Relatório do Trabalho Final da Disciplina Desenvolvimento de Software Dirigido a Modelos – Data 07/02/2022

Eduardo Alves de Oliveira

Link GitHub dos arquivos: https://github.com/eduaopec/UFF_DSDM/Trabalho_Final_Entregue/

Motivação

Criação de um DSML simplificada para a modelagem de aplicações de Internet das Coisas (IoT) em um conjunto definido de eletrodomésticos (geladeira e tv) em uma Smart Home.

Conceitos

Os conceitos do domínio (entidades e relações) tem como referência uma arquitetura IoT chamada de “RILA” ("*Reference IoT Layered Architecture*" - https://architexturez.net/pst/az-cf-176871-1454077830).

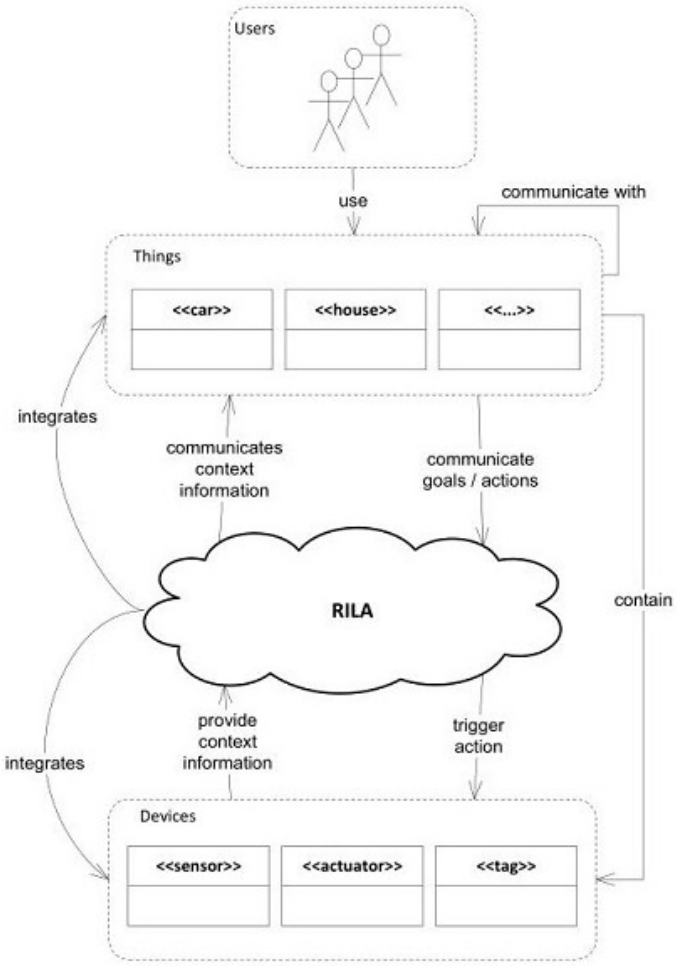


Figura 1 – Arquitetura RILA

Através da leitura da “RILA” (figura 1) foram extraídos os seguintes conceitos: Aplicação, Coisa (Geladeira e Tv) e Dispositivo (Sensor, Atuador).

Outros conceitos foram definidos para atender a proposta desse DSML simplificado, são eles: Cloud (armazenamento), Mensageria (contexto de comunicação) e Mensagem (comunicação de ações e metas).

Conceitos e Propriedades da Linguagem

Antes de se chegar aos conceitos e propriedades foi criado um diagrama de objetos (figura 2) com o objetivo de se encontrar a linguagem abstrata da DSML.

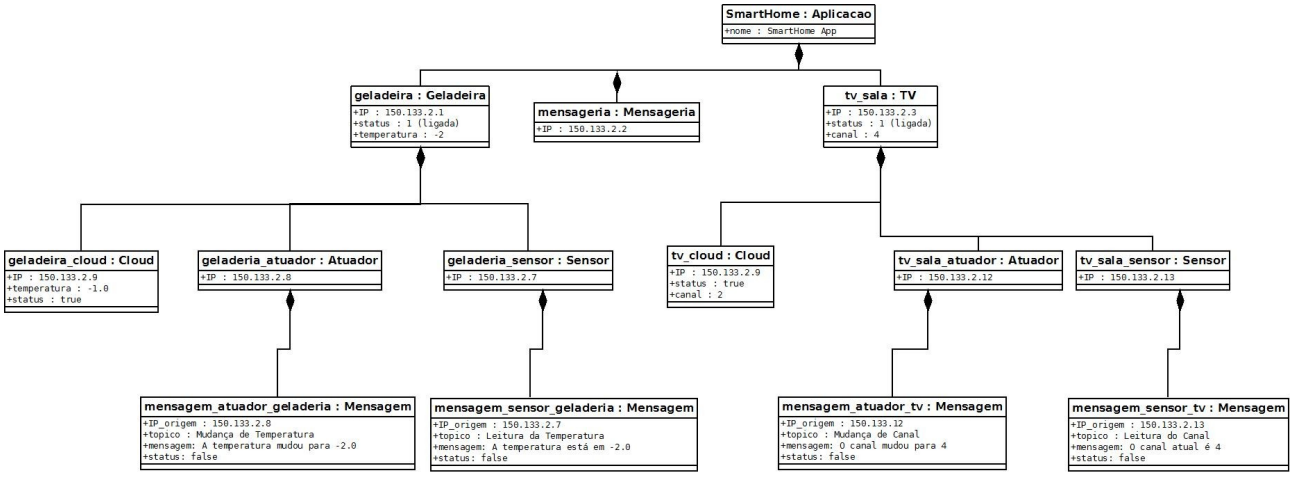


Figura 2 – Diagrama de Objetos da DSML

A seguir temos a descrição de todos os conceitos com suas propriedades intrínsecas e extrínsecas da DSML.

Conceito (classes)	Propriedades Intrínseca (atributos)	Propriedade Extrínseca (associações entre as classes)
Aplicação	Nome: String	
Geladeira	Ip: String Status: Boolean Temperatura: Float	Associada a classe aplicação.
TV	Ip: String Status: Boolean Canal: Int	Associada a classe aplicação.
Mensageria	Ip: String	Associada a classe aplicação.
Cloud	Ip: String Status: Boolean Temperatura: Float Canal: Int	Associada as classes Tv e Geladeira.
Sensor	Ip: String	Associada as classes Tv e Geladeira.
Atuador	Ip: String	Associada as classes Tv e Geladeira.
Mensagem	Ip_origem: String Topico: String Mensagem: String Status: Boolean	Associada as classes Sensor e Atuador.

Tabela 1 – Conceitos e Propriedades da DSML

Diagrama da Linguagem Concreta

A criação da linguagem concreta foi através da abertura de uma nova configuração do Sirius no Eclipse.

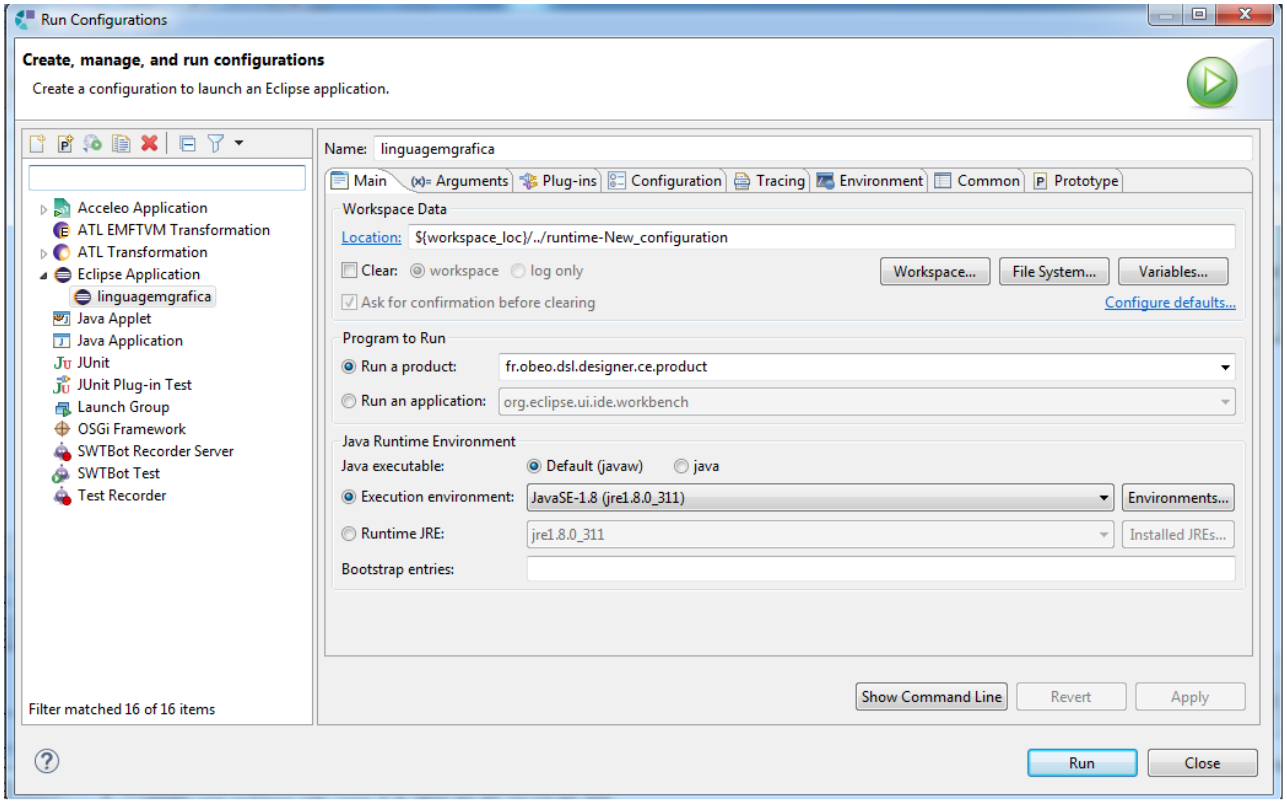


Figura 3 – Configuração para criação da Linguagem Concreta (gráfica)

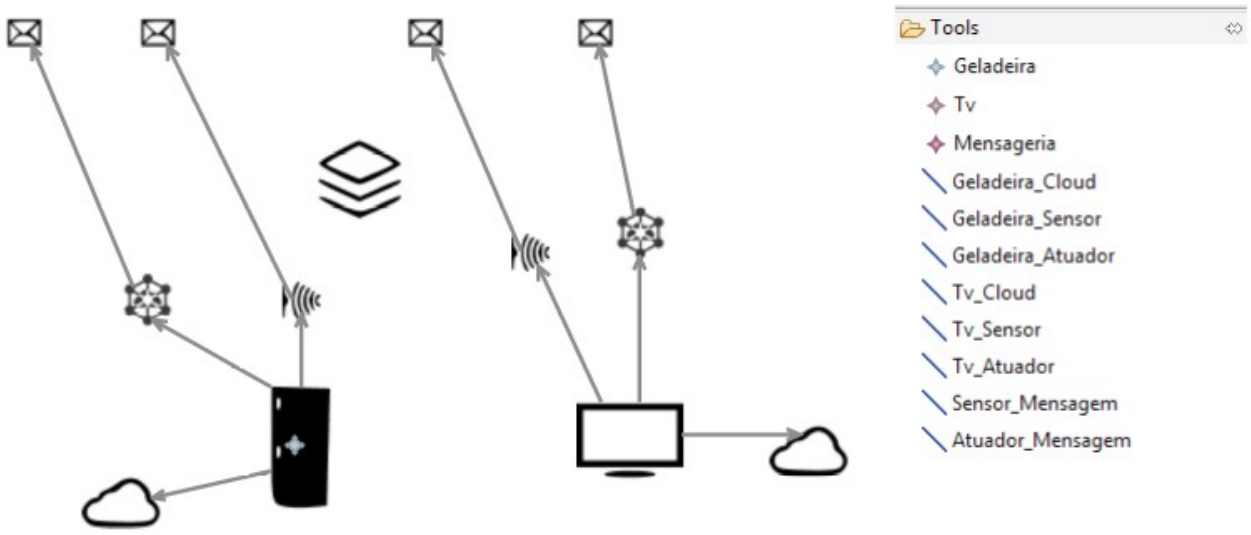


Figura 4 – Linguagem Concreta (gráfica) da DSML

Após a criação das linguagens abstratas e concretas, veio a etapa da transformação M2M, onde através dos metamodelos PIM e PSM foram definidas as regras de transformação utilizando a ATL.

PIM (Modelo Independente de Plataforma) – Metamodelo Rila

Baseado nas informações obtidas na descrição da linguagem abstrata (ver tabela 1) foi desenhado um diagrama de classes (figura 5) que representa meu metamodelo Rila. Sendo esse a representação PIM do Rila.

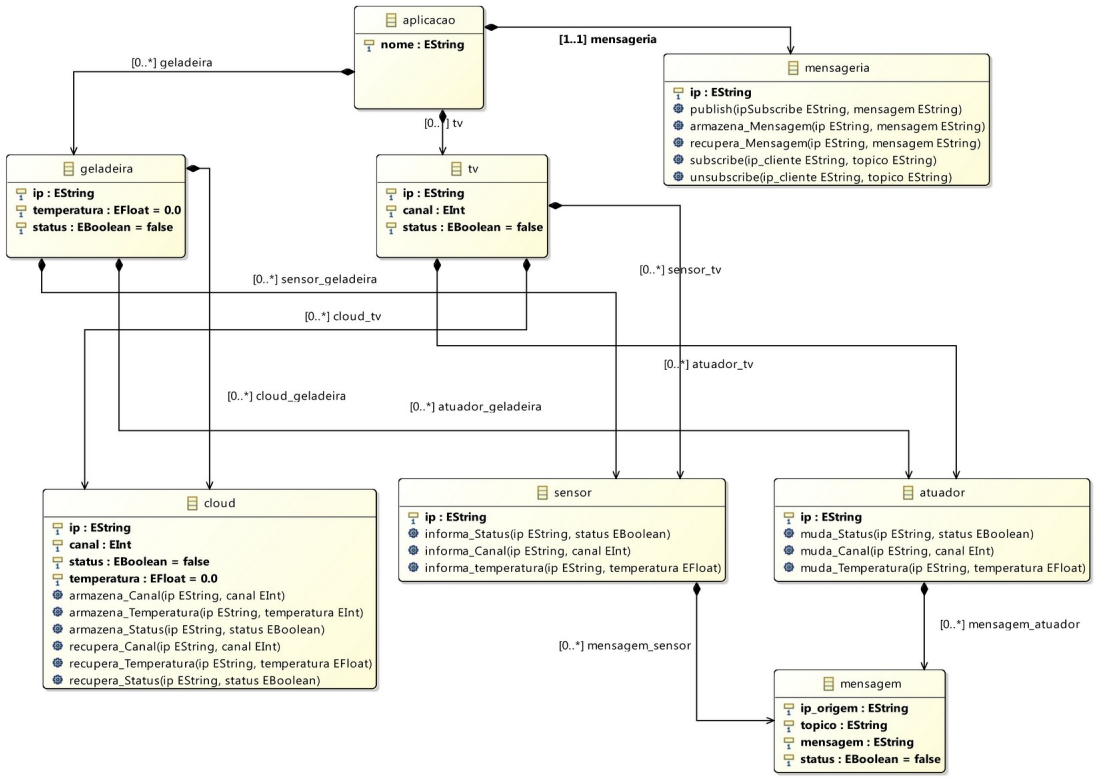


Figura 5 – Metamodelo Rila (PIM)

PSM (Modelo Específico de Plataforma) – Metamodelo Rila com protocolo MQTT

Antes de definir o metamodelo PSM busquei uma solução técnica para o requisito de comunicação previsto no RILA.

A solução encontrada foi o protocolo MQTT (*Message Queuing Telemetry Transport).* Este protocolo foi desenvolvido com objetivo de interligar máquinas com um modelo de comunicação bem leve. Consome poucos recursos e é baseada na filosofia publicador/assinante (ver figura 6), onde os clientes se comunicam de forma desacoplada.

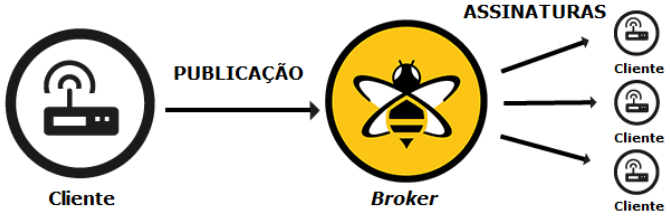


Figura 6 – Descrição da solução de comunicação MQTT

A seguir, na figura 7, apresento o modelo PSM já com o conceito do Broker do MQTT.

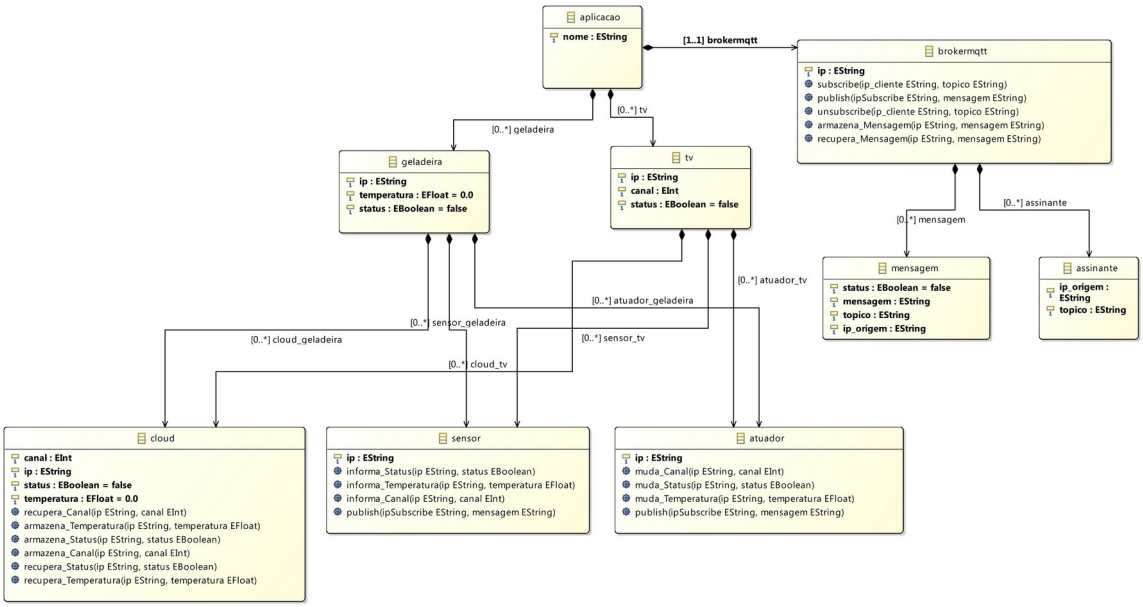
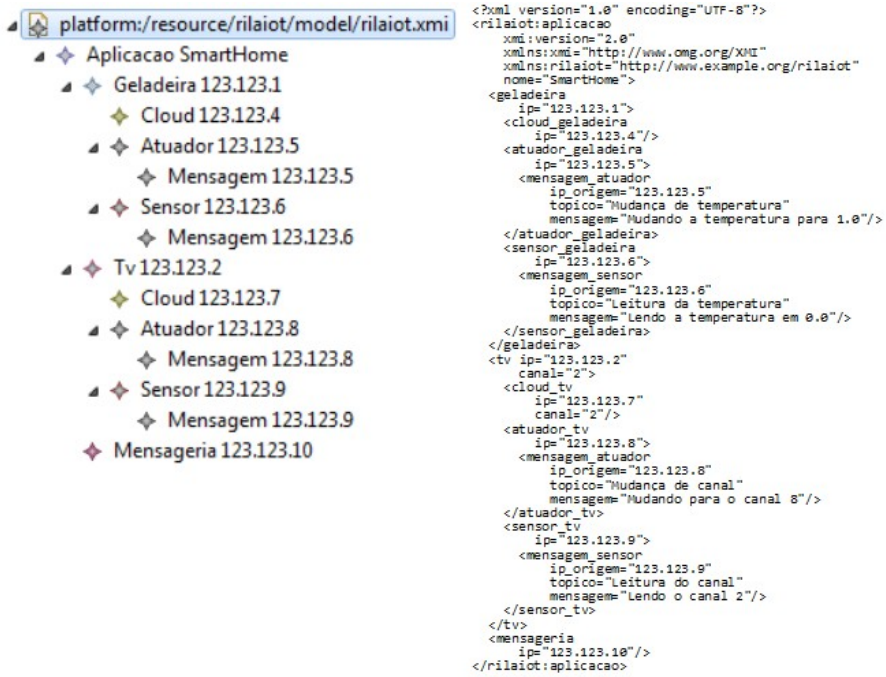


Figura 7 – Metamodelo Rila com protocolo MQTT (PSM)

XMI do PIM

Antes de rodar o código ATL de transformação é necessário criar um arquivo XMI de entrada (XMI do PIM), para ser utilizado como fontes para a geração do arquivo XMI de saída (XMI do PSM).

A seguir temos a figura 8 que exibe o XMI do PIM que foi utilizado como exemplo para a criação das regras da ATL e a geração do arquivo de saída XMI do PSM.




```
<?xml version="1.0" encoding="UTF-8"?>
<rilaiot:aplicacao
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:rilaiot="http://www.example.org/rilaiot"
    nome="SmartHome">
  <geladeira
      ip="123.123.1">
    <cloud_geladeira
        ip="123.123.4"/>
    <atuador_geladeira
        ip="123.123.5">
      <mensagem_atuador
          ip_origem="123.123.5"
          topico="Mudança de temperatura"
          mensagem="Mudando a temperatura para 1.0"/>
    </atuador_geladeira>
    <sensor_geladeira
        ip="123.123.6">
      <mensagem_sensor
          ip_origem="123.123.6"
          topico="Leitura da temperatura"
          mensagem="Lendo a temperatura em 0.0"/>
    </sensor_geladeira>
  </geladeira>
  <tv ip="123.123.2"
      canal="2">
    <cloud_tv
        ip="123.123.7"
        canal="2"/>
    <atuador_tv
        ip="123.123.8">
      <mensagem_atuador
          ip_origem="123.123.8"
          topico="Mudança de canal"
          mensagem="Mudando para o canal 8"/>
    </atuador_tv>
    <sensor_tv
        ip="123.123.9">
      <mensagem_sensor
          ip_origem="123.123.9"
          topico="Leitura do canal"
          mensagem="Lendo o canal 2"/>
    </sensor_tv>
  </tv>
  <mensageria
      ip="123.123.10"/>
</rilaiot:aplicacao>
```

Figura 8 – XMI do PIM

Linguagem ATL de transformação M2M

Para que seja possível ocorrer o M2M do PIM para o PSM, é preciso definir um conjunto de regras utilizando a linguagem ATL. A seguir temos o código criado na linguagem ATL para realizar o M2M (ver figura 9).

```
-- @path rilaiot=/rilaiot/model/rilaiot.ecore
-- @path rilaiotmqtt=/rilaiotmqtt/model/rilaiotmqtt.ecore

module rilaiot2rilaiotmqtt;
create OUT : rilaiotmqtt from IN : rilaiot;

helper def: getMensagens(rila:rilaiot!mensagem) : Sequence(rilaiotmqtt!mensagem)
=
	rilaiotmqtt!mensagem.allInstances()->iterate(men; conjunto:
Sequence(rilaiotmqtt!mensagem) = Sequence{}|
	conjunto -> append (men)

);

helper def: getAssinantes(rila:rilaiot!mensagem) : Sequence(rilaiotmqtt!assinante)
=
	rilaiotmqtt!assinante.allInstances()->iterate(ass; conjunto:
Sequence(rilaiotmqtt!assinante) = Sequence{}|
	conjunto -> append (ass)

);

rule aplicacao2aplicacao {
	from
		a : rilaiot!aplicacao
	to
		a1 : rilaiotmqtt!aplicacao (
			nome <- a.nome,
			tv <- a.tv,
			geladeira <- a.geladeira,
			brokermqtt <- a.mensageria
		)
}

rule tv2tv {
	from
		a : rilaiot!tv
	to
		a1 : rilaiotmqtt!tv (
			ip <- a.ip,
			canal <- a.canal,
			status <- a.status,
			sensor_tv <- a.sensor_tv,
			atuador_tv <- a.atuador_tv,
			cloud_tv <- a.cloud_tv
		)
}

rule sensortv2sensortv {
	from
		a : rilaiot!sensor
	to
```

```
            a1 : rilaiotmqtt!sensor (
                  ip <- a.ip
            )
}

rule atuadortv2atuadortv {
      from
            a : rilaiot!atuador
      to
            a1 : rilaiotmqtt!atuador (
                  ip <- a.ip
            )
}

rule cloudtv2cloudtv {
      from
            a : rilaiot!cloud
      to
            a1 : rilaiotmqtt!cloud (
                  ip <- a.ip,
                  canal <- a.canal,
                  temperatura <- a.temperatura,
                  status <- a.status
            )
}

rule geladeira2geladeira {
      from
            a : rilaiot!geladeira
      to
            a1 : rilaiotmqtt!geladeira (
                  ip <- a.ip,
                  temperatura <- a.temperatura,
                  status <- a.status,
                  sensor_geladeira <- a.sensor_geladeira,
                  atuador_geladeira <- a.atuador_geladeira,
                  cloud_geladeira <- a.cloud_geladeira
            )
}

rule aplicacao2brokermqtt {
      from
            a : rilaiot!mensageria
      to
            a1 : rilaiotmqtt!brokermqtt (
                  ip <- a.ip,
                  mensagem <- thisModule.getMensagens(a),
                  assinante <- thisModule.getAssinantes(a)
)
}

rule mensagem2mensagem {
      from
            a : rilaiot!mensagem
      to
            a1 : rilaiotmqtt!mensagem (
                  ip_origem <- a.ip_origem,
                  topico <- a.topico,
                  mensagem <- a.mensagem,
                  status <- a.status
            ),
            a2: rilaiotmqtt!assinante (
```

```
                ip_origem <- a.ip_origem,
                topico <- a.topico
            )
}
```

Figura 9 – Regras de Transformação do Rila no ATL (M2M)

Para gerar o XMI do PSM foi criada uma nova configuração no Sirius para rodar as regras da ATL (figura 10).

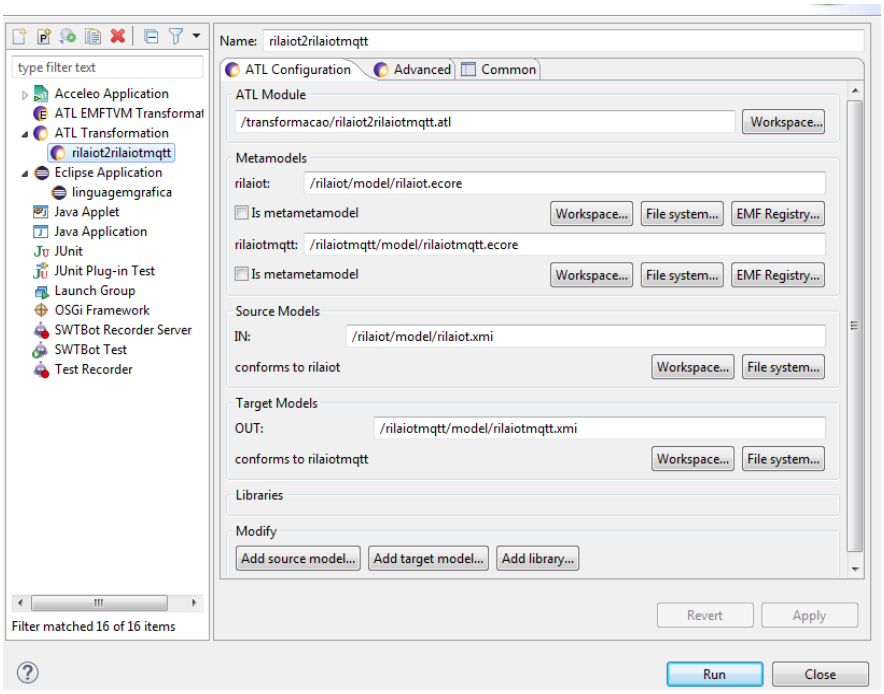
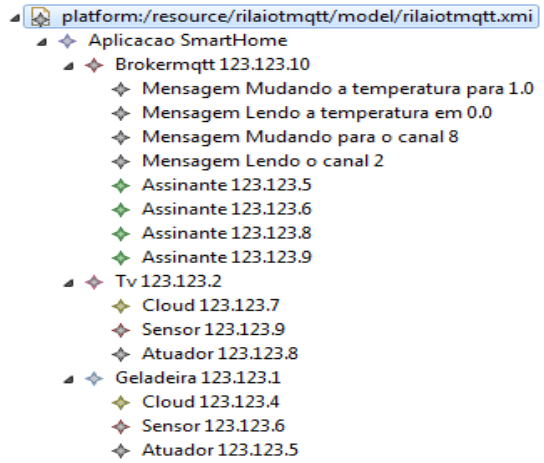


Figura 10 – Configuração para rodar o ATL (M2M)

XMI do PSM

A seguir, na figura 11, temos o XMI de saída construído pelas regras de transformação, baseado no PSM.




```
<?xml version="1.0" encoding="ISO-
8859-1"?>
<rilaiotmqtt:aplicacao
xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:rilaiotmqtt="http://www.exam
ple.org/rilaiotmqtt"
nome="SmartHome">
  <brokermqtt ip="123.123.10">
    <mensagem mensagem="Mudando a
temperatura para 1.0"
topico="Mudança de temperatura"
ip_origem="123.123.5"/>
    <mensagem mensagem="Lendo a
temperatura em 0.0"
topico="Leitura da temperatura"
ip_origem="123.123.6"/>
    <mensagem mensagem="Mudando
para o canal 8" topico="Mudança de
canal" ip_origem="123.123.8"/>
    <mensagem mensagem="Lendo o
canal 2" topico="Leitura do canal"
ip_origem="123.123.9"/>
    <assinante
ip_origem="123.123.5"
topico="Mudança de temperatura"/>
    <assinante
ip_origem="123.123.6"
topico="Leitura da temperatura"/>
    <assinante
ip_origem="123.123.8"
topico="Mudança de canal"/>
    <assinante
ip_origem="123.123.9"
topico="Leitura do canal"/>
  </brokermqtt>
  <tv ip="123.123.2" canal="2">
    <cloud_tv canal="2"
ip="123.123.7"/>
    <sensor_tv ip="123.123.9"/>
    <atuador_tv ip="123.123.8"/>
  </tv>
  <geladeira ip="123.123.1">
    <cloud_geladeira
ip="123.123.4"/>
    <sensor_geladeira
ip="123.123.6"/>
    <atuador_geladeira
ip="123.123.5"/>
  </geladeira>
</rilaiotmqtt:aplicacao>
```

Figura 11 – XMI do PSM

Geração de código Java (transformação M2T) com Acceleo

Para gerar o código java foi preciso criar uma nova configuração para o Acceleo no Eclipse. Através dessa configuração, rodou-se o código para a criação das classes java (ver figura 12).

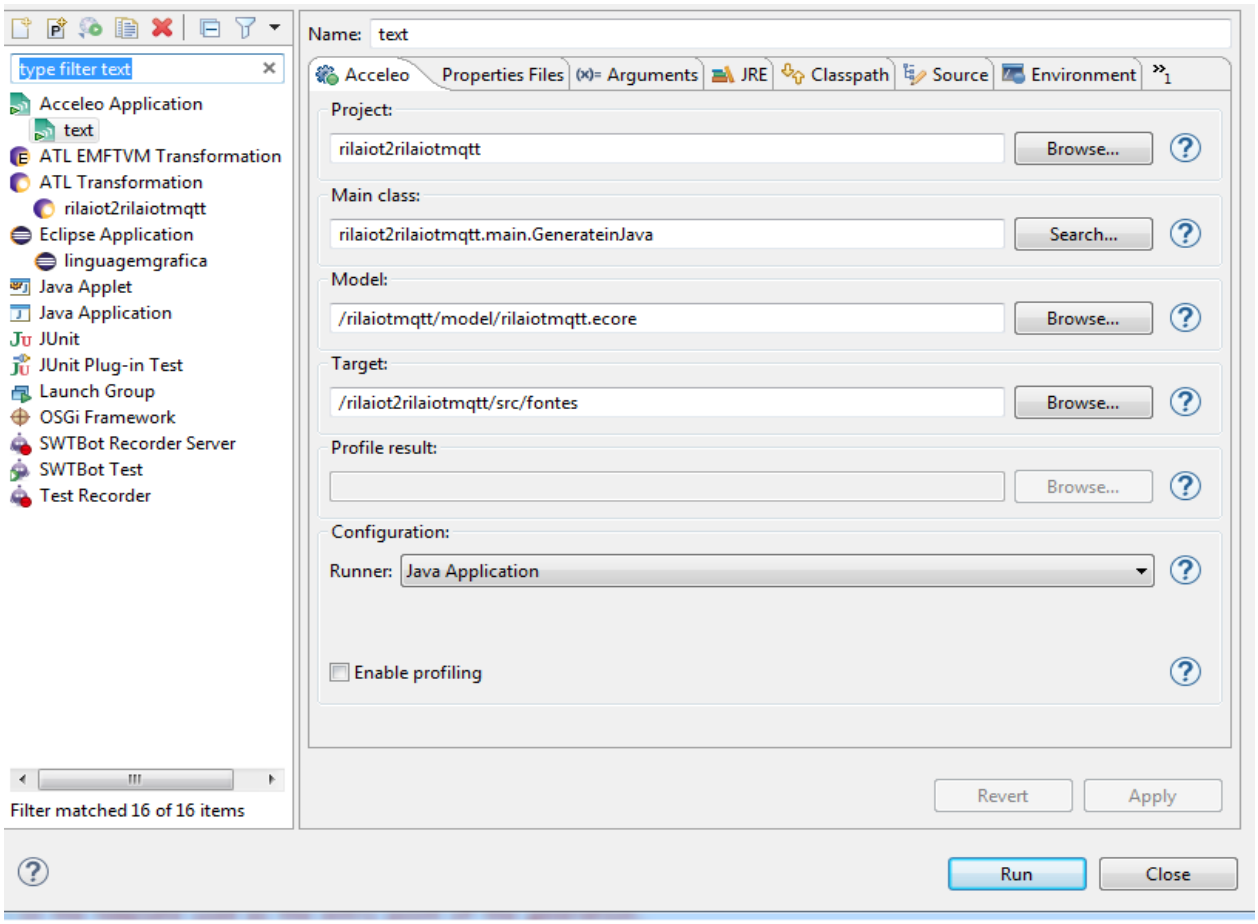


Figura 12 – Configuração para rodar o Acceleo (M2T)

A seguir (figura 13) apresento o conteúdo do arquivo generateinJava.mtl para a geração do código do metamodelo PSM

```
[comment encoding = UTF-8 /]
[module generateinJava('http://www.example.org/rilaiotmqtt')]


[template public generateElement(raiz :
rilaiotmqtt::aplicacao)]
[comment @main/]

[for (i : rilaiotmqtt::geladeira | raiz.geladeira)]
[file ('fontes/'+i.ip+'.java', false, 'UTF-8')]
     package fontes;
 public class [i.ip/]{}
[/file]
[file ('fontes/'+i.status+'.java', false, 'UTF-8')]
     package fontes;
 public class [i.status/]{}
[/file]
[file ('fontes/'+i.temperatura+'.java', false, 'UTF-8')]
     package fontes;
 public class [i.temperatura/]{}
[/file]
[/for]

[/template]
```

Figura 13 – Conteúdo do arquivo generateinJava.mtl

Após realizar a configuração de duas classes registerPackages e registerResourceFactories (ver Figura 14) no arquivo GenerateinJava.java executei o código do arquivo generateinJava.mtl e retornou um erro (ver figura 15).

Infelizmente não consegui solucionar esse erro a tempo da apresentação, e por consequência não consegui gerar as classes Java do meu metamodelo do PSM.

```
    /**
     * This can be used to update the resource set's package registry with all needed
EPackages.
     *
     * @param resourceSet
     *            The resource set which registry has to be updated.
     * @generated NOT
     */
    @Override
    public void registerPackages(ResourceSet resourceSet) {
        super.registerPackages(resourceSet);

        if (!isInWorkspace(rilaiotmqtt.RilaiotmqttPackage.class)) {

resourceSet.getPackageRegistry().put(rilaiotmqtt.RilaiotmqttPackage.eINSTANCE.getNsURI(
), rilaiotmqtt.RilaiotmqttPackage.eINSTANCE);
        }


/**
     * This can be used to update the resource set's resource factory registry with all
needed factories.
     *
     * @param resourceSet
     *            The resource set which registry has to be updated.
     * @generated NOT
     */
    @Override
    public void registerResourceFactories(ResourceSet resourceSet) {
```

```
        super.registerResourceFactories(resourceSet);

        if (!isInWorkspace(rilaiotmqtt.RilaiotmqttPackage.class)) {

resourceSet.getPackageRegistry().put(rilaiotmqtt.RilaiotmqttPackage.eINSTANCE.getNsURI(
), rilaiotmqtt.RilaiotmqttPackage.eINSTANCE);
        }
```

Figura 14 – Configuração das classes registerPackages e registerResourceFactories

```
The generation failed to generate any file because there are no model
elements that matches at least the type of the first parameter of one of
your main templates.
The problem may be caused by a problem with the registration of your
metamodel, please see the method named "registerPackages" in the Java
launcher of your generator. It could also come from a missing [comment
@main/]

in the template used as the entry point of the generation.
```

Figura 15 – Erro ao rodar o código do arquivo generateinJava.mtl

Conclusões

Após a realização do trabalho proposto cheguei as seguintes conclusões:

1. Muita dificuldade para usar o Sirius;
2. Falta de experiência prévia no ambiente Eclipse e linguagem Java dificultaram bastante o andamento do trabalho;
3. Muito tempo gasto com retrabalho na reconstrução dos metamodelos e reinstalação do ambiente (problemas de infraestrutura);
4. Pouca documentação/tutorias disponíveis na internet.

Trabalhos Futuros

Como trabalhos futuros sugiro as seguintes iniciativas:

1. Aprimorar o conhecimento no Sirius;
2. Obter mais experiência no ambiente Eclipse e na linguagem Java;
3. Evoluir o metamodelo Rila para atender a mais requisitos da arquitetura.