

# Specification of Air Traffic Control\*

Kristian Kalsing<sup>†</sup>  
Software Verification Research Centre  
School of Information Technology  
The University of Queensland

October 3, 1999

## Abstract

This report presents a case study of air traffic control (ATC). A formal specification of an ATC system is developed. The ATC model represents the air traffic controllers and the actual ATC system as a whole.

A simulator is specified as an extension to the ATC specification. The simulator comprises operations that are external to the ATC system such as flight movements, etc.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The Air Traffic Control System</b>	<b>2</b>
2.1	Description . . . . .	2
2.2	Operational Concept . . . . .	3
<b>3</b>	<b>Formal Specification</b>	<b>5</b>
3.1	The Sum Language . . . . .	5
3.2	The ATC system . . . . .	5
3.2.1	State and Initialisation . . . . .	6
3.2.2	Operations . . . . .	10
3.3	The Simulator . . . . .	14
3.3.1	State and Initialisation . . . . .	14
3.3.2	Operations . . . . .	14
<b>4</b>	<b>Discussions</b>	<b>17</b>
4.1	Handover . . . . .	17
4.2	Topology . . . . .	18

---

\*This work represents parts of an honours project at the Software Verification Research Centre, The University of Queensland. Supervisor is Peter A. Lindsay.

<sup>†</sup>M.Sc. student, Technical University of Denmark, email: [kristian@kalsing.dk](mailto:kristian@kalsing.dk).

<b>5</b>	<b>Conclusions</b>	<b>18</b>
<b>A</b>	<b>Mathematical Notation</b>	<b>19</b>
A.1	Sets . . . . .	19
A.2	Numbers . . . . .	19
A.3	Binary Relations . . . . .	19
A.4	Functions . . . . .	20
A.5	Sequences . . . . .	20
<b>B</b>	<b>ATC Specification</b>	<b>21</b>
<b>C</b>	<b>Simulator Specification</b>	<b>26</b>

## 1 Introduction

In this report we present a formal model of air traffic control (ATC). The model is not an attempt to give a complete specification of an ATC system.

The model covers aspects that are relevant for our particular purpose: to build and simulate an ATC system which can be used for animation in order to perform safety analysis on an algorithm that calculates reasonable delays for all the aircraft in the system.

This report merely presents the Sum specifications of the ATC system and a purpose built simulator.

## 2 The Air Traffic Control System

In this section we first describe the ATC system in general. Then we describe the operational concept of our simplified model of the system.

### 2.1 Description

In general terms, an ATC system together with the air traffic controllers who run it are responsible for directing aircraft safely through the respective ATC region. The main objective is to make sure that all aircraft are separated in time and spatial location.

In particular, we consider arrivals to an airport. The ATC region consists of a sequence of airspaces beginning where the aircraft start to approach the airport and ending where the aircraft have landed and is located at a gate.

All airspaces in the sequence has exactly one controller assigned to it. A controller is responsible for all the aircraft in the airspace that he/she is assigned to.

Figure 1 shows a typical flow chart of the arrivals. An aircraft goes through several states as it approaches an airport. At all times a particular controller (enroute, approach, etc.) is responsible for directing the aircraft. We will refer to this flow as the *arrivals flow*.

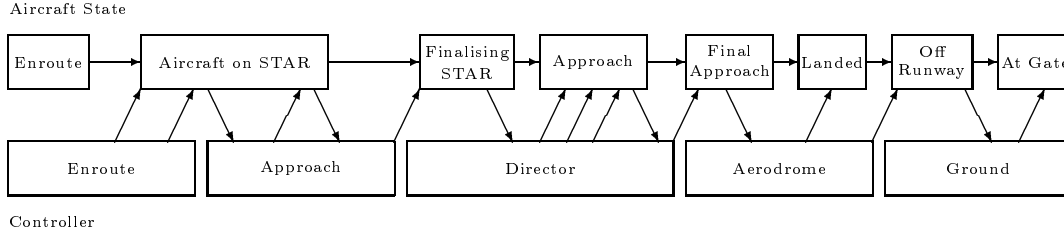


Figure 1: Arrivals Flow Chart

There is communication between the aircraft and the controller at certain points in the arrivals flow (in the flow chart communication is represented by an arrow between a controller and an aircraft state). The controllers call the aircraft to give them speed adjustments, tell the pilot to *delay* the flight or tell the pilot to contact the next controller in a *handover*. The pilots call the controllers mainly to announce their current position or to confirm instructions.

## 2.2 Operational Concept

In this section we will describe our interpretation of an ATC system. Our specifications will not cover all aspects of an ATC system. However, it will be comprehensive enough to model a simple system that can be used to simulate aircraft going through the arrivals flow.

We model the arrivals flow by having a sequence of controllers. We do not distinguish between controllers and their airspaces. When an aircraft is located in an airspace we simply say that the aircraft is at the respective controller.

Within each controller we have a fixed number of positions flights can occupy. The positions do not represent spatial locations but are positions in time. The number of a particular position represents the number of time units till next handover.

We do not monitor the spatial locations of aircraft. We assume that the spatial separation is taken of. We use the abstraction of modelling positions in time rather than spatial positions because it really is what we are interested in when building the simulator.

Strictly the model should take the spatial separation into account. However, spatial separation is not relevant for our particular purpose of the simulation, hence we have made the abstraction that we merely consider positions in time and assume that parts of the ATC system we have not specified handle the spatial separation.

Being that close to the airport (as we are in the arrivals flow) the aircraft will approach the airport following a nearly one-dimensional path. Hence separation in time normally implies spatial separation too. However, we may encounter situations where flights are overtaking each other. In those cases we assume that the required spatial separation is fully satisfied.

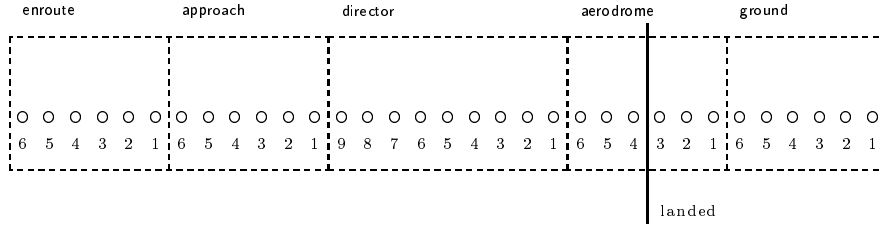


Figure 2: Arrivals Flow Chart as Modelled

The simple arrivals flow is shown in figure 2. The aircraft states (see figure 1) are not relevant to our model. At a certain position in the flow the aircraft will have landed. This position is usually located at the **aerodrome** controller.

In our model, under normal operation, an aircraft will arrive at the first controller (the **enroute** controller), move through all the positions, and finally be removed from the system at the last position at the last controller (the **ground** controller). At this point the aircraft will be located at a gate and will no longer be of interest.

We will refer to the basic setup in figure 2 as the *topology* of the system. The topology defines the sequence of controllers, the number of positions, and determines the landing position.

Our specification will consist of two models. As indicated in figure 3, one module specifies the ATC system and the other models the simulator. The model of the ATC system covers the controllers' behaviour and actions performed by the system itself. The simulator models the behaviour of aircraft.

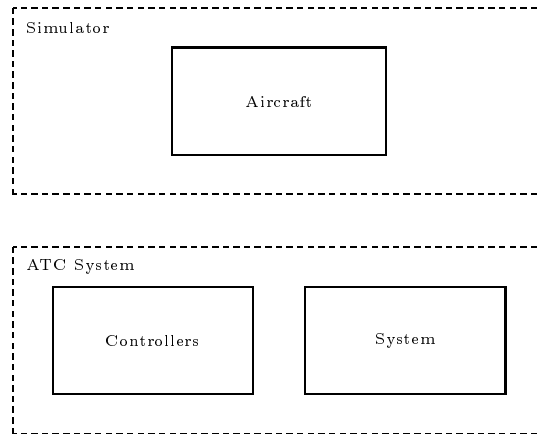


Figure 3: Specification Structure

### 3 Formal Specification

The models of the ATC system and the simulator are formally specified using the Sum specification language. The next section briefly describes Sum.

The following two sections give a formal presentation of the two models. Each model is presented gradually beginning with the state variables and the invariants. Essential concepts in the model are related to concepts in our domain as presented in the previous section.

#### 3.1 The Sum Language

Sum [3] is a variant of the well known Z specification language. While closely related to Z, Sum extends Z to facilitate the production of modular specifications and ease specification readability. Relevant to this case study, Sum particularly provides facilities at the specification level for:

- Modular and parameterised specifications. Modules may be imported, giving visibility to the referenced entities.
- Distinguished state machines represented by modules through the use of pre-defined *state*, *init* and *schema* schemas:
  - *state* schemas represent the state encapsulated by the module.
  - *init* schemas initialise the respective state.
  - *schema* schemas capture state transitions.
- Explicit preconditions in schemas by using the prefix *pre*.

In this particular case study the specification comprises two modules, the ATC system and the simulator. The simulator module will import the ATC module.

#### 3.2 The ATC system

Recall, that in our model the ATC system represents both the actual system *and* the controllers.

### 3.2.1 State and Initialisation

The state of the ATC system and its respective invariant are specified as follows:

*state*

---

$flow : \text{iseq } Controller$   
 $positions : Controller \rightarrow \mathbb{N}_1$   
 $delay\_pos : Controller \rightarrow \mathbb{P} \mathbb{N}_1$   
 $capacity : Controller \rightarrow \mathbb{N}$   
 $landing\_ctrl : Controller$   
 $landing\_pos : \mathbb{N}_1$   
 $control : Aircraft \rightarrow Controller$   
 $current\_pos : Aircraft \rightarrow \mathbb{N}_1$   
 $remaining\_t : Aircraft \rightarrow \mathbb{N}$   
 $delay : Aircraft \rightarrow \mathbb{N}$   
 $contact : Aircraft \leftrightarrow Controller$   
 $waiting : \mathbb{P} Aircraft$   
 $open : \mathbb{B}$   
 $t\_to\_landing : Aircraft \rightarrow \mathbb{Z}$   
 $max\_delay : Aircraft \rightarrow \mathbb{N}$

---

$\text{dom } control = \text{dom } current\_pos$   
 $\text{dom } control = \text{dom } remaining\_t$   
 $\text{dom } control = \text{dom } delay$   
 $\text{ran } control \subseteq \text{ran } flow$   
 $\text{dom } contact \subseteq \text{dom } control$   
 $\text{ran } contact \subseteq \text{ran } flow$   
 $waiting \subseteq \text{dom } control$   
 $\forall a : \text{dom } control \bullet current\_pos(a) \leq positions(control(a))$   
 $\forall c : \text{ran } flow \bullet$   
 $\quad c \in \text{dom } positions \wedge c \in \text{dom } delay\_pos \wedge c \in \text{dom } capacity$   
 $\forall c : \text{ran } flow \bullet \forall p : delay\_pos(c) \bullet p \leq positions(c)$   
 $flow \neq \langle \rangle \Rightarrow$   
 $\quad landing\_ctrl \in \text{ran } flow \wedge$   
 $\quad landing\_pos \leq positions(landing\_ctrl)$   
 $\forall a : \text{dom } control \bullet a \in waiting \Rightarrow current\_pos(a) = 1$   
 $open \Rightarrow flow \neq \langle \rangle$   
 $\forall a : \text{dom } control \bullet t\_to\_landing(a) =$   
 $\quad t\_to\_end(control(a), current\_pos(a), positions, flow) -$   
 $\quad t\_to\_end(landing\_ctrl, landing\_pos, positions, flow)$   
 $\forall a : \text{dom } control \bullet max\_delay(a) =$   
 $\quad remaining\_t(a) - t\_to\_landing(a)$

---

The variables *flow* and *positions* represent a sequence of controllers and the number of positions within each controller (recall, that a position is the time till next handover). Together with *landing\_ctrl* and *landing\_pos*, which specify the controller and the position within that particular controller the actual landing takes place, they define the topology of the arrivals flow.

The first controller in the sequence is the last one an aircraft encounters under normal operation. The positions within a controller are numbered from 1 to  $n$ , where  $n$  is the total number of positions. Position 1 is the last position a flight occupies within a controller when passing through that particular controllers airspace.

Those four variables could just as well have been specified as constants as that is what they basically would be under normal circumstances. When simulating scenarios from the real world we would never change those variables. We chose to put them among the state variables in order to allow a wider range of experiments with the system. We will have the opportunity to run the same scenarios on different topologies. For example, we can systematically analyse how a different number of controllers would handle a particular scenario.

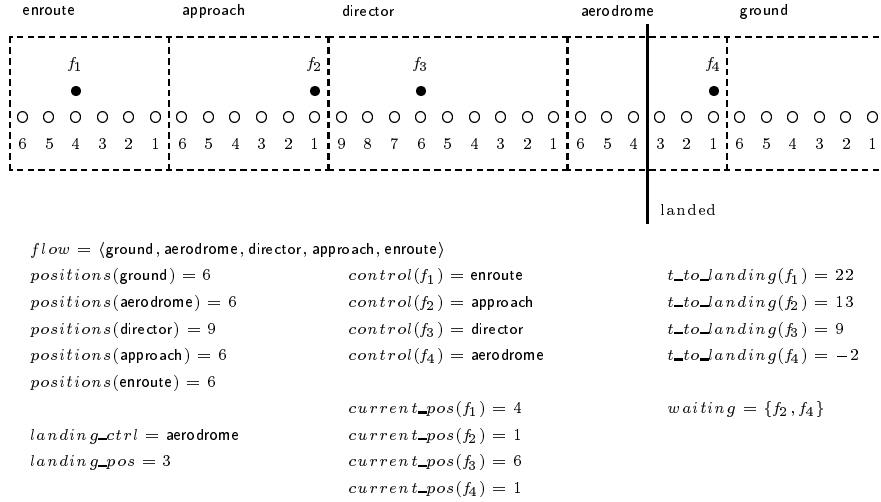


Figure 4: Instantiation of State Variables

The variable *delay\_pos* specifies for each controller, a set of positions where it is possible to delay a flight. These positions represent some sort of holding patterns.

The capacity of a controller is specified by *capacity*. There is a limit on how many flights a controller can handle. We allow *capacity* and *delay\_pos* to be changed during normal operation due to weather conditions, etc.

The current position of an aircraft is represented by *control* and *current\_pos*. The first variable gives us the controller and the second the position within that controller.

The relation *contact* represents whether communication has been established between a controller and an aircraft. Notice, that *control* gives the controller to whom the aircraft is visible, which does not necessarily mean that the controller can communicate with the respective aircraft.

The variable *remaining\_t* is the remaining time an aircraft can stay in the air due to fuel limitations, etc.

The variable *delay* specifies for each aircraft the current instructions to delay. Zero means that none is given, any other number denotes how many time units a flight should delay as soon as it reaches a position where delay is possible.

The set *waiting* is the set of aircrafts waiting to be handed over. They have reached the last position (1) within a controller's airspace and await the controller's handover.

The boolean variable *open* tells us whether the airport is open or closed. When an airport is closed no aircraft can be added to the system. But the system will still handle the aircraft that were already in the system when the airport was closed.

The variables *t\_to\_landing* and *max\_delay* are secondary variables (their values are derived from one or more primary state variables). Respectively, for each aircraft in the system they give the number of time units from the current position to the landing and the maximum possible delay the aircraft can be given.

Figure 4 shows an example of how some of the state variables are used. There are four flights in the system.

We introduce a number of auxiliary functions in order to ease the specification:

$$\begin{array}{|l}
\text{next\_ctrl :} \\
\hline
\text{Controller} \times \text{iseq Controller} \rightarrow \text{Controller} \\
\hline
\forall c1, c2 : \text{Controller}; \text{flw} : \text{iseq Controller} \bullet \\
\text{next\_ctrl}(c1, \text{flw}) = c2 \Leftrightarrow \\
\exists i : \mathbb{N} \bullet c1 = \text{flw}(i) \wedge c2 = \text{flw}(i - 1) \\
\\
\text{entrance\_ctrl :} \\
\hline
\text{iseq Controller} \rightarrow \text{Controller} \\
\hline
\forall \text{flw} : \text{iseq Controller} \bullet \\
\text{entrance\_ctrl}(\text{flw}) = \text{last flw} \\
\\
\text{exit\_ctrl :} \\
\hline
\text{iseq Controller} \rightarrow \text{Controller} \\
\hline
\forall \text{flw} : \text{iseq Controller} \bullet \\
\text{exit\_ctrl}(\text{flw}) = \text{head flw} \\
\\
\text{number\_in\_seq :} \\
\hline
\text{Controller} \times \text{iseq Controller} \rightarrow \mathbb{N} \\
\hline
\forall c : \text{Controller}; \text{flw} : \text{iseq Controller}; n : \mathbb{N} \bullet \\
\text{number\_in\_seq}(c, \text{flw}) = n \Leftrightarrow \text{flw}(n) = c
\end{array}$$



$sum\_of\_pos :$ $\mathbb{N} \times (Controller \rightarrow \mathbb{N}) \times iseq\ Controller \rightarrow \mathbb{N}$
$\forall i : \mathbb{N};\ psn : Controller \rightarrow \mathbb{N};\ flw : iseq\ Controller \bullet$ $sum\_of\_pos(i, psn, flw) =$ if $i = 0$ then $0$ else $psn(flw(i)) + sum\_of\_pos(i - 1, psn, flw)$
$t\_to\_end :$ $Controller \times \mathbb{N} \times (Controller \rightarrow \mathbb{N}) \times iseq\ Controller \rightarrow \mathbb{N}$
$\forall c : Controller;\ p : \mathbb{N};\ psn : Controller \rightarrow \mathbb{N};\ flw : iseq\ Controller \bullet$ $t\_to\_end(c, p, psn, flw) =$ $p + sum\_of\_pos(number\_in\_seq(c, flw) - 1, psn, flw)$

With *next\_ctrl*, *entrance\_ctrl* and *exit\_ctrl* we can get the next controller in the flow, the first controller, and the last controller, respectively.

*number\_in\_seq* tells us what number in the flow a controller is. From an arbitrary position in the flow we can get the distance (in time) to the end of the flow with *t\_to\_end*.

All the five auxiliary functions are partial. This implies that there are a lot of cases where we cannot predict what the functions will return when applied to certain input. However, our model will only pass input to these functions that gives sensible output (e.g. we will never ask for the next controller given the last controller, we will never ask for the entrance controller of an empty sequence, etc.). In order words, the way the functions are used in the specification ensures deterministic behaviour. Practically the preconditions of the operations in the model ensure that the auxiliary functions will always return deterministic results.

The state invariant puts restrictions on the system state. The essential relationships between the objects in the system state are:

- each aircraft that is visible to a controller has a current position within that controller, its remaining time in the air is known, and it has a current instruction to delay (possibly zero);
- aircraft can only be visible to controllers within the arrivals flow;
- controllers can only have communication with aircraft within the system;
- only aircraft located within the system can be waiting to be handed over;
- aircraft can only occupy positions that actually exist;
- for all controllers in the arrivals flow we know their number of positions, their set of positions where delay is possible, and their capacity;
- the set of delay positions for a controller has to be actual positions within that controller;

- an arrivals flow has to have a known landing controller and a landing position;
- an aircraft awaiting handover (or removal) is in position 1 of some controller;
- an open airport requires at least one controller;
- the secondary variable  $t\_to\_landing$  is specified as the difference between the aircraft's distance from last position in the flow and the landing position's distance from the last position;
- the secondary variable  $max\_delay$  is specified as the difference between the aircraft's remaining time in the air and its total time to landing.

Notice, that we did not demand the number of aircraft within a controller to stay within the capacity limit. This specification should be interpreted as a domain model. In such a specification we model all possible behaviours. In the real world the capacity can be exceeded, naturally it should be in the domain model (specifying that the capacity limit can not be exceeded is a (safety) requirement to the system and is not a part of the domain model).

Initially no topology is set up, there are no aircraft in the system, no current instructions to delay and the airport is closed.

$init$ $flow' = \langle \rangle$ $positions' = \emptyset$ $delay\_pos' = \emptyset$ $capacity' = \emptyset$ $control' = \emptyset$ $current\_pos' = \emptyset$ $remaining\_t' = \emptyset$ $delay' = \emptyset$ $contact' = \emptyset$ $waiting' = \emptyset$ $\neg open'$
---

### 3.2.2 Operations

The topology of the arrivals flow is reset with the two operations *ResetFlow* and *ResetPositions*.

$op \text{ Reset\_Flow}$
$cs? : \text{iseq Controller}$ $c? : \text{Controller}$ $p? : \mathbb{N}$
$\text{pre}(\text{control} = \emptyset \wedge c? \in \text{ran } cs? \wedge p? \leq \text{positions}(c?))$ $\text{changes\_only } \{\text{flow}, \text{landing\_ctrl}, \text{landing\_pos}\}$ $\text{flow}' = cs?$ $\text{landing\_ctrl}' = c?$ $\text{landing\_pos}' = p?$

$op \text{ Reset\_Positions}$
$c? : \text{Controller}$ $n? : \mathbb{N}$
$\text{pre}(c? \notin \text{ran control})$ $\text{changes\_only } \{\text{positions}\}$ $\text{positions}' = \text{positions} \oplus \{c? \mapsto n?\}$

With *Reset\_Flow* we can define the sequence of controllers and the compulsory landing controller and position. With *Reset\_Positions* we define the number of positions within a controller. As stated in the invariant, the landing controller and position have to exist for any non-empty sequence of controllers.

The arrivals flow cannot be reset whilst aircraft are located in the system. Similarly the number of positions can not be reset for a controller if there are one or more aircraft located within that particular controller's airspace. Under normal operation we would not wish to reset these variables, anyway.

With *Reset\_Capacity* we can reset the current capacity of a controller. There are no restrictions on when that can be done.

$op \text{ Reset\_Capacity}$
$c? : \text{Controller}$ $n? : \mathbb{N}$
$\text{changes\_only } \{\text{capacity}\}$ $\text{capacity}' = \text{capacity} \oplus \{c? \mapsto n?\}$

The operation *Reset\_Delay\_Pos* is used to reset the set of positions within a controller that allow delay. *Add\_Delay\_Pos* and *Delete\_Delay\_Pos* are used to alter the set.

$\text{op Reset\_Delay\_Pos}$ <hr/> $c? : \text{Controller}$ $ps? : \mathbb{PN}$ <hr/> $\text{pre}(\forall p : ps? \bullet p \leq \text{positions}(c?))$ $\text{changes\_only } \{ \text{delay\_pos} \}$ $\text{delay\_pos}' = \text{delay\_pos} \oplus \{ c? \mapsto ps? \}$
---

$\text{op Add\_Delay\_Pos}$ <hr/> $c? : \text{Controller}$ $p? : \mathbb{N}_1$ <hr/> $\text{pre}(p? \notin \text{delay\_pos}(c?) \wedge p? \leq \text{positions}(c?))$ $\text{changes\_only } \{ \text{delay\_pos} \}$ $\text{delay\_pos}' = \text{delay\_pos} \oplus \{ c? \mapsto (\text{delay\_pos}(c?) \cup \{ p? \}) \}$
---

$\text{op Delete\_Delay\_Pos}$ <hr/> $c? : \text{Controller}$ $p? : \mathbb{N}_1$ <hr/> $\text{pre}(p? \in \text{delay\_pos}(c?))$ $\text{changes\_only } \{ \text{delay\_pos} \}$ $\text{delay\_pos}' = \text{delay\_pos} \oplus \{ c? \mapsto (\text{delay\_pos}(c?) \setminus \{ p? \}) \}$
--

The operation *Handover\_Flight* hands over an aircraft from its current controller to the next controller in the sequence. The aircraft has to be waiting for the handover (be in position 1 and not currently on hold).

An aircraft that reaches position 1 within a controller is seen as an external event, thus it will be a task of the simulator to extend the set *waiting* with the aircraft.

$\text{op Handover\_Flight}$ <hr/> $a? : \text{Aircraft}$ <hr/> $\text{pre}(a? \in \text{waiting} \wedge \text{control}(a?) \neq \text{exit\_ctrl}(\text{flow}))$ $\text{changes\_only } \{ \text{control}, \text{current\_pos}, \text{waiting} \}$ $\text{control}' = \text{control} \oplus$ $\quad \{ a? \mapsto \text{next\_ctrl}(\text{control}(a?), \text{flow}) \}$ $\text{current\_pos}' = \text{current\_pos} \oplus$ $\quad \{ a? \mapsto \text{positions}(\text{next\_ctrl}(\text{control}(a?), \text{flow})) \}$ $\text{waiting}' = \text{waiting} \setminus \{ a? \}$
---

It is not possible for an aircraft to be handed over from the last controller (ground) in the arrivals flow. In that case we use the operation *Remove\_Flight* to remove the

aircraft from the system, meaning that the aircraft has been located at a gate and is no longer of interest in the arrivals flow. The latter operation can be seen as special case of the previous one.

$\text{op } \textit{Remove\_Flight}$	_____
$a? : \textit{Aircraft}$	
$\text{pre}(a? \in \textit{waiting} \wedge \textit{control}(a?) = \textit{exit\_ctrl}(\textit{flow}))$	
$\text{changes\_only } \{\textit{control}, \textit{current\_pos}, \textit{remaining\_t}, \textit{delay}, \textit{contact}, \textit{waiting}\}$	
$\textit{control}' = (\{a?\} \triangleleft \textit{control})$	
$\textit{current\_pos}' = (\{a?\} \triangleleft \textit{current\_pos})$	
$\textit{remaining\_t}' = (\{a?\} \triangleleft \textit{remaining\_t})$	
$\textit{delay}' = (\{a?\} \triangleleft \textit{delay})$	
$\textit{contact}' = (\{a?\} \triangleleft \textit{contact})$	
$\textit{waiting}' = \textit{waiting} \setminus \{a?\}$	

When a controller instructs a flight to delay we use the operation *Delay\_Flight*. As input we give the aircraft and the number of time units the aircraft is to be delayed.

$\text{op } \textit{Delay\_Flight}$	_____
$a? : \textit{Aircraft}$	
$t? : \mathbb{N}$	
$\text{pre}(a? \in \text{dom } \textit{control} \wedge a? \text{ contact } \textit{control}(a?))$	
$\text{changes\_only } \{\textit{delay}, \textit{waiting}\}$	
$\textit{delay}' = \textit{delay} \oplus \{a? \mapsto t?\}$	
$\textit{waiting}' = \textit{waiting} \setminus \{a?\}$	

Notice, that this operation represents an *instruction to delay*, that does not mean that the aircraft actually will delay. The aircraft might not currently be in a position that allows delay. However, the aircraft will still receive the instruction and keep it until a new one is given.

If the flight previously was waiting to be handed over it will be removed from the set *waiting* (if its current position does not allow delay the simulator will put it back in *waiting*).

Instructions to delay can only be given if the aircraft is visible to a controller and communication is established between the aircraft and its current controller.

Finally we introduce two operations *Open\_Entrance* and *Close\_Entrance* to open and close the airport respectively.

$\text{op } \textit{Open\_Entrance}$	_____
$\text{pre}(\neg \textit{open})$	
$\text{changes\_only } \{\textit{open}\}$	
$\textit{open}'$	

$\begin{array}{l} \text{op } \textit{Close\_Entrance} \\ \text{pre}(\textit{open}) \\ \text{changes\_only } \{\textit{open}\} \\ \neg \textit{open}' \end{array}$
---

A closed airport cannot receive any more aircraft to its arrivals flow. However, aircraft that are already in the system when the airport is closed will still be handled as usual.

### 3.3 The Simulator

The simulator module imports the ATC module, meaning that the simulator in terms of the specification is an extension to the ATC system. We model the behaviour of aircraft including communication with these.

#### 3.3.1 State and Initialisation

No additional variables are introduced in the simulator.

$\begin{array}{l} \text{state} \\ \textit{ATC.state} \end{array}$
---

$\begin{array}{l} \text{init} \\ \textit{ATC.init} \end{array}$
---

#### 3.3.2 Operations

The operation *Add\_Flight* adds a flight to the system. This simulates the normal case where an aircraft arrives to the airport and enters the arrivals flow. Flights can be added only when the airport is open.

$\begin{array}{l} \text{op } \textit{Add\_Flight} \\ a? : \textit{Aircraft} \\ rt? : \mathbb{N} \\ \\ \text{pre}(\textit{open} \wedge a? \notin \text{dom } \textit{control}) \\ \text{changes\_only } \{\textit{control}, \textit{current\_pos}, \textit{remaining\_t}, \textit{delay}\} \\ \textit{control}' = \textit{control} \oplus \{a? \mapsto \textit{entrance\_ctrl}(\textit{flow})\} \\ \textit{current\_pos}' = \textit{current\_pos} \oplus \{a? \mapsto \textit{positions}(\textit{entrance\_ctrl}(\textit{flow}))\} \\ \textit{remaining\_t}' = \textit{remaining\_t} \oplus \{a? \mapsto rt?\} \\ \textit{delay}' = \textit{delay} \oplus \{a? \mapsto 0\} \end{array}$
--

When a flight is added to the system it becomes visible to the first controller in the arrivals flow, its current position is set to the first position in the flow within

that controller, its remaining time in the air is given, and its current instruction to delay is set to zero.

With *Flight\_Appears* and *Flight\_Disappears* we can simulate the cases where flights for more or less mysterious reasons suddenly appears or disappears at random positions in the arrivals flow.

$\text{op } \textit{Flight\_Appears}$
$a? : \textit{Aircraft}$
$c? : \textit{Controller}$
$p? : \mathbb{N}$
$rt? : \mathbb{N}$
$\text{pre}(a? \notin \text{dom } \textit{control} \wedge c? \in \text{ran } \textit{flow} \wedge p? \leq \text{positions}(c?))$
$\text{changes\_only } \{\textit{control}, \textit{current\_pos}, \textit{remaining\_t}, \textit{delay}\}$
$\textit{control}' = \textit{control} \oplus \{a? \mapsto c?\}$
$\textit{current\_pos}' = \textit{current\_pos} \oplus \{a? \mapsto p?\}$
$\textit{remaining\_t}' = \textit{remaining\_t} \oplus \{a? \mapsto rt?\}$
$\textit{delay}' = \textit{delay} \oplus \{a? \mapsto 0\}$
$\text{op } \textit{Flight\_Disappears}$
$a? : \textit{Aircraft}$
$\text{pre}(a? \in \text{dom } \textit{control})$
$\text{changes\_only } \{\textit{control}, \textit{current\_pos}, \textit{remaining\_t}, \textit{delay}, \textit{contact}, \textit{waiting}\}$
$\textit{control}' = (\{a?\} \triangleleft \textit{control})$
$\textit{current\_pos}' = (\{a?\} \triangleleft \textit{current\_pos})$
$\textit{remaining\_t}' = (\{a?\} \triangleleft \textit{remaining\_t})$
$\textit{delay}' = (\{a?\} \triangleleft \textit{delay})$
$\textit{contact}' = (\{a?\} \triangleleft \textit{contact})$
$\textit{waiting}' = \textit{waiting} \setminus \{a?\}$

These operations can be used to simulate failure in the radars or more rare (but still possible) scenarios such as a flight that all of a sudden with no reason turns around and leaves the system while approaching the airport.

Communication between an aircraft and a controller is established and abandoned with *Establish\_Contact* and *Abandon\_Contact* respectively.

$\text{op } \textit{Establish\_Contact}$
$a? : \textit{Aircraft}$
$c? : \textit{Controller}$
$\text{pre}(a? \in \text{dom } \textit{control} \wedge c? \in \text{ran } \textit{flow} \wedge (a?, c?) \notin \textit{contact})$
$\text{changes\_only } \{\textit{contact}\}$
$\textit{contact}' = \textit{contact} \cup \{(a?, c?)\}$

$\text{op } \textit{Abandon\_Contact}$ <hr/> $a? : \textit{Aircraft}$ $c? : \textit{Controller}$ <hr/> $\text{pre}(a? \textit{contact } c?)$ $\text{changes\_only } \{ \textit{contact} \}$ $\textit{contact}' = \textit{contact} \setminus \{(a?, c?)\}$
---

The aircraft has to be visible to the system and the controller has to be part of the arrivals flow for the communication to be established.

The operation *Tick* is intended to be the default operation in the simulator.

$\text{op } \textit{Tick}$ <hr/> $\text{pre}(\textit{waiting} = \emptyset)$ $\forall a : \text{dom } \textit{control} \bullet$ $\textit{remaining\_t}' = \textit{remaining\_t} \oplus \{a \mapsto (\textit{remaining\_t}(a) - 1)\} \wedge$ $\text{if } \textit{delay}(a) = 0 \text{ then}$ $\quad \text{if } \textit{current\_pos}(a) = 1 \text{ then}$ $\quad \quad \text{changes\_only } \{ \textit{remaining\_t}, \textit{waiting} \} \wedge$ $\quad \quad \textit{waiting}' = \textit{waiting} \cup \{a\}$ $\quad \text{else}$ $\quad \quad \text{changes\_only } \{ \textit{remaining\_t}, \textit{current\_pos} \} \wedge$ $\quad \quad \textit{current\_pos}' = \textit{current\_pos} \oplus \{a \mapsto (\textit{current\_pos}(a) - 1)\}$ $\quad \text{fi}$ $\text{else}$ $\quad \text{if } \textit{current\_pos}(a) \in \textit{delay\_pos}(\textit{control}(a)) \text{ then}$ $\quad \quad \text{changes\_only } \{ \textit{remaining\_t}, \textit{delay} \} \wedge$ $\quad \quad \textit{delay}' = \textit{delay} \oplus \{a \mapsto (\textit{delay}(a) - 1)\}$ $\quad \text{else}$ $\quad \quad \text{if } \textit{current\_pos}(a) = 1 \text{ then}$ $\quad \quad \quad \text{changes\_only } \{ \textit{remaining\_t}, \textit{waiting} \} \wedge$ $\quad \quad \quad \textit{waiting}' = \textit{waiting} \cup \{a\}$ $\quad \quad \text{else}$ $\quad \quad \quad \text{changes\_only } \{ \textit{remaining\_t}, \textit{current\_pos} \} \wedge$ $\quad \quad \quad \textit{current\_pos}' = \textit{current\_pos} \oplus \{a \mapsto (\textit{current\_pos}(a) - 1)\}$ $\quad \quad \text{fi}$ $\quad \text{fi}$ $\text{fi}$
---

The operation can only be activated if no aircraft are waiting to be handed over. This forces the controllers to perform the handover before time goes by.

When *Tick* is activated all aircraft have their remaining time in the air decreased by one time unit. Whether or not to move an aircraft is decided by dividing its current state into six different cases. Table 1 explains the different cases in the operation.



Instruction to delay	Current position	At delay position	Action
no	1	-	The aircraft is waiting to be handed over. The set <i>waiting</i> is updated.
no	$> 1$	-	The aircraft moves one position forwards. <i>current_pos</i> is updated.
yes	-	yes	The aircraft delays. The current instruction to delay ( <i>delay</i> ) is decreased by 1 time unit.
yes	1	no	The aircraft is waiting to be handed over. The set <i>waiting</i> is updated.
yes	$> 1$	no	The aircraft moves one position forwards. <i>current_pos</i> is updated.

Table 1: The Six Cases of the *Tick* Operation

## 4 Discussions

In this section we will discuss a few shortcomings in the model.

### 4.1 Handover

The main shortcoming in the specification is how the handover is modelled. In our model the handover can be described as follows:

1. The simulator moves the aircraft into the last position (1) of a controller.
2. If the aircraft does not have a current instruction to delay or the position it is occupying disallows delay, the simulator will add the aircraft to the set *waiting*, meaning that it is waiting to be handed over.
3. At some point the ATC system will perform the handover. The aircraft will be moved to the first position within the next controller.

The problem is that there is no guarantee when the ATC system will perform the actual handover. What happens if the controller refuses to hand over a flight? In our model the flight will stay in the same position and await the handover.

The simulator will freeze until the handover is done (the *Tick* operation requires that no aircraft are waiting to be handed over to be activated).

With our simulator the handovers have to be done automatically in order to perform a smooth continuous simulation. This is not a satisfactory solution as it does not correspond to what actually happens in the real world.

Another model could have allowed the flight to move on into the next controller's airspace when it reaches the last position within an airspace. But what would the

role of the handover be in that case? If the aircraft can move on without the controller's permission then the handover operation is superfluous in our model.

## 4.2 Topology

Another issue to discuss is the number of controllers in the arrivals flow. In our model we require at least one controller to open an airport. A more realistic scenario would be to demand a larger number of controllers as a minimum. The five controllers in figure 1 (enroute, approach, director, aerodrome and ground) seem to be a reasonable number. Most airports have this structure.

Nevertheless, we chose the minimum of one controller in order to come up with a more general and generic model. We want to be able to experiment with the number of controllers and even try topologies that are very uncommon at present.

## 5 Conclusions

Together the two modules specify a domain model of an ATC region. We have modelled the ATC system including its controllers and modelled the aircraft's behaviour as a part of the simulator.

Based on the model we can formulate requirements to the system. The requirements model will be a formal derivation of the domain model.

The objective is to develop an algorithm that calculates reasonable delays for all aircraft in the system in order to obtain an optimal flow through the system and to ensure that all safety requirements are satisfied.

Our model has an appropriate selection of operations to build testing scenarios for that algorithm. We can test the algorithm on several different scenarios:

- normal operation, where aircraft arrive regularly;
- less normal operation, where aircraft arrive in an unusual pattern;
- the odd flight appears in the middle of the arrivals flow;
- a flight suddenly disappears from the system;
- communication between controller and aircraft is not established as expected;
- contact between a controller and an aircraft is suddenly lost;
- or any combination of the above.

In order to perform a safety analysis on the handover process we would need to come up with another model. We need to specify what the simulator should do with an aircraft that a controller has forgotten to hand over.

## A Mathematical Notation

This section explains some of the mathematical notations used in the specifications. The section is not a complete Sum glossary, it only contains notation used in this particular report.

### A.1 Sets

Let  $S$  and  $T$  be sets; and  $t$  an expression.

$t \in S$	Set membership.
$t \notin S$	Set non-membership.
$S \subseteq T$	Set inclusion.
$S \cap T$	Set intersection.
$S \cup T$	Set union.
$S \setminus T$	Set difference.
$\mathbb{P}S$	Powerset: the set of all subsets of $S$ .
$\#S$	Size of a finite set.

### A.2 Numbers

$\mathbb{Z}$	The set of integers.
$\mathbb{N}$	The set of natural numbers.
$\mathbb{N}_1$	The set of strictly positive natural numbers.
$m \dots n$	The set of integers between $m$ and $n$ inclusive.

### A.3 Binary Relations

A binary relation is modelled by a set of ordered pairs. Hence operators for sets can be used on relations. Let  $X$  and  $Y$  be sets;  $x : X$ ;  $y : Y$ ;  $S$  be a subset of  $X$ ;  $T$  be a subset of  $Y$ ; and  $R$  a relation between  $X$  and  $Y$ .

$X \leftrightarrow Y$	The set of relations between $X$ and $Y$ .
$x \underline{R} y$	$x$ is related by $R$ to $y$ .
$\text{dom } R$	The domain of a relation $R$ : the set of $x$ components that are related to some $y$ .
$\text{ran } R$	The range of a relation $R$ : the set of $y$ components that some $x$ is related to.
$S \triangleleft R$	Domain restriction: the relation $R$ with its domain restricted to the set $S$ .
$S \triangleleft\!\!\!\!\!\diagup R$	Domain exclusion: the relation $R$ with the members of $S$ excluded from its domain.
$R \triangleright T$	Range restriction: the relation $R$ with its range restricted to the set $T$ .
$R \triangleright\!\!\!\!\!\diagup T$	Range exclusion: the relation $R$ with the members of $T$ excluded from its range.
$R_1 \oplus R_2$	Overriding.

#### A.4 Functions

As functions are relations, all the operators defined above for relations also apply to functions. Let  $X$  and  $Y$  be sets.

$X \rightharpoonup Y$	The set of partial functions from $X$ to $Y$ . Note that the domain of a partial function does not necessarily contain the whole of $X$ , but it may.
$X \rightarrow Y$	The set of total functions from $X$ to $Y$ .

#### A.5 Sequences

Let  $X$  be a set; and  $A$  a sequence with elements taken from  $X$ .

$\text{iseq}$	The set of finite injective sequences whose elements are drawn from $X$ .
$\#A$	The length of a sequence $A$ .
$\langle \rangle$	The empty sequence.
$\text{set } A$	The set of elements in the sequence $A$ .
$\text{head } A$	The first element of a non-empty sequence $A$ .
$\text{last } A$	The final element of a non-empty sequence $A$ .

## B ATC Specification

This appendix provides the complete specification of the ATC system encapsulated in a module.

$ATC[Controller; Aircraft]$
$next\_ctrl :$ $Controller \times iseq\ Controller \rightarrow Controller$
$\forall c1, c2 : Controller; flw : iseq\ Controller \bullet$ $next\_ctrl(c1, flw) = c2 \Leftrightarrow$ $\exists i : \mathbb{N} \bullet c1 = flw(i) \wedge c2 = flw(i - 1)$
$entrance\_ctrl :$ $iseq\ Controller \rightarrow Controller$
$\forall flw : iseq\ Controller \bullet$ $entrance\_ctrl(flw) = last\ flw$
$exit\_ctrl :$ $iseq\ Controller \rightarrow Controller$
$\forall flw : iseq\ Controller \bullet$ $exit\_ctrl(flw) = head\ flw$
$number\_in\_seq :$ $Controller \times iseq\ Controller \rightarrow \mathbb{N}$
$\forall c : Controller; flw : iseq\ Controller; n : \mathbb{N} \bullet$ $number\_in\_seq(c, flw) = n \Leftrightarrow flw(n) = c$
$sum\_of\_pos :$ $\mathbb{N} \times (Controller \rightarrow \mathbb{N}) \times iseq\ Controller \rightarrow \mathbb{N}$
$\forall i : \mathbb{N}; psn : Controller \rightarrow \mathbb{N}; flw : iseq\ Controller \bullet$ $sum\_of\_pos(i, psn, flw) =$ $\text{if } i = 0 \text{ then } 0 \text{ else } psn(flw(i)) + sum\_of\_pos(i - 1, psn, flw)$
$t\_to\_end :$ $Controller \times \mathbb{N} \times (Controller \rightarrow \mathbb{N}) \times iseq\ Controller \rightarrow \mathbb{N}$
$\forall c : Controller; p : \mathbb{N}; psn : Controller \rightarrow \mathbb{N}; flw : iseq\ Controller \bullet$ $t\_to\_end(c, p, psn, flw) =$ $p + sum\_of\_pos(number\_in\_seq(c, flw) - 1, psn, flw)$

*state*

$flow : \text{iseq } Controller$   
 $positions : Controller \rightarrow \mathbb{N}_1$   
 $delay\_pos : Controller \rightarrow \mathbb{P} \mathbb{N}_1$   
 $capacity : Controller \rightarrow \mathbb{N}$   
 $landing\_ctrl : Controller$   
 $landing\_pos : \mathbb{N}_1$   
 $control : Aircraft \rightarrow Controller$   
 $current\_pos : Aircraft \rightarrow \mathbb{N}_1$   
 $remaining\_t : Aircraft \rightarrow \mathbb{N}$   
 $delay : Aircraft \rightarrow \mathbb{N}$   
 $contact : Aircraft \leftrightarrow Controller$   
 $waiting : \mathbb{P} Aircraft$   
 $open : \mathbb{B}$   
 $t\_to\_landing : Aircraft \rightarrow \mathbb{Z}$   
 $max\_delay : Aircraft \rightarrow \mathbb{N}$

$\text{dom } control = \text{dom } current\_pos$   
 $\text{dom } control = \text{dom } remaining\_t$   
 $\text{dom } control = \text{dom } delay$   
 $\text{ran } control \subseteq \text{ran } flow$   
 $\text{dom } contact \subseteq \text{dom } control$   
 $\text{ran } contact \subseteq \text{ran } flow$   
 $waiting \subseteq \text{dom } control$   
 $\forall a : \text{dom } control \bullet current\_pos(a) \leq positions(control(a))$   
 $\forall c : \text{ran } flow \bullet$   
 $\quad c \in \text{dom } positions \wedge c \in \text{dom } delay\_pos \wedge c \in \text{dom } capacity$   
 $\forall c : \text{ran } flow \bullet \forall p : delay\_pos(c) \bullet p \leq positions(c)$   
 $flow \neq \langle \rangle \Rightarrow$   
 $\quad landing\_ctrl \in \text{ran } flow \wedge$   
 $\quad landing\_pos \leq positions(landing\_ctrl)$   
 $\forall a : \text{dom } control \bullet a \in waiting \Rightarrow current\_pos(a) = 1$   
 $open \Rightarrow flow \neq \langle \rangle$   
 $\forall a : \text{dom } control \bullet t\_to\_landing(a) =$   
 $\quad t\_to\_end(control(a), current\_pos(a), positions, flow) -$   
 $\quad t\_to\_end(landing\_ctrl, landing\_pos, positions, flow)$   
 $\forall a : \text{dom } control \bullet max\_delay(a) =$   
 $\quad remaining\_t(a) - t\_to\_landing(a)$

*init*

$flow' = \langle \rangle$   
 $positions' = \emptyset$   
 $delay\_pos' = \emptyset$   
 $capacity' = \emptyset$   
 $control' = \emptyset$   
 $current\_pos' = \emptyset$   
 $remaining\_t' = \emptyset$   
 $delay' = \emptyset$   
 $contact' = \emptyset$   
 $waiting' = \emptyset$   
 $\neg open'$

*op Reset\_Flow*

$cs? : \text{iseq Controller}$   
 $c? : \text{Controller}$   
 $p? : \mathbb{N}$

$\text{pre}(control = \emptyset \wedge c? \in \text{ran } cs? \wedge p? \leq \text{positions}(c?))$   
 $\text{changes\_only } \{flow, landing\_ctrl, landing\_pos\}$   
 $flow' = cs?$   
 $landing\_ctrl' = c?$   
 $landing\_pos' = p?$

*op Reset\_Positions*

$c? : \text{Controller}$   
 $n? : \mathbb{N}$

$\text{pre}(c? \notin \text{ran } control)$   
 $\text{changes\_only } \{positions\}$   
 $positions' = positions \oplus \{c? \mapsto n?\}$

*op Reset\_Capacity*

$c? : \text{Controller}$   
 $n? : \mathbb{N}$

$\text{changes\_only } \{capacity\}$   
 $capacity' = capacity \oplus \{c? \mapsto n?\}$

---

*op* *Reset\_Delay\_Pos* \_\_\_\_\_

*c?* : *Controller*

*ps?* :  $\mathbb{P}\mathbb{N}$

---

$\text{pre}(\forall p : ps? \bullet p \leq \text{positions}(c?))$

$\text{changes\_only } \{ \text{delay\_pos} \}$

$\text{delay\_pos}' = \text{delay\_pos} \oplus \{ c? \mapsto ps? \}$

---



---

*op* *Add\_Delay\_Pos* \_\_\_\_\_

*c?* : *Controller*

*p?* :  $\mathbb{N}_1$

---

$\text{pre}(p? \notin \text{delay\_pos}(c?) \wedge p? \leq \text{positions}(c?))$

$\text{changes\_only } \{ \text{delay\_pos} \}$

$\text{delay\_pos}' = \text{delay\_pos} \oplus \{ c? \mapsto (\text{delay\_pos}(c?) \cup \{p?\}) \}$

---



---

*op* *Delete\_Delay\_Pos* \_\_\_\_\_

*c?* : *Controller*

*p?* :  $\mathbb{N}_1$

---

$\text{pre}(p? \in \text{delay\_pos}(c?))$

$\text{changes\_only } \{ \text{delay\_pos} \}$

$\text{delay\_pos}' = \text{delay\_pos} \oplus \{ c? \mapsto (\text{delay\_pos}(c?) \setminus \{p?\}) \}$

---



---

*op* *Handover\_Flight* \_\_\_\_\_

*a?* : *Aircraft*

---

$\text{pre}(a? \in \text{waiting} \wedge \text{control}(a?) \neq \text{exit\_ctrl}(\text{flow}))$

$\text{changes\_only } \{ \text{control}, \text{current\_pos}, \text{waiting} \}$

$\text{control}' = \text{control} \oplus$

$\{ a? \mapsto \text{next\_ctrl}(\text{control}(a?), \text{flow}) \}$

$\text{current\_pos}' = \text{current\_pos} \oplus$

$\{ a? \mapsto \text{positions}(\text{next\_ctrl}(\text{control}(a?), \text{flow})) \}$

$\text{waiting}' = \text{waiting} \setminus \{ a? \}$

---



*op Remove\_Flight*

$a? : \text{Aircraft}$

$\text{pre}(a? \in \text{waiting} \wedge \text{control}(a?) = \text{exit\_ctrl}(\text{flow}))$   
 $\text{changes\_only} \{ \text{control}, \text{current\_pos}, \text{remaining\_t}, \text{delay}, \text{contact}, \text{waiting} \}$   
 $\text{control}' = (\{a?\} \triangleleft \text{control})$   
 $\text{current\_pos}' = (\{a?\} \triangleleft \text{current\_pos})$   
 $\text{remaining\_t}' = (\{a?\} \triangleleft \text{remaining\_t})$   
 $\text{delay}' = (\{a?\} \triangleleft \text{delay})$   
 $\text{contact}' = (\{a?\} \triangleleft \text{contact})$   
 $\text{waiting}' = \text{waiting} \setminus \{a?\}$

*op Delay\_Flight*

$a? : \text{Aircraft}$

$t? : \mathbb{N}$

$\text{pre}(a? \in \text{dom control} \wedge a? \text{ contact control}(a?))$   
 $\text{changes\_only} \{ \text{delay}, \text{waiting} \}$   
 $\text{delay}' = \text{delay} \oplus \{a? \mapsto t?\}$   
 $\text{waiting}' = \text{waiting} \setminus \{a?\}$

*op Open\_Entrance*

$\text{pre}(\neg \text{open})$

$\text{changes\_only} \{ \text{open} \}$

$\text{open}'$

*op Close\_Entrance*

$\text{pre}(\text{open})$

$\text{changes\_only} \{ \text{open} \}$

$\neg \text{open}'$

## C Simulator Specification

This appendix provides the complete specification of the simulator encapsulated in a module.

<i>Simulator</i>	
<i>import ATC</i>	
<i>state</i>	<i>ATC.state</i>
<i>init</i>	<i>ATC.init</i>
<i>op Add_Flight</i>	
<i>a?</i> : <i>Aircraft</i>	
<i>rt?</i> : $\mathbb{N}$	
	$\text{pre}(\text{open} \wedge a? \notin \text{dom control})$ $\text{changes\_only} \{ \text{control}, \text{current\_pos}, \text{remaining\_t}, \text{delay} \}$ $\text{control}' = \text{control} \oplus \{ a? \mapsto \text{entrance\_ctrl}(\text{flow}) \}$ $\text{current\_pos}' = \text{current\_pos} \oplus \{ a? \mapsto \text{positions}(\text{entrance\_ctrl}(\text{flow})) \}$ $\text{remaining\_t}' = \text{remaining\_t} \oplus \{ a? \mapsto \text{rt}? \}$ $\text{delay}' = \text{delay} \oplus \{ a? \mapsto 0 \}$
<i>op Flight_Appears</i>	
<i>a?</i> : <i>Aircraft</i>	
<i>c?</i> : <i>Controller</i>	
<i>p?</i> : $\mathbb{N}$	
<i>rt?</i> : $\mathbb{N}$	
	$\text{pre}(a? \notin \text{dom control} \wedge c? \in \text{ran flow} \wedge p? \leq \text{positions}(c?))$ $\text{changes\_only} \{ \text{control}, \text{current\_pos}, \text{remaining\_t}, \text{delay} \}$ $\text{control}' = \text{control} \oplus \{ a? \mapsto c? \}$ $\text{current\_pos}' = \text{current\_pos} \oplus \{ a? \mapsto p? \}$ $\text{remaining\_t}' = \text{remaining\_t} \oplus \{ a? \mapsto \text{rt}? \}$ $\text{delay}' = \text{delay} \oplus \{ a? \mapsto 0 \}$

*op Flight\_Disappears*

*a?* : *Aircraft*

$\text{pre}(a? \in \text{dom } \textit{control})$

$\text{changes\_only } \{ \textit{control}, \textit{current\_pos}, \textit{remaining\_t}, \textit{delay}, \textit{contact}, \textit{waiting} \}$

$\textit{control}' = (\{a?\} \triangleleft \textit{control})$

$\textit{current\_pos}' = (\{a?\} \triangleleft \textit{current\_pos})$

$\textit{remaining\_t}' = (\{a?\} \triangleleft \textit{remaining\_t})$

$\textit{delay}' = (\{a?\} \triangleleft \textit{delay})$

$\textit{contact}' = (\{a?\} \triangleleft \textit{contact})$

$\textit{waiting}' = \textit{waiting} \setminus \{a?\}$

*op Establish\_Contact*

*a?* : *Aircraft*

*c?* : *Controller*

$\text{pre}(a? \in \text{dom } \textit{control} \wedge c? \in \text{ran } \textit{flow} \wedge (a?, c?) \notin \textit{contact})$

$\text{changes\_only } \{ \textit{contact} \}$

$\textit{contact}' = \textit{contact} \cup \{(a?, c?)\}$

*op Abandon\_Contact*

*a?* : *Aircraft*

*c?* : *Controller*

$\text{pre}(a? \text{ contact } c?)$

$\text{changes\_only } \{ \textit{contact} \}$

$\textit{contact}' = \textit{contact} \setminus \{(a?, c?)\}$

*op Tick*

```

pre(waiting =  $\emptyset$ )
 $\forall a : \text{dom } \textit{control} \bullet$ 
  remaining_t' = remaining_t  $\oplus$  { a  $\mapsto$  (remaining_t(a) - 1) }  $\wedge$ 
  if delay(a) = 0 then
    if current_pos(a) = 1 then
      changes_only { remaining_t, waiting }  $\wedge$ 
      waiting' = waiting  $\cup$  { a }
    else
      changes_only { remaining_t, current_pos }  $\wedge$ 
      current_pos' = current_pos  $\oplus$  { a  $\mapsto$  (current_pos(a) - 1) }
    fi
  else
    if current_pos(a)  $\in$  delay_pos(control(a)) then
      changes_only { remaining_t, delay }  $\wedge$ 
      delay' = delay  $\oplus$  { a  $\mapsto$  (delay(a) - 1) }
    else
      if current_pos(a) = 1 then
        changes_only { remaining_t, waiting }  $\wedge$ 
        waiting' = waiting  $\cup$  { a }
      else
        changes_only { remaining_t, current_pos }  $\wedge$ 
        current_pos' = current_pos  $\oplus$  { a  $\mapsto$  (current_pos(a) - 1) }
      fi
    fi
  fi
fi

```

## References

- [1] Brenton Atchison, Peter Lindsay, and David Tombs. *Using Formal Methods for Software Safety Assurance*. Software Verification Research Centre, The University of Queensland, 1999.
- [2] Juan C. Bicarregui, John S. Fitzgerald, Peter A. Lindsay, Richard Moore, and Brian Ritchie. *Proof in VDM: A Practitioner's Guide*. Springer-Verlag, 1994.
- [3] The Cogito Group. *The Sum Reference Manual*. Software Verification Research Centre, 1997.
- [4] Daniel Hazel, Paul Strooper, and Owen Traynor. *Requirements Engineering and Verification using Specification Animation*. Software Verification Research Centre, The University of Queensland, 1998.
- [5] Michael S. Nolan. *Fundamentals of Air Traffic Control*. Wadsworth Publishing Company, 1994.
- [6] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1996.
- [7] J. M. Spivey. *The Z Notation*. Prentice Hall, 1989.