realizada pelo helper `calcKMInterface` o cálculo de da posição de cada interface. Por exemplo, se um pipeline tem um comprimento de 100 km e um volume de 1.000 m³, então se houver um fluido com volume de 500 m³, a propriedade `kmInterface` do fluido será calculada para 50 km.

Dessa forma os segmentos novos são criados a partir dos fluidos criados. Cada segmento novo recebe um nome que é igual ao do segmento original concatenado com um número sequencial.

O PSM gerado pode ser visualizado na figura abaixo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PipeSim3:PipelineSystem xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:PipeSim3="http://www.example.org/PipeSim3" name="OSBRA">
  <pipeline name="PLN2PIRp" rugosity="0.045" elasticity="1899731.0"
poisson="0.3">
    <segment diameter="20.0" thickness="0.25" name="PLN2PIRp_0"
initialKm="0.0" finalKm="52.0">
      <station km="0.0" name="PLN" direction="Sending"/>
      <station km="45.0" name="XV" direction="Sending"/>
      <fluid name="S500" id="2702200001" density="840.0" volume="10000.0"
viscosity="345.0" vaporPressure="0.0" kmInterface="52.0"/>
      <instrument tag="PT001" type="Pressure" km="0.0">
        <value>72.0</value>
      </instrument>
      <instrument tag="FT001" type="Flow" km="0.0">
        <value>900.0</value>
      </instrument>
    </segment>
    <segment diameter="20.0" thickness="0.25" name="PLN2PIRp_1"
initialKm="52.0" finalKm="99.0">
      <station km="95.0" name="PIR" direction="Receiving"/>
      <fluid name="GLNA" id="2702200001" density="720.0" volume="9075.0"
viscosity="1.0" vaporPressure="0.0" kmInterface="99.0"/>
      <instrument tag="PT002" type="Pressure" km="99.0">
        <value>7.0</value>
      </instrument>
    </segment>
  </pipeline>
</PipeSim3:PipelineSystem>
```



```

    </segment>
    <elevationprofile initialKm="0.0" initialElevation="700.0"
finalKm="99.0" finalElevation="854.0" name="PLN2PIR"/>
    <calculategh/>
  </pipeline>
</PipeSim3:PipelineSystem>

```

Figura 10 - XMI da saída do Projeto ATL.

Na visualização do eCore, a saída do projeto ATL pode ser visualizada abaixo.

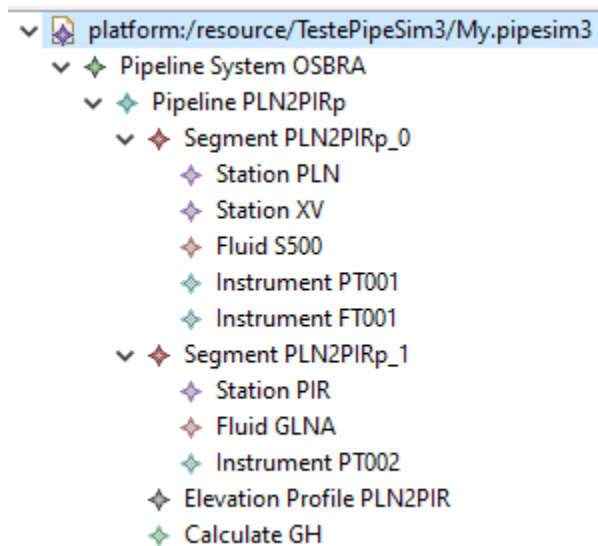


Figure 11 : Visualização do modelo Ecore

Uma vez que o PSM tenha sido gerado, faz-se necessário a transformação M2T definida pelo Acceleo, cujo código está sendo mostrado abaixo.

```

[comment encoding = UTF-8 /]
[module generate('http://www.example.org/PipeSim3')]

[query public hasGH(pipe:Pipeline) : Boolean =
    not pipe.calculategh->isEmpty()
/]

[query public getFirstPipelineSystem(pipe:Pipeline) : PipelineSystem =
    pipe.ancestors(PipelineSystem)->at(1)
/]

```

```

[template public generatePipelineSystem(aPipelineSystem : PipelineSystem)]
[comment @main/]

[UnitConversion('UnitConversion')/]
[generateInicializaGeral(aPipelineSystem)/]

[comment para cada sistema podem haver um ou mais pipelines/]
[for (p : Pipeline | aPipelineSystem.pipeline)]
    [generatePipeline(p)/]
    [for (s : Segment | p.segment)]
        [generateSegment(s)/]
        [for (st : Station | s.station)]
            [generateStations(st)/]
        [/for]
        [for (inst : Instrument | s.instrument)]
            [generateInstruments(inst)/]
        [/for]
        [for (f : Fluid | s.fluid)]
            [generateFluids(f)/]
        [/for]
    [/for]
    [for (e : ElevationProfile | p.elevationprofile)]
        [generateElevationProfile(e)/]
    [/for]
[/for]

[comment na versão atual do PSM, somente um pipelineSystem é possível/]
[file (aPipelineSystem.name + '.vb', false, 'UTF-8')]

Public Class [aPipelineSystem.name/]
[comment para cada sistema, podem existir mais de um pipeline /]

    Public name as String = "[aPipelineSystem.name/]"
    [for (pipe : String | aPipelineSystem.pipeline.name)]
        Public [pipe/] as New [pipe/]
    [/for]
    Public listPipeline as New List(of Object)({ [for
(aPipelineSystem.pipeline.name) separator(', ')] [self/][ /for] })

    Public Sub New()

    End Sub

    Public Sub New(name as String)
        Me.name = name
    End Sub

End Class
[/file]

```

```
[/template]

[template public generatePipeline(aPipe:Pipeline) ? (not hasGH(aPipe)) post
(trim())]

[file (aPipe.name + '.vb', false, 'UTF-8')]

Public Class [aPipe.name/]

    Public name as String = "[aPipe.name/]"
    Public elasticity = [aPipe.elasticity/]
    Public poisson = [aPipe.poisson/]
    Public rugosity = [aPipe.rugosity/]
    [for (seg : String | aPipe.segment.name)]
    Public [seg/] As New[seg/]
    [/for]
    Public listSegments As New List(of Object)({ [for
(aPipe.segment.name) separator(', ')] [self/][[/for] ])
    Public elevationProfile As [aPipe.elevationprofile.name/]

    Public Sub New()

    End Sub

    Public Sub New(name as String,
                    elasticity as Double,
                    poisson as Double,
                    rugosity as Double,
                    listSegments as List(of Object),
                    elevationProfile as Object)

        Me.name = name
        Me.elasticity = elasticity
        Me.poisson = poisson
        Me.rugosity = rugosity
        Me.listSegments = listSegments
        Me.elevationProfile = elevationProfile

    End Sub

End Class
[/file]
[/template]
```

```

[template public generatePipeline(aPipe:Pipeline) ? ( hasGH(aPipe)) post
(trim())]

[file (aPipe.name + '.vb', false, 'UTF-8')]
Public Class [aPipe.name/]

    Public name as String = "[aPipe.name/]"
    Public elasticity = [aPipe.elasticity/]
    Public poisson = [aPipe.poisson/]
    Public rugosity = [aPipe.rugosity/]
    [for (seg : String | aPipe.segment.name)]
    Public [seg/] As New [seg/]
    [/for]
    Public listSegments As New List(of Object)({ [for
(aPipe.segment.name) separator(', ')] [self/][[/for] ]})
    Public elevationProfile As New [aPipe.elevationprofile.name/]
    Public f As new FrictionFactor

    Public Sub New()

    End Sub
    Public Sub New(name as String,
                    elasticity as Double,
                    poisson as Double,
                    rugosity as Double,
                    listSegments as List(of Object),
                    elevationProfile as Object)

        Me.name = name
        Me.elasticity = elasticity
        Me.poisson = poisson
        Me.rugosity = rugosity
        Me.listSegments = listSegments
        Me.elevationProfile = elevationProfile
    End Sub
End Class
[/file]
[if (hasGH(aPipe))]
    [generateReynolds('Reynolds')/]
    [generateCalculations('Calculations')/]
    [generateFrictionFactor('FrictionFactor')/]
    [generateCalculateGH('CalculateGH',getFirstPipelineSystem(aPipe))/]
[/if]

[/template]

```

```

[template public generateSegment(aSegment:Segment) post (trim())]

[file (aSegment.name + '.vb', false, 'UTF-8')]

Public Class [aSegment.name/]

    Public name as String = "[aSegment.name/]"
    Public diameter = [aSegment.diameter/]
    Public thickness = [aSegment.thickness/]
    Public firstKm = [aSegment.initialKm/]
    Public lastKm = [aSegment.finalKm/]
    [for (seg : String | aSegment.station.name)]
    Public [seg/] As New [seg/]
    [/for]
    [for (seg : String | aSegment.instrument.tag)]
    Public [seg/] As New [seg/]
    [/for]
    Public listStation As New List(of Object)({ [for
(aSegment.station.name) separator(', ')] [self/][[/for] })
    Public listInstrument As New List(of Object)({ [for
(aSegment.instrument.tag) separator(', ')] [self/][[/for] })
    Public fluid As New [aSegment.fluid.name/]

    Public Sub New()

    End Sub
    Public Sub New(Byval name as String,
                    Byval diameter as Double,
                    Byval thickness as Double,
                    Byval firstKm as Double,
                    Byval lastKm as Double,
                    Byval listStation as List(of Object),
                    Byval listInstrument as List(of Object),
                    Byval fluid as Object)

        Me.name = name
        Me.diameter = diameter
        Me.thickness = thickness
        Me.firstKm = firstKm
        Me.lastKm = lastKm
        Me.listStation = listStation
        Me.listInstrument = listInstrument
        Me.fluid = fluid
    End Sub
End Class
[/file]
[/template]

```

```

[template public
generateElevationProfile(aElevationProfile:ElevationProfile) post (trim())]

[file (aElevationProfile.name + '.vb', false, 'UTF-8')]

Public Class [aElevationProfile.name/]

    Public name as String = "[aElevationProfile.name/]"
    Public initialElevation as Double =
[aElevationProfile.initialElevation/]
    Public finalElevation as Double = [aElevationProfile.finalElevation/]
    Public firstKm as Double = [aElevationProfile.initialKm/]
    Public lastKm as Double = [aElevationProfile.finalKm/]
    Public invertProfile as Boolean = [aElevationProfile.invertProfile/]

    Public Sub New()

    End Sub

    Public Sub New(Byval name as String,
                    Byval initialElevation as Double,
                    Byval finalElevation as Double,
                    Byval firstKm as Double,
                    Byval lastKm as Double,
                    Byval invertProfile as Boolean
                    )

        Me.name = name
        Me.initialElevation = initialElevation
        Me.finalElevation = finalElevation
        Me.firstKm = firstKm
        Me.lastKm = lastKm
        Me.invertProfile = invertProfile

    End Sub

End Class

[/file]
[/template]

[template public generateStations(aStation:Station) post (trim())]

[file (aStation.name + '.vb', false, 'UTF-8')]

Public Class [aStation.name/]

    Public name as String = "[aStation.name/]"
    Public km as Double = [aStation.km/]

```

```

Public direction as String = "[aStation.direction/]"

Public Sub New()

End Sub

Public Sub New(Byval name as String,
               Byval km as Double,
               Byval direction as String)

    Me.name = name
    Me.km = km
    Me.direction = direction

End Sub

End Class

[/file]
[/template]

[template public generateInstruments(aInstrument:Instrument) post (trim())]

[file (aInstrument.tag + '.vb', false, 'UTF-8')]

Public Class [aInstrument.tag/]

    Public name as String = "[aInstrument.tag/]"
    Public km as Double = [aInstrument.km/]
    Public type as String = "[aInstrument.type/]"
    Public value as Double = [aInstrument.value/]

    Public Sub New()
    End Sub

    Public Sub New(Byval name as String,
                  Byval km as Double,
                  Byval type as String,
                  Byval value as Double)

        Me.name = name
        Me.km = km
        Me.type = type
        Me.value = value

    End Sub

End Class

[/file]
[/template]

```

```

[template public generateFluids(aFluid:Fluid) post (trim())]

[file (aFluid.name + '.vb', false, 'UTF-8')]

Public Class [aFluid.name/]

    Public name as String = "[aFluid.name/]"
    Public density as Double = [aFluid.density/]
    Public id as String = "[aFluid.id/]"
    Public kmInterface as Double = [aFluid.kmInterface/]
    Public vaporPressure as Double = [aFluid.vaporPressure/]
    Public viscosity as Double = [aFluid.viscosity/]
    Public volume as Double = [aFluid.volume/]

    Public Sub New()

    End Sub

    Public Sub New(Byval name as String,
                    Byval id as String,
                    Byval viscosity as Double,
                    Byval density as Double,
                    Byval kmInterface as Double,
                    Byval vaporPressure as Double,
                    Byval volume as Double)

        Me.name = name
        Me.density = density
        Me.id = id
        Me.kmInterface = kmInterface
        Me.vaporPressure = vaporPressure
        Me.viscosity = viscosity
        Me.volume = volume

    End Sub

End Class

[/file]
[/template]

[comment]
-----
---- Static Class -----
[/comment]

[comment]
Inicializa geral

```



```

[/comment]

[template public generateInicializaGeral(pipe : PipelineSystem) ]
[comment TODO Auto-generated template stub/]
[file ('InicializaGeral.vb' , false, 'UTF-8')]
Public Class InicializaGeral

    Public [pipe.name/] As New [pipe.name/]
[comment]
    [for (p : Pipeline | pipe.pipeline)]
    Public [p.name/] As New [p.name/]
    [/for]

[for (p : Pipeline | pipe.pipeline)]
    Public [p.name/] As New [p.name/]
    [for (s : Segment | p.segment)]
    Public [s.name/] As New [s.name/]
        [for (st : Station | s.station)]
        Public [st.name/] As New [st.name/]
        [/for]
        [for (inst : Instrument | s.instrument)]
        Public [inst.tag/] As New [inst.tag/]
        [/for]
        [for (f : Fluid | s.fluid)]
        Public [f.name/] As New [f.name/]
        [/for]
    [/for]
    [for (e : ElevationProfile | p.elevationprofile)]
    Public [e.name/] As New [e.name/]
    [/for]
[/for][/comment]

    Public Sub New()

    End Sub

'[/protected (pipe.name)]
'Main code
'[/protected]

End Class
[/file]
[/template]

[comment]
Most import function of this work!
[/comment]

```

```

[template public generateCalculateGH(p : String, aPipelineSystem:
PipelineSystem) ]
[comment TODO Auto-generated template stub/]
[file (p + '.vb', false, 'UTF-8')]
Public Class CalculateGH

    Public listaInstrumentosPressao As New List(Of Object)
    Public listaInstruemntosVazao As New List(Of Object)

    ' construtor
    Public Sub New()

    End Sub

    Public Sub CalculateAllGH(Byval pipelineSystem as Object)
        Console.WriteLine("Sistema " & pipelineSystem.name)
        For Each pipe in pipelineSystem.listPipeline
            CalculateGH(pipe)
        Next
    End Sub

    Public Sub CalculateGH(ByVal pipeline as Object)

        Console.WriteLine("Duto " & pipeline.name)

        Dim finalPressure As Double = 0

        'ordena a lista de segmentos pelo primeiro km
        Dim segmentosOrdenado As IEnumerable(Of Object) =
CType(pipeline.listSegments, IEnumerable(Of Object)).OrderBy(Function(x)
x.FirstKm).ToList

        'pega lista de instrumentos
        listaInstrumentosPressao = segmentosOrdenado.First.listInstrument

        'pega a cota inicial
        Dim cotaInicial As Double =
pipeline.elevationProfile.initialElevation

        'Pega o valor do primeiro instrumento de pressão
        Dim firstInstPressure =
listaInstrumentosPressao.OrderBy(Function(x) x.km).Where(Function(y)
y.type.ToString = "Pressure").First.value

        'Pega o valor do primeiro instrumento de vazão
        Dim firstInstFlow = listaInstrumentosPressao.OrderBy(Function(x)
x.km).Where(Function(y) y.type.ToString = "Flow").First.value

        'Pega a densidade do fluido

```

```

Dim dens As Double = segmentosOrdenado.First.fluid.density / 1000

'Calcula o head inicial
Dim headInicial As Double = 10 * firstInstPressure / dens +
cotaInicial

Dim headFinal As Double = 0

Console.WriteLine("Valor do head inicial = " &
Math.Round(headInicial))
Console.WriteLine("Valor da pressao inicial = " &
firstInstPressure)

'Calcula o número de Reynolds para cada segmento e calcula o fator
de atrito
For i As Integer = 0 To segmentosOrdenado.Count - 1

    Dim visc As Double = segmentosOrdenado(i).fluid.viscosity

    dens = segmentosOrdenado(i).fluid.density / 1000

    Dim reynolds As New Reynolds(visc, firstInstFlow,
segmentosOrdenado(i).diameter, segmentosOrdenado(i).thickness)

    Dim f As New FrictionFactor(visc, firstInstFlow,
segmentosOrdenado(i).diameter, segmentosOrdenado(i).thickness,
pipeline.rugosity)

    Dim fatorAtrito As Double =
f.calculateFrictionFactor(segmentosOrdenado(i).diameter,
segmentosOrdenado(i).thickness)

    Dim diametroInterno As Double =
UnitConversion.ConvertInches2Meter(reynolds.calcInternalDiameter(segmentosO
rdenado(i).diameter, segmentosOrdenado(i).thickness))

    Dim velocity As Double =
reynolds.calcVelocity(segmentosOrdenado(i).diameter,
segmentosOrdenado(i).thickness, firstInstFlow)

    Dim hf As Double = fatorAtrito * (segmentosOrdenado(i).lastKm -
segmentosOrdenado(i).firstKm) * 1000 / diametroInterno * velocity ^ 2 / (2
* 9.81)

    If i = 0 Then
        headFinal = (headInicial - hf)
        finalPressure = convertHead2Pressure(headFinal, dens)
    Else
        headInicial = convertPressure2Head(finalPressure, dens)
    End If
End For

```

```

        headFinal = headInicial - hf
        finalPressure = convertHead2Pressure(headFinal, dens)
    End If

    Console.WriteLine("Numero de Reynolds para o segmento " &
segmentosOrdenado(i).name & " = " & Math.Round(reynolds.getReynolds()))
    Console.WriteLine("Fator de atrito para o segmento " &
segmentosOrdenado(i).name & " = " & Math.Round(fatorAtrito, 4))
    Console.WriteLine("Head inicial = " & Math.Round(headInicial))
    Console.WriteLine("Head final = " & Math.Round(headFinal))
    Console.WriteLine("Densidade = " & Math.Round(dens,3))
Next

    Console.WriteLine("Pressão final = " &
Math.Round(convertHead2Pressure(headFinal -
pipeline.elevationProfile.finalElevation, dens), 2))

End Sub

Public Function convertPressure2Head(Byval pressure as double,
                                     Byval density
as double) as double

    Return 10 * pressure / density

End Function

Public Function convertHead2Pressure(Byval head as double,
                                     Byval density
as double) as double

    Return density * head / 10

End Function

End Class
[/file]
[/template]

[template public generateCalculations(p : String) ]
    [comment TODO Auto-generated template stub/]
[file (p + '.vb', false, 'UTF-8')]

Public MustInherit Class Calculations

    ''' <summary>
    ''' Calculate velocity in SI
    ''' </summary>

```

```

''' <param name="externalDiameter">in inches</param>
''' <param name="thickness">in inches</param>
''' <returns>Returns velocity in m/s</returns>
Public Function calcVelocity(ByVal externalDiameter As Double,
                             ByVal thickness As Double,
                             Byval flow as Double) As
Double

    Dim internalDiameter As Double =
UnitConversion.ConvertInches2Meter(
    calcInternalDiameter(externalDiameter, thickness))

    Dim area As Double = calcArea(internalDiameter)

    Dim flowPerSecond As Double = flow / 3600

    Return flowPerSecond / area

End Function

''' <summary>
'''
''' </summary>
''' <param name="Dint"> Internal Diameter </param>
''' <returns>Area of pipeline section</returns>
Private Function calcArea(ByVal Dint As Double) As Double

    Return Math.PI * Dint ^ 2 / 4

End Function

''' <summary>
''' Calculate internal diameter
''' </summary>
''' <param name="externalDiameter"> External Diameter</param>
''' <param name="thickness">Thickness of the pipeline wall</param>
''' <returns>internal diameter</returns>
Public Function calcInternalDiameter(ByVal externalDiameter As Double,
                                     ByVal thickness As Double)

    Return externalDiameter - 2 * thickness

End Function

End Class

[/file]
[/template]

```

```

[template public generateReynolds(p : String) ]
[file (p + '.vb', false, 'UTF-8')]

Public Class Reynolds
    Inherits Calculations

    Public visc As Double
    Public velocity As Double
    Public internalDiameter As Double

    Public Sub New()

    End Sub

    Public Sub New(ByVal visc As Double,
                    ByVal flow As Double,
                    ByVal externalDiameter As Double,
                    ByVal thickness As Double)

        Me.visc = visc
        Me.velocity = calcVelocity(externalDiameter, thickness, flow)
        Me.internalDiameter =
UnitConversion.ConvertInches2Meter(calcInternalDiameter(externalDiameter,
thickness))

    End Sub

    ''' <summary>
    ''' Calculates adimensional Reynolds Number
    ''' </summary>
    ''' <param name="velocity">must be in SI (m/s)</param>
    ''' <param name="internalDiameter">must be in SI (m)</param>
    ''' <returns>Reynolds Number</returns>
    Public Function calcReynolds(ByVal velocity As Double,
                                ByVal internalDiameter As Double)

        Return velocity * internalDiameter / (visc * 0.000001)

    End Function

    Public Function getReynolds()

        Return velocity * internalDiameter / (visc * 0.000001)

    End Function
End Class
[/file]
[/template]

```

```

[template public generateFrictionFactor(p : String) ]
    [comment TODO Auto-generated template stub/]
[file (p + '.vb', false, 'UTF-8')]
Public Class FrictionFactor
    Inherits Calculations

    Public rey as Reynolds
    Public reynoldsNumber as double
    Public visc as Double
    Public flow as Double
    Public velocity as Double
    Public internalDiameter as Double
    Public externalDiameter as Double
    Public thickness as Double
    Public rugosity As Double

    Public Sub New()

    End Sub

    Public Sub New(Byval visc as Double,
                    Byval flow as Double,
                    Byval externalDiameter as Double,
                    Byval thickness as Double,
                    ByVal rugosity As Double)

        Me.visc = visc
        Me.flow = flow
        Me.externalDiameter = externalDiameter
        Me.thickness = thickness

        velocity = calcVelocity(externalDiameter, thickness, flow)

        internalDiameter =
UnitConversion.ConvertInches2Meter(calcInternalDiameter(externalDiameter,
thickness))

        rey = new Reynolds(visc, flow, externalDiameter, thickness)
        reynoldsNumber = rey.getReynolds()

    End Sub

    ''' <summary>
    ''' Calcula o fator de atrito
    ''' </summary>
    ''' <param name="externalDiameter">em polegadas </param>
    ''' <param name="thickness">em polegadas</param>

```

```

''' <returns></returns>
Public Function calculateFrictionFactor(ByVal externalDiameter As
Double,
                                     ByVal thickness As Double) As
Double

    If flow = 0 Then Return 0

    'Dim velocity As Double = calcVelocity(externalDiameter, thickness)

    'Dim internalDiameter As Double =
UniConversion.ConvertInches2Meter(calcInternalDiameter(externalDiameter,
thickness))

    'Dim reynolds As Double = calcReynolds(velocity, internalDiameter)

    Select Case reynoldsNumber
        Case < 1400
            Return calcLaminarFrictionFactor(velocity,
internalDiameter)
        Case > 2750
            Return calcTurbulentFrictionFactor(velocity,
internalDiameter)
        Case Else
            Return 0.0457
    End Select

End Function

''' <summary>
''' Calc friction factor in case number of Reynolds is below lower
limit
''' units are the same as described in reynolds calculation
''' </summary>
''' <param name="velocity">must be in SI (m/s)</param>
''' <param name="internalDiameter">must be in SI</param>
''' <returns></returns>
Private Function calcLaminarFrictionFactor(ByVal velocity As Double,
                                     ByVal internalDiameter As Double) As
Double

    'Dim reynolds As Double = calcReynolds(velocity, internalDiameter)

    Return 64 / reynoldsNumber

End Function

Private Function calcLaminarFrictionFactor() As Double

```



```

        Return 64 / reynoldsNumber

    End Function

    ''' <summary>
    ''' Calculate friction factor for turbulent condition according to
    miller
    ''' </summary>
    ''' <param name="velocity">must be in SI</param>
    ''' <param name="internalDiameter">must be in SI</param>
    ''' <returns></returns>
    Private Function calcTurbulentFrictionFactor(ByVal velocity As Double,
                                                ByVal internalDiameter As
    Double) As Double

        'Dim Reynolds As Double = calcReynolds(velocity, internalDiameter)

        Return 0.25 * Math.Log10(rugosity / 1000 / internalDiameter / 3.7 +
    5.74 * (reynoldsNumber ^ -0.9)) ^ -2

    End Function

    Private Function calcTurbulentFrictionFactor() As Double

        Return 0.25 * Math.Log10(rugosity / 1000 / internalDiameter / 3.7 +
    5.74 * (reynoldsNumber ^ -0.9)) ^ -2

    End Function
End Class
[/file]
[/template]

[template public UnitConversion(p : String) ]
    [comment TODO Auto-generated template stub/]
[file (p + '.vb', false, 'UTF-8')]
Public Class UnitConversion

    Public Shared Function ConvertInches2Meter(ByVal value As Double) As
    Double

        Return value * 0.0254

    End Function

End Class
[/file]
[/template]

```

Para testar a DSL os arquivos gerados foram inseridos em um projeto Vb.Net no Visual Studio 2017 como um projeto de console. Tal como pode ser visto na Figura 9. Com exceção do arquivo Module1.vb que é criado automaticamente com o projeto, todos os demais arquivos foram importados da saída do Acceleio.

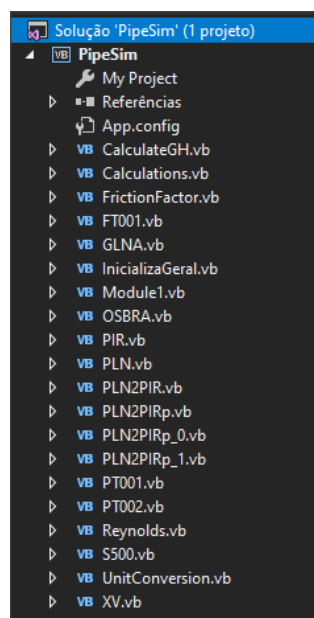


Figura 9: Arquivos gerados automaticamente, com exceção do Module1.vb

O módulo principal, module1, contém o seguinte código:

```

Module Module1

    Sub Main()

        Dim init As New InicializaGeral

        Dim GH1 As New CalculateGH()

        GH1.CalculateAllGH(init.OSBRA)

        Console.ReadKey()

    End Sub

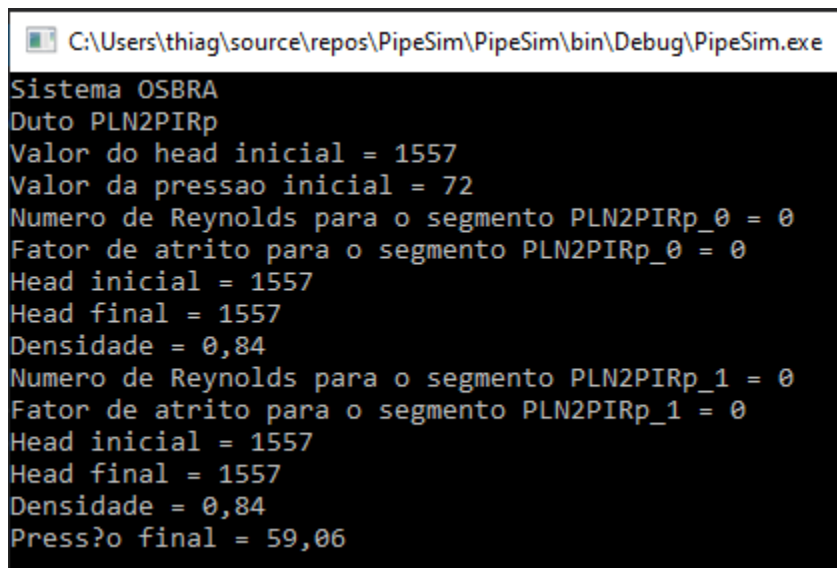
End Module

```

A classe inicializaGeral, apenas instancia o PipelineSystem que no exemplo chama-se OSBRA. As demais classes são instanciadas de acordo com as associações do modelo PSM.

A classe GH1 contém o código necessário para executar a funcionalidade CalculateGH.

Ao executar o programa a saída pode ser observada na Figura 10.



```

C:\Users\thiag\source\repos\PipeSim\PipeSim\bin\Debug\PipeSim.exe
Sistema OSBRA
Duto PLN2PIRp
Valor do head inicial = 1557
Valor da pressao inicial = 72
Numero de Reynolds para o segmento PLN2PIRp_0 = 0
Fator de atrito para o segmento PLN2PIRp_0 = 0
Head inicial = 1557
Head final = 1557
Densidade = 0,84
Numero de Reynolds para o segmento PLN2PIRp_1 = 0
Fator de atrito para o segmento PLN2PIRp_1 = 0
Head inicial = 1557
Head final = 1557
Densidade = 0,84
Press?o final = 59,06

```

Figura 10: Saída do programa PipeSim para o modelo OSBRA.

Para testar alguns casos, o arquivo xmi foi modificado para simular situações em que o duto esteja parado em operação e com valores de densidade diferentes.

Caso 1: Primeiro caso com massas específicas iguais e vazão de 900 m³/h.

```
<?xml version="1.0" encoding="UTF-8"?>
<PipeLang:PipelineSystem xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:PipeLang="http://www.example.org/PipeLang" name="OSBRA">
  <pipeline name="PLN2PIRp" rugosity="0.045" elasticity="1899731.0"
poisson="0.3">
    <segment diameter="20.0" thickness="0.25" name="PLN2PIRs"/>
    <elevationprofile initialKm="0.0" initialElevation="700.0"
name="PLN2PIR" finalKm="99.0" finalElevation="854.0"/>
    <instrument tag="PT001" type="Pressure" km="0.0">
      <value>72.0</value>
    </instrument>
    <instrument tag="FT001" type="Flow" km="0.0">
      <value>900.0</value>
    </instrument>
    <instrument tag="PT002" type="Pressure" km="99.0">
      <value>7.0</value>
    </instrument>
    <station km="0.0" name="PLN" direction="Sending"/>
    <station km="45.0" name="XV" direction="Sending"/>
    <station km="95.0" name="PIR" direction="Receiving"/>
    <fluid name="S500" density="840.0" volume="10000.0" id="2702200001"
viscosity="345.0" vaporPressure="0.0"/>
    <fluid name="GLNA" density="720.0" volume="-1.0" id="2702200001"
viscosity="1.0" vaporPressure="0.0"/>
    <calculategh/>
  </pipeline>
</PipeLang:PipelineSystem>
```

```
C:\Users\thiag\source\repos\PipeSim\PipeSim\bin\Debug\PipeSim.exe
Sistema OSBRA
Duto PLN2PIRp
Valor do head inicial = 1557
Valor da pressao inicial = 72
Numero de Reynolds para o segmento PLN2PIRp_0 = 1863
Fator de atrito para o segmento PLN2PIRp_0 = 0,0457
Head inicial = 1557
Head final = 1145
Densidade = 0,84
Numero de Reynolds para o segmento PLN2PIRp_1 = 642661
Fator de atrito para o segmento PLN2PIRp_1 = 0,0125
Head inicial = 1145
Head final = 1043
Densidade = 0,84
Press?o final = 15,92
```

Figura 11: Saída do caso 1

A saída para esse caso apresenta o mesmo head na interface e um head final menor como era de se esperar dadas as condições iniciais mencionadas para este caso.

Caso 2: Primeiro caso com massas específicas iguais e vazão de 0 m³/h, duto parado.

```
<?xml version="1.0" encoding="UTF-8"?>
<PipeLang:PipelineSystem xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:PipeLang="http://www.example.org/PipeLang" name="OSBRA">
  <pipeline name="PLN2PIRp" rugosity="0.045" elasticity="1899731.0"
poisson="0.3">
    <segment diameter="20.0" thickness="0.25" name="PLN2PIRs"/>
    <elevationprofile initialKm="0.0" initialElevation="700.0"
name="PLN2PIR" finalKm="99.0" finalElevation="854.0"/>
    <instrument tag="PT001" type="Pressure" km="0.0">
      <value>72.0</value>
    </instrument>
    <instrument tag="FT001" type="Flow" km="0.0">
      <value>0.0</value>
    </instrument>
    <instrument tag="PT002" type="Pressure" km="99.0">
      <value>7.0</value>
    </instrument>
```

```

    <station km="0.0" name="PLN" direction="Sending"/>
    <station km="45.0" name="XV" direction="Sending"/>
    <station km="95.0" name="PIR" direction="Receiving"/>
    <fluid name="S500" density="840.0" volume="10000.0" id="2702200001"
viscosity="345.0" vaporPressure="0.0"/>
    <fluid name="GLNA" density="840.0" volume="-1.0" id="2702200001"
viscosity="1.0" vaporPressure="0.0"/>
    <calculategh/>
  </pipeline>
</PipeLang:PipelineSystem>

```

```

C:\Users\thiag\source\repos\PipeSim\PipeSim\bin\Debug\PipeSim.exe
Sistema OSBRA
Duto PLN2PIRp
Valor do head inicial = 1557
Valor da pressao inicial = 72
Numero de Reynolds para o segmento PLN2PIRp_0 = 0
Fator de atrito para o segmento PLN2PIRp_0 = 0
Head inicial = 1557
Head final = 1557
Densidade = 0,84
Numero de Reynolds para o segmento PLN2PIRp_1 = 0
Fator de atrito para o segmento PLN2PIRp_1 = 0
Head inicial = 1557
Head final = 1557
Densidade = 0,84
Press?o final = 59,06

```

Figura 12: Saída do caso 2 com vazão zero.

O duto estando parado e as densidades sendo iguais espera-se que o head seja constante.

Caso 3: Massas específicas diferentes, vazão zero.

```

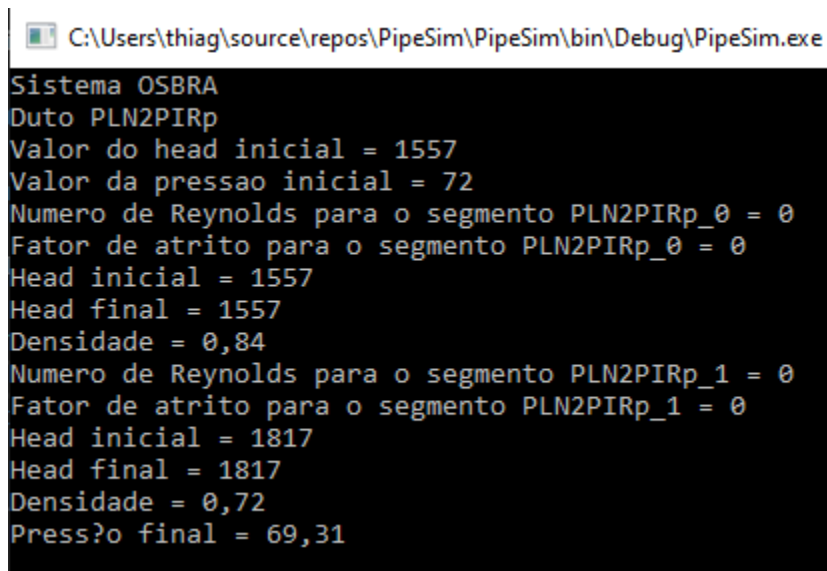
<?xml version="1.0" encoding="UTF-8"?>
<PipeLang:PipelineSystem xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:PipeLang="http://www.example.org/PipeLang" name="OSBRA">
  <pipeline name="PLN2PIRp" rugosity="0.045" elasticity="1899731.0"
poisson="0.3">

```

```

    <segment diameter="20.0" thickness="0.25" name="PLN2PIRs"/>
    <elevationprofile initialKm="0.0" initialElevation="700.0"
name="PLN2PIR" finalKm="99.0" finalElevation="854.0"/>
    <instrument tag="PT001" type="Pressure" km="0.0">
        <value>72.0</value>
    </instrument>
    <instrument tag="FT001" type="Flow" km="0.0">
        <value>0.0</value>
    </instrument>
    <instrument tag="PT002" type="Pressure" km="99.0">
        <value>7.0</value>
    </instrument>
    <station km="0.0" name="PLN" direction="Sending"/>
    <station km="45.0" name="XV" direction="Sending"/>
    <station km="95.0" name="PIR" direction="Receiving"/>
    <fluid name="S500" density="840.0" volume="10000.0" id="2702200001"
viscosity="345.0" vaporPressure="0.0"/>
    <fluid name="GLNA" density="720.0" volume="-1.0" id="2702200001"
viscosity="1.0" vaporPressure="0.0"/>
    <calculategh/>
</pipeline>
</PipeLang:PipelineSystem>

```



```

C:\Users\thiag\source\repos\PipeSim\PipeSim\bin\Debug\PipeSim.exe
Sistema OSBRA
Duto PLN2PIRp
Valor do head inicial = 1557
Valor da pressao inicial = 72
Numero de Reynolds para o segmento PLN2PIRp_0 = 0
Fator de atrito para o segmento PLN2PIRp_0 = 0
Head inicial = 1557
Head final = 1557
Densidade = 0,84
Numero de Reynolds para o segmento PLN2PIRp_1 = 0
Fator de atrito para o segmento PLN2PIRp_1 = 0
Head inicial = 1817
Head final = 1817
Densidade = 0,72
Press?o final = 69,31

```

Figura 13: Saída do caso 3 com vazão zero, e massas específicas diferentes

Como a massa específica do segundo produto é menor espera-se que na interface ocorra um degrau positivo, pois o head é inversamente proporcional a densidade.

Caso 4: Massas específicas diferentes, vazão de 900 m³/h

```
<?xml version="1.0" encoding="UTF-8"?>
<PipeLang:PipelineSystem xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:PipeLang="http://www.example.org/PipeLang" name="OSBRA">
  <pipeline name="PLN2PIRp" rugosity="0.045" elasticity="1899731.0"
poisson="0.3">
    <segment diameter="20.0" thickness="0.25" name="PLN2PIRs"/>
    <elevationprofile initialKm="0.0" initialElevation="700.0" name="PLN2PIR"
finalKm="99.0" finalElevation="854.0"/>
    <instrument tag="PT001" type="Pressure" km="0.0">
      <value>72.0</value>
    </instrument>
    <instrument tag="FT001" type="Flow" km="0.0">
      <value>900.0</value>
    </instrument>
    <instrument tag="PT002" type="Pressure" km="99.0">
      <value>7.0</value>
    </instrument>
    <station km="0.0" name="PLN" direction="Sending"/>
    <station km="45.0" name="XV" direction="Sending"/>
    <station km="95.0" name="PIR" direction="Receiving"/>
    <fluid name="S500" density="840.0" volume="10000.0" id="2702200001"
viscosity="345.0" vaporPressure="0.0"/>
    <fluid name="GLNA" density="720.0" volume="-1.0" id="2702200001"
viscosity="1.0" vaporPressure="0.0"/>
    <calculategh/>
  </pipeline>
</PipeLang:PipelineSystem>
```



```
C:\Users\thiag\source\repos\PipeSim\PipeSim\bin\Debug\PipeSim.exe
Sistema OSBRA
Duto PLN2PIRp
Valor do head inicial = 1557
Valor da pressao inicial = 72
Numero de Reynolds para o segmento PLN2PIRp_0 = 1863
Fator de atrito para o segmento PLN2PIRp_0 = 0,0457
Head inicial = 1557
Head final = 1145
Densidade = 0,84
Numero de Reynolds para o segmento PLN2PIRp_1 = 642661
Fator de atrito para o segmento PLN2PIRp_1 = 0,0125
Head inicial = 1336
Head final = 1234
Densidade = 0,72
Press?o final = 27,39
```

Figura 14: Saída do caso 4 com vazão de 900 m³/h e massas específicas diferentes.

Esse caso mostra de forma similar ao caso 3 que existe uma descontinuidade do head na interface, pois conforme já foi mencionado, a densidade do segundo produto é menor o que leva a um head maior. Uma observação importante é que a perda de carga é maior com vazão então, a pressão final para esse caso está menor do que a pressão final para o caso 3, como era de se esperar.

Conclusões e Recomendações para trabalhos futuros

Muita dificuldade em obter informações específicas sobre o uso das ferramentas na internet, também tive muitos problemas na utilização das mesmas, pois eu tinha pouca familiaridade com as mesmas.

Recomenda-se o uso de controle de versão se possível dentro da própria ferramenta, uma vez que tentativas de mudanças nos metamodelos acarretaram a corrupção dos projetos. A tentativa de usar o git por fora entrou em conflito com o controle de versão que aparentemente existe dentro do eclipse.

Como trabalho futuro, outras funcionalidades podem ser adicionadas, tais como os cálculos de correção de vazão para pressão e temperatura, integração da vazão para obtenção do volume, verificação de quebra de coluna, algoritmos de detecção e localização de vazamento. Inserção da temperatura no gradiente hidráulico, etc.

Não consegui trabalhar eficientemente com métodos, a geração automática dos mesmos não foi possível.

A Codificação do Acceleo deve ser trocada para geração de strings/mensagens em português.

Não consegui trabalhar com enumerados na transformação M2M.

Não consegui listar os atributos em uma coleção de forma automática no acceleo, tal como pude observar em exemplos com java, cujo metamodelo era UML ou ECore.

Com a geração automática dos arquivos, o código pode ser facilmente testado em um projeto de console. Eu entendi que poderia utilizar o mesmo “pipeline de ferramentas” para gerar modelos de vários dutos e guardá-los como xmi. Quando quisesse simular algum modelo já teria os arquivos prontos.

Um ponto de quebra de paradigma para mim, foi que cada objeto do modelo virou uma classe no código gerado. Então, como exemplo, se eu tiver 30 instrumentos no modelo, cada instrumento vai virar uma classe. A possibilidade de previamente instanciar cada modelo de antemão pelo Acceleo, foi bem interessante.

Em virtude da minha não familiaridade prévia com os diversos framework e a ferramenta principal, Eclipse, fizeram com que eu tivesse perdido por várias vezes os modelos que estava trabalhando, principalmente em virtude de alterações no metamodelo.

Eu achei que fazer o ciclo todo do trabalho desde o PIM até a geração do código me fizeram entender melhor o que cada etapa de transformação estaria fazendo, o que me fez alterar por diversas vezes os modelos, PIM e PSM.

Nas primeiras versões que eu fiz no Sirius tinha criado no editor uma interface gráfica com caixa de ferramentas, possibilitando arrastar os objetos na hora da criação do modelo. Conforme mencionado, em virtude das diversas alterações que eu fiz no PIM, acabei não recriando a mesma em função do tempo.

Como trabalho futuro, seria interessante colocar restrições e validações no modelo, tais como pode ser feito no editor do Sirius, por exemplo. Para o POC realizado, uma pressão e vazão a montante são requeridos. Caso o usuário não tivesse criado o modelo com estes instrumentos na posição de origem, o gradiente hidráulico não poderia ter sido calculado.

Algumas boas práticas de engenharia de software poderiam ter sido implementadas tais como segregar a saída, no presente exemplo, a saída de console está dentro da classe CalculoGH. Poderia ter sido criado interfaces de modo a retirar essa saída de dentro da classe. Um outro exemplo seria a inclusão de alguma variação do padrão de projeto factory, pensando

em futuras implementações. Por exemplo, o cálculo do fator de atrito possui outras equações passíveis de serem implementadas.

Também poderiam ter sido criadas “protected areas” nas classes, pois algumas extensões ou inclusões de novos métodos por usuários, nas classes pode ser desejado.

Os artefatos gerados nesse trabalho pelo Papyrus estão no Github no seguinte endereço: <https://github.com/thiagoaramaki/DSDM>. Os artefatos contidos neste repositório estão descritos abaixo.

- O projeto do Sirius está no arquivo PipeLang.zip

- O projeto ATL está no arquivo PipeLang2PipeSim.zip

- O projeto Acceleo está no arquivo TesteFinal2.zip

As instâncias dos modelos de exemplo do PIM e do PSM estão no arquivo zipado runtime-New_configuraron.zip. Dentro desse zip, procurar a pasta OSBRA0 para a instância do modelo PIM e a pasta TestePipeSim3 para a instância do modelo PSM.

- Este documento está com o nome DSL - PipeSim.pdf

- A apresentação a ser realizada está com o PipeSim.pdf

Os demais arquivos não mencionados dentro do repositório são de aulas anteriores, na criação de metamodelos usando estereótipos UML.