Insulin Dosage Advisor System

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Pedro Faria – ei11167@fe.up.pt
Rafaela Faria – ei12129@fe.up.pt
Rui Figueira – ei11021@fe.up.pt

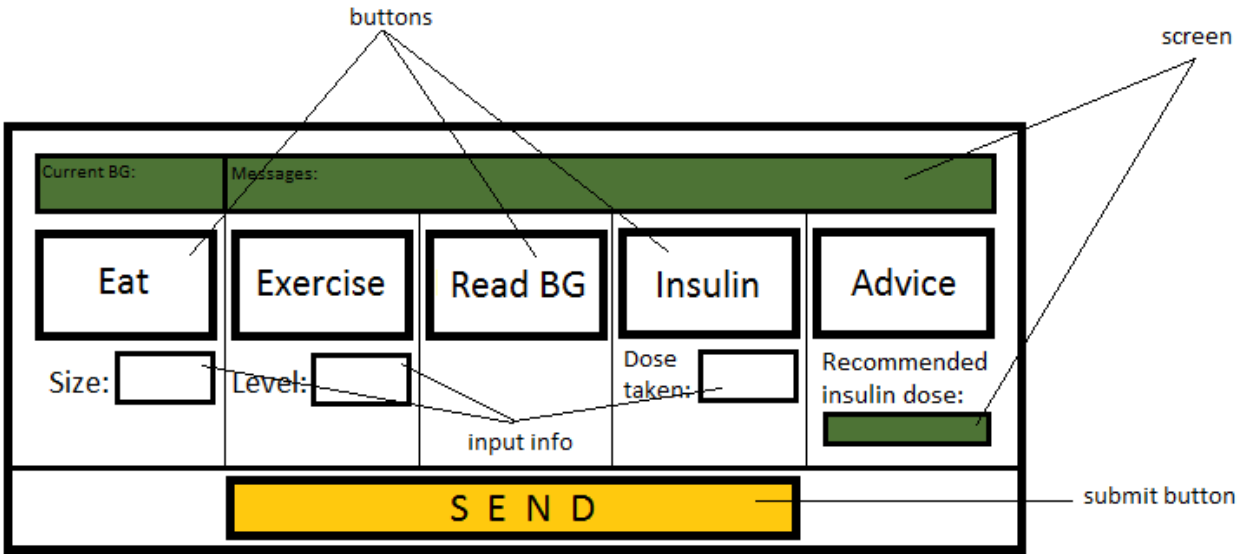
Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

19 de Dezembro de 2014

Ìndice

1. Descrição informal do sistema e lista de requisitos

1.1. Descrição informal do sistema

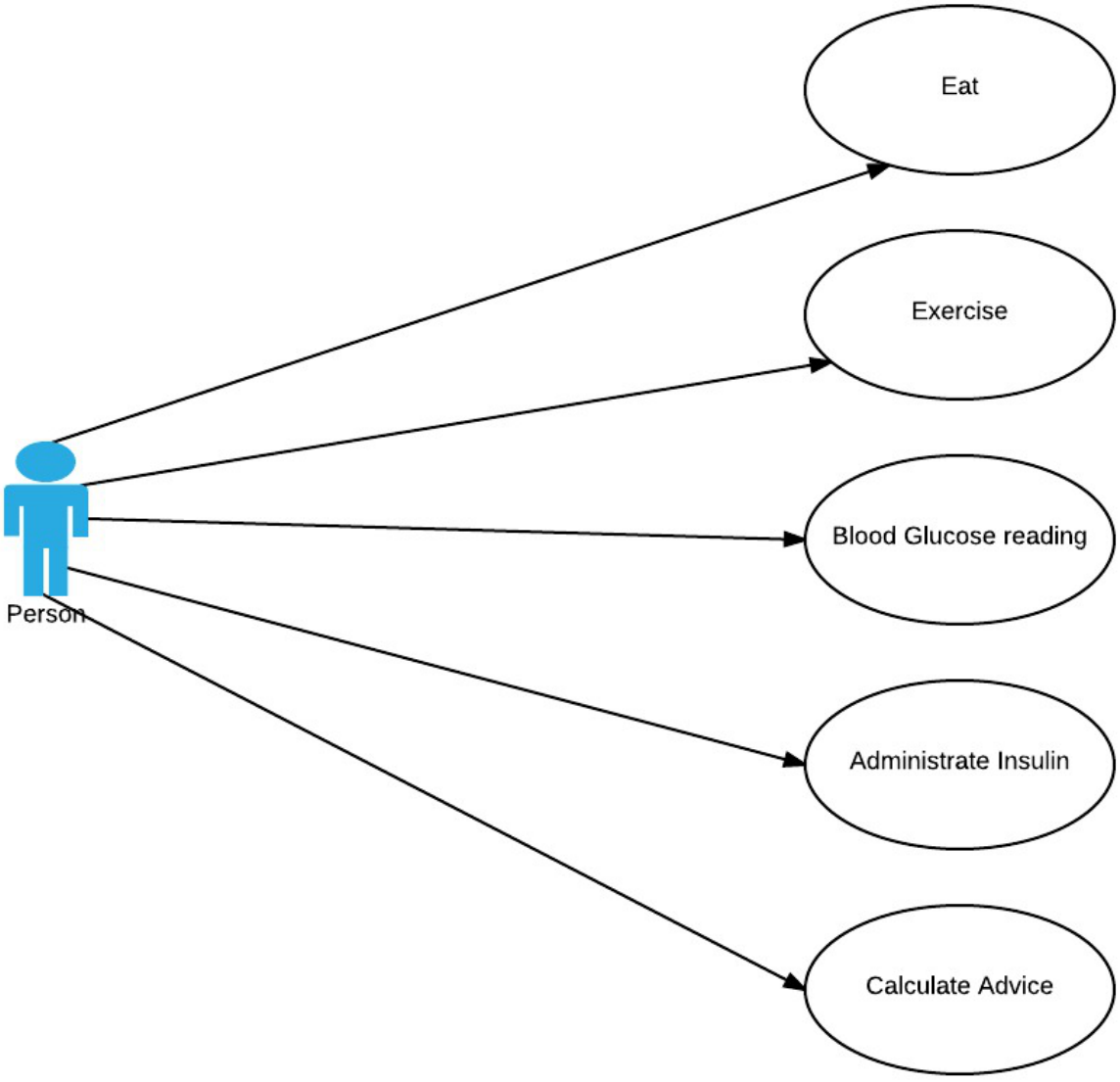


1.2. Lista de requisitos

Id	Priority	Description
R1	Obrigatório	A pessoa pode comer, introduzindo a hora da refeição e o seu tamanho (tamanho: 0 - nada, 1 - ligeira, 2 - normal, 3 - grande)
R2	Obrigatório	A pessoa pode fazer exercício físico, introduzindo a hora e o nível de esforço (nível: 0 - nenhum, 1 - mínimo, 2 - normal, 3 - intenso)
R3	Obrigatório	A pessoa pode verificar o seu nível de glicose no sangue, introduzindo a hora do pedido
R4	Obrigatório	A pessoa pode administrar insulina, introduzindo a hora do acto e a dose injectada
R5	Obrigatório	A pessoa pode pedir um conselho de dose a tomar, introduzindo a hora do pedido

2. UML

2.1. Modelo de casos de uso



Cenários dos casos de uso

Cenário	Eat
Descrição	Cenário que ocorre quando a pessoa come.
Pré-Condições	Verificar se a acção actual, uma variável que é actualizada após cada acção, é do tipo <EAT> ou <NORMAL>
Steps	1. Actualiza a glicose no sangue, consoante o tamanho da refeição (aumenta a glicose no sangue). 2. Actualiza a variável "lastMealTime" para a hora recebida (para cálculo de jejum) 2. Verifica se a glicose actualizada ultrapassa o limite máximo saudável. 3. Caso esse limite seja ultrapassado, actualiza a acção seguinte para <INSULIN>. 4. Caso contrário, actualiza-se para <NORMAL>.

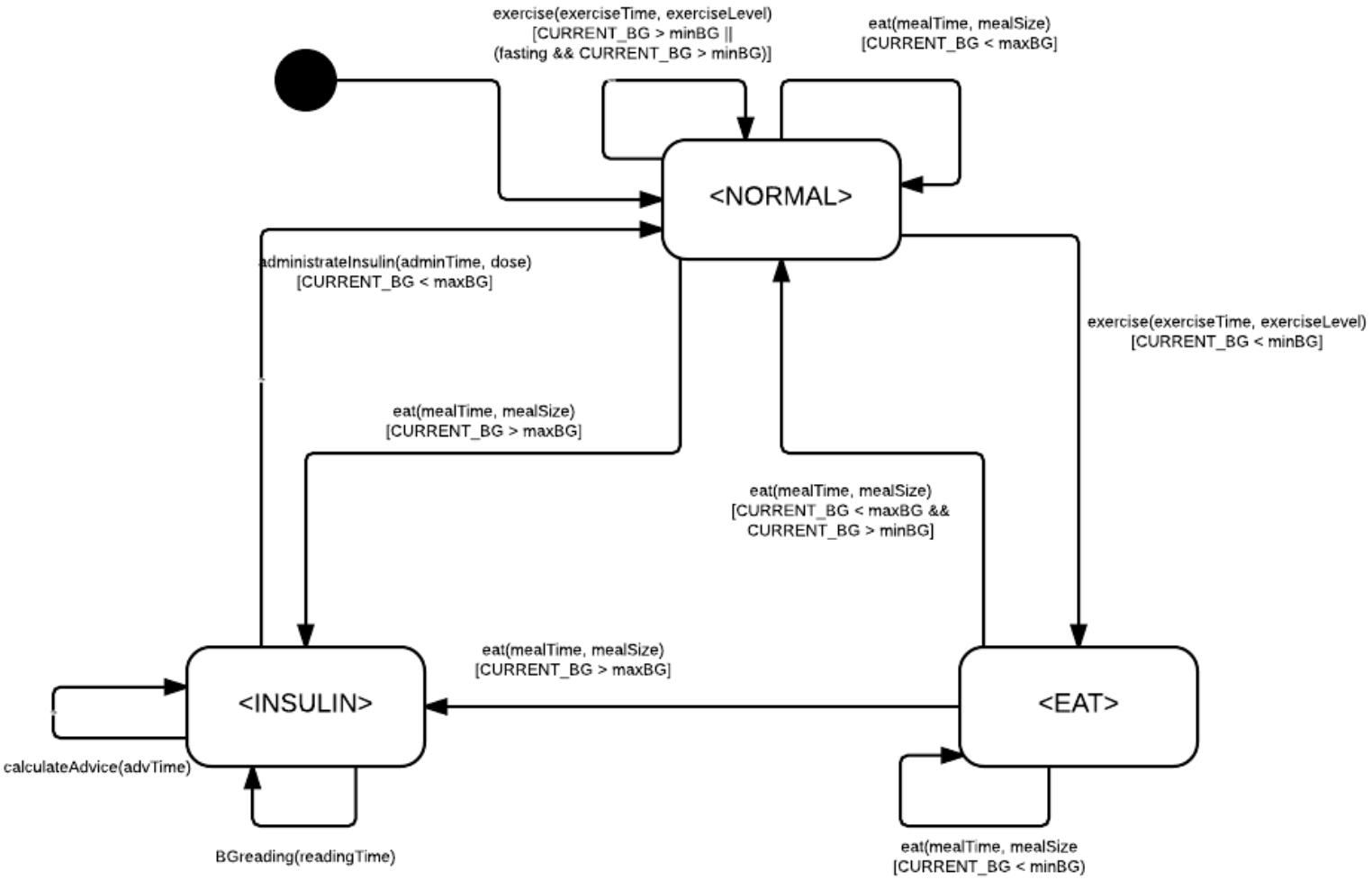
Cenário	Exercise
Descrição	Cenário que ocorre quando a pessoa faz exercício físico.
Pré-Condições	Verificar se a acção actual é do tipo <NORMAL>
Steps	1. Actualiza a glicose no sangue, consoante o tamanho da refeição (diminui a glicose no sangue). 2. Verifica se a glicose actualizada é menos que o limite mínimo saudável. 3. Caso a glicose esteja abaixo do limite mínimo, actualiza a acção seguinte para <EAT>. 4. Caso contrário, actualiza-se para <NORMAL>.

Cenário	Blood Glucose Reading
Descrição	Cenário que ocorre quando a pessoa mede a glicose no sangue.
Steps	1. Através da hora da medição e da hora da última refeição calcula a possibilidade de jejum. 2. Caso se verifique jejum e a glicose no sangue seja superior 130, actualiza a acção seguinte para <INSULIN>. 3. Caso tenha comido há menos de 6 horas (não está em jejum) e a glicose no sangue seja superior a 200, actualiza a acção seguinte para <INSULIN>. 4. Caso não esteja em jejum, verifica se a glicose no sangue é superior ao limite mínimo. 5. Se não for, actualiza a acção seguinte para <EAT>. 6. Se estiver tudo dentro dos limites, actualiza a acção para <NORMAL>.

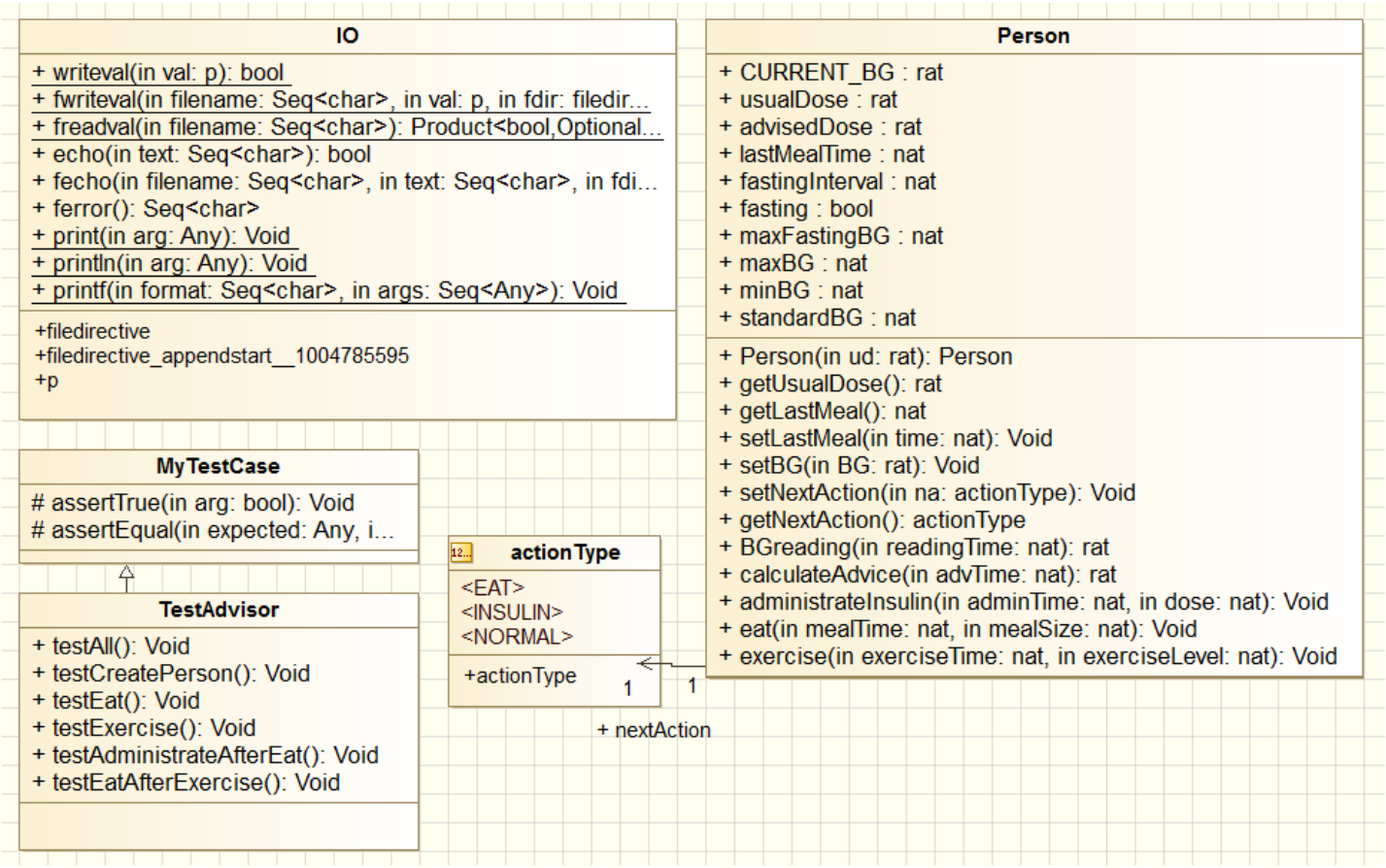
Cenário	Administrate Insulin
Descrição	Cenário que ocorre quando a pessoa toma insulina.
Pré-Condições	Verificar se a acção actual é do tipo <INSULIN> Verificar se a dose a administrar é a sua dose usual ou uma dose sugerida caso essa exista.
Steps	1. Actualiza a glicose no sangue, conforme a dose de insulina. 2. Actualiza a proxima acção conforme a glicose no sangue.

Cenário	Calculate Advice
Descrição	Cenário que ocorre quando a pessoa pede um conselho da dose de insulina a tomar.
Pré-Condições	A última refeição só pode estar distanciada uma hora (no máximo) do pedido.
Steps	1. Verificar se o valor da glicose no sangue é menor que o mínimo aceitável. 2. Caso se verifique, actualiza a próxima acção para <EAT>. 3. Caso contrário, verifica se a glicose está entre os limites aceitáveis. 4. Se estiver, actualiza a próxima acção para <NORMAL> 5. Se não estiver, significa que está acima do valor máximo, e calcula a dose que vai recomendar retornando-a.

2.2. Modelo da máquina de estados



2.3. Modelo de classes



Class	Descrição
Person	Modelo principal. Define o estado das variáveis e todas as operações do utilizador.
MyTestCase	Super-classe para a classe de testes. Define o assertEquals e o assertTrue.
TestAdvisor	Define os testes e cenários para pessoa.

3. Modelo VDM++ formal

3.1. Class Person

```
class Person
/*
	Contains the core model and functions of insulinAdvisor.
	Defines the state variables and operations available to the user.
*/
types

public actionType = <EAT> | <INSULIN> | <NORMAL>;

instance variables

	public CURRENT_BG : rat := 80;
	public usualDose : rat;
	public advisedDose : rat := 0;

	public nextAction : actionType := <NORMAL>;

	public lastMealTime : nat := 0;
	public fastingInterval : nat := 6; -- max interval without eating
	public fasting : bool := false;

	-- standard values
	public maxFastingBG : nat := 130;
	public maxBG : nat := 200;
	public minBG : nat := 70;
	public standardBG : nat := 100;

operations
	-- new Person constructor
	public Person: rat ==> Person
		Person(ud) == (
		usualDose := ud;
		advisedDose := ud;
		return self);

-- getters and setters
public getUsualDose: () ==> rat
	getUsualDose() == return usualDose;

public getLastMeal: () ==> nat
	getLastMeal() == return lastMealTime;
```

```
public setLastMeal: nat ==> ()
	setLastMeal(time) == lastMealTime := time;

public setBG: rat ==> ()
	setBG(BG) == CURRENT_BG := BG;

public setNextAction: actionType ==> ()
	setNextAction(na) == nextAction := na;

public getNextAction: () ==> actionType
	getNextAction() == return nextAction;

-- receives the time of reading and calculates the fasting variable and sets next
action accordingly
-- returns the current glucose on blood
public BGreading: nat ==> rat
	BGreading(readingTime) == (

				if((readingTime - getLastMeal()) > fastingInterval) then(
					fasting := true)
				else (
					fasting := false
				);

				if((fasting and CURRENT_BG >= maxFastingBG) or (not fasting
and CURRENT_BG >= maxBG)) then (
					IO`println("BG over maximum, administrate insulin!");
					setNextAction(<INSULIN>);
				)
				else if(CURRENT_BG <= minBG) then (
					IO`println("BG under minimum, you must eat!");
					 setNextAction(<EAT>);
			)
			else(
				setNextAction(<NORMAL>);
			);

				return CURRENT_BG;
	);

-- receives the time of function call and calculates the recommended dose to
administrate
-- returns the recommended dose to administrate or an info message
public calculateAdvice: nat ==> rat
	calculateAdvice(advTime) == (
		if((advTime - getLastMeal() > 1))
			then (
			IO`println("Can't advice, last meal was 1h+ ago");
```

```
                    return 0)
                else(
                    if(CURRENT_BG < minBG) then (

                                IO`println("Dont administrate - BG under minimum, you
must eat!");

                                setNextAction(<EAT>);

                                return 0;

                                )

                                else if((CURRENT_BG >= minBG) and (CURRENT_BG <=
standardBG))then(

                                IO`println("Dont administrate - BG withing expected
levels!");

                                setNextAction(<NORMAL>);

                                return 0;

                                )

                                else(

                                advisedDose := ((CURRENT_BG-100)/10);

                                return advisedDose;

                                )
                    )
                    );

-- receives the time and dose of administration and sets currentBG and next action
accordingly
public administrateInsulin: nat * nat ==> ()
        administrateInsulin(adminTime, dose) == (
                setBG(CURRENT_BG - (dose * 10));

                if(BGreading(adminTime) > 0) then (return;)     --refreshes person's state
        )
        pre (nextAction = <INSULIN> and (dose = usualDose or dose = advisedDose and
advisedDose > 0));

-- receives the time and the size of the meal and sets currentBG and next action
accordingly
```

```
-- mealSize: 0 - nothing, 1 - light, 2 - normal, 3 - large
public eat: nat * nat ==> ()
	eat(mealTime, mealSize) == (

	setBG((CURRENT_BG + (10 * mealSize)));

	if(CURRENT_BG > maxBG) then setNextAction(<INSULIN>);

	if(mealSize > 0) then	setLastMeal(mealTime);

	if(BGreading(mealTime) > 0) then (return;) --refreshes person's state
	)
	pre nextAction in set elems [ <EAT> , <NORMAL> ] ;

-- receives the time and the level of the exercise and sets currentBG and next action
accordingly
-- exerciseLevel: 0 - none, 1 - minimal, 2 - normal, 3 - heavy
public exercise: nat * nat ==> ()
	exercise(exerciseTime, exerciseLevel) == (

			setBG((CURRENT_BG - (8 * exerciseLevel)));

			if(CURRENT_BG < minBG) then setNextAction(<EAT>);

			if(BGreading(exerciseTime) > 0) then (return;)
	)
	pre nextAction = <NORMAL>;


end Person
```

4. Modelo de validação

4.1. Class 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
```

4.2. Class TestAdvisor

```
class TestAdvisor is subclass of MyTestCase
/*
  Contains the test cases for the person.
  Illustrates a scenario-based testing approach.
  The test cases cover all usage scenarios as well as all states and transitions.
*/
```

```
operations

-- run all the tests
public testAll: () ==> ()
	testAll() == (
		IO`println("testCreatePerson");
		testCreatePerson();
		IO`println("testCreatePerson passed");
		IO`println(" ");
		IO`println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");
		IO`println("testEat");
		testEat();
		IO`println("testEat passed");
		IO`println(" ");
		IO`println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");
		IO`println("testExercise");
		testExercise();
		IO`println("testExercise passed");
		IO`println(" ");
		IO`println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");
		IO`println("testAdministrateAfterEat");
		testAdministrateAfterEat();
		IO`println("testAdministrateAfterEat passed");
		IO`println(" ");
		IO`println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");
		IO`println("testEatAfterExercise");
		testEatAfterExercise();
		IO`println("testEatAfterExercise passed");
		IO`println("-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+");

		);

-- test to verify if inicial values of the new instance of person are ok
public testCreatePerson: () ==> ()
	testCreatePerson() == (

	dcl p: Person := new Person(5);

	assertEqual(p.BGreading(1), 80);
	assertEqual(p.calculateAdvice(1) , 0);
	assertEqual(p.getNextAction() , <NORMAL>);

	);

-- tests to verify if person operations are working as expected
public testEat : () ==> ()
```

```
	testEat() == (
		dcl p1: Person := new Person(5);
		dcl p2: Person := new Person(5);

		IO`println("p1: Eating: 1h, size 3");
		p1.eat(1, 3);
		IO`println("p1: Reading at 1h -> after eating");
		assertEqual(p1.BGreading(1), 110);
		IO`println("p1:Calculate advice at 2h ->");
		assertEqual(p1.calculateAdvice(2), 1);
		IO`println("p1:Calculate advice at 3h ->");
		assertEqual(p1.calculateAdvice(3), 0);

		IO`println("p2: Eating: 1h, size 3");
		p2.eat(1, 3);
		IO`println("p2: Eating: 1h, size 3");
		p2.eat(2, 3);
		IO`println("p2: Reading at 2h -> after eating");
		assertEqual(p2.BGreading(2), 140);
		IO`println("p2:Calculate advice at 2h ->");
		assertEqual(p2.calculateAdvice(2), 4);
		IO`println("p2:Calculate advice at 3h -> Still reads");
		assertEqual(p2.calculateAdvice(3), 4);

		IO`println("p2: Eating: 3h, size 3");
		p2.eat(3, 3);
		IO`println("p2: Eating: 4h, size 3");
		p2.eat(4, 3);
		IO`println("p2: BG at 4h, after 4 meals");
		assertEqual(p2.BGreading(4), 200);
		IO`println("p2:Calculate advice at 4h -> Needs Insulin");
		assertEqual(p2.calculateAdvice(4), 10);

	);

public testExercise : () ==> ()
	testExercise() == (
		dcl p1: Person := new Person(5);
		dcl p2: Person := new Person(5);

		IO`println("p1: Exercise: 1h, level 3");
		p1.exercise(1, 3);
		IO`println("p1: Reading at 1h -> ");
		assertEqual(p1.BGreading(1), 56);


		IO`println("Exercise: 1h, level 1");
		p2.exercise(1, 1);
```

```
		IO`println("p2: Reading at 1h -> ");
		assertEqual(p2.BGreading(1), 72);

);

public testAdministrateAfterEat : () ==> ()
	testAdministrateAfterEat() == (
	dcl p: Person := new Person(5);
		IO`println("p: Eating: 1h, size 3");
		p.eat(1, 3);
		IO`println("p: Eating: 1h, size 3");
		p.eat(2, 3);
		IO`println("p: Eating: 3h, size 3");
		p.eat(3, 3);
		IO`println("p: Eating: 4h, size 3");
		p.eat(4, 3);
		IO`println("p: BG at 4h, after 4 meals");
		assertEqual(p.BGreading(4), 200);
		IO`println("p:Calculate advice at 4h -> Needs Insulin");
		assertEqual(p.calculateAdvice(4), 10);
		IO`println("p:Administrate at 4h advised amount");
		p.administrateInsulin(4, 10);
		IO`println("p:State after administrate is NORMAL");
		assertEqual(p.getNextAction(), <NORMAL>);
);

public testEatAfterExercise : () ==> ()
	testEatAfterExercise() == (
	dcl p: Person := new Person(5);

		IO`println("p: Exercise: 1h, level 3");
		p.exercise(1, 3);
		IO`println("p: Reading at 1h -> ");
		assertEqual(p.BGreading(1), 56);

		IO`println("p: Eating: 2h, size 3 -> Still under");
		p.eat(2, 1);
		assertEqual(p.getNextAction(), <EAT>);
		IO`println("p: Eating: 3h, size 3 -> OK");
		p.eat(3, 3);
		assertEqual(p.getNextAction(), <NORMAL>);
);

end TestAdvisor
```

5. Verificação do modelo

5.1. Exemplo de verificação invariante

No.	PO Name	Type
4	Person`Person(rat)	state invariant holds
5	Person`setLastMeal(nat)	state invariant holds
6	Person`setBG(rat)	state invariant holds
7	Person`setNextAction(actionType)	state invariant holds
8	Person`BGreading(nat)	state invariant holds
9	Person`calculateAdvice(nat)	state invariant holds
10	Person`calculatedAdvice(nat)	state invariant holds

No. 4 -

```
public Person: rat ==> Person
              Person(ud) == (
              usualDose := ud;
              advisedDose := ud;
              return self);
```

No. 5 -

```
public setLastMeal: nat ==> ()
       setLastMeal(time) == lastMealTime := time;
```

No. 6 -

```
public setBG: rat ==> ()
       setBG(BG) == CURRENT_BG := BG;
```

No. 7 -

```
public setNextAction: actionType ==> ()
	setNextAction(na) == nextAction := na;
```

No. 8 e 9 -

```
public BGreading: nat ==> rat
	BGreading(readingTime) == (
		if((readingTime - getLastMeal()) > fastingInterval) then(
			fasting := true)
		else (
			fasting := false
		);
		if((fasting and CURRENT_BG >= maxFastingBG) or (not fasting and
CURRENT_BG >= maxBG)) then (
			IO`println("BG over maximum, administrate insulin!");
			setNextAction(<INSULIN>);
		)
		else if(CURRENT_BG <= minBG) then (
			IO`println("BG under minimum, you must eat!");
			setNextAction(<EAT>);
		)
		else(
			setNextAction(<NORMAL>);
		);
		return CURRENT_BG;
);
```

No. 10 -

```
public calculateAdvice: nat ==> rat
	calculateAdvice(advTime) == (
		if((advTime - getLastMeal() > 1))
			then (
			IO`println("Can't advice, last meal was 1h+ ago");
			return 0)
		else(
			if(CURRENT_BG < minBG) then (

	IO`println("Dont administrate - BG under minimum, you must eat!");

	setNextAction(<EAT>);

	return 0;

	)

	else if((CURRENT_BG >= minBG) and (CURRENT_BG <= standardBG))then(

	IO`println("Dont administrate - BG withing expected levels!");

	setNextAction(<NORMAL>);

	return 0;

	)

	else(

	advisedDose := ((CURRENT_BG-100)/10);

	return advisedDose;

	)
			)
			);
```

6. Conclusão

O modelo desenvolvido cobre as principais funções que deveriam ser implementadas num dispositivo de aconselhamento de dose de insulina.

Do nosso ponto de vista, pela análise do artigo disponibilizado no enunciado, acreditamos ter atingido os objectivos pretendidos.

Devido à vaga informação e existência de poucos exemplos de projectos nesta linguagem, houve dificuldades no arrancar do projecto e desenvolvimento de uma maior complexidade do mesmo.

7. Referências

1. Métodos Formais em Engenharia de Software [EIC0039-1S] - Slides FEUP, https://moodle.up.pt/course/view.php?id=3192
2. Overture tool web site, http://overturetool.org
3. http://en.wikipedia.org/wiki/Insulin