Modelação Formal de um Simulador para Air Traffic Control em VDM++

(Tema 10)

Turma 5

Jorge Miguel Reis - ei08053@fe.up.pt

Rui Grandão Rocha - ei11010@fe.up.pt

Métodos Formais em Engenharia de Software

Mestrado Integrado em Engenharia Informática e Computação

17 de Dezembro de 2014

Índice

1. Informal system description and list of requirements

1.1 Descrição do Sistema

O sistema a modelar em VDM++ neste projecto é um simulador de controlo de tráfego aéreo. Os aviões devem circular numa mapa bidimensional, que representa o radar ATC, de acordo com *clock ticks*. Entram no mesmo pelos seus limites e orientados para um certo objectivo que pode ser: sair do radar pelos limites norte, sul, este ou oeste; ou aterrar no aeroporto local, devendo entrar com direcção correcta na pista de aterragem. O controlador de tráfego aéreo deve também evitar colisões entre aviões, ordenando que os mesmos rodem Xº *Clockwise* ou *Counterclockwise*, tendo em conta que apenas conseguem rodar 45º por *tick*.

1.2 Lista de Requisitos

Id	**Prioridade**	**Descrição**
R1	Obrigatório	O controlador aéreo deve visualizar a informação do radar, ou seja, informação dos aviões dentro da área controlada e respectiva posição e orientação.
R2	Obrigatório	Devem surgir aviões, entrando na área do radar por uma das suas extremidades, orientados para o seu interior.
R3	Obrigatório	O controlador aéreo deve verificar qual o objectivo do avião (sair para N, S, W, E ou aterrar) e orientá-lo de forma correcta.
R4	Obrigatório	O controlador aéreo deve prever colisões entre aviões e desviá-los atempadamente.
R5	Obrigatório	Para desviar ou orientar os aviões, o controlador aéreo deve enviar comandos de rotação com o número de graus e direcção (*Clockwise* ou *Counterclockwise*).
R6	Obrigatório	O controlador aéreo deve orientar para a pista do aeroporto pelo sentido correcto os aviões que tenham o objectivo de aterrar.
R7	Opcional	Quando necessário, o controlador aéreo deve dar indicação aos aviões para aumentarem ou diminuirem a sua velocidade.

2. Visual UML model

2.1 Use Case Model

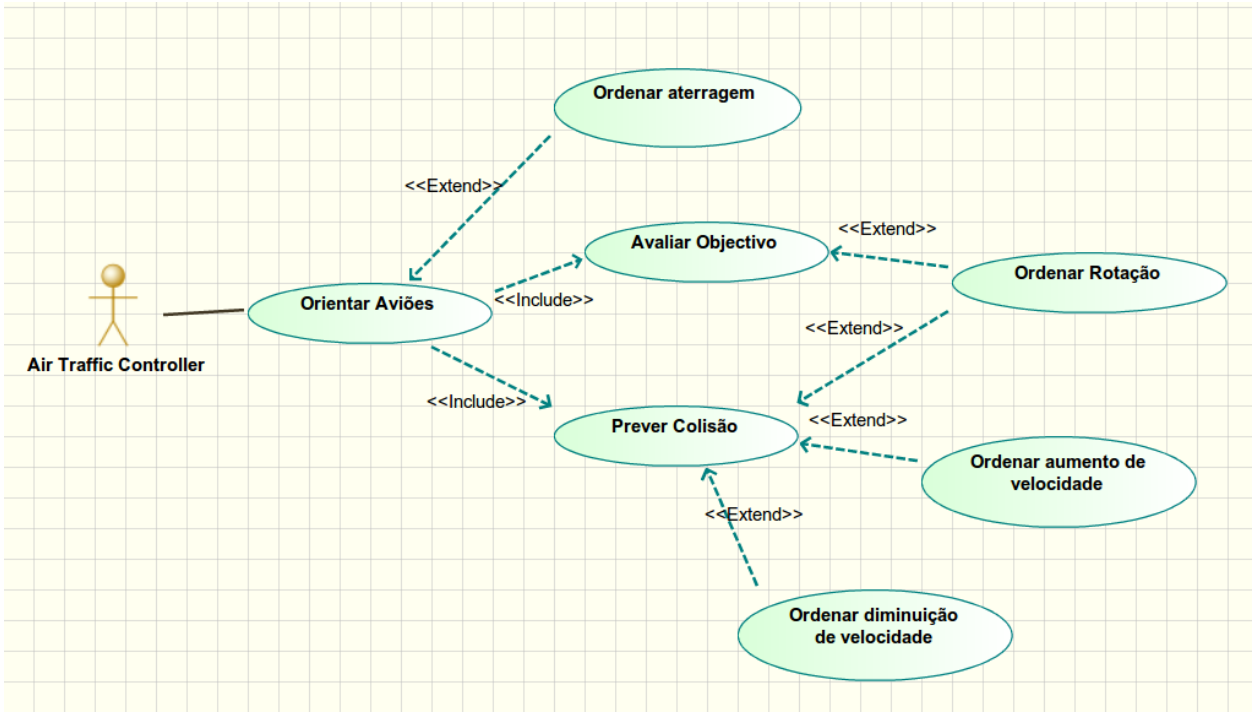


Figura 1: Principais casos de uso aplicados ao ATC (Air Traffic Controller). Os casos de uso respeitantes ao simulador são ocultados mas referidos de seguida.

Cenário	**Simular Aparecimento de Aviões**
Descrição	Entrada de aviões na área do radar, por uma das extremidades.
Pré-Condições	1. A posição do avião é uma posição válida no mapa (grid). 2. A posição inicial é uma posição extrema do mapa (grid). 3. Não existe nenhum avião na mesma posição. 4. O objectivo do avião deverá ser: N, S, E, W ou L (aterrar). 5. O avião entra no mapa orientado para o seu objectivo.
Pós-Condições	1. O objectivo (Goal) do avião foi atribuído correctamente. 2. Os valores de direcção encontra-se dentro dos limites

	normais.

Cenário	**Avaliar Objectivo**
Descrição	Verificar se o avião se encontra orientado para o seu objectivo.
Pré-Condições	1. O avião encontra-se na área do radar.
Pós-Condições	1. Caso não haja perigo de colisão, o avião deverá ficar orientado para o seu objectivo.

Cenário	**Prever Colisões**
Descrição	O controlador aéreo deve prever se no *clock tick* seguinte, tendo em conta a direcção actual dos aviões, existe o perigo de colisão.
Pré-Condições	1. Devem existir aviões no mapa.
Pós-Condições	1. Caso haja perigo de colisão, o controlador deverá ter enviado uma ordem de rotação ao avião para uma zona sem perigo.

Cenário	**Ordenar rotação**
Descrição	Pedido de rotação do avião em um número de graus indicado (múltiplo de 45º, CW ou CCW)
Pré-Condições	1. O ângulo de rotação pedido deve ser superior a zero. 2. O ângulo de rotação pedido deve ser múltiplo de 45. 3. O sentido da rotação deve ser CW ou CCW.
Pós-Condições	1. O estado final após a rotação não deve originar uma possível colisão entre aviões.

Cenário	**Ordenar aumento de velocidade**
Descrição	Aumentar velocidade do avião para uma velocidade indicada.
Pré-Condições	1. A velocidade indicada deve ser superior à velocidade actual. 2. A velocidade indicada deve ser igual ou inferior à velocidade máxima.

Pós-Condições	1. A velocidade final do avião deve ser igual ou superior à velocidade mínima. 2. A velocidade final do avião deve ser igual ou inferior à velocidade máxima. 3. A velocidade final deve ser igual à velocidade indicada. 4. A velocidade final deve ser superior à velocidade anterior.

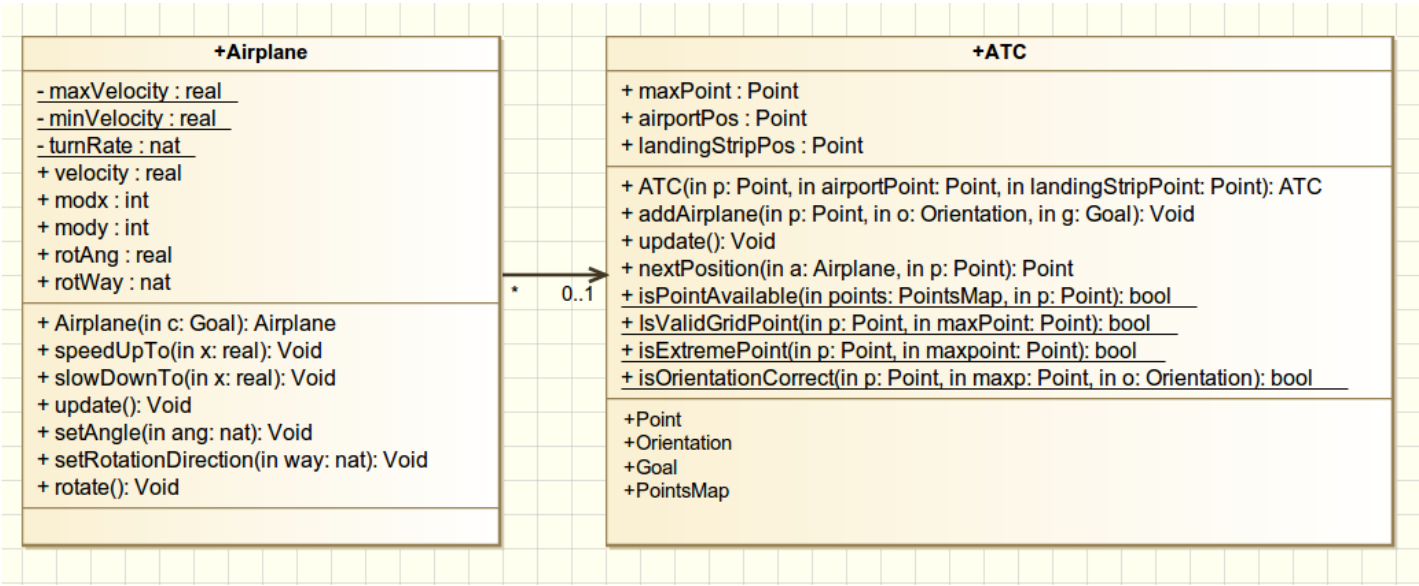
Cenário	**Ordenar diminuição de velocidade**
Descrição	Diminuir velocidade do avião para uma velocidade indicada.
Pré-Condições	1. A velocidade indicada deve ser inferior à velocidade actual. 2. A velocidade indicada deve ser igual ou superior à velocidade mínima.
Pós-Condições	1. A velocidade final do avião deve ser igual ou superior à velocidade mínima. 2. A velocidade final do avião deve ser igual ou inferior à velocidade máxima. 3. A velocidade final deve ser igual à velocidade indicada. 4. A velocidade final deve ser inferior à velocidade anterior.

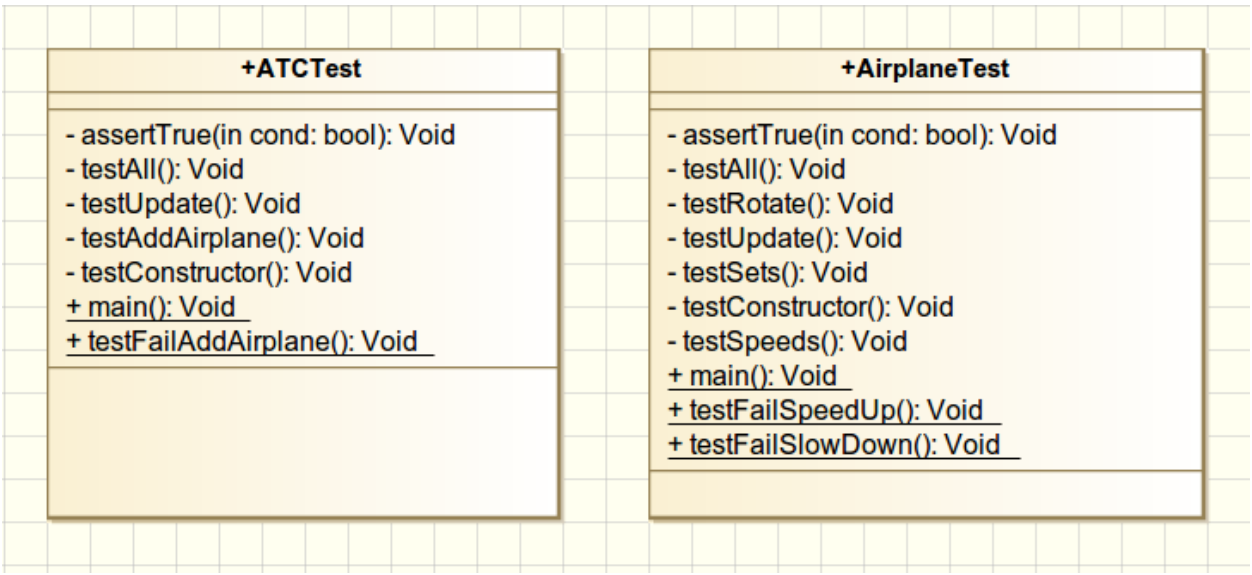
Cenário	**Update da Simulação**
Descrição	Simula a actualização das posições dos aviões após um *clock tick*.
Pré-Condições	1. Devem existir aviões no radar.
Pós-Condições	1. A nova posição do avião deverá ser igual à soma da anterior com o vector direcção multiplicado pela velocidade do avião. 2. Apenas pode existir um avião em cada posição (quadrícula) do mapa, para evitar colisões. 3. A orientação do avião não é alterada neste processo. 4. O valor da velocidade mantém-se após este processo.
Steps	1. Avaliar se todos os aviões seguem na direcção do seu objectivo e ordenar rotações caso necessário. 2. Prever colisões e ordenar rotações nesse caso. 3. Alterar posição do avião para a nova, com base na sua velocidade e direcção.

Cenário	**Ordenar Aterragem**
Descrição	Simula a aterragem de um avião na pista.
Pré-Condições	1. O avião deve entrar na pista de aterragem com orientação para o aeroporto. 2. A velocidade actual do avião deve ser igual à velocidade mínima.
Pós-Condições	1. O avião deve desaparecer do radar após aterrar.

2.2 Class Model

Class	**Descrição**
Airplane	Define avião, que será observado e manipulado no radar do ATC.
ATC	Define o controlador aéreo e respectivo radar, assim como funções para controlo dos aviões presentes no mesmo.
AirplaneTest	Define os casos de teste e de utilização para a classe Airplane.
ATCTest	Define os casos de teste e de utilização para a classe ATC.


+Airplane
- maxVelocity : real
- minVelocity : real
- turnRate : nat
+ velocity : real
+ modx : int
+ mody : int
+ rotAng : real
+ rotWay : nat
+ Airplane(in c: Goal): Airplane
+ speedUpTo(in x: real): Void
+ slowDownTo(in x: real): Void
+ update(): Void
+ setAngle(in ang: nat): Void
+ setRotationDirection(in way: nat): Void
+ rotate(): Void
*
0..1
+ATC
+ maxPoint : Point
+ airportPos : Point
+ landingStripPos : Point
+ ATC(in p: Point, in airportPoint: Point, in landingStripPoint: Point): ATC
+ addAirplane(in p: Point, in o: Orientation, in g: Goal): Void
+ update(): Void
+ nextPosition(in a: Airplane, in p: Point): Point
+ isPointAvailable(in points: PointsMap, in p: Point): bool
+ IsValidGridPoint(in p: Point, in maxPoint: Point): bool
+ isExtremePoint(in p: Point, in maxpoint: Point): bool
+ isOrientationCorrect(in p: Point, in maxp: Point, in o: Orientation): bool
+Point
+Orientation
+Goal
+PointsMap


+ATCTest
- assertTrue(in cond: bool): Void
- testAll(): Void
- testUpdate(): Void
- testAddAirplane(): Void
- testConstructor(): Void
+ main(): Void
+ testFailAddAirplane(): Void
+AirplaneTest
- assertTrue(in cond: bool): Void
- testAll(): Void
- testRotate(): Void
- testUpdate(): Void
- testSets(): Void
- testConstructor(): Void
- testSpeeds(): Void
+ main(): Void
+ testFailSpeedUp(): Void
+ testFailSlowDown(): Void

3. Formal VDM++ model

3.1 Class Airplane

```
class Airplane

    values
        private maxVelocity: real = 1;                      -- The plane's max
velocity
        private minVelocity: real = 0.1;              -- The plane's min velocity
        private turnRate: nat = 0;                                  -- Number of ticks
the plane takes to turn after a command is received, 0 means its instant

    instance variables
        public velocity: real := 1.0;                 -- The plane's actual velocity

        public goal: ATC`Goal;
 -- (1) Destination. N,W,E,S or L | L is for landing

        public modx: int := 0;                        -- Modifier for x coordinate
        public mody: int := 0;                        -- Modifier for y coordinate

        public rotAng: real := 0;        -- Current rotation angle ( if 0 no rotation )
        public rotWay: nat := 0;                                    -- Rotation
way. ( 1 = clockwise | 2 = counter-clockwise | 0 = no rotation )

    operations
       -- contrutor. recebe um objectivo c e cria um aviao cujo objectivo e realizar
c(1)
        public Airplane: ATC`Goal ==> Airplane
              Airplane(c) == (
                     goal := c;
                     if( c = <N> ) then (
                            modx := 0;
                            mody := 1;
                     ) else if( c = <S> ) then (
                            modx := 0;
                            mody := -1;
                     ) else if( c = <E> ) then (
                            modx := 1;
                            mody := 0;
                     ) else if( c = <W> ) then (
                            modx := -1;
                            mody := 0;
                     );
```

```
                return self
        )
        pre c in set elems[<N>, <W>, <E>, <S>, <L>]
        post modx >= -1 and
                        modx <= 1 and
                        mody >= -1 and
                        mody <= 1;

-- faz um aviao acelerar ate uma velocidade final de x
public speedUpTo: real ==> ()
        speedUpTo(x) ==
                velocity := x
        pre x > velocity and
                        x <= maxVelocity
        post     velocity >= minVelocity and
                                velocity <= maxVelocity and
                                velocity = x and
                                velocity > velocity~;

-- faz um aviao acelerar ate uma velocidade final de x
public slowDownTo: real ==> ()
        slowDownTo(x) ==
                velocity := x
        pre x < velocity and
                        x >= minVelocity
        post     velocity >= minVelocity and
                                velocity <= maxVelocity and
                                velocity = x and
                                velocity < velocity~;

-- actualiza a posicao actual do aviao tendo em conta o sentido do movimento
public update: () ==> ()
        update() == (
                if( rotAng > 0 and rotWay > 0) then (
                        rotate()
                )
        );

-- Sets the airplane rotation angle
public setAngle: nat ==> ()
        setAngle(ang) == (
                rotAng := ang
        )
pre ang >= 0 and ang rem 45 = 0
post rotAng >= 0 and rotAng rem 45 = 0;

-- Sets the airplane rotation direction
public setRotationDirection: nat ==> ()
```

```
	setRotationDirection(way) == (
		rotWay := way
	)
pre way >= 0 and way < 3
post rotWay >= 0 and rotWay < 3;

-- faz o aviao rodar
-- arg1: angulo
-- arg2: sentido ( 1 = clockwise | 2 = counter-clockwise )
public rotate: () ==> ()
	rotate() == (
		rotAng := rotAng - 45;
		if( rotWay = 1 ) then (    -- clockwise
			if( modx = -1 and mody = 0 ) then
				mody := 1
			else if( modx = -1 and mody = 1 ) then
				modx := 0
			else if( modx = 0 and mody = 1 ) then
				modx := 1
			else if( modx = 1 and mody = 1 ) then
				mody := 0
			else if( modx = 1 and mody = 0 ) then
				mody := -1
			else if( modx = 1 and mody = -1 ) then
				modx := 0
			else if( modx = 0 and mody = -1 ) then
				modx := -1
			else if( modx = -1 and mody = -1 ) then
				mody := 0
		) else if( rotWay = 2 ) then (    -- counter-clockwise
			if( modx = -1 and mody = 0 ) then
				mody := -1
			else if( modx = -1 and mody = 1 ) then
				mody := 0
			else if( modx = 0 and mody = 1 ) then
				modx := -1
			else if( modx = 1 and mody = 1 ) then
				modx := 0
			else if( modx = 1 and mody = 0 ) then
				mody := 1
			else if( modx = 1 and mody = -1 ) then
				mody := 0
			else if( modx = 0 and mody = -1 ) then
				modx := 1
			else if( modx = -1 and mody = -1 ) then
				modx := 0
		)
	)
```

```
    pre rotAng > 0 and rotAng rem 45 = 0 and
               rotWay > 0 and
               rotWay < 3
    post rotWay > 0 and
                rotWay < 3 and
                rotAng >= 0 and rotAng rem 45 = 0;

end Airplane
```

3.2 Class ATC

```
class ATC

types
    -- representa um ponto na "grelha"
    public Point ::
                    X: real
                    Y: real;

    -- representa a orientacao actual de por exemplo um aviao
    public Orientation = <N> | <S> | <W> | <E>;
    -- representa o objectivo de um aviao | <L> -> Land
    public Goal = <N> | <S> | <W> | <E> | <L>;

    public PointsMap = map Point to Airplane;
-- a uma posicao apenas pode estar associado uma aviao

instance variables
    public points: PointsMap := {|->};
                                                        -- grelha de pontos
    public maxPoint : Point := mk_Point(10,10);
                    -- ponto maximo da grelha ( dimensoes maximas )
        inv forall p in set dom points & IsValidATCPoint(p, maxPoint);
-- invariante que garante que cada ponto presente na grelha e um ponto
-- que pertence ao conjunto de todos os pontos entre (0,0) e (maxPoint.X, maxPoint.Y)

    public airportPos : Point := mk_Point(0,0);
                                -- posicao do aeroporto no ATC
    public landingStripPos : Point := mk_Point(0,0);
                    -- posicao da pista de aterragem do aeroporto

operations
```

```
    -- contrutor, cria um ATC
    public ATC : Point * Point * Point ==> ATC
            ATC(p, airportPoint,landingStripPoint) ==
            (
                    -- set das variaveis do ATC
            airportPos := airportPoint;
                    landingStripPos := landingStripPoint;
        maxPoint := p;

        return self;
         )
        pre IsValidATCPoint(airportPoint, maxPoint) and             -- verifica se
posicao do aeroporto e valida
                    IsValidATCPoint(landingStripPoint, maxPoint)                --
verifica se posicao da pista e valida
        post maxPoint = p;

        -- adiciona um aviao a grelha do ATC isto representa o evento de um aviao
entrar no espaco
        -- aereo controlado por este mesmo ATC
        public addAirplane : Point * Orientation * Goal ==> ()
            addAirplane(p, o, g) == (
                    points := { p |-> new Airplane(g)};
            )
        pre IsValidATCPoint(p, maxPoint) and
        -- aviao adicionado numa posicao valida do ATC
                    isExtremePoint(p, maxPoint) and
                            -- posicao e um extremo do ATC
                    isPointAvailable(points, p) and
                            -- nao existe nenhum aviao/aeroporto/pista na posicao
                    isOrientationCorrect(p, maxPoint, o)
        -- aviao entra no mapa na orientaao correcta
        post points(p).goal = g;


        -- metodo principal do ATC
        -- verifica se vai haver colisoes e desvia um dos avioes
        -- garante que cada aviao continua a ir em direcao ao seu objectivo
        public update : () ==> ()
            update() == (
                    for all point in set dom points do
                    (
                            -- evaluateGoal e rodar se necessário

                                    dcl airplane : Airplane := points(point);
                            if (nextPosition(airplane, point) in set dom points) then
            -- prever proxima posicao e verificar se esta desocupada
```

```
                                (airplane.setAngle(45);

                                        -- se estiver, roda 45 graus para desviar
                                airplane.setRotationDirection(1)
                                );
                        -- update
                        airplane.update();

                                                -- update ao aviao
                        points := {point} <-: points;
                        points := points ++ { nextPosition(airplane, point) |->
airplane };             -- altera a posicao do aviao
                )
        )
        pre points <> {|->};            -- mapeamento nao deve estar vazio
        -- numero de avioes deve permanecer igual

    -- retorna posicao do aviao no fim do proximo tick
    public nextPosition : Airplane * Point ==> Point
        nextPosition(a, p) == (
                dcl point : Point := mk_Point(p.X + a.modx, p.Y + a.mody);
                return point;
        )
        pre IsValidATCPoint(p, maxPoint);

functions

        -- verifica se um dado ponto esta livre (ainda nao existe no mapa)
    public isPointAvailable : PointsMap * Point -> bool
        isPointAvailable(points, p) == (
        if p in set dom points then
                false
        else
                true
    );

    -- confirma se um dado ponto pertence ao alcance do ATC
    public IsValidATCPoint : Point * Point -> bool
        IsValidATCPoint(p, maxPoint) ==
        maxPoint.X >= p.X and p.X >= 0 and maxPoint.Y >= p.Y and p.Y >=0;

  -- confirma se um ponto e um ponto de limite de alcance do ATC
  -- estes pontos representam onde os avioes podem "entrar"
  public isExtremePoint : Point * Point -> bool
    isExtremePoint(p, maxpoint) == (
        p.X = 0 or p.X = maxpoint.X or p.Y = 0 or p.Y = maxpoint.Y
    );
```

```
   -- dado um aviao que entrou no radar do ATC verifica se entrou com a orientacao
correcta
   public isOrientationCorrect : Point * Point * Orientation -> bool
        isOrientationCorrect(p, maxp, o) == (
               -- x = 0
               ( p.X = 0 and p.Y = 0 and ( o = <N> or o = <E> ) ) or
               ( p.X = 0 and p.Y = maxp.Y and ( o = <S> or o = <E> ) ) or
               ( p.X = 0 and p.Y > 0 and p.Y < maxp.Y and o = <E> ) or
               -- x = max
               ( p.X = maxp.X and p.Y = 0 and ( o = <N> or o = <W> ) ) or
               ( p.X = maxp.X and p.Y = maxp.Y and ( o = <S> or o = <W> ) ) or
               ( p.X = maxp.X and p.Y > 0 and p.Y < maxp.Y and o = <W> ) or
               -- y = 0
               ( p.Y = 0 and p.X > 0 and p.X < maxp.X and o = <N> ) or
               -- y = max
               ( p.Y = maxp.Y and p.X > 0 and p.X < maxp.X and o = <S> )
        );

end ATC
```

4. Model validation

4.1 Class AirplaneTest

```
class AirplaneTest

operations
    -- verifica se uma determinada condicao e valida
    -- pre condicao: a condicao tem de ser verdadeira
    private assertTrue: bool ==> ()
        assertTrue(cond) == return
        pre cond;

    -- Testa todos os casos de teste presentes excepto os que estao desenhado
especificamente para falhar
    private testAll: () ==> ()
        testAll() == (
              testConstructor();              -- Testes ao construtor
              testSpeeds();                   -- Testes de velocidade
              testSets();                     -- Testes aos sets do avião
              testRotate();                   -- Testes de rotação do avião
              testUpdate();                   -- Testes ao update do avião
        );

    -- testa a rotacao de um aviao
    private testRotate: () ==> ()
        testRotate() == (
              dcl a : Airplane := new Airplane(<N>);         -- aviao base --

              -- Roda uma vez no sentido dos ponteiros do relogio
              a.setAngle( 45 );
              a.setRotationDirection( 1 );

              a.rotate();

              assertTrue( a.rotAng = 0 );
              assertTrue( a.modx = 1 and a.mody = 1 );

              -- Roda duas vezes ( 90 graus ) contra os ponteiros do relogio
              a.setAngle( 90 );
              a.setRotationDirection( 2 );
```

```
            a.rotate();

            assertTrue( a.rotAng = 45 );
            assertTrue( a.rotWay = 2 );
            assertTrue( a.modx = 0 and a.mody = 1 );

            a.rotate();

            assertTrue( a.rotAng = 0 );
            assertTrue( a.rotWay = 2 );
            assertTrue( a.modx = -1 and a.mody = 1 );
    );

-- testa o metodo que actualiza a informacao do aviao a cada tick
private testUpdate: () ==> ()
    testUpdate() == (
            dcl a : Airplane := new Airplane(<N>);         -- aviao base --

            a.update();

            assertTrue( a.rotAng = 0 );
            assertTrue( a.rotWay = 0 );
            assertTrue( a.modx = 0 and a.mody = 1 );

            a.setAngle( 45 );
            a.setRotationDirection( 1 );

            a.update();

            assertTrue( a.rotAng = 0 );
            assertTrue( a.modx = 1 and a.mody = 1 );
    );

-- testa as funcoes de set a variaveis
private testSets: () ==> ()
    testSets() == (
            dcl a : Airplane := new Airplane(<N>);         -- aviao base --

            -- Angle
            a.setAngle( 45 );
            assertTrue( a.rotAng = 45 );
            a.setAngle( 90 );
            assertTrue( a.rotAng = 90 );
            a.setAngle( 0 );
            assertTrue( a.rotAng = 0 );

            -- Rotation Direction
            a.setRotationDirection( 1 );
```

```
            assertTrue( a.rotWay = 1 );
            a.setRotationDirection( 2 );
            assertTrue( a.rotWay = 2 );
            a.setRotationDirection( 0 );
            assertTrue( a.rotWay = 0 );
    );

-- testa o constructor da classe aviao
private testConstructor: () ==> ()
    testConstructor() == (
            dcl a1 : Airplane := new Airplane(<E>);
            dcl a2 : Airplane := new Airplane(<S>);
            dcl a3 : Airplane := new Airplane(<W>);
            dcl a4 : Airplane := new Airplane(<N>);

            assertTrue( a1.modx = 1 and a1.mody = 0 );
            assertTrue( a2.modx = 0 and a2.mody = -1 );
            assertTrue( a3.modx = -1 and a3.mody = 0 );
            assertTrue( a4.modx = 0 and a4.mody = 1 );
    );

-- testa as funcoes de manuseamento de velocidades
private testSpeeds: () ==> ()
    testSpeeds() == (
            dcl a : Airplane := new Airplane(<N>);         -- aviao base --

            a.slowDownTo( 0.5 );
            assertTrue( a.velocity = 0.5 );
            a.speedUpTo( 0.9 );
            assertTrue( a.velocity = 0.9 );
            a.speedUpTo( 1.0 );
            assertTrue( a.velocity = 1.0 );
    );

-- corre todos os casos de teste
public static main: () ==> ()
    main() == (
            new AirplaneTest().testAll();
    );

-- caso de teste desenhado para falhar na chamada da funcao speedUpTo do aviao
-- falha ao introduzir uma velocidade superior a velocidade maxima dos avioes
-- violando assim uma pre condicao
public static testFailSpeedUp: () ==> ()
    testFailSpeedUp() == (
            dcl a : Airplane := new Airplane(<N>);         -- aviao base --
            a.speedUpTo( 3 );
```

```
        );

    -- caso de teste desenhado para falhar na chamada da funcao slowDownTo do aviao
    -- falha ao introduzir uma velocidade inferior a velocidade minima dos avioes
    -- violando assim uma pre condicao
    public static testFailSlowDown: () ==> ()
    testFailSlowDown() == (
        dcl a : Airplane := new Airplane(<N>);         -- aviao base --
        a.slowDownTo( 0 );
    );

end AirplaneTest
```

4.2 Class ATCTest

```
class ATCTest

operations
    -- verifica se uma determinada condicao e valida
    -- pre condicao: a condicao tem de ser verdadeira
    private assertTrue: bool ==> ()
        assertTrue(cond) == return
        pre cond;

    -- Testa todos os casos de teste presentes excepto os que estao desenhado
especificamente para falhar
    private testAll: () ==> ()
        testAll() == (
              testConstructor();
              testAddAirplane();
              testUpdate();
        );

    -- testa o metodo que actualiza o estado do ATC
    private testUpdate: () ==> ()
        testUpdate() == (
              -- declaring a new ATC
              dcl maxP : ATC`Point := mk_ATC`Point( 10, 10 );
              dcl airportP : ATC`Point := mk_ATC`Point( 5, 5 );
              dcl landingStripP : ATC`Point := mk_ATC`Point( 6, 5 );
              dcl g : ATC := new ATC( maxP, airportP, landingStripP );
```

```
            -- declaring an airplane
            dcl p : ATC`Point := mk_ATC`Point( 5, 0 );
            dcl o : ATC`Orientation := <N>;
            dcl goal : ATC`Goal := <N>;

            -- adding the airplane to the ATC
            g.addAirplane(p, o , goal);

            -- updating the ATC so that the airplane moves
            g.update();

            assertTrue( mk_ATC`Point(5, 1) in set dom g.points );
    );

-- testa a funcionalidade de adicionar um aviao ao espaco aereo do ATC
private testAddAirplane: () ==> ()
    testAddAirplane() == (
            -- declaring a new ATC
            dcl maxP : ATC`Point := mk_ATC`Point( 10, 10 );
            dcl airportP : ATC`Point := mk_ATC`Point( 5, 5 );
            dcl landingStripP : ATC`Point := mk_ATC`Point( 6, 5 );
            dcl g : ATC := new ATC( maxP, airportP, landingStripP );

            -- declaring an airplane
            dcl p : ATC`Point := mk_ATC`Point( 5, 0 );
            dcl o : ATC`Orientation := <N>;
            dcl goal : ATC`Goal := <N>;

            -- adding the airplane to the ATC
            g.addAirplane(p, o , goal);
    );

-- testa o construtor do ATC
private testConstructor: () ==> ()
    testConstructor() == (
            -- declaring a new ATC
            dcl maxP : ATC`Point := mk_ATC`Point( 10, 10 );
            dcl airportP : ATC`Point := mk_ATC`Point( 5, 5 );
            dcl landingStripP : ATC`Point := mk_ATC`Point( 6, 5 );
            dcl g : ATC := new ATC( maxP, airportP, landingStripP );

            assertTrue( g.maxPoint = maxP );
            assertTrue( g.airportPos = airportP );
            assertTrue( g.landingStripPos = landingStripP );
    );

-- executa todos os testes
public static main: () ==> ()
```

```
        main() == (
                new ATCTest().testAll();
        );

    -- adiciona um aviao propositadamente no meio do ATC de modo a mostrar a efeciencia
    -- das pre condicoes da funcao addAirplane
    public static testFailAddAirplane: () ==> ()
        testFailAddAirplane() == (
                -- declaring a new ATC
                dcl maxP : ATC`Point := mk_ATC`Point( 10, 10 );
                dcl airportP : ATC`Point := mk_ATC`Point( 5, 5 );
                dcl landingStripP : ATC`Point := mk_ATC`Point( 6, 5 );
                dcl g : ATC := new ATC( maxP, airportP, landingStripP );

                -- declaring an airplane
                dcl p : ATC`Point := mk_ATC`Point( 5, 5 );
                dcl o : ATC`Orientation := <N>;
                dcl goal : ATC`Goal := <N>;

                -- adding the airplane to the ATC
                g.addAirplane(p, o , goal);
        );

end ATCTest
```

5. Model verification

5.1 Example of domain verification

Um exemplo de “proof obligation” gerada pelo Overture:

Nº	PO Name	Type
6	ATC`update	legal map application

Código em análise (parte relevante sublinhada):

```
dcl airplane : Airplane := points(point);
```

Neste caso, é verificado se o “*point”* passado consta do domínio da *map* “*points”*.

5.2 Example of invariant verification

Outro exemplo de “proof obligation” gerada pelo Overture:

Nº	PO Name	Type
3	ATC`addAirplane	state invariant holds

Código em análise (parte relevante sublinhada):

```
public addAirplane : Point * Orientation * Goal ==> ()
              addAirplane(p, o, g) == (
                     points := { p |-> new Airplane(g)};
              )
```

Invariante verificada:

```
inv forall p in set dom points & IsValidATCPoint(p, maxPoint);
```

Assim, é possível observar que, na adição de um elemento “*Point”* no *map* “*points”*, é verificado se os seus componentes não ultrapassam os valores de “*maxPoint”* (ponto máximo na *Grid*).

Conclusions

O modelo desenvolvido cobre a maior parte dos requisitos propostos. Dada a complexidade do problema e utilização de um paradigma e linguagem novos, detalhes como o aumento e diminuição de velocidade foram descartados pois resultariam em casos complicados de análise de colisões. Assim, apesar de especificado, no desenvolvimento assumiu-se velocidade constante. Tais aspectos poderiam ser melhorados com mais tempo e domínio da linguagem de programação utilizada.

References

1. Quick Overview of VDM Operators, http://kurser.iha.dk/eit/tivdm1/Quick_Overview_of_VDM_Operators.pdf
2. Overture tool, http://overturetool.org
3. Modelio - website e documentação, http://www.modelio.org/