Formal Verification and Validation of Aircraft Departure Process in Air Traffic Control System Using VDM++

Shahid Yousaf, Nazir Ahmad Zafar, and Sher Afzal Khan

***Abstract*—The safety critical systems are those whose failure may cause the loss of human life, serious injuries and financial disasters. The air traffic control system (ATC) is a safety critical system and rise in the air traffic volume make it more critical and complex which caused for the unwanted delay in the aircraft departure process. To ensure its safety it requires advance methodologies for its designing process. The VDM++ is an emerging technique based on mathematical notation having object oriented features and used to specify and verify the software system. This technique is used to formalize the departure process of ATC system. The departure process is controlled by the air traffic controllers. The air traffic controllers control the aircraft traffic. This departure process is made possible with the help of the ground and local controller. Initially, the aircraft is under the control of ground controller. Further the control is transfer to the local controller. These both controllers communicate to aircraft and also to each other for safe and secure movement of aircraft and finally the aircraft depart from the airport.**

***Index Terms*—VDM, VDM++, ATC, formal method.**

I. Introduction

Computers are increasingly being used in safety critical system, interactive control systems for transport (like road, rail and air) medical equipment and power stations and process plants are example of it [1]. The air traffic control system (ATC), a part of the airspace system, has a responsibility to manage the complex mixture of air traffic from general, commercial, corporate and military aviation [2]. ATC system is a name of service which provides guideline to aircrafts, prevents collision and manages secure and orderly traffic flow. It is a vast network consisting of equipment and people, which ensure the safe operation of aircrafts [3]. The primary objective of ATC system is the safety of aircrafts and its passenger. To ensure the safety we must satisfy that there is no conflicting situation in any stage of the aircraft departure and arrival procedure. The air traffic controllers play important role to avoiding the collision of aircrafts. The term collision refers to both mid-air and ground collision. The effectiveness of a system means the departure and landing of any aircraft without any collision.

The ATC system is safety critical system where the miner mistake can cause intolerable loss. It needs to develop carefully to prevent the precious lives. ATC system provides verity of service to aircraft like weather updates, flight profile information, emergency relief services and navigational etc.

A number of failures in the safety critical systems have been reported in [4], [5], [6], [7]. Therac-25 system failed due to the error in software, this system was a radiation therapy machine controlled by computer which overdoses the six peoples. In 1996 another disaster happened when the European Ariane 5 launcher crashed after 40 sec of its take off. The inquiry board consisting of European space agency and CNES (French National Centre for Space Studies) documented that this explosion was the consequence of software error. The F-16 is fighter jet that has faced number of accidents. According to the inquiry report the reason of these accidents are the combination of both onboard computer controller and human error. Many more example of safety critical system failures have been documented which caused the loss of precious human life and equipment.

The major reason of the software failure is an inconsistent and ambiguous specification. A significant problem of developing software for safety critical systems is how to guarantee that the functional behavior of a developed software system will satisfy the corresponding functional requirements and will not violate the safety requirements of overall system. In order to solve this problem, it is important to analyze thoroughly the safety properties of the overall system, to achieve accurate software functional requirements and to verify properly the implementation of the software. But it is difficult to analyze all of the above mentioned properties of the software system by the traditional software development methods. Because these development methods are based on the natural languages which are inherently ambiguous and hence it is not possible to write the unambiguous specification from the ambiguous language [8]. In [9] it is described that formal methods are widely recognized as a mean to write precise, consistent and unambiguous specification and are helpful to analyze thoroughly the properties of the overall system due to which we can achieve accurate software functional requirements. Formal methods research began in the 1960s, which focused on establishing mathematical and rigorous approaches to program construction and analysis [10], [11], [12], [13]. Formal methods are mathematically-based techniques, often supported by reasoning tools that can offer a rigorous and effective way to model, design and analyze computer systems [14]. Typical techniques used in formal methods are invariants, proof obligations, and a calculus for refining specifications or proving properties about specifications and implementations, and the relationship between a specification and its implementation [15]. Due to the above

Manuscript received September 10, 2012; revised October 23, 2012.

Shahid Yousaf is with Department of Computer Science & IT, The University of Lahore, Pakistan (e-mail: shahid.usaf@yahoo.com).

Nazir Ahmad Zafar is with the Department of Computer Science, College of CS & IT, King Faisal University, Al Hassa, Saudi Arabia (e-mail: nazafar@kfu.edu.sa).

Sher Afzal khan is with the Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan (e-mail: sher.afzal@awkum.edu.pk).

DOI: 10.7763/IJET.2012.V4.478

distinguish properties of the formal methods we have used the formal methods “VDM++” for specification of air traffic control system. It is a formal specification language, derived from VDM, it is extends by providing object-orientation, parallel and real time features [16]. In this work, we model the system for the ground level aircraft departure process with the help of ground and local controller, which are the parts of air traffic controllers. An air traffic controller collaborates with other controllers to hand off an aircraft, after successfully hand off the aircraft communicates with other controllers.

The organization of this paper is as fallow. In section II formal methods are explored. Formal modeling using VDM++ is given in section III and finally the last section IV will describe the conclusion.

II. Formal Methods

The software engineers and designers always try to develop reliable and consistent software but the behavior of software is often a surprise for them. Mostly software systems fail and cannot fulfill the requirements, there are many reasons of their failure but most of them fail due to incomplete and ambiguous requirement specification, which are hardly difficult to handle by using traditional software development process. Formal methods are used as an emerging technology which handles these problems in systematic way using logical arguments. Formal methods [17] 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. Further, formal methods [18] refer to mathematical rigorous techniques and tools for the specification, design and verification of software and hardware systems. Single formal technique is not well suitable for all kinds of problems, therefore integration of methods in same time is necessary. Formal specification provides the power to clearly express the meanings, across natural languages barriers, just like legal languages, which were developed to prevent misinterpretation of law. Formal specification was developed to prevent misinterpretation of specifications.

III. Formal Modeling Using VDM++

Modeling is an engineering technique, which plays an important role to proceed the project toward the success. The modeling clearly reflects the behavior of real world system. The ATC is a highly complex and safety critical system that requires visualization, complexity management and communication. VDM++ will help in handling complexities and drawbacks of existing ATC systems and help to clearly understand the domain.

A. Aircraft

The aircraft is a core part of this modeling and other two are supporting of it. It is defined as class *AirCraft.* The types which are used in this class are *string* and *Aircraft*, where the *string* is sequence of character and the *Aircraft* is a composite type which has aircraft id and callsign.

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

The instance variables used in the specification are given below. *“LC”* is the object of the local controller and “*GCC*” is the object of ground controller which allow accessing all the instance variables of the ground and local controller in this class.

```
instance variables
LC:LocalController;
GCC:Groundcontroller;
public Aircrafts:set of Aircraft;
public NIL:string;
public Taxiway:string;
public TaxiwayQ:seq of string;
public RTTaxiclcQ:seq of string;
public RunwayQ:seq of string;
public Taxiin:seq of string;
public RTdepartclcQ:seq of string;
public Runsonrunway:string;
private CancelDepart:set of string;
private Recoverplan: seq of string;
private FinalDept:seq of string;
```

Operations are the key features of the specification. The following operations are modeled to perform certain task.

1) *Request for Taxi Clearance:* In the operation denoted by *RequestTaxiclarence(craftin:string)* the aircraft sends request for taxi clearance. The pre-condition ensures that prior to this, aircraft must be a registered aircraft, departure list has been assigned to it and it does not send request for taxi clearance. The post-condition includes it to those aircrafts which are waiting for taxi clearance.

```
RequestTaxiclarence(craftin:string)
ext wr RTTaxiclcQ:seq of string
    rd Aircrafts:set of Aircraft
    wr GCC:Groundcontroller
pre  exists a in set Aircrafts & a. callsign = craftin
     and craftin not in set elems RTTaxiclcQ
     and craftin in set elems GCC.Deplist
post RTTaxiclcQ = RTTaxiclcQ~ ^ [craftin];
```

2) *Taxing:* The pre-condition of this operation ensures that the aircraft must be a registered aircraft, it does not belong to taxing aircrafts and it has permit of taxi clearance. The post-condition of this operation includes it to those aircrafts which are in taxing.

```
Taxing(craftin:string)
ext wr GCC:Groundcontroller
    wr Taxiin:seq of string
    rd Aircrafts:set of Aircraft
pre exists a in set Aircrafts & a.callsign = craftin
    and craftin not in set elems Taxiin
and craftin in set elems GCC.issutaxiclec post Taxiin
= Taxiin~ ^ [craftin];
```

3) *Aircaft on Runway:* The pre-condition of this operation ensures that the aircraft is registered aircraft, it does not belong to those aircrafts which are in runway queue and it belongs to those aircraft which are in start departure queue. The post-condition includes it to runway queue list and discarded it from the select for departure queue.

```
AircraftonRunway(craftin:string)
ext wr RunwayQ:seq of string
    wr LC:LocalController
    rd Aircrafts:set of Aircraft
pre exists a in set Aircrafts & a.callsign = craftin
    and craftin not in set elems RunwayQ
    and craftin in set elems LC.SDepartQ
post RunwayQ = RunwayQ~ ^ [craftin]
     and  LC.SDepartQ = tl LC.SDepartQ~;
```

4) *Request for Departure Clearance:* For the request of departure clearance the operation denoted by

RequestDepartureClerance(craftin:string) is defined. The pre-condition ensures that before sending a request for departure clearance the aircraft must be a registered aircraft, also belongs to *RuwayQ* and does not have permission for taxi clearance. The post-condition includes it for the request of departure clearance.

```
RequestDepartureClerance(craftin:string)
ext rd RunwayQ:seq of string
    wr RTdepartclcQ:seq of string
    rd Aircrafts:set of Aircraft
pre exists a in set Aircrafts & a.callsign = craftin
    and craftin not in set elems RTdepartclcQ
    and craftin in set elems RunwayQ
post RTdepartclcQ = RTdepartclcQ~ ^ [craftin];
```

5) *Start Departure:* In this operation the aircraft runs on the runway. The start departure operation is denoted by *Startdeparture(craftin:string).* The pre-condition ensures that the aircraft must be a registered aircraft, it has clearance for departure and there is no aircraft on the runway. In the post-condition the aircraft runs on the runway and the runway is busy.

```
Startdeparture(craftin:string)
ext wr Runsonrunway:string
    wr LC:LocalController
    rd Aircrafts:set of Aircraft
pre exists a in set Aircrafts & a.callsign = craftin
    and craftin in set elems LC.Gdepartclc
    and  Runsonrunway = NIL
post Runsonrunway = craftin
     and Runsonrunway <> NIL;
```

6) *Diversion:* In this operation departure of aircraft is cancelled and diverts it to any safe area.

```
Deviation(craftin:string)
ext wr CancelDepart:set of string
pre true
post CancelDepart=CancelDepart~ union {craftin};
```

7) *Recovered Plan:* In this operation the canceled departure is planed again for departure, this is the aircraft whose departure is canceled.

```
Recoverdplan(craftin:string)
ext wr CancelDepart:set of string
    wr Recoverplan: seq of string
pre craftin in set CancelDepart
post Recoverplan = Recoverplan~ ^[craftin]
     and CancelDepart = tl CancelDepart~;
```

8) *Final Departure:* Final departure procedure is described in this operation.

```
Finaldepart(craftin:string)
ext wr Recoverplan: seq of string
    wr FinalDept:seq of string
pre craftin in set elems Recoverplan
    or Runsonrunway = craftin
post FinalDept = FinalDept~ ^ [craftin]
     and Recoverplan= tl Recoverplan~;
end AirCraft
```

B. Ground Controller

The ground controller is responsible for the movement of aircraft at ground. It is defined as class *Groundcontroller* and types used in this class are *string* and *Date.*

```
class Groundcontroller
types
public string = seq of char;
public Date=token;
```

The instance variables used in this specification class are given below, where *"AC"* and *"LC"* are the objects respectively of the Aircraft and local controller which allows accessing all the instance variables of these in this class.

```
instance variables
public Deplist:seq of string;
public RTRunwayQ:seq of string;
AC:AirCraft;
LC:LocalController;
public craft_id:string;
public ArngDepQ:seq of string;
public issutaxiclec : seq of string;
```

1) *Issue Departure Information List:* To issue the departure information list the operation denoted by *issuDepInfolist (callsign: string, d_time: Date, route: string, destination: string)* is defined. The pre-condition ensures that prior to this aircraft must reside in taxiway queue and departure information list is not assigned to it. In the post-condition information (callsign, departure time, rout and destination) of aircraft is added to departure list.

```
operations
issuDepInfolist(craftin:string,callsign:string,d_t
ime:Date,route:
string,destination:string)
ext wr Deplist:seq of string
    wr AC:AirCraft
pre craftin in set elems AC.TaxiwayQ
  and craftin not in set elems Deplist
post Deplist =Deplist~ ^
[callsign,d_time,route,destination];
```

2) *Check the Aircraft is in Taxiway:* This operation returns the status of the craft weather it is on taxiway or not, if the craft is on taxiway then it returns true else false

```
isaircraftonTaxiway(craftin:string)Query:bool
ext wr AC:AirCraft
pre true
post Query <=> craftin in set elems AC.TaxiwayQ;
```

3) *Request to assign runway:* The operation denoted by *RequestToAssignRunway(craftin:string)* is defined for the request to assign runway. Pre-condition is check over this operation which ensures that the aircraft must be a registered aircraft, it does not belong to sequence of those aircrafts which have already been assigned a runway and it must belong to those aircrafts which have sent request for taxi clearance. The post-condition includes it to those aircrafts which have send request for the assignment of runway.

```
RequestToAssignRunway(craftin:string)
ext wr AC:AirCraft
    wr RTRunwayQ:seq of string
pre exists a in set AC.Aircrafts & a.callsign = craftin
    and craftin not in set elems RTRunwayQ
    and craftin in set elems AC.RTTaxiclcQ
post RTRunwayQ = RTRunwayQ~ ^ [craftin];
```

4) *Issue Taxi Clearance:* The operation denoted by *Issutaxiclearance(craftin:string)* is defined so that taxi clearance is granted to aircraft. The pre-condition of this operation ensures that it must be a registered aircraft, it does not belong to those aircrafts which already have taxi clearance and runway is assigned to it. The post-condition ensures taxi clearance.

```
Issutaxiclearance(craftin:string)
ext wr issutaxiclec:seq of string
    wr LC:LocalController
    wr AC:AirCraft
pre exists a in set AC.Aircrafts & a.callsign = craftin
    and craftin not in set elems issutaxiclec
    and craftin in set dom LC.AssignedRunway
post issutaxiclec= issutaxiclec~ ^ [craftin];
```

5) *Arrange Aircraft for Departure:* The operation denoted by *Arrangefordepart (craftin:string)* is defined for the arrangement of aircraft for departure process. The pre-condition of this operation ensures that the aircraft must be a registered aircraft, it does not belongs to departure arrange queue and it has assigned runway. The post-condition of the operation is included the aircraft into arrange departure queue.

```
Arrangefordepart(craftin:string)
ext wr ArngDepQ:seq of string
    wr AC:AirCraft
    wr LC:LocalController
```

```
pre exists a in set AC.Aircrafts & a.callsign = craftin
    and craftin not in set elems ArngDepQ
    and craftin in set dom LC.AssignedRunway
post ArngDepQ=ArngDepQ~ ^ [craftin];
end Groundcontroller
```

C. Local Controller

The local controller is defined as class *LocalController* and string type is also used in this class. Types used in this class are *string*, *date and runway* where *runway* is a composite data type of *Runwayno* and *status*. *Rstatus* is enumerated type and the values *<Available>* and *<NotAvailable>* are called quote types. Similarly the *Takeaction* is also enumerated type and value *<Actionlist>* is quote type.

```
class LocalController
types
  public string=seq of char;
  public Date=token;
       Rstatus=<Available>|<NotAvailable>;
        Takeaction=<Actionlist>;
        Runway::Runwayno:string
                status:Rstatus;
```

The instance variables used in this specification given below, "*AC* "is the object of the Aircraft which allows accessing all the instance variables of the Aircraft and "*GCC*" is the object of the ground controller for accessing the all variables of the ground controller.

```
instance variables
     GCC:Groundcontroller;
     AC:AirCraft;
     Runways:seq of Runway;
     public AssignedRunway:map string to string;
     public SDepartQ:seq of string;
     public Gdepartclc:seq of string;
```

1) *Functions:* The function denoted by *isavailable (runwayidin: seq of Runway)pos:nat* is modeled which returns the position of that runway whose status is available. *runwayidin* is an input variable of type sequence of *Runway* and *pos* is an output variable of type natural number.

```
functions
isavailable(runwayidin:seq of Runway)pos:nat
pre true
post runwayidin(pos).status = <Available> and forall
i in set {1,...,pos-1} & runwayidin(i).status <>
<Available>;
```

2) *Update Pre-Departure Information List:* For updating pre-departure information list, the aircraft must have assigned the departure list, i.e., it belongs to the Deplist then the departure list is updated, i.e., new departure time, new route and destination will be assigned to craft.

```
operations
UpdatePreDepartureInfo(callsign:string,d_time:Date
,route:string,destination:string)
  ext wr GCC:Groundcontroller
pre callsign in set elems GCC.Deplist
post GCC.Deplist = GCC.Deplist ++
{1|->callsign,2|->d_time,3|->route,4|->destination
};
```

3) *Assign Runway to Aircraft:* The operation denoted by *AssignRunway()* is defined to assign the runway to aircraft. The pre-condition ensures that before assigning the runway to aircraft the runway must be available then in the post-condition the runway is assigned to aircraft which is the first one who sends the request for the assignment of runway.

```
AssignRunway()
ext wr AC:AirCraft
  wr GCC:Groundcontroller
  wr AssignedRunway:map string to string
  rd Runways:seq of Runway
pre let pos = isavailable(Runways)
    in pos <> 0
post let pos = isavailable(Runways)
     in AssignedRunway = AssignedRunway~ munion {hd
(GCC.RTRunwayQ) |-> Runways(pos).Runwayno};
```

4) *Select for Departure:* The pre-condition ensures that before the selection of departure the aircraft which is going to be selected must be a registered aircraft, it also belongs to arrange for departure queue and does not belong to select for departure queue. The post-condition select to it for departure and discard it from those aircraft which are in arrange departure queue.

```
SelectforDepart(craftin:string)
ext wr SDepartQ:seq of string
    wr GCC:Groundcontroller
    wr AC:AirCraft
pre exists a in set AC.Aircrafts & a.callsign = craftin
    and craftin in set elems GCC.ArngDepQ
    and craftin not in set elems SDepartQ
post SDepartQ=SDepartQ~ ^[craftin]
     and GCC.ArngDepQ=tl GCC.ArngDepQ~;
```

5) *Grant Departure Clearance:* Departure clearance is granted in the operation which is denoted by *GrantDepartureclearence(craftin:string).* The pre-condition is a constraint on this operation that checked that aircraft which is going to take permission should be a registered aircraft, does not have permission but has sent request for departure clearance. The post-condition grant permission to aircraft for departure and this aircraft is discarded from the list of those aircraft which have sent request for departure clearance.

```
GrantDepartureclearence(craftin:string)
ext wr AC:AirCraft
    wr Gdepartclc:seq of string
pre exists a in set AC.Aircrafts & a.callsign = craftin
    and craftin not in set elems Gdepartclc
    and craftin in set elems AC.RTdepartclcQ
post Gdepartclc = Gdepartclc~ ^[craftin]
     and AC.RTdepartclcQ= tl AC.RTdepartclcQ~;
```

6) *Check Emergency Condition:* In this operation emergency situation is checked if any type of emergency is found then new action is performed by aircraft this action is provided by action list that what will be done in this emergency situation.

```
EmergencyCondition(craftin:string)Newaction:Takeac
tion
ext wr AC:AirCraft
pre true
post exists a in set AC.Aircrafts & a.callsign =
craftin
     and Newaction = <Actionlist>;
end LocalController
```

IV. Conclusion

The use of formal methods in the industrial area is increasing rapidly, especially in the critical and complex domain of systems. Formal methods are standard modeling techniques for systems engineers to analyze, specify, design and verify complex systems through mathematical notations that can be used to specify system requirements. Formal methods provide the facility to investigate the issues at early stages of the systems development. It enhances and improves the quality of system removes the semantic gape between the system design and implementation. The air traffic control system is a safety critical system where we have strong need of dependability, avoidance of mistakes and detection of errors and define the limits. In this research we formalized the departure process of air traffic control system which increases the trust of formal methods in applying the safety critical systems. This specification is written in VDM++ which is formal method's approach with an object oriented feature. This specification is unambiguous and reliable. This is also semantically and syntactically true and proved by VDM++ toolset.

REFERENCES

[1] A. Hussy, D. Leadbetter, P. Lindsay, A. Neal, and M. Humphres, "Modeling and hazard identification in air traffic control user-interface," Software Verification Research Center, Technical report no. 0014. Queensland University, April, 2000.

[2] D. W. Christopher, S. M. Anne, P. Raja, and P. M. James, *The future of air traffic control human operators and automation*, National academy press, 1998.

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

[4] N. G. Leveson and C. S. Turner, "An investigation of the therac-25 accidents," vol. 26, no. 7, July. pp. 18-41, IEEE, 1993.

[5] J. L. Lyons, ARINAE 5 Flight 501 failure report by the inquiry board, July 1996.

[6] P. G. Neumann, "Risks digest forum on risks to the public in computers and related systems," vol. 27, no. 1, pp. 7-17, *ACM*., 2002.

[7] P. G. Neumann, "Risks to the public computers and related system," ACM SIGSOFT software engineering note, ACM., 2000.

[8] A. Nadeem, "Automated testing of object oriented systems using VDM++ and UML communication diagrams," Ph.D. Thesis, 2007.

[9] D. Kiper and J. E. Tomayko, "Techniques for safety critical software development," *Proceedings of the thirty-first annual hawaii international conference on system sciences*, vol. 3, IEEE Computer Society. 1998.

[10] E. W. Dijkstra, "A constructive approach to the problem of program correctness," *BIT numerical mathematics*, vol. 8, no. 3, pp. 174-186.

[11] R. W. Floyd, "Assigning meaning to programs," in *Proceedings of the symposium in applied mathematics*, vol. 19, pp. 19-37, 1968.

[12] C. A. R .Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*. vol. 12, no. 10, pp. 576 -580, ACM., 1969.

[13] B. J. Raymond, "Use of formal methods in the development of safety critical control software," Ph.D. Thesis, 2002.

[14] D. Craigen, S. Gerhart, T. Ralston, and K. Summerskill, "An international survey of industrial applications of formal methods," vol. 2, 1993.

[15] K. Ralf, "Limits of formal methods," *Formal aspects of computing*, vol. 9, no. 4, pp. 379-394, Springer, 1997.

[16] E. Durr, K. J. Van and S. Goldsack, "Using VDM++ in the development of a large industrial application," in *Proceedings of the International Symposium of on-board real-time software,* 1996.

[17] J. C. Kelly, "Formal Methods specification and verification guidebook for software and computer System," NASA. vol. 1. 1998.

[18] L. Zhang and L. Wang. "Aspect-oriented formal specification for multimedia systems," in *Proceedings of the 2008 IEEE/ACS international conference on computer systems and applications*, pp. 260-267, 2008.