Formal Analysis of Arrival Procedure of Air Traffic Control System

Shahid Yousaf[1], Sher Afzal Khan[2], Nazir Ahmad Zafar[3], Farooq Ahmad[1], and Muazzam Ali Khan[2]

[1]Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan
[2]Department of Computer sciences, Abdul Wali Khan University, Mardan, Pakistan
[3]Department of Computer Science, King Faisal University, Al Hassa, Saudi Arabia
Email: shahid.yousaf@ucp.edu.pk; muazzam@awkum.edu.pk; drfarooq@ucp.edu.pk; nazafar@kfu.edu.sa

Abstract: The air traffic control (ATC) is safety, monetary and environmental critical system. Its failure may cause the loss of human life, severe injuries, loss of money and environmental issues. The complexity of such systems requires formal modeling and step by step design processes. In this paper we investigate the use of formal method VDM++ to specify and verify the arrival procedure of aircrafts. The control along arrival procedure changes from the ramp to the gate controller to make possible the safe arrival. For the specification the bottom up approach is used to model the system. Initially, aircraft, ramp and gate controller are specified, then all are combine for their synchronize approach. The specification and syntactical verification are performed by VDM++ which is an object oriented model based formal approach.

[Yousaf S, Khan SA, Zafar NA and Farooq A and Khan MA. **Formal Analysis of Arrival Procedure of Air Traffic Control System.** *Life Sci J* 2012;9(4):3094-3098] (ISSN:1097-8135). http://www.lifesciencesite.com. 454

.

Keywords: VDM++; air traffic control system; formal specification; formal method.

1. Introduction

The major concern of air traffic control system is to ensure the safe operation of private and commercial aircrafts [5]. ATC is heavily dependent upon the capabilities of human operator; some accidents in ATC were documented by "human error" with the causal factor involving the perception, memory, decision making, communication and resource management [4]. Therefore formal analysis is very essential for proving safety properties of ATC system. Formal methods are used to remove the ambiguities in specification of system and have been applied to specify and verify the complex systems. The above mentioned reason motivated us to use formal methods to design ATC system. The work of S. Ahmad and V. Saxena [1] used the Sami formal notation UML which cannot be verified systematically to ensure a specification's accuracy [9]. VDM++ has the following advantages to design air traffic control system.

1). This specification technique is more comprehensive form than other methods. 2) It gives a precise definition of what is going to build. 3) In our research, VDM++ helped to clarify the key ideas of ATC system. 4) It provided a precise way of defining the data and underlying functions of the ATC system. 5) It also provided us a way to specify the interface between components of the entire system under development in a precise manner.

For the safe arrival process aircrafts communicate with the air traffic controllers. In this arrival procedure initially, the aircraft is under the control of ramp controller all the activities during arrival process of aircraft are controlled by this controller. Initially the aircraft will send request to the ramp controller for the entrance of ramp area and on this request the ramp controller grant the permission to aircraft then after this aircraft enter into ramp area. The gate controller arranged all the aircraft in sequence at ramp and then control is transfer to gate controller. Just likewise the ramp controller, gate controller controls the activities of aircraft. In this paper, we have used the extend version of Vienna Development Method (VDM) that is VDM++ to formalize the arrival procedure of aircraft. The organization of this paper is as fallow. In section 2, an introduction to formal methods is presented. Formal modeling of the arrival procedure is given using VDM++ in section 3. Finally, conclusion and future works is discussed in section 4.

2. Formal Methods

Formal methods [13-15] consist of the set of techniques and tools based on mathematical modeling and formal logic that are used to specify and verify requirements and designs for computer systems and software as presented in various application [16-26]

2.1 Classification of Formal Methods

Formal methods are used for both software and hardware designing or software- hardware co-designing [6, 10]. Classification of formal methods with respect to the use of it, is given below as discussed in [10].

Writing Formal Specification:

Formal methods are used to reason about mathematical objects. However, hardware circuits are not mathematical objects, they are real world physical objects. Therefore, it is necessary to develop

mathematical model of system and also describe the properties of that system [7]. The formal specification of a system is written in term of mathematical notation which is precise and unambiguous.

Proving Properties about the Specification: The requirement specification normally given in informal languages when we write it in formal specification language then this is an error-prone. This formal specification is used for proving the properties of the system.

Deriving Implementation from a given Specification:

Once the specification is written then it is helpful to design models which automatically derives the implementation of system with the full requirements. This idea actually belongs to the fifth generation of the programming language, i.e., PROLOG [8], where the implementation phase and system specification is very closely related to each other.

However, specifications are often given in a declarative manner and not in constructive manner. This means that these specifications only describe what the system should do but not how this function can be achieved. It is certainly impossible to derive correct program from declarative specification since these problems are inherently undecidable thus the machine can never solve them. Therefore, the construction of appropriate implementation always remains a creative task for human being [7].

Verifying a specification w.r.t. a given implementation:

It is possible that the description of the system which is automatically derived may be less detailed. However, the design steps that are used to refine the description of the system must not effect the validity of the system specification. However, it must be checked that the abstract implementations satisfy the original specification. This is a formal verification process. The formal verification can be applied in two different ways. One method is based on the automated theorem proving for the certain formal language. In 1980 another technique was developed which is called model checking. In model checking the description of the system is not given in the logic. The procedure of model checking is task to evaluate the specification in interpretation.

2.2 Application of Formal Methods

Formal specification techniques are applicable in many real time systems but are most applicable in the development of critical systems and standards [9].

2.2.1 Security Critical Systems

Security critical system involves authorize use of system. For the verification of network protocol, formal methods are used [9]. The network security is essential for every organization either it is private or government sector because intruder effect the networks which can cause for loss of precious information and resource, therefore the use of formal methods for writing specification of protocol are helpful to achieve security goals of protocols. Some security models are formalized and the verified by using formal automated tools [11].

2.2.2 Complex Systems

Formal approach is used to develop the complex systems [2]. This is the only technique, which gives us precisely specifying models of system for the complex software systems [3].

2.2.3 Safety Critical Systems

Safety critical systems are also called life critical systems because in the safety critical systems failure of the system or software might be dangerous for life. Criticality is often expressed in terms of:

- Reliability
- Availability
- Maintainability
- Safety
- Security

Critical systems make expensive methods worthwhile and needs experience. The most common Examples of safety-critical systems are given below.

Medicine: The medicine is critical area where we cannot afford the failure of the system because the failure of the system means failure of the life or some bad effect on the life. Following are the some machine and system used in medicine.

- Heart-lung machines
- Mechanical ventilation systems
- Infusion pumps and Insulin pumps
- Radiation therapy machines
- Robotic surgery machines
- Defibrillator machines

Nuclear Engineering: Nuclear reactor control system has a close relationship with safety critical systems. Even the miner mistake can cause the inerrable lose of life.

Transport: The transportation system is implemented almost in every country of the world. The railway signaling and control system belongs to safety critical systems. If the problem occurs in this system, the lives would be in danger zone.

Aviation: Aviation includes all those activities that are manmade flying devices like aircraft and fighter jet. Following are the some systems related to it.

- Air traffic control systems
- Avionics, particularly fly-by-wire systems
- Radio navigation RAIM
- Aircrew life support systems

- Flight planning to determine fuel requirements for a flight

3. Formal Modeling Using VDM++

Formal modeling is being increasing mentioned in some safety-related standards as a possible method of improving dependability. The formal specification of the air traffic control system is defined as: Three main entities, aircraft, gate controller and ramp controller are defined.

3.1 Aircraft

Types and definition are same as for the departure procedure the class is defined with name *AirCraft.*

```
class AirCraft
types
public string = seq of char;
public Aircraft: ACid:string
callsign:string
```

3.2 Instance Variables

The instance variables used in the specification are given below. *RC,* and *GC* are respectively objects of ramp and gate controller, which allow accessing all the instance variables of these controllers in the class.

```
instance variables
public Aircrafts:set of Aircraft;
Public NIL:string;
public RTTaxiWayQ:seq of string;
TaxiWayQ:seq of string;
public RTRampQ:seq of string;
public RTTaxiclcQ:seq of string;
public Assignedgate:map string to string;
public RequestGate:seq of string;
RTPgateQ: seq of string;
Reached: seq of string;
pGateQ: seq of string;
RC:Rampcontroller;
GC:GateController;
```

3.3 Possible Operations

The following operations are modeled to perform certain task for the arrival procedure.

Request to Enter Ramp:

The operation denoted by *Request To EnterRamp (craftin:string)* is defined, where aircraft sends request to enter in the ramp area. The pre-condition of this operation ensures that the aircraft must be a registered aircraft, it does not belong to those aircrafts which have sent request to enter ramp. It must reside on taxiway. The post-condition includes it to those aircraft which have sends request for enter ramp area.

```
RequestToEnterRamp(craftin:string)
ext wr RTRampQ:seq of string
   rd Aircrafts:set of Aircraft
   wr TaxiWayQ:seq of string
pre exists a in set Aircrafts & a.ACid = craftin
   and craftin not in set elems RTRampQ
   and craftin in set elems TaxiWayQ
post RTRampQ =RTRampQ~ ^ [craftin]
    and TaxiWayQ=tl TaxiWayQ~;
```

Enter in Ramp Area:

The operation denoted by *Enter Ramp (craftin:string)* is defined where aircraft can enter into the ramp area. The pre-condition ensures that only that aircraft can enter into ramp area which is registered and have clearance to enter ramp area. In the post condition aircraft enter into ramp area and its permission to enter ramp is discarded.

```
EnterRamp(craftin:string)
ext wr Ramp:string
   wr RC:Rampcontroller
   rd NIL:string
pre exists a in set Aircrafts & a.ACid = craftin
   and craftin in set elems RC.GClcrampQ
   and Ramp=NIL
post  Ramp=craftin
    and RC.GClcrampQ=tl RC.GClcrampQ~;
```

Request to Assign Gate:

The operation denoted by *Request Assigngate (craftin:string)* is defined in the specification where aircraft sends requests to gate controller for the assigning of gate. The pre-condition shows that there are three invariants on this operation first one is that the aircraft which sent a request to assign the gate should not be part of those aircrafts which have already requested to assign gate and second is that it does not have already assigned gate, third is that this aircraft also be a valid aircraft, i.e., it belongs to those aircrafts which are registered and last is that aircraft belong to those aircrafts which are in ramp queue. In the post-condition request of aircraft is confirmed.

```
RequestAssigngate(craftin:string)
ext  wr RequestGate:seq of string
    --rd Assignedgate:map string to string
    wr RC:Rampcontroller
   -- wr GC:GateController
    rd Aircrafts:set of Aircraft
pre  craftin not in set elems RequestGate
   -- and craftin not in set dom GC.Assignedgate
    and exists a in set Aircrafts & a.ACid = craftin
    and craftin  in set elems RC.RampQ
post RequestGate=RequestGate~ ^ [craftin];
```

Request to Pass from Gate:

The operation denoted by *RequesttoPassGate (craftin:string)* is defined where aircraft send request to pass from the gate.

```
RequesttoPassGate (craftin:string)
ext wr RTPgateQ: seq of string
   wr GC:GateController
   rd Aircrafts:set of Aircraft
pre  exists a in set Aircrafts & a.ACid = craftin
```

```
    and craftin not in set elems RTPgateQ
    and craftin in set dom GC.Assignedgate
post RTPgateQ= RTPgateQ~ ^ [craftin];
```

Pass from Gate:

To pass from the gate the aircraft must have permission to pass from the gate, it is also registered aircraft then the aircraft can pass from the gate.

```
PassFromGate(craftin:string)
ext wr pGateQ:seq of string
    wr GC:GateController
    rd Aircrafts:set of Aircraft
pre exists a in set Aircrafts & a.ACid = craftin
    and craftin not in set elems pGateQ
    and craftin  in set elems GC.PFclearence
post pGateQ=pGateQ~ ^ [craftin]
     and GC.PFclearence= tl GC.PFclearence~;
```

Final Arrived:

This operation keep the record of those aircrafts which are arrived.

```
arrived(craftin:string)
ext wr Reached:set of string
    wr pGateQ:seq of string
    rd Aircrafts:set of Aircraft
pre exists a in set Aircrafts & a.ACid = craftin
    and craftin in set elems pGateQ
    and craftin not in set Reached
post Reached= Reached~ union {craftin}
     and pGateQ=tl pGateQ~;
end AirCraft
```

3.4 Ramp Controller

The ramp controller defined as class *RampController* and the type used in this class is string.

```
class Rampcontroller
types
public string = seq of char;
```

The instance variable used in this specification given below. "AC" is the object of the Aircraft which allow accessing all the instance variables of the Aircraft and "GC" is the object of the ground controller for accessing the all variables of the ground controller.

```
instance variables
AC:AirCraft;
public RampQ:seq of string;
GClcrampQ: seq of string;
```

Grant Clearance to Enter Ramp: The operation denoted by *GrantClearanceTo EnterRamp (craftin:string)* is defined so that permission for enter ramp is granted to aircraft. The pre-condition of this operation ensures that the aircraft must be registered before enter in the ramp area, it does not belong to those aircraft which already have clearance to enter ramp and this aircraft have sent request for entering ramp. In the post-condition clearance is granted to aircraft and its request is discarded which it has send for enter ramp.

```
operations
GrantClearanceToEnterRamp(craftin:string)
ext wr AC:AirCraft
    wr GClcrampQ: seq of string
pre exists a in set AC.Aircrafts & a.ACid = craftin
     and craftin in set elems AC.RTRampQ
     and craftin not in set elems GClcrampQ
post GClcrampQ = GClcrampQ~ ^ [craftin]
     and AC.RTRampQ= tl AC.RTRampQ~;
```

Sequencing at Ramp: For the arrangement of aircraft at ramp, the operation denoted by *SequenceATRamp (craftin:string)* is defined. Pre-condition ensures that there must be an aircraft in the ramp variable, this aircraft must not be in the ramp queue and it also be a registered aircraft. Post-condition promoted it to the ramp queue and variable ramp become free.

```
SequenceATRamp(craftin:string)
ext wr RampQ:seq of string
    wr GClcrampQ: seq of string
    wr AC:AirCraft
pre   exists a in set AC.Aircrafts & a.ACid = craftin
      and craftin not in set elems RampQ
post RampQ= RampQ~ ^ [craftin]
     and AC.Ramp=AC.NIL;
end Rampcontroller
```

3.5 Gate Controller

The gate controller is responsible to assign the gate to aircraft without assigning the gate the craft cannot proceed for the departure. It is defined as class 'GateController'.

The types which are used are "string" and "Gate" the Gate is a composite type which has gate id "Gid" and status of gate "status ''.

```
class GateController
types
public string = seq of char;
Gstatus = <FREE>|<BUSY>;
Gate::   Gid:string
            status:Gstatus;
```

The instance variables used in the specification are given below. "AC" is the object of the Aircraft, which allows accessing all the instance variables of the Aircraft.

```
 instance variables
AC:AirCraft;
Gates:seq of Gate;
public Assignedgate:map string to string;
public PFclearence:seq of string;
```

Functions: *Isavailable* function is modeled formally, which returns the position of that gate whose status is free.

```
functions
isavailable(gateidin:seq of Gate)pos:nat
pre true
```

```
post gateidin(pos).status = <FREE> and forall i in set
{1,...,pos-1} & gateidin(i).status <> <FREE>;
```

Assign Gate: The gate assigning process is catered in the following operation. Here first it is checked that the aircraft which has sent request for gate assignment that should not be assigned the gate, then the gate will assign to aircraft if the gate status is free otherwise the gate will not be assigned to the aircraft.

```
operations
AssignGate()
ext wr AC:AirCraft
   wr Assignedgate:map string to string
   wr Gates:seq of Gate
pre let pos = isavailable(Gates)
   in pos <> 0
post let pos = isavailable(Gates)
    in Assignedgate = Assignedgate~ munion {hd
(AC.RequestGate) |-> Gates(pos).Gid};
```

Clearance to pass from the Gate: This operation is defined in which aircraft is allowed to pass from the gate.

```
ClearancetoPasFromGate(craftin:string)
ext wr PFclearence:seq of string
   wr AC:AirCraft
pre craftin not in set elems PFclearence
   and craftin in set dom Assignedgate
   and exists a in set AC.Aircrafts & a.ACid = craftin
post PFclearence=PFclearence~ ^ [craftin]
    and AC.RTPushBack= tl AC.RTPushBack~
end GateController.
```

4. Conclusion

From the model of aircraft control system along the arrival, we revealed that the use of formal method for such system is necessary. The formalized structure gave the primary and fundamental basis for safety critical systems. It also provided necessary and excellent basis for fault tolerance and reliable structure of the system. The method ensured the consistency, reliability and safety of the model. All the above properties can reduce the failure ratio of air traffic control system. In the development the bottom up approach is used i.e., initially the basic components like aircraft; ramp and ground controller are specified. Further, for synchronized affect they all are composed.

References

[1]. Ahmad, S. and Saxena, V. 2008. Design of formal air traffic control system through UML. Ubiquitous computing and communication journal. Vol. 3, No. 2, pp. 1-10

[2]. Andres, C., Rafael, G. and Nunez, M. 2008. Using formal methods to develop complex information system: a practical/theoretical experience. Proceedings of ACM symposium on applied computing, pp. 848-849, USA.

[3]. Gabbar, A. 2006. Modern formal methods and application. Springer.

[4]. Hanh, T.T.B. and Hung, D. V. 2007. Verification of an air traffic control system with probabilistic real time model-checking. UNU-IIST report.

[5]. Kacem, A. H. and Kacem, N. H. 2007. From formal specification to model checking of MAS using CSP and SPIN. Internation journal of computing and information sciences, Vol. 5, No. 1, pp. 35-44.

[6]. McDermid, J., Galloway, A., Burton, S., Clark, J., Toyn, I., Tracey, N. and Valentine, S. 1998. Towards industrially applicable formal methods: three small steps, and one giant leap. Second international conference on formal engineering methods, pp. 76-88, Australia.

[7]. NASA. 1998. Formal Methods specification and verification guidebook for software and computer System. Vol. 1.

[8]. Reagan, G.O. 2008. A brief history of computing. Springer.

[9]. Razali, R., Snook, C. F., Poppleton, M. R., Garratt, P. W. and Walters, R. J. 2007. Experimental comparison of the comprehensibility of a UML-based formal specification versus a textual one. 11th international conference on evaluation and assessment in software engineering, pp. 1-11.

[10]. Schneider, K. 2004. Verification of reactive systems: formal methods and algorithms. Texts in theoretical computer science series, Springer.

[11]. Sidhu, D., Chung, A. and Blumer, T. 1998. Experience with formal methods in protocol development, Proceedings of the IFIP TC/WG6.1 second international conference on formal description techniques for distributed systems and communication protocols, pp. 437-453.

[12]. Sterling, L., Shapiro, E. and Warren, D. H. D. 1986. The art of prolog advanced programming techniques. Second edition, The MIT press cambridge, massachusetts london, England.

[13]. Ahmad, F. and S. A. Khan (2012). "Module-based architecture for a periodic job-shop scheduling problem." Computers & Mathematics with Applications.

[14]. Ali, G., S. A. Khan, et al. (2012). "Formal modeling towards a dynamic organization of multi-agent systems using communicating X-machine and Z-notation." Indian Journal of Science and Technology 5(7).

[15]. Gul Afzal Khan, S. A. K., Nazir Ahmad Zafar, F.A.S.I. (2012). "A Review of different Approaches of Land Cover Mapping." Life Sci J 9(4).

[16]. Khan, S. A., A. A. Hashmi, et al. (2012). "Semantic Web Specification using Z-Notation." Life Science Journal 9(4).

[17]. Khan, S. A. and N. A. Zafar (2007). "Promotion of local to global operation in train control system." Journal of Digital Information Management 5(4): 231.

[18]. Khan, S. A. and N. A. Zafar (2009). Towards the formalization of railway interlocking system using Z-notations, IEEE.

[19]. Khan, S. A. and N. A. Zafar (2011). "Improving moving block railway system using fuzzy multi-agent specification language." Int. J. Innov. Computing, Inform. Control 7(7).

[20]. Khan, S. A., N. A. Zafar, et al. (2011). "Extending promotion to operate controller based on train's operation." International J. Phy. Sci 6(31): 7262 - 7270.

[21]. Khan, S. A., N. A. Zafar, et al. (2011). "Petri net modeling of railway crossing system using fuzzy brakes." International J. Phy. Sci 6(14): 3389-3397.

[22]. M, F. and S. A. Khan (2012). "Specification and Verification of Safety Properties along a Crossing Region in a Railway Network Control." Applied Mathematical Modelling, 10.1016/j.apm.2012.10.047.

[23]. Raza, M. I., Q. J. Zaib, et al. (2012). "Meticulous analysis of Semantic Data Model -An optimal approach for ERD." J. Basic. Appl. Sci. Res. 8(2): 8344-8354.

[24]. Yousaf, S., N. A. Zafar, et al. (2010). Formal analysis of departure procedure of air traffic control system, IEEE.

[25]. Zafar, N. A., S. A. Khan, et al. (2012). "Formal Modeling towards the Context Free Grammar." Life Science Journal 9(4).

[26]. Zafar, N. A., S. A. Khan, et al. (2012). "Towards the safety properties of moving block railway interlocking system." Int. J. Innovative Comput., Info & Control.

11/9/2012