Modelação de uma Smart Grid em VDM++

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Informática e Computação

Métodos Formais de Engenharia de Software

João Manuel Ferreira Trindade ● 201109221 ● ei11118@fe.up.pt

Paulo Bordalo Marcos ● 201100759 ● ei12131@fe.up.pt

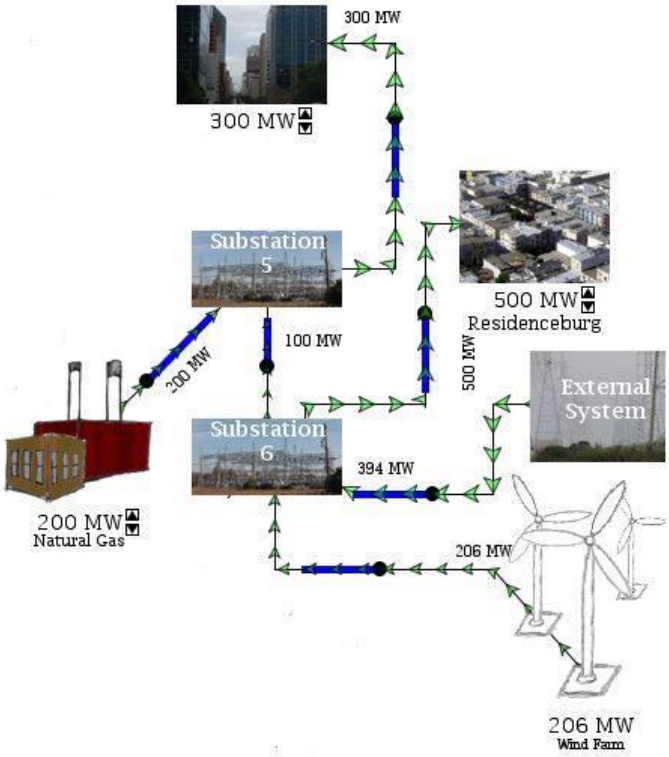
Dezembro 2014

Índice

Descrição Informal do Sistema e Lista de Requisitos 3
Descrição Informal 3
Lista de Requisitos 3
Modelo Visual UML 4
Modelo de Casos de Uso 4
Máquina de Estados 5
Diagrama de Classes 6
Modelação Formal VDM++ 6
Classe EffectiveNode 6
Classe Powerline 8
Classe SmartCity 9
Classe NodeFactory 11
Validação do Modelo 12
Classe MyTestCase 12
Classe TestSmartGrid 12
Conclusões 18
Bibliografia 18

Descrição Informal do Sistema e Lista de Requisitos

Descrição Informal

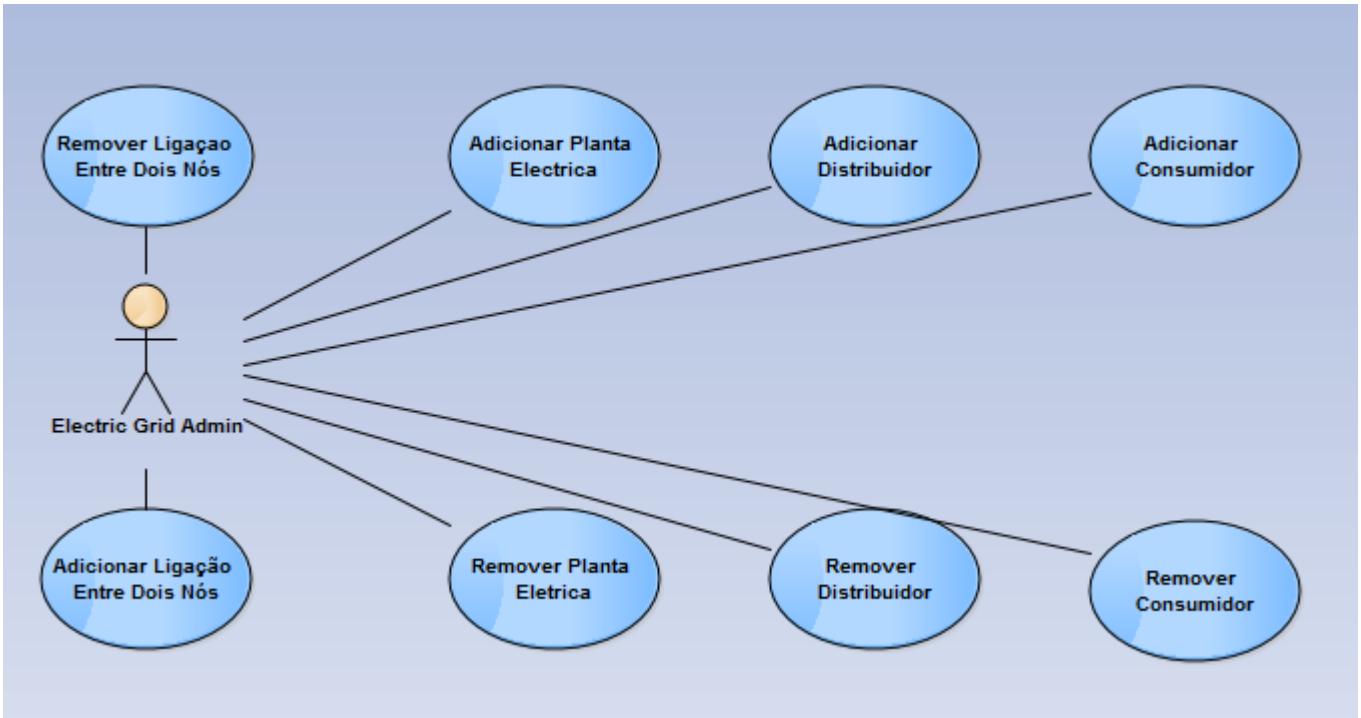


Lista de Requisitos

- Não é possível desligar uma planta eléctrica que deixe parte do sistema sem energia
- Deve existir redundância caso uma planta falhe
- Uma planta eléctrica não pode estar directamente ligada a um consumidor
- Não pode passar mais corrente eléctrica do que aquela suportada pela linha
- Não é possível armazenar energia eléctrica

Modelo Visual UML

Modelo de Casos de Uso



Cenário	Adicionar Consumidor
Descrição	Quero adicionar um consumidor à cidade
Pre-Condições	-
Pós-Condições	A soma dos consumos de todos os consumidores, incluído o adicionado tem de ser inferior ao conjunto de todas as produções
Excepções	-

Cenário	Remover Planta Electrica
Descrição	Quero remover uma planta eléctrica da cidade
Pre-Condições	-
Pós-Condições	A soma dos consumos de todos os consumidores tem de ser inferior a soma das produções que abastecem cada um, já sem a planta removida.
Excepções	-

Cenário	Remover Distribuidor
Descrição	Quero remover um distribuidor da cidade
Pre-Condições	-
Pós-Condições	A soma dos consumos de todos os consumidores tem de ser inferior a soma das produções que abastecem cada um, já sem o distribuidor.
Excepções	-

Cenário	Adicionar Ligação Entre Dois Nós
Descrição	Quero ligar dois nós
Pre-Condições	O valor de energia a transportar pela liga não pode ser maior do que a sua capacidade máxima
Pós-Condições	-
Excepções	-

Cenário	Remover Ligação Entre Dois Nós
Descrição	Quero remover a ligação entre dois
Pre-Condições	A soma dos consumos de todos os consumidores tem de ser inferior a soma das produções que abastecem cada um, já sem a ligacao
Pós-Condições	-
Excepções	-

Máquina de Estados

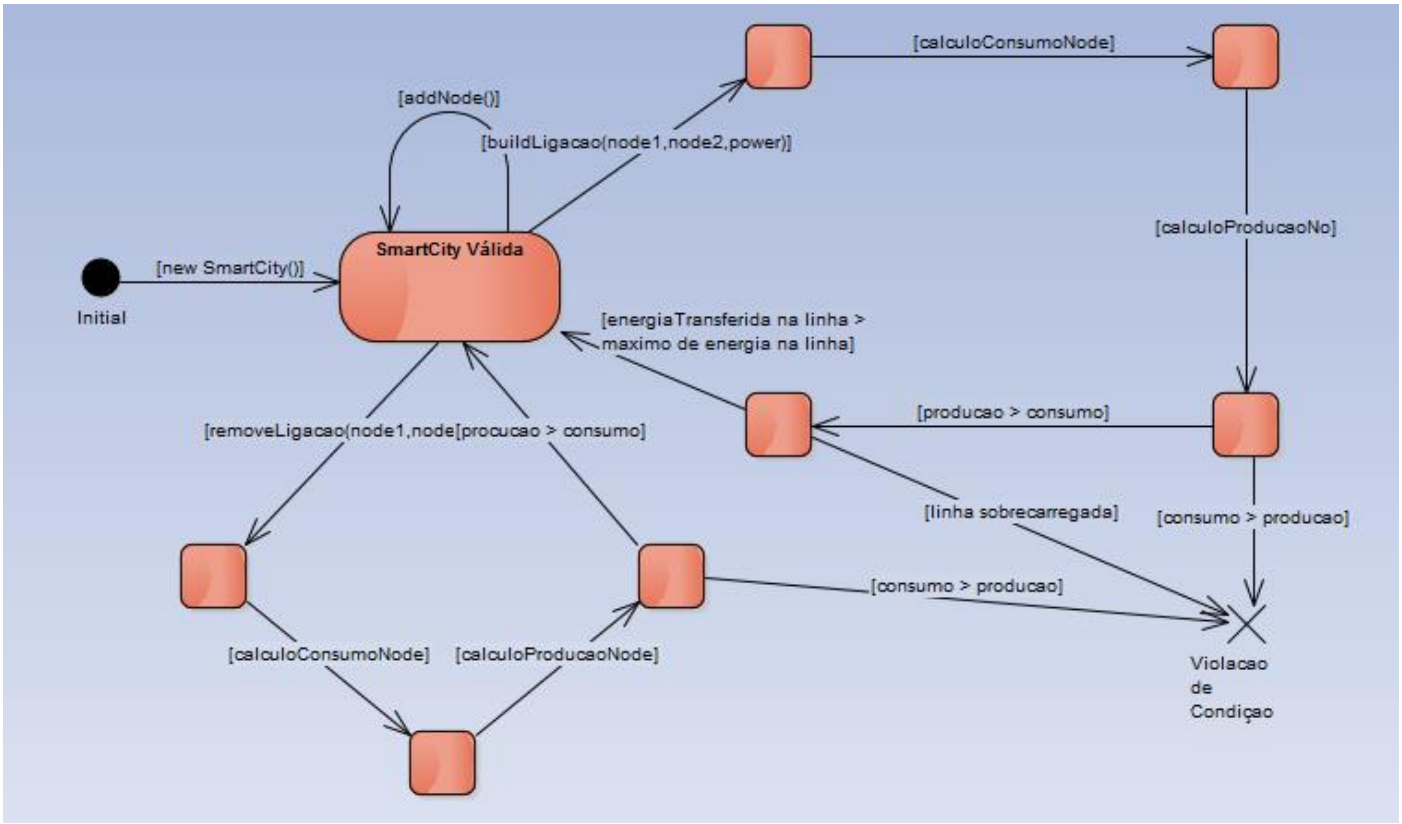
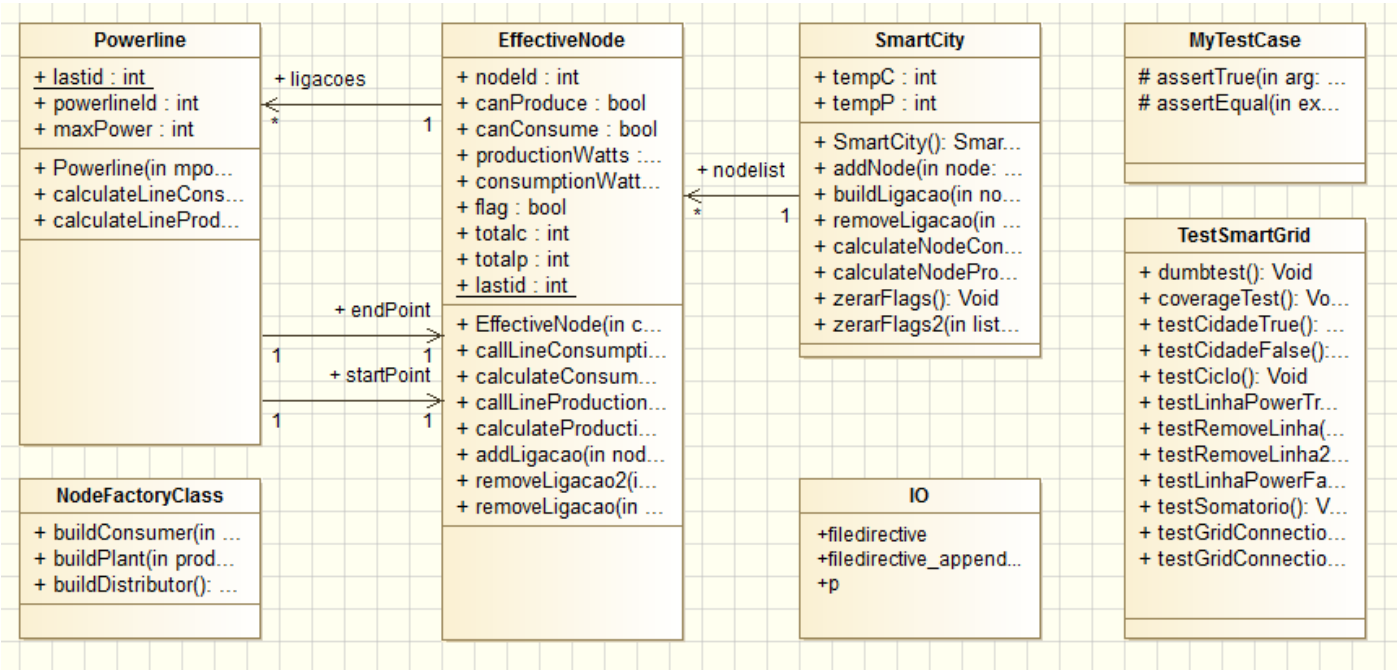


Diagrama de Classes



Classe	Descrição
Powerline	Representa uma linha de transporte de corrente eléctrica
EffectiveNode	Representa um nó da rede (planta, distribuidor, consumidor)
SmartCity	Representação da Cidade, conjunto de nós
NodeFactoryClass	Classe Factory para criar nós da rede
TestSmartGrid	Classe com testes para o modelo
MyTestCase	Superclasse de testes. Define assertTrue e assertEqual. Criada por Prof.Pascoal Faria (Faria)

Modelação Formal VDM++

Classe EffectiveNode

```
class EffectiveNode

types

instance variables
	public nodeId : int;
	public canProduce : bool;
	public canConsume : bool;
	public productionWatts : int;
	public consumptionWatts : int;
	public ligacoes : set of Powerline;
	public flag : bool;
	public totalc : int;
	public totalp : int;
	public static lastid : int := 0;

operations
	public EffectiveNode : bool * bool * int * int ==> EffectiveNode
		EffectiveNode(cp, cc, pw, cw) ==
		(	lastid := lastid + 1;
			nodeId := lastid;
			canProduce := cp;
			canConsume := cc;
```

```
                productionWatts := pw;
                consumptionWatts := cw;
                ligacoes := {};
                flag := false;
                totalc := 0;
                totalp := 0;
                return self;
        )
        post canProduce = false => productionWatts = 0 and canConsume = false =>
consumptionWatts = 0;

    -- Calculo do consumo de cada linha do nó
    public callLineConsumption : set of Powerline * int * int ==> int
        callLineConsumption(linhas, somatorio, myid) ==
            (if (linhas = {}) then (return somatorio)
            else
            ( flag := true;
                let l1 in set linhas in callLineConsumption(linhas\{l1},somatorio +
l1.calculateLineConsumption(),myid)
            )
        );

    -- Calculo do consumo do nó
    public calculateConsumption : () ==> int
            calculateConsumption() ==
                (if (canConsume = true and not(flag)) then
                (           flag := true;
                            return consumptionWatts;
                )
                else
                (
                        flag := true;
                        return callLineConsumption(ligacoes, 0, nodeId);
                )
            );



    -- Calculo da energia produzida em cada linha do nó
    public callLineProduction : set of Powerline * int * int ==> int
        callLineProduction(linhas, somatorio, myid) ==
            (if (linhas = {}) then (return somatorio)
            else
            ( flag := true;
                let l1 in set linhas in callLineProduction(linhas\{l1},somatorio +
l1.calculateLineProduction(),myid)
            )
        );

    -- Calculo da energia produzida no nó
    public calculateProduction : () ==> int
            calculateProduction() ==
            (           if (canProduce = true and not(flag)) then
                        (           flag := true;
                                    return productionWatts;
                        )
                        else
                        (
                                flag := true;
                                return callLineProduction(ligacoes, 0, nodeId);
                        )
            );
```

```
	-- adicionar ligacao a um nó
	public addLigacao : EffectiveNode * int ==> ()
		addLigacao(node_final, maxAmmountOfPower) ==
				(	dcl line : Powerline := new
Powerline(maxAmmountOfPower,self,node_final);
					dcl linedois : Powerline := new
Powerline(maxAmmountOfPower,node_final,self);

					ligacoes := ligacoes union {line};
					node_final.ligacoes := node_final.ligacoes union {linedois};
				)
				pre (	(canProduce = true => node_final.canProduce = false)
						and (canConsume = true =>
node_final.canConsume = false)
						and (canConsume = true => node_final.canProduce
<> true)
						and (canProduce = true => node_final.canConsume
<> true)
						and (canProduce = true => productionWatts  <
maxAmmountOfPower)
						and (canConsume = true => consumptionWatts <
maxAmmountOfPower)
						and (node_final.canProduce = true =>
node_final.productionWatts  < maxAmmountOfPower)
						and (node_final.canConsume = true =>
node_final.consumptionWatts < maxAmmountOfPower)
					);

		-- remove ligacao de um nó, funcao recursiva
		public removeLigacao2 : EffectiveNode * set of Powerline * set of Powerline ==>
set of Powerline
		removeLigacao2(node_final,lista,listafinal) ==
			(	if (lista = {}) then (return listafinal)
				else
				( let linha1 in set lista in
						(		if( (linha1.endPoint = self and
linha1.startPoint = node_final) or (linha1.startPoint = self and linha1.endPoint = node_final)
)	then
	removeLigacao2(node_final,lista\{linha1},listafinal)
	else
	removeLigacao2(node_final,lista\{linha1},listafinal union {linha1})
							)
				)
		);

		-- remove ligacao de um nó
		public removeLigacao : EffectiveNode ==> ()
		removeLigacao(node_final) ==
		(	ligacoes := removeLigacao2(node_final,ligacoes,{});
			node_final.ligacoes := removeLigacao2(self,node_final.ligacoes,{});
		);
end EffectiveNode
```

Classe Powerline

```
class Powerline

types

instance variables
	public static lastid : int := 0;
	public powerlineId : int ;
```

```
	public maxPower : int;
	public startPoint : EffectiveNode;
	public endPoint : EffectiveNode;

inv maxPower > 0;
inv startPoint <> endPoint ;

operations
	public Powerline : int * EffectiveNode * EffectiveNode ==> Powerline
	Powerline (mpower, node1, node2 ) ==
		(	lastid:=lastid + 1;
			powerlineId := lastid;
			maxPower := mpower;
			startPoint := node1;
			endPoint := node2;
			return self;
	);
	-- calculo do consumo na linha
	public calculateLineConsumption : () ==> int
	calculateLineConsumption() ==
		(	--if (not(startPoint.flag))  then (return startPoint.calculateConsumption()
)
		 if (not(endPoint.flag)) then (return endPoint.calculateConsumption())
			else return 0;
	);

	-- calculo da producao na linha
	public calculateLineProduction : () ==> int
	calculateLineProduction() ==
		(	--if (not(startPoint.flag))  then (return startPoint.calculateProduction())
			 if (not(endPoint.flag)) then (return endPoint.calculateProduction())
			else return 0;
	);

end Powerline
```

Classe SmartCity

```
class SmartCity

instance variables
	public nodelist : set of EffectiveNode;
	public tempC : int;
	public tempP : int;

operations
	public SmartCity : () ==> SmartCity
		SmartCity() ==
		(	nodelist := {};
			tempC := 0;
			tempP := 0;
			return self;
		);

		public addNode : EffectiveNode ==> ()
		addNode(node) ==
			nodelist:= nodelist union {node};

		-- metodo construtor de ligacao entre dois nós, recebe o max power que a ligacao
pode ter
		public buildLigacao: EffectiveNode * EffectiveNode * int ==> ()
		buildLigacao(node1,node2,maxp) ==
			(	node1.addLigacao(node2,maxp);

				if(node1.canConsume = true) then
```

```
                (       zerarFlags();
                        tempC := node2.calculateConsumption();
                        zerarFlags();
                        tempP := node2.calculateProduction();
                        zerarFlags();
                )else
                (       if(node1.canProduce = false) then
                                (       zerarFlags();
                                        tempC := node1.calculateConsumption();
                                        zerarFlags();
                                        tempP := node1.calculateProduction();
                                        zerarFlags();
                                )
                                else
                                (       tempC := 0;
                                        tempP := 0;
                                )
                );

)
post (tempC <= tempP);

-- metodo que remove a ligacao entre dois nós
public removeLigacao: EffectiveNode * EffectiveNode ==> ()
removeLigacao(node1,node2) ==
(       node1.removeLigacao(node2);

        if(node1.canProduce) then
        (       zerarFlags();
                tempC := node2.calculateConsumption();
                zerarFlags();
                tempP := node2.calculateProduction();
                zerarFlags();
        )else
        (       if(not(node1.canConsume)) then
                (       zerarFlags();
                        tempC := node1.calculateConsumption();
                        zerarFlags();
                        tempP := node1.calculateProduction();
                        zerarFlags();
                )       else
                        (               tempC := 0;
                                        tempP := 0;
                        )
        );
)
post (tempC <= tempP);

-- calculo do consumo de um nó da cidade
public calculateNodeConsumption: EffectiveNode ==> int
calculateNodeConsumption(node) ==
(               dcl consumption : int := 0;
                zerarFlags();
                consumption := node.calculateConsumption();
                zerarFlags();
                return consumption;
);

-- calculo da producao de um nó na cidade
public calculateNodeProduction: EffectiveNode ==> int
calculateNodeProduction(node) ==
(               dcl production : int := 0;
                zerarFlags();
                production := node.calculateProduction();
                zerarFlags();
```

```
                        return production;
                );

                -- metodo para zerar flags de cada nó. deve ser chamado depois do calculo de
producao/consumo
                public zerarFlags : ()  ==> ()
                zerarFlags() ==
                        zerarFlags2(nodelist);

                -- mettodo para zear flags. chamada recursiva
                public zerarFlags2 : set of EffectiveNode ==> ()
                zerarFlags2(lista) ==
                (               if (lista = {}) then (return)
                                else
                                (               let node1 in set lista in
                                                (               node1.flag := false;
                                                                zerarFlags2(lista\{node1});
                                                )
                                )
                );

                --public printflags : ()  ==> ()
                --printflags() ==
                --(    printflags2(nodelist);
                --     return;
                --);

                --public printflags2 : set of EffectiveNode  ==> ()
                --printflags2(lista) == (
                --     if (lista = {}) then return
                --     else
                --     (
                --                     let node1 in set lista in
                --                             (               IO`print(node1.flag);
                --                                             IO`print(" ");
                --                                             printflags2(lista\{node1});
                --                             )
                --     )
                --);

end SmartCity
```

Classe NodeFactory

```
class NodeFactoryClass
types
values
instance variables
operations
        -- metodo construtor de consumidores
        public buildConsumer : int ==> EffectiveNode
                                buildConsumer(consumedPower) ==
                                        return new EffectiveNode(false,true,0,consumedPower);

                -- metodo construtor de plantas geradoras de energia
                public buildPlant : int ==> EffectiveNode
                        buildPlant(producedPower) ==
                                return new EffectiveNode(true,false,producedPower,0);

                -- metodo construtor de distribuidores
                public buildDistributor : () ==> EffectiveNode
                buildDistributor() ==
                        return new EffectiveNode(false,false,0,0);

end NodeFactoryClass
```

Validação do Modelo

Classe MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

operations

	-- Simulates assertion checking by reducing it to pre-condition checking.
	-- If 'arg' does not hold, a pre-condition violation will be signaled.
	protected assertTrue: bool ==> ()
	assertTrue(arg) ==
		return
	pre arg;

	-- Simulates assertion checking by reducing it to post-condition checking.
	-- If values are not equal, prints a message in the console and generates
	-- a post-conditions violation.
	protected assertEqual: ? * ? ==> ()
	assertEqual(expected, actual) ==
		if expected <> actual then (
		IO`print("Actual value (");
		IO`print(actual);
		IO`print(") different from expected (");
		IO`print(expected);
		IO`println(")\n")
		)
	post expected = actual

end MyTestCase
```

Classe TestSmartGrid

```
class TestSmartGrid is subclass of MyTestCase

operations

		public coverageTest:() ==> ()
			coverageTest() == (
			testCidadeTrue();
			testCiclo();
			testLinhaPowerTrue();
			testRemoveLinha();
			testRemoveLinha2();
			testSomatorio();
			testGridConnectionCorrect();
	);

-- Testa uma cidade com 3 centrais, 5 consumidores, 4 distribuidores. Tem que passar em todas
as pre/pos-condicoes
public testCidadeTrue:() ==> ()
		testCidadeTrue() == (
			dcl fabrica : NodeFactoryClass := new NodeFactoryClass() ;
			dcl porto : SmartCity := new SmartCity();


			dcl reatorNuclear : EffectiveNode := fabrica.buildPlant(900);
			dcl barragem : EffectiveNode := fabrica.buildPlant(1200);
```

```
                dcl centralEolica : EffectiveNode := fabrica.buildPlant(700);

                dcl distribuidor1 : EffectiveNode := fabrica.buildDistributor();
                dcl distribuidor2 : EffectiveNode := fabrica.buildDistributor();
                dcl distribuidor3 : EffectiveNode := fabrica.buildDistributor();
                dcl distribuidor4 : EffectiveNode := fabrica.buildDistributor();

                dcl consumidor1 : EffectiveNode := fabrica.buildConsumer(500);
                dcl consumidor2 : EffectiveNode := fabrica.buildConsumer(113);
                dcl consumidor3 : EffectiveNode := fabrica.buildConsumer(500);
                dcl consumidor4 : EffectiveNode := fabrica.buildConsumer(200);
                dcl consumidor5 : EffectiveNode := fabrica.buildConsumer(1001);

                dcl somatorio : int;

                porto.addNode(distribuidor1); porto.addNode(distribuidor2);
porto.addNode(distribuidor3); porto.addNode(distribuidor4);
                porto.addNode(consumidor1); porto.addNode(consumidor2);
porto.addNode(consumidor3); porto.addNode(consumidor4); porto.addNode(consumidor5);
                porto.addNode(reatorNuclear); porto.addNode(barragem);
porto.addNode(centralEolica);

                IO`println("Adicionado Ligacao Dist1 Dist2
");porto.buildLigacao(distribuidor1,distribuidor2,1000);
                IO`println("Adicionado Ligacao Dist1 Dist3
");porto.buildLigacao(distribuidor1,distribuidor3,1000);
                IO`println("Adicionado Ligacao Dist1 Dist4
");porto.buildLigacao(distribuidor1,distribuidor4,1000);
                IO`println("Adicionado Ligacao Dist2 Dist3
");porto.buildLigacao(distribuidor2,distribuidor3,1000);
                IO`println("Adicionado Ligacao Dist2 Dist4
");porto.buildLigacao(distribuidor2,distribuidor4,1000);
                IO`println("Adicionado Ligacao Dist3 Dist4
");porto.buildLigacao(distribuidor3,distribuidor4,1000);
                IO`println("Adicionada Ligacao NuclearPlant Dist3
");porto.buildLigacao(distribuidor3,reatorNuclear,1000);
                IO`println("Adicionada Ligacao Consumidor Dist2
");porto.buildLigacao(consumidor1,distribuidor2,1000);
                IO`println("Adicionada Ligacao Consumidor2 Dist4
");porto.buildLigacao(consumidor2,distribuidor4,1000);
                IO`println("Adicionada Ligacao WindPlant Dist2
");porto.buildLigacao(centralEolica,distribuidor1,1000);
                IO`println("Adicionada Ligacao Consumidor3 Dist4
");porto.buildLigacao(consumidor3,distribuidor4,1000);
                IO`println("Adicionada Ligacao Consumidor4 Dist1
");porto.buildLigacao(distribuidor1,consumidor4,1000);
                IO`println("Adicionada Ligacao BarragemDoAlqueva Dist2
");porto.buildLigacao(barragem,distribuidor4,1201);
                IO`println("Adicionada Ligacao Consumidor5 Dist1
");porto.buildLigacao(distribuidor1,consumidor5,1002);

                somatorio := porto.calculateNodeConsumption(distribuidor2);
                assertEqual(somatorio,2314);
                IO`println(somatorio);
                somatorio := porto.calculateNodeProduction(distribuidor2);
                assertEqual(somatorio,2800);
                IO`println(somatorio);
        );

-- Testa uma cidade com 3 centrais, 5 consumidores, 4 distribuidores. Tem que falhar a
adicionar um 3º consumidor porque a rede fica sem capacidade.
public testCidadeFalse:() ==> ()
        testCidadeFalse() == (
                dcl fabrica : NodeFactoryClass := new NodeFactoryClass() ;
                dcl porto : SmartCity := new SmartCity();
```

```
                dcl reatorNuclear : EffectiveNode := fabrica.buildPlant(900);
                dcl barragem : EffectiveNode := fabrica.buildPlant(1200);
                dcl centralEolica : EffectiveNode := fabrica.buildPlant(700);
                dcl centralEolica2 : EffectiveNode := fabrica.buildPlant(700);

                dcl distribuidor1 : EffectiveNode := fabrica.buildDistributor();
                dcl distribuidor2 : EffectiveNode := fabrica.buildDistributor();
                dcl distribuidor3 : EffectiveNode := fabrica.buildDistributor();
                dcl distribuidor4 : EffectiveNode := fabrica.buildDistributor();

                dcl consumidor1 : EffectiveNode := fabrica.buildConsumer(500);
                dcl consumidor2 : EffectiveNode := fabrica.buildConsumer(113);
                dcl consumidor3 : EffectiveNode := fabrica.buildConsumer(500);
                dcl consumidor4 : EffectiveNode := fabrica.buildConsumer(200);
                dcl consumidor5 : EffectiveNode := fabrica.buildConsumer(1001);

                porto.addNode(distribuidor1); porto.addNode(distribuidor2);
porto.addNode(distribuidor3); porto.addNode(distribuidor4);
                porto.addNode(consumidor1); porto.addNode(consumidor2);
porto.addNode(consumidor3); porto.addNode(consumidor4); porto.addNode(consumidor5);
                porto.addNode(reatorNuclear); porto.addNode(barragem);
porto.addNode(centralEolica); porto.addNode(centralEolica2);

                IO`println("Adicionado Ligacao Dist1 Dist2
");porto.buildLigacao(distribuidor1,distribuidor2,1000);
                IO`println("Adicionado Ligacao Dist1 Dist3
");porto.buildLigacao(distribuidor1,distribuidor3,1000);
                IO`println("Adicionado Ligacao Dist1 Dist4
");porto.buildLigacao(distribuidor1,distribuidor4,1000);
                IO`println("Adicionado Ligacao Dist2 Dist3
");porto.buildLigacao(distribuidor2,distribuidor3,1000);
                IO`println("Adicionado Ligacao Dist2 Dist4
");porto.buildLigacao(distribuidor2,distribuidor4,1000);
                IO`println("Adicionado Ligacao Dist3 Dist4
");porto.buildLigacao(distribuidor3,distribuidor4,1000);

                IO`println("Adicionada Ligacao Plant Dist3
");porto.buildLigacao(distribuidor3,reatorNuclear,1000);
                --IO`println("Adicionada Ligacao CentralEolica Dist1
");porto.buildLigacao(distribuidor1,centralEolica,1000);

                IO`println("Adicionada Ligacao Consumidor1 Dist2
");porto.buildLigacao(consumidor1,distribuidor2,1000);
                IO`println("Adicionada Ligacao Consumidor2 Dist4
");porto.buildLigacao(consumidor2,distribuidor4,1000);
                IO`println("Adicionada Ligacao Consumidor3 Dist4
");porto.buildLigacao(consumidor3,distribuidor4,1000);

        );

-- Testa uma cidade com ligações circulares. Nao deve ter problemas a calcular o somatorio.
Assert deve dar verdadeiro.
    public testCiclo:() ==> ()
        testCiclo() == (
                --dcl nuclearPlant : EffectiveNode := new EffectiveNode(true,false,900,0);
                dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
                dcl dist2 : EffectiveNode := new EffectiveNode(false,false,0,0);
                dcl dist3 : EffectiveNode := new EffectiveNode(false,false,0,0);
                dcl dist4 : EffectiveNode := new EffectiveNode(false,false,0,0);
                dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,500);
                dcl consumidor2 : EffectiveNode := new EffectiveNode(false,true,0,113);
                dcl consumidor3 : EffectiveNode := new EffectiveNode(false,true,0,500);
                dcl consumidor4 : EffectiveNode := new EffectiveNode(false,true,0,200);
```

```
		dcl consumidor5 : EffectiveNode := new EffectiveNode(false,true,0,1001);
		dcl somatorio : int;
		dist1.addLigacao(dist2,1000);
		dist1.addLigacao(dist3,1000);
		dist1.addLigacao(dist4,1000);
		dist2.addLigacao(dist3,1000);
		dist2.addLigacao(dist4,1000);
		dist3.addLigacao(dist4,1000);
		dist2.addLigacao(consumidor,1000);
		dist4.addLigacao(consumidor2,1000);
		dist4.addLigacao(consumidor3,1000);
		dist1.addLigacao(consumidor4,1000);
		dist3.addLigacao(consumidor5,1002);
		somatorio := dist2.calculateConsumption();
		assertEqual(somatorio,2314);
		IO`println(somatorio);
	);


	public testLinhaPowerTrue:() ==> ()
	testLinhaPowerTrue() == (
		dcl porto : SmartCity := new SmartCity();
		dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
		dcl nuclearPlant : EffectiveNode := new EffectiveNode(true,false,900,0);
		dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,500);


		porto.addNode(dist1); porto.addNode(nuclearPlant);
porto.addNode(consumidor);

		porto.buildLigacao(dist1,nuclearPlant,1000);
		IO`println("Adicionado Consumidor de 500W a um Dist1 com uma linha que
suporta no Maximo 501W ");porto.buildLigacao(dist1,consumidor,501);
	);

	public testRemoveLinha:() ==> ()
	 testRemoveLinha() == (
		dcl porto : SmartCity := new SmartCity();
		dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
		dcl nuclearPlant : EffectiveNode := new EffectiveNode(true,false,900,0);
		dcl nuclearPlant2 : EffectiveNode := new EffectiveNode(true,false,1000,0);
		dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,500);
		dcl consumidor2 : EffectiveNode := new EffectiveNode(false,true,0,500);
		dcl somatorio : int;

		porto.addNode(dist1); porto.addNode(nuclearPlant);
porto.addNode(nuclearPlant2); porto.addNode(consumidor);  porto.addNode(consumidor2);

		porto.buildLigacao(dist1,nuclearPlant,1000);
		porto.buildLigacao(dist1,consumidor,1000);
		porto.buildLigacao(dist1,nuclearPlant2,1001);
		porto.buildLigacao(dist1,consumidor2,1000);

		porto.zerarFlags();
		somatorio := dist1.calculateProduction();
		IO`println(somatorio);
		porto.zerarFlags();

		porto.removeLigacao(dist1,nuclearPlant);

		porto.zerarFlags();
		somatorio := dist1.calculateProduction();
		IO`println(somatorio);
		porto.zerarFlags();
```

```
		porto.zerarFlags();
		somatorio := dist1.calculateConsumption();
		IO`println(somatorio);
		porto.zerarFlags();

		porto.removeLigacao(dist1,consumidor);

		porto.zerarFlags();
		somatorio := dist1.calculateConsumption();
		IO`println(somatorio);
		porto.zerarFlags();
	);

	public testRemoveLinha2:() ==> ()
	 testRemoveLinha2() == (
		dcl porto : SmartCity := new SmartCity();
		dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
		dcl nuclearPlant : EffectiveNode := new EffectiveNode(true,false,900,0);
		dcl nuclearPlant2 : EffectiveNode := new EffectiveNode(true,false,1000,0);
		dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,500);
		dcl consumidor2 : EffectiveNode := new EffectiveNode(false,true,0,500);
		dcl somatorio : int;

		porto.addNode(dist1); porto.addNode(nuclearPlant);
porto.addNode(nuclearPlant2); porto.addNode(consumidor);  porto.addNode(consumidor2);

		porto.buildLigacao(dist1,nuclearPlant,1000);
		porto.buildLigacao(dist1,consumidor,1000);
		porto.buildLigacao(dist1,nuclearPlant2,1001);
		porto.buildLigacao(dist1,consumidor2,1000);

		porto.zerarFlags();
		somatorio := dist1.calculateProduction();
		IO`println(somatorio);
		porto.zerarFlags();

		porto.removeLigacao(nuclearPlant,dist1);

		porto.zerarFlags();
		somatorio := dist1.calculateProduction();
		IO`println(somatorio);
		porto.zerarFlags();

		porto.zerarFlags();
		somatorio := dist1.calculateConsumption();
		IO`println(somatorio);
		porto.zerarFlags();

		porto.removeLigacao(consumidor,dist1);

		porto.zerarFlags();
		somatorio := dist1.calculateConsumption();
		IO`println(somatorio);
		porto.zerarFlags();
	);

	public testLinhaPowerFalse:() ==> ()
	testLinhaPowerFalse() == (
		dcl porto : SmartCity := new SmartCity();
		dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
		dcl nuclearPlant : EffectiveNode := new EffectiveNode(true,false,900,0);
		dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,500);

		porto.addNode(dist1); porto.addNode(nuclearPlant);
porto.addNode(consumidor);
```

```
                porto.buildLigacao(dist1,nuclearPlant,1000);
                IO`println("Adicionado Consumidor de 500W a um Dist1 com uma linha que
suporta no Maximo 499W ");porto.buildLigacao(dist1,consumidor,499);
        );


    public testSomatorio:() ==> ()
        testSomatorio() == (
            dcl porto : SmartCity := new SmartCity();
            dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
            dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,100);
            dcl consumidor2 : EffectiveNode := new EffectiveNode(false,true,0,100);
            dcl nuclearplant : EffectiveNode := new EffectiveNode(true,false,500,0);
            dcl somatorio : int;
            porto.addNode(dist1);
            porto.addNode(consumidor);
            porto.addNode(consumidor2);
            porto.addNode(nuclearplant);
            porto.buildLigacao(dist1,nuclearplant,1000);
            porto.buildLigacao(dist1,consumidor,1000);
            porto.buildLigacao(dist1,consumidor2,1000);


            somatorio := porto.calculateNodeConsumption(dist1);
            assertEqual(somatorio,200);
            IO`println(somatorio);
        );


    public testGridConnectionCorrect: () ==> ()
        testGridConnectionCorrect() == (
            dcl nuclearPlant : EffectiveNode := new EffectiveNode(true,false,900,0);
            dcl dist1 : EffectiveNode := new EffectiveNode(false,false,0,0);
            dcl dist2 : EffectiveNode := new EffectiveNode(false,false,0,0);
            dcl consumidor : EffectiveNode := new EffectiveNode(false,true,0,100);

            nuclearPlant.addLigacao(dist1,1500);
            dist1.addLigacao(consumidor,1500);
            dist1.addLigacao(dist2,1500);
        );


        public testGridConnectionBadLayout :() ==> ()
            testGridConnectionBadLayout() == (
                dcl nuclearPlant : EffectiveNode := new
EffectiveNode(true,false,900,0);
                dcl consumidor : EffectiveNode := new
EffectiveNode(false,true,0,100);

                nuclearPlant.addLigacao(consumidor, 1000);
            );

end TestSmartGrid
```

Conclusões

Após efectuar este trabalho em VDM++ através do IDE Overture, chegamos à conclusão que com certeza existem melhores linguagens/plataformas para desenvolver
um trabalho deste género. A sintaxe desta linguagem, provém duma mistura de linguagens conhecidas (como C, Java, Pascal, SQL, Scheme...), o que nos leva a
olha recorrentemente para exemplos feitos anteriormente com receio de algo escapar no nosso escopo de visão. A documentação era pouca, mesmo quase nenhuma tendo
como excepção o manual disponibilizado pelos criadores que foi uma óptima ajuda, mas não conseguindo cobrir um vasto campo de problemas que vão surgindo.

O IDE Overture, mesmo que tendo por base o conhecido
Eclipse, afasta-se de bastante deste principalmente a nível de usabilidade. O Overture parecia ter consciência própria e efectuar alguns comandos apenas quando
se encontrava bem-disposto. Casos como exportar o UML (que exportava todas as classes menos uma, após alguma insistência e sem qualquer alteração de uma linha
de código lá por assumir aquilo que era suposto.

Em termos do âmbito da disciplina, foi bastante enriquecedor, deixando-nos mais atentos a quaisquer possíveis condições que escapam no dia-a-dia de um programador.
Acredito que seria mais criativo, interessante e útil se usada uma linguagem com alguma adesão.

Bibliografia

Faria, J. P. (s.d.). *Formal Modeling of a Vending Machine in VDM++.* Obtido de Formal Modeling of a Vending Machine in VDM++.