Zafar *Complex Adapt Syst Model* (2016) 4:4
DOI 10.1186/s40294-016-0014-y

Complex Adaptive Systems Modeling
a SpringerOpen Journal

RESEARCH Open Access

Formal specification and analysis of take-off procedure using VDM-SL

Nazir Ahmad Zafar*

*Correspondence:
nazafar@gmail.com
Department of Computer Science, COMSATS Institute of Information Technology, Sahiwal Campus, Sahiwal 57000, Pakistan

Abstract

Purpose: Air traffic management system is a complex adaptive and safety critical system which requires considerable attention for its modelling and verification. Currently Air traffic control (ATC) systems are heavily dependent upon human intervention at airport causing accidents and delays because of failure of communication. The purpose of this study is to develop, plan, manage and verify aircrafts movement procedures at the airport surface that prevent delays and collisions.

Methods: The airport surface is decomposed into blocks and represented by the graph relation. The state space of the system is described by identifying all the possible components of the system. The ground and local controls monitor queues of the aircrafts moving from taxiway to take-off. It is insured that once an aircraft is inserted into a queue, it is eventually removed from it after the next queue has become available. The take-off procedure is provided using graph theory and Vienna Development Method Specification Language (VDM-SL) and analyzed using VDM-SL toolbox.

Results: Formal specification of graph-based model, taxiways, aircrafts, runways and controllers is provided in static part of the model. The state space analysis describing take-off algorithms is provided by defining optimal paths and possible operations in dynamic model expediting the departure procedure. The model is developed by a series of refinements following the stepwise development approach.

Conclusions: The delays at airport surface require effective safety and guidance protocols to control air traffic at the airport. In static model, the safety criteria are described in terms of invariants over the data types carrying critical information. The safety is insured by defining pre/post conditions in description of operations for changing state space of the system. Although the proposed study is focussed more on the safety component, however, the efficiency is not ignored.

Keywords: Air traffic control, Modelling, Graph theory, Formal analysis, VDM-SL, Validation

Background

Introduction

Air traffic control (ATC) system is highly a safety critical system because its failure may cause a huge loss in terms of deaths or financial losses. Air traffic management system (ATFM) is a complex adaptive system more precisely an example of a complex socio-technical system. This is because each airport comprises of interactions between a

variety of facilities including, technical systems, users, humans, multiple airlines, policies, rules and procedure which are embedded in a large network of other airports. The system is characterized by a large number of interconnected parts for which it is difficult to predict the behaviour and the existence of many different stakeholders. Because of a large increase in movement of population and consequently a significant increase in air traffic, next generation ATC systems are suggested to improve efficiency by not compromising at existing safety standards (Erzberger 2006). Although partial support to automated ATC system is available, however, still it is heavily dependent upon human interaction causing accidents and delays because of failure of communication (Shorrock and Kirwan 2002). Therefore, developing an automated ATC system enabling aircrafts to move safely and freely in air and at airport is globally a challenging task (Debbache 2001). Further, we believe that modeling of an ATC system is an open research area because of its complexity and increasing demand due to increasing trend in the population. It is surprising to note that the airports have historically been more dangerous than the airspace in terms of collisions. For example, the number of collisions occurred at airport surface are three times larger than the number of collisions occurred in the airspace (SCSK 2013). It is noted that a small change in ATFM may have a large impact on the overall ATC and management system at the airport. Further, the quantity of fuel burned is proportional to the waiting time of an aircraft at the airport increasing emissions significantly at the airport in terms of various pollutants including greenhouse gasses (Marshall and Joseph 1992). Therefore, we require an effective and automated monitoring and guiding mechanism to expedite and control air traffic at the airports. Consequently, modelling and development of safe and efficient ATC system being highly safety critical in nature has raised various research questions. For example:

- How can we build procedures at the airport surface that prevent delays and collisions while at the same time meeting the increased demand of air traffic?
- How can we develop, plan, manage and optimize the routes for in air traffic by coordinating among all the local and ground controllers for the safe and efficient operation of the entire system?
- How can we design algorithms that can account for the various requirements, for example, air lines priorities, weather conditions, runtime changes to expedite the takeoff and landing procedures?
- How can we develop an automated, embedded, networked IT-based solution with complete monitoring and guidance system by benefiting from the sensing and acting technologies to coordinate among all the stakeholders and decision makers?

This paper is focussed to address first two questions partly by considering airport model, aircrafts, controllers and other necessary sub-systems which are required to complete the formal model. There are two main sub-systems, i.e., ground and local controllers at an airport for air traffic management which are very less focussed by the scientific community. Functionality of ground controller is to share information with other sub-systems, for example, to define priorities among various operators and to provide an active decision support functions for route predictions. Local controllers are deployed for executing processes of aircrafts waiting at the airport surface for runway, taking off,

landing or flying over the airport. Aircrafts are allowed to push back whenever are ready and the runway capacity is constrained. The aircrafts can take off only one at a time from a single runway while many aircrafts are waiting in the queue. For this purpose safe and efficient computer models are required to be developed to improve pushback and taxiing algorithms to expedite the take off procedure.

In this paper, formal procedure of managing air traffic at airport from taxiing to take-off is provided using graph theory and VDM-SL. The detailed information, for example, wind speed and direction, aircraft type, aircraft weight, weather conditions which may change a runway configuration, in reality are not considered in defining the take-off procedure. Such simplifications are made because our objective is to describe a simple and abstract model which can be applied to any real world ATC system after refinement. The VDM-SL is applied because of its detailed descriptive power and rigorous computer tool (SCSK 2013). The VDM-SL toolbox (SCSK 2013) is used for analysing properties of ATC system in terms of invariants over the data types and pre/post conditions over the operations.

The airport surface is represented by a graph relation. Taxiways are represented as paths by adding more information about state space of the system in VDM-SL. Formal specification of permissible aircrafts and runways is described as mappings. Main possible operations are defined to describe taxiing to take-off procedure. The safety criteria are described in terms of invariants over the data types carrying the critical information. The pre/post conditions of operations are verified for consistency and correctness. The model is based on next generation ATC systems which are used to shift from traditional ground stations to modern navigation systems (Erzberger and Heere 2009). The preliminary results were presented in (Yousaf et al. 2012) in which arrival and departure procedures were described using VDM++. The model was improved by linking graph theory and Z notation at an abstract level of specification. The improved model is different from the existing work (Zafar 2014) due to various reasons. For example, the proposed model is modified based on the errors identified in the existing work. Further, few unnecessary queues were defined in the take-off procedure which are removed to increase simplicity of the model. Few assumptions are reviewed in functionality of ground and local controllers as in real ATC systems. For example, it is supposed that taxiing aircraft is under both the ground and local controllers. The same assumption was taken in (Zafar 2014) but it was not described correctly in the formal specification. The model is refined in depth and is near to implementation now for a realistic ATC system. Finally, a detailed model is described using VDM-SL, results are visualized by validation techniques available in VDM-SL toolbox to increase a confidence. Rest of the paper is organized as: An introduction to modelling complex systems is presented in next subsection. Problem statement and methods are given in section "Methods". Formal description and analysis of the algorithm is described in section "Results and discussion". The relevant work is discussed critically in section "Related work". Conclusion and future work are discussed in section "Conclusions".

Modelling complex adaptive systems

Large scale socio-technical systems such as industrial networks, financial systems, energy systems, environmental systems, infrastructures, democratic systems, exist

everywhere in our society, which are causing a resource and climate crisis. We desperately need to transform our production and consumption patterns by investigating transformation procedures leading to a sustainable development at the globe. Above large scale systems and the ecosystems in which they exist are known to be the complex adaptive systems (CAS). In another definition, CAS are a dynamic network of many agents, for example, cells, species, individuals, firms, nations, acting in parallel, constantly acting and reacting to what the other agents are doing (Holland 2006). Many natural systems, for example, brains, immune systems, ecologies, societies, are characterized by apparently complex behaviour having a large number of components are CAS. Parallel and distributed computing systems, artificial intelligence systems, artificial neural networks, evolutionary programs are examples of artificial CAS. The control of CAS is highly dispersed and decentralized which needs to have a coherent behaviour (Armano and Javarone 2013). The overall behavior of the system is achieved by a huge number of decisions made by many individual agents in every moment in competition and cooperation among the agents themselves (Boulaire et al. 2015). The analysis of CAS can be done by a combination of applied, theoretical and experimental methods, for example, mathematical modelling and computer-based simulation.

Complex Adaptive Systems can be characterized by various properties as discussed below.

The first one property is emergence which represents that the agents interact apparently in a random way rather than in a controlled or planned way showing the behaviour of the system in general and behaviour of the agents within the system. Co-evolution is another interesting property which means that all the systems exist within their own environment and are part of the environment as well. The change of system changes the environment and vice versa as required. Further, a complex adaptive system does not have to be optimal and there is always a trade off among efficiency and effectiveness introducing a suboptimal property. Ambiguity and inconsistency in CAS introduce a property named, variety, because of contradictions to create new possibilities evolved in the environment. The relationships between the agents are more important, in general, than the agents themselves that is why connectivity is a critical property to the survival of a system. Complex Adaptive Systems are not complicated and have emerging patterns governed by the simple rules and principles. A small change in the initial conditions of the system can have significant impact introducing iteration property that is in fact a feedback loop after a periodical time intervals. Self organisation is an important property of CAS as there is no hierarchy, command, planning or managing of control but there is a periodical self organisation of the agents to find the best behaviour with the environment. Most of the systems are nested within the other systems renaming CAS as the nested system. Considering the above characteristics, we can imagine that CAS are all around us. Most importantly, CAS are a model for thinking about the real world phenomena not a model for predicting real world behaviours (Armano and Javarone 2013). Because of the complex nature of the CAS, it requires rigorous tools which are complex themselves to create and understand such systems.

Complex Adaptive Systems are investigated through the use of computer simulations tools including agent-based methodologies and complex networks which are being widely appreciated in the natural and social sciences. Agent-based methodologies (ABM)

has caused a revolution in the area of computer modelling and simulation as we can develop models to analyse, test, predict and replicate the behaviour of complex systems. ABM simulates CAS from the detailed level representing the actions and interactions of various individual agents in the artificial world. Agent-based models are models representing computer code, recursive mathematical rules, applied to a given well-defined set of inputs. ABM models are not restricted to represent general statistical models or driving equations of a system but can represent explicitly the micro interactions and patterns of behaviour of the agents (Boulaire et al. 2015). Statistical and mathematical analysis techniques are still important as they play a critical role in developing and testing the ABM. Agent based modelling is a persistent approach consisting of collection of states, rules and agents which interacts with each other mutually modifying their states by following the certain rules. The agents in CAS are components of the system. The air and water molecules in a weather system, and flora and fauna in the ecosystem are examples of agents in whether and ecosystems. The agents in CAS interact and connect with each other in unpredictable and unplanned manners.

Complex networks are effective models to describe CAS such as biological systems, neural networks, social systems, chemical systems, and the World Wide Web which are few examples of the systems those are composed by a large number of interconnected dynamical systems. Initially, such systems can be described by graphs whose nodes represent the agents, and edges represent the relationship between the agents. Complex networks have a support to cope with the structural properties of the CAS by defining topology of the system and then describing rules that govern the behaviour of the agents. There are many questions which arise when studying and applying complex networks, for example, how to model large and dynamical systems which interact through a complex topology to behave collectively.

Both ABM and complex networks modelling are based on testing and simulation techniques for verification of the CAS. Unfortunately the testing and simulation techniques are lacking of in verifying the correctness of the systems. The number of testing and simulations in such systems increases exponentially to achieve a required level of confidence due to their complexity. Further, when an enhancement to the system is required regression testing will be needed to perform which recommends that the complete set of simulations must be re-performed. Formal specification and analysis techniques help to ensure that the models and simulations are correct and reliable to overcome the disadvantages of testing and simulation (North 2014). Therefore, it is required to apply formal approaches which provide an exhaustive support for verification of algorithms before the simulation.

Methods

It is supposed that ground control monitors aircrafts moving from gate to taxiway. The local control is responsible for taxiing to runway and take-off. After reaching the runway, aircraft is put into the queue and then takes-off after the final permission. As the configuration of runways for arriving and departing aircrafts is dependent on the airport, that is why, such issues are out of scope of this research. The detailed information in modeling of the ATC system, for example, aircraft speed, height, location, weight and type, weather conditions, wind speed and direction which may change a runway configuration,

in reality are not considered. This is because the models can be refined by providing such details in the further refinement of the model. The extension of the model will be easier as we have used composite type structure in the model which supports extension of the model. The airport surface is divided into blocks which are nodes of the graph relation. In reality, the topology must be a weighted graph in which an edge may represent time to move from one node to another. We have supposed an un-weighted graph to describe an abstract and simple model. Further, a loop-free route is assumed in the formal specification. On the other hand, if a loop is in a scenario because of cancellation of a flight or due to any other reason, human intervention is supposed to resolve this issue. The objective is to find and assign optimal routes with minimum delays meeting the real safety standards by defining a set of queues and sequence of operations. Once an aircraft is inserted into a queue, it should eventually be removed from the queue after the next queue becomes available. In other words, the formal system does not allow any situation where an aircraft is inserted into a queue, and never removed from that queue. A semi-formal procedure for the queue management is presented in Table 1. The second column of the Table is used for queue description. The columns 3 and 4 of the Table show that once an aircraft is inserted into a queue, ultimately it is removed from the queue as soon as possible.

The VDM-SL is used for the formal specification which is supported by the VDM-SL Toolbox. The tool contains various facilities, for example, a syntax checker, a static semantics checker, an interpreter and a code generator to C++. As VDM-SL is a non-executable language in general that is why the interpreter supports only a subset of the language.

Results and discussion

Formal model using VDM-SL

Formal specification of the algorithm is described in this section to achieve the objectives of expediting the traffic flow management for safe and efficient operation of the ATC system.

Static model

The static model of the ATC system consists of ground and local controllers which are further analysed to describe airport topology, taxiways, runways and aircrafts. A smallest unit *Block* of the airport surface is represented by the *token* type which is in fact node in the airport topology. The token type consists of a countable infinite set of distinct

Table 1 Aircrafts Queues Management

#	Queue description	Queue management	
		Inserted into	Removed from
1	Taxiway permission	taxiwayPermission	–
2	Taxiway assigned	taxiwaysAssigned	taxiwayPermission
3	Taxiing	Taxiing	taxiwaysAssigned
4	Runway permission	runwayPermission	Taxiing
5	Runway assign	runwaysAssigned	runwayPermission
6	On runway	Onrunway	runwaysAssigned
7	Take-off	–	Onrunway

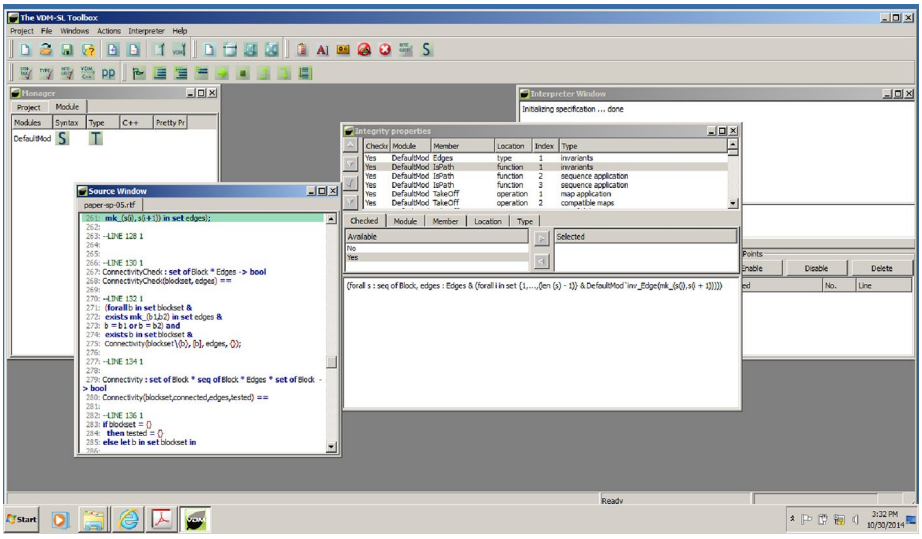

Fig. 1 Snapshot of specification using VDM-SL tool

values termed as tokens. The only operations that can be performed on tokens are equality and inequality. The connectivity of two blocks is represented by an *Edge* which is a product type. The values of a product type are called tuples. A tuple is a fixed length list in which ith element of the tuple must belong to the ith element of the product type. As the block is supposed enough smallest, it is supposed that no block is connected to itself in the definition of edge. The set of all edges of the topology is represented as *Edges*. A set is an unordered collection of values and all are of the same type. All sets in VDM-SL are assumed as finite that means it contains only a finite number of elements. The elements of a set type can be complex, for example, can be relation, functions or sets themselves. If the data types contain values which should not be allowed then it is possible to restrict the values in a type by means of an invariant. In the invariants, it is stated that *Edges* relation is symmetric. It is supposed that a *Block* is connected with, at most, four blocks of the topology.

```
Block = token;

Edge = Block * Block

    inv edge ==

        let mk_(b1,b2) = edge in

        b1 <> b2 ;

Edges = set of Edge

    inv edges ==

        (forall mk_(b1,b2) in set edges &

        mk_(b2,b1) in set edges) and

        (forall mk_(b,-) in set edges &

        card {b1|mk_(b1,b2) in set edges & b2=b}<=4);
```

The topology is defined by a composite type which consists of two components namely blocks and edges.

Composite types are similar to record types in programming languages. The elements of composite type are almost similar to that of tuples that is product types. The difference between the composite type and the product type is that the different components of a composite type can be directly selected by means of corresponding selector functions whereas product type elements are selected by indices. Another difference is that a tuple must have at least two entries whereas records can be empty. An ordered pair (b1, b2) in the set of edges means that an aircraft can move from block b1 to the block b2.

```
Topology :: blocks : set of Block
            edges : Edges
    inv top ==
       (forall mk_(b1,b2) in set top.edges &
       b1 in set top.blocks and b2 in set top.blocks) and
       (forall b in set top.blocks &
       exists mk_(b1,b2) in set top.edges & b=b1 or b=b2) and
       ConnectivityCheck(top.blocks, top.edges);
```

Invariants

- It is stated that both blocks of an edge in the topology are in the set of blocks.
- For any block of the topology, there is an edge for which the block is an end point.
- The *ConnectivityCheck* function evaluates true if the set of blocks form a path in the topology.

A function type is used in VDM-SL which has a type from a type A to a type B that associates with each element of A to a unique element of B. A function value can be thought similar to a function in the programming languages. The *ConnectivityCheck* function is used to check that the blocks occupied by an aircraft generate a path at the topology. The function takes two arguments, i.e., set of blocks and set of edges and verifies the property.

```
ConnectivityCheck : set of Block * Edges -> bool
ConnectivityCheck(blockset,edges) ==
       (forall b in set blockset &
       exists mk_(b1,b2) in set edges &
       b = b1 or b = b2) and
       exists b in set blockset &
       Connectivity(blockset\{b},[b],edges,{});
```

The connectivity function checks if a given set of blocks forms a path. An arbitrary block is selected from the list of ordered blocks treated so far. For each of the remaining blocks, it is investigated after adding a block to the list at appropriate place, whether the list forms a path. The block can be added in the head or tail of the list. When no block is added to the list, it is investigated whether all the blocks have been added to the list. If this is the case, the set forms a path otherwise it does not generate any path.

```
Connectivity : set of Block * seq of Block * Edges * set of Block  -> bool

Connectivity(blockset,connected,edges,tested) ==

      if blockset = {}

        then tested = {}

      else let b in set blockset in

        if mk_(b, hd connected) in set edges

        then Connectivity(blockset\{b} union tested,[b]^connected,edges,{})

          else if mk_(connected(len connected), b) in set edges

        then Connectivity(blockset\{b} union tested,connected^[b],edges,{})

        else Connectivity(blockset\{b},connected,edges,tested union {b});
```

Taxiway consists of identifier, path, priority and state. The quote type in VDM-SL corresponds to enumerated type in Pascal. *Priority* is defined as type construction that allows enumerated types which is union of *LOW* and *HIGH*. Similarly, *State* of a taxiway is union of *CLEAR* and *OCCUPIED*.

```
Taxiway:: taxiwayid: TaxiwayId

          taxiway: seq of Block

          priority: Priority

          taxistate: State;

TaxiwayId = Block;

Priority = <HIGH>|<LOW>;

State = <CLEAR>|<OCCUPIED>;
```

The *Taxiways* is defined as a composite type which consists of four components, i.e., *taxiways*, *optimals*, *suboptimals* and *edges*. The *taxiways* is defined as a mapping from taxiway identifier to *Taxiway*. A map type from a type A to a type B is a type that associates with each element of A to a unique element of B. A map value can be assumed as an unordered collection of pairs in which the first element is a key which is used to get the second element of the pair. The set of key elements is called domain while the set of all information values is called range of the map. A sequence is an ordered collection of elements of the same type. A sequence type is the type of finite sequences of elements of a type which may include an empty sequence. Optimal and sub-optimal taxiways are defined as sequence types.

```
Taxiways:: taxiways: map TaxiwayId to Taxiway
           optimals: seq of Taxiway
           suboptimals: seq of Taxiway
           edges: Edges
 inv tws ==
 forall tid in set dom tws.taxiways & tws.taxiways(tid).taxiwayid = tid and
 forall op in set elems tws.optimals & IsPath (op.taxiway, tws.edges) and
 forall op in set elems tws.optimals & op.priority = <HIGH> and
 forall so in set elems tws.suboptimals & IsPath (so.taxiway,tws.edges) and
 forall so in set elems tws.suboptimals & so.priority = <LOW> and
 elems tws.optimals inter elems tws.suboptimals={} and
 rng tws.taxiways = elems tws.optimals inter elems tws.suboptimals;
```

Invariants

- An identifier in the domain of *taxiways* mapping is same as in the *Taxiway* composite data type.
- An optimal taxiway forms a path in the topology which is verified by function *IsPath* given below.
- The optimal taxiways are for high priority aircrafts
- A suboptimal taxiway forms a path in the topology which is verified similar to the optimals taxiways.
- The suboptimal taxiways are of low priority.
- As a taxiway is optimal or suboptimal therefor the intersection of both types of taxiways is empty.
- The set of all the taxiways of the airport is union of optimal and suboptimal taxiways.

The *IsPath* function takes sequence of blocks and edges as an input and checks if the sequence forms a path in the edge relation. It is stated that any two consecutive elements in the path sequence constitute an edge in the graph relation.

```
IsPath : seq of Block * Edges -> bool
IsPath(s, edges) ==
       (forall i in set {1,...,(len s-1)} &
       mk_(s(i),s(i+1)) in set edges);
```

Runway consists of runway identifier, priority and its state. The set of runways is defined by a mapping from runway identifier to *Runway*. In the invariants, it is stated that an identifier element in the domain of the *runways* mapping is in the *Runway* data type.

```
Runway:: runwayid: RunwayId
         priority: Priority
         rwaystate: State;
RunwayId = Block;
Runways = map RunwayId to Runway
    inv rws ==
        forall rwid in set dom rws & rws(rwid).runwayid = rwid;
```

An aircraft consists of identifier, route and priority. The allocated *route* is optimal or suboptimal based on the priority. In the invariants, it is stated that a route is always non-empty. The state of the path is assumed as occupied. Finally, if the priority is high then route is optimal otherwise it is sub-optimal.

```
Aircraft:: aircraftid: AircraftId
           route: Taxiway
           priority: Priority
    inv acft==
        acft.route.taxiway <> [] and
        acft.route.taxistate = <OCCUPIED> and
        acft.route.priority = <HIGH> <=> acft.priority = <HIGH> and
        acft.route.priority = <LOW> <=> acft.priority = <LOW>;
AircraftId = token;
```

The *Aircrafts* consists of *aircrafts* and *edges*. The *aircratfs* is a mapping from aircraft identifier to *aircraft*. In the invariants, it is stated that an identifier in the domain of aircrafts is in the *Aircraft*. For every aircraft in the range of *aircrafts* mapping, its route is well-defined path in the airport topology.

```
Aircrafts:: aircrafts : map AircraftId to Aircraft
            edges: Edges
    inv acfts ==
        forall acid in set dom acfts.aircrafts &
          acfts.aircrafts(acid).aircraftid = acid and
        forall acid in set dom acfts.aircrafts & IsPath
        (acfts.aircrafts(acid).route.taxiway, acfts.edges);
```

Ground control is used for monitoring and guiding aircrafts moving from gate to enter taxiway. The ground controller consists of *aircrafts, taxiways*, *taxiwayPermission*, *taxiwaysAssigned* and *taxiing*. The *taxiwayPermission* is a sequence type representing aircrafts having permission for taxiing. The *taxiwaysAssigned* is a mapping to record

aircrafts which are assigned taxiways. The *taxiing* mapping is to record information about aircrafts which are on the taxiways.

```
GroundController:: aircrafts: Aircrafts
                   taxiways: Taxiways
                   taxiwayPermission: seq of AircraftId
                   taxiwaysAssigned: map AircraftId to TaxiwayId
                   taxiing: map AircraftId to Taxiway
inv gc ==
         forall aid1 in set elems gc.taxiwayPermission &
         exists aid2 in set dom gc.aircrafts.aircrafts & aid1 = aid2 and
         forall aid in set dom gc.taxiwaysAssigned &
         exists tid in set dom gc.taxiways.taxiways &
           gc.taxiwaysAssigned(aid) = tid and
         len gc.taxiwayPermission <=L1 and
         forall i, j in set inds gc.taxiwayPermission &
         i<> j=> gc.taxiwayPermission(i)<> gc.taxiwayPermission(j) and
         dom gc.taxiwaysAssigned inter elems gc.taxiwayPermission={} and
         card dom gc.taxiwaysAssigned < L2 and
         dom gc.taxiing inter dom gc.taxiwaysAssigned = {} and
         elems gc.taxiwayPermission inter dom gc.taxiing = {} and
         card dom gc.taxiing < L3;
L1,L2,L3:int = 10;
```

Invariants: (i) An aircraft having permission for taxiing is in the set of aircrafts in ground controller. (ii) Any taxiway allotted to an aircraft for taxiing is in the set of available taxiways. (iii) The total number of aircrafts in the permission queue should not be greater than the permissible limit. (iv) Any two aircrafts in the queue having permissions for taxiing are distinct. (v) The intersection of elements of queue having permission and which are assigned taxiways is empty. (vi) The total number of aircrafts which are assigned taxiways does not exceed the allowed limit. (vii) The intersection of taxiing aircrafts and aircrafts which are assigned taxiways is empty. (viii) The intersection of taxiing aircrafts and aircrafts having permission for taxiing is empty. (ix) The total number of taxiing aircrafts does not exceed the limit.

The local controller consists of *aircrafts*, *runways*, *taxiing*, *runwayPermission*, *runwaysAssigned* and *onrunway*. The *runwayPermission* is used to represent aircrafts in the queue which have permission for take-off. The *runwaysAssigned* is a mapping from aircraft identifier to runway identifier to represent aircrafts which are assigned the runways. The *onrunway* is a mapping from aircraft on the runways.

```
LocalController:: aircrafts: Aircrafts
                  runways: Runways
                  taxiing: map AircraftId to Taxiway
                  runwayPermission: seq of AircraftId
                  runwaysAssigned: map AircraftId to RunwayId
                  onrunway: map AircraftId to Runway
    inv lc ==
         forall aid1 in set elems lc.runwayPermission &
         exists aid2 in set dom lc.aircrafts.aircrafts & aid1 = aid2 and
         forall aid in set dom lc.runwaysAssigned &
         exists rid in set dom lc.runways & lc.runwaysAssigned(aid)=rid and
         card dom lc.taxiing <=L3 and
         dom lc.taxiing subset elems lc.runwayPermission and
         forall i, j in set inds lc.runwayPermission &
         i<> j => lc.runwayPermission(i)<> lc.runwayPermission(j) and
         len lc.runwayPermission <= L4 and
         elems lc.runwayPermission inter dom lc.runwaysAssigned = {} and
         card dom lc.runwaysAssigned <=L5 and
         dom lc.onrunway subset dom lc.runwaysAssigned and
         elems lc.runwayPermission inter dom lc.onrunway = {} and
         card dom lc.onrunway <=L6;
L4,L5,L6:int = 10;
```

Invariants: (i) A runway allotted to an aircraft in the queue having permission is in the set of available runways. (ii) Any aircraft having permission for take-off is in the set of aircrafts allowed at airport. (iii) The total number of taxiing aircrafts does not exceed the limit. (iv) The set of taxiing aircrafts is subset of aircrafts having permissions. (v) Any two aircrafts in the queue having permissions are distinct. (vi) The total number of aircrafts in the permission queue does not exceed its limit. (vii) The intersection of aircrafts having permission for taxiing and aircrafts which are assigned runways is empty. (viii) The total number of aircrafts which are assigned runways does not exceed the allowed limit. (ix) The set of aircrafts which are on runways is subset of the aircrafts assigned the runways. (x) The intersection of aircrafts on runways and aircrafts having permission is empty. (xi) The total number of aircrafts which are on runways does not exceed the permissible limit.

Dynamic model

It is possible to make a state definition if global variables are desired in the formal specification. The components of the state are a collection of global variables which can be accessed inside the state operations. A state in a module is initialised before defining

any of the state operation. Formal specification of the queues management is described below by defining operations over state controllers. A state identifier is declared of a specific type. The invariant inv clause is a Boolean expression denoting a property which must hold for the state variables at all times. The init clause denotes a condition which must hold initially.

For every queue, it is checked if the size of next queue is less than its maximum bound and previous queue is not empty then the first aircraft in the previous is moved to the next queue. In this way, starvation is avoided and efficiency is achieved. Each operation is linked with the previous one by defining the pre-conditions. The correctness of an operation is described in terms of post-conditions. When an aircraft is on a taxiway it will be in the record of both the ground and local controllers. The state variables are initialized following the invariants.

```
state Controllers of
        gcontroller: GroundController
        lcontroller: LocalController
    inv mk_Controllers(gcontroller, lcontroller) ==
         forall aid in set dom gcontroller.taxiing &
         aid in set dom lcontroller.taxiing and
         forall aid in set dom lcontroller.taxiing &
         aid in set dom gcontroller.taxiing and
         forall aid in set dom gcontroller.taxiwaysAssigned &
         aid not in set dom lcontroller.runwaysAssigned and
         forall aid in set dom lcontroller.runwaysAssigned &
         aid not in set dom gcontroller.taxiwaysAssigned and
         dom gcontroller.taxiing inter dom lcontroller.onrunway = {}
init conts ==
  conts = mk_Controllers(mk_GroundController(mk_Aircrafts({|->},{}),
          mk_Taxiways({|->},[],[ ],{}),[],{|->},{|->}),
          mk_LocalController(mk_Aircrafts({|->},{}),{|->},{|->},[],
          {|->},{|->}))
end
```

An explicit operation consists of a statement using a block statement. The statement may access any state variables as required, that is, for reading or writing. An implicit operation is described using an optional pre-condition, and a mandatory post-condition. In the operation given below, an aircraft sends a request to ground controller, the controller accepts request after verification of its identity and adds in the list of aircrafts having permission for taxiing. The procedure is described by an implicit operation represented by *taxiwayPermission* which takes aircraft identifier as input and state of the controller is updated by defining its post conditions before checking pre conditions. In

the specification, external clause is introduced by keyword ext. Its purpose is to restrict access of the operation to only those components which are specified. Further, its purpose is to specify the mode of access, that is, either read or write.

```
TaxiwayPermission(aid: AircraftId)
     ext wr gcontroller: GroundController
 pre aid in set dom gcontroller.aircrafts.aircrafts and
     aid not in set elems gcontroller.taxiwayPermission and
     len gcontroller.taxiwayPermission < L1
 post gcontroller.taxiwayPermission = gcontroller.taxiwayPermission^[aid];
```

Pre-conditions (i) The aircraft must be in the set of aircrafts under the ground controller to verify that the aircraft is known to the system. (ii) The aircraft is not already in the list of aircrafts having permission for taxiing. (iii) An aircraft is provided the permission if size of the queue having permission is less than the maximum permissible limit.

Post-conditions List of aircrafts having permission for taxiing is updated by sequence concatenation operator by adding the aircraft *aid* at the end of list.

After having permission, taxiway is assigned to the aircraft. The ground controller checks various conditions such as priority, availability of taxiway, size of current queue and then assigns the taxiway to the aircraft using the *TaxiwayAssign* operation.

```
TaxiwayAssign(aid: AircraftId)
     ext wr gcontroller: GroundController
 pre aid in set dom gcontroller.aircrafts.aircrafts and
     aid in set elems gcontroller.taxiwayPermission and
     aid not in set dom gcontroller.taxiwaysAssigned and
     card dom gcontroller.taxiwaysAssigned < L2
  post let acft = gcontroller.aircrafts.aircrafts(aid) in
     (acft.priority=<HIGH> and card dom gcontroller.taxiwaysAssigned<L2)=>
      exists i in set inds gcontroller.taxiways.optimals &
      gcontroller.taxiwaysAssigned = gcontroller.taxiwaysAssigned munion
        {aid |-> gcontroller.taxiways.optimals(i)} and
     (acft.priority=<LOW> and card dom gcontroller.taxiwaysAssigned <L2)=>
      exists i in set inds gcontroller.taxiways.suboptimals &
      gcontroller.taxiwaysAssigned = gcontroller.taxiwaysAssigned munion
        {aid |-> gcontroller.taxiways.optimals(i)} and
      gcontroller.taxiwayPermission = tl gcontroller.taxiwayPermission;
```

Pre-conditions (i) The aircraft is known to the system. (ii) The aircraft is not assigned a taxiway. (iii) There exits a taxiway with clear state. (iv) The aircraft is assigned a taxiway if the queue size does not exceed the maximum allowed limit.

Post-conditions (i) If priority of the aircraft is high, an optimal taxiway with clear state is assigned. (ii) If priority is low, a suboptimal taxiway with clear state is assigned. (iii) The aircraft is removed from the list of aircrafts having permission for taxiing.

An aircraft sends a request to the ground controller for taxiing. If state of the assigned taxiway is clear then the aircraft is allowed for taxiing.

```
Taxiing(aid: AircraftId)
      ext wr gcontroller: GroundController
      wr lcontroller : LocalController
   pre aid in set dom gcontroller.aircrafts.aircrafts and
      aid not in set dom lcontroller.aircrafts.aircrafts and
      aid in set dom gcontroller.taxiwaysAssigned and
      aid not in set dom gcontroller.taxiing and
      card dom gcontroller.taxiing < L3 and
      exists tid in set dom gcontroller.taxiways.taxiways &
      gcontroller.taxiways.taxiways(tid).taxistate = <CLEAR>
   post exists tid in set dom gcontroller.taxiways.taxiways &
      gcontroller.taxiways.taxiways(tid).taxistate = <CLEAR> =>
      let gct = gcontroller.taxiing in
      gct = gct munion {aid|-> gcontroller.taxiways.taxiways(tid)} and
      gcontroller.taxiwaysAssigned={aid}<-:gcontroller.taxiwaysAssigned and
      let lcas=lcontroller.aircrafts.aircrafts in
      lcas=lcas munion {aid|->gcontroller.taxiwaysAssigned(aid)};
```

Pre-conditions (i) The aircraft is under the ground controller. (ii) The aircraft is not under the local controller. (iii) The aircraft is assigned a taxiway. (iv) The aircraft is not already taxiing at any taxiway. (v) The total number of taxiing aircrafts is less than maximum permissible limit. (vi) There exists a taxiway with clear state.

Post-conditions (i) The taxiing mapping in ground controller is updated by munion operator of mapping in VDM-SL. (ii) The aircraft is removed from the list of aircrafts which are assigned taxiways. (iii) The aircrafts is added in the list of aircrafts under local controller. This is because taxiing aircraft must be in the record of both the ground and local controllers.

The runway permission procedure is defined below by the operation *RunwayPermission*. Before an aircraft is given permission for runway, the queue size of permission list is verified.

```
RunwayPermission(aid: AircraftId)
      ext wr gcontroller: GroundController
      wr lcontroller : LocalController
 pre aid in set dom lcontroller.aircrafts.aircrafts and
      aid in set dom lcontroller.taxiing and
      aid not in set elems lcontroller.runwayPermission and
      len lcontroller.runwayPermission < L4
 post lcontroller.runwayPermission = lcontroller.runwayPermission^[aid] and
      gcontroller.taxiing = {aid} <-: gcontroller.taxiing and
      lcontroller.taxiing = {aid} <-: lcontroller.taxiing and
      gcontroller.aircrafts.aircrafts = {aid} <-:
        gcontroller.aircrafts.aircrafts;
```

Pre-conditions (i) The aircraft is under the local controller. (ii) The aircraft is in the queue of taxiing aircrafts. (iii) The aircraft is not already in the list of aircrafts having permission for runway. (iv) The size of the queue having permission for runway is less than the limit.

Post-conditions (i) The list of aircrafts having permission for take-off is updated by adding the aircraft at the end of the list. (ii–iii) The aircraft is removed from the taxiing aircrafts under both the ground and local controllers. (iv) The aircraft is removed from the list of aircrafts under the ground controller.

Runway assigning procedure is described below. It is noted that after leaving taxiway, the aircraft is only under the local controller. That is why in the operation external clause refers to only local controller.

```
RunwayAssign(aid: AircraftId)
      ext wr lcontroller : LocalController
  pre aid in set dom lcontroller.aircrafts.aircrafts and
      aid in set elems lcontroller.runwayPermission and
      aid not in set dom lcontroller.runwaysAssigned and
      card dom lcontroller.runwaysAssigned < L5
  post let acft = lcontroller.aircrafts.aircrafts(aid) in
     (acft.priority=<HIGH> and card dom lcontroller.runwaysAssigned<L5)=>
      exists i in set dom lcontroller.runways &
      lcontroller.runways(i).priority = <HIGH> =>
      lcontroller.runwaysAssigned = lcontroller.runwaysAssigned munion
        {aid |-> lcontroller.runways(i)} and
      (acft.priority=<LOW> and card dom lcontroller.runwaysAssigned<L5)=>
      exists i in set dom lcontroller.runways &
      lcontroller.runways(i).priority = <LOW> =>
      lcontroller.runwaysAssigned = lcontroller.runwaysAssigned munion
        {aid |-> lcontroller.runways(i)} and
      lcontroller.runwayPermission = tl lcontroller.runwayPermission;
```

Pre-conditions (i) The aircraft is known to the system. (ii) The aircraft has permission for runway. (iii) The aircraft is not assigned any runway. (iv) The aircraft is assigned a runway if the queue size does not exceed the limit.

Post-conditions (i) If aircraft priority is high and queue does not exceed the limit, an optimal runway is assigned. (ii) If priority is low and queue does not exceed the limit, a suboptimal runway is assigned. (iii) The aircraft is removed from the list of aircrafts having permission for runway.

After an aircraft is assigned a runway, it is in the queue of aircrafts which are waiting for take-off. Formal description of the take-off procedure is given below.

```
TakeOff(aid: AircraftId)
      ext wr lcontroller : LocalController
  pre aid in set dom lcontroller.aircrafts.aircrafts and
      aid in set dom lcontroller.runwaysAssigned and
      aid not in set dom lcontroller.onrunway and
      card dom lcontroller.onrunway < L6 and
      exists rid in set dom lcontroller.runways &
      lcontroller.runways(rid).rwaystate = <CLEAR>
 post exists rid in set dom lcontroller.runways &
      lcontroller.runways(rid).rwaystate = <CLEAR> =>
      let orw = lcontroller.onrunway in
      orw = orw munion {aid|-> lcontroller.runways(aid)} and
      lcontroller.runwaysAssigned={aid}<-:lcontroller.runwaysAssigned and
      lcontroller.aircrafts.aircrafts={aid}<-:
        lcontroller.aircrafts.aircrafts;
```

Pre-conditions (i–ii) The aircraft is under the local controller and assigned a runway. (iii) The aircraft is not already on runway. (iv) The total number of aircrafts on runway is less than maximum permissible limit. (v) There exists a taxiway which is in the clear state.

Post-conditions (i) If a runway is in clear state then the *onrunway* mapping is updated. (ii–iii) The aircraft is removed from the list and local controller.

Model analysis

We know there does not exist any computer tool which may promise about complete correctness of a computer model. An art of writing formal specification does not provide any guarantee about complete correctness of a model. However, if specification is analysed and supported by rigorous computer tools, it increases confidence by identifying potential errors at the early stages of software development.

The ATC take off procedure is formalized using VDM-SL because it is a formal specification language used both at abstract and detailed level. A model in VDM-SL can provide an understanding between developer and a customer which may help to understand and stabilize the requirements. In this way, a high level of confidence in correctness and conformance of a model can be obtained at early stages of the software development. One of the most important features of VDM-SL is that it can be integrated with existing technologies which is required at current stage of development in formal methods.

Two principal concerns that arise in systems development are validation and verification. Validation addresses whether the system that is produced actually fulfills the requirements. Verification, on the other hand, attempts to establish whether the product of the particular phase of the software process meets the requirements established

during the previous phase. Informally, the difference between validation and verification can be expressed raising the questions.

- Validation: "Are we building the right system?"
- Verification: "Are we building the system right?"

Our experience in developing this model has shown that it is easy to propose a model free of syntax and type errors but to prove that the model represents the required behavior of the system being modeled is not a simple task. Consequently the model must be verified and validated to be an acceptable model of the system. Validation is necessarily an informal process since the user's requirements are informal. On the other hand, verification can be done by formal as well as by informal ways. By validation, we mean the activities which increase the modeller's confidence. Validation is done through animation and testing using interpreter and debugger to increase confidence that the specification reflects the informal description. For this purpose, we developed several scenarios. Aircrafts were introduced and state space of the airport was described. And then analysis of the formal definitions was done using the VDM-SL toolbox by systematic testing.

The model is evaluated using the VDM-SL toolbox to ensure that it complies with the requirements. The VDM-SL toolbox is used to write, develop and analyse formal specification written in VDM-SL as shown in Fig. 1. Syntax and type checking is performed to verify that the specification obeys the rules available in VDM-SL. We developed many contradictory examples for the definitions to prove that our formal definition captured the required behavior. We observed that even a formal definition is checked and analyzed by the toolbox but it does not give guarantee to be correct. As a result, we suggest that analysis should be done by modeler producing counter examples as seriously as the formal specification of a system is described.

Test coverage is another important facility in VDM-SL toolbox by which it is possible to list precisely which parts of the formal specification have not been covered by given a test suite. Although test coverage facility, in VDM-SL toolbox, does not give guarantee about correctness of a model even if 100 % test coverage is achieved, but it increases the confidence in the model as it can be checked which part of the model is not executed. It is mentioned that test suits were developed in such a way full test coverage were achieved. It is to be noted that a single test case cannot produce 100 % test coverage. It is the modeler responsibility to make a combination of test cases by which all the formal specification can be executed. Initially, some inconsistencies were observed in the model and then the model was adjusted after series of refinements until we achieved the required behavior of the system. Dynamic types checking, invariants, pre-conditions and post-conditions are done for run-time errors. The use of VDM-SL toolbox has eased the model development, as we were able to check the specification and thereby could observe the consequences of our definitions.

Related work

Literature review of ATC system

In most relevant, planning function is developed for taxi operation to address uncertainties based on the real data using scenarios-based approach (Koeners et al. 2011).

In another relevant work, a genetic algorithm for minimum cost and maximum-flow is developed and tested to maximize capacity at the surface of an airport (Garcia et al. 2002). Here, only a part of the planning function is evaluated based on simulation to show performance of the algorithm. In (Rademaker and Koeners 2011), taxi time is claimed to be reduced by defining uncertainties in taxi process and aircrafts queue at the runways. In another work (Medina et al. 2010), a model for estimating the ramp congestion delay is described by employing managed gate operation computer tool, however, validation or verification is not provided to prove its correctness. An applicability of Z notation to radar plot processing in the multi-radar automatic tracking system of ATC System is investigated at the London ATC Centre in (Simcox 1989). An informal pseudo code description of the radar plot processing function was converted into a formal specification using Z language. The specification was partly validated using an RSRE Z syntax and type checking tool. In studies conducted by NASA (Michael and Steven 2012), (Yang and Kuchar 1997), collaborative air traffic flow using multi-agents is developed by providing information to controllers and air carriers to manage the airports. Traffic limitations are relaxed and developed by simulating airport surface for aircrafts movement in (Hanh and Hung 2007). The drawback of these approaches is use of simulation that is several strategies were used to select various routes increasing its complexity. In another work, a study was conducted to investigate the applicability of software fault tolerance techniques to ATC systems (Moulding and Smith 1992). Most of the work was concerned with exploring the requirements and to validate for a short term conflict alert ATC function using the combined CORE and VDM approach. A simple case study of ATC system is presented in (Fitzgerald and Larsen 2009) to introduce some structures of VDM-SL. A fusion of intelligent computing is applied for development of ATFM using advantages of the meta-level control approach in (Alves et al. 2008). Probabilistic timed automata and PRISM tool are used to verify and analyse the properties of ATC system (Kwiatkowska et al. 2004). In (Hall 1999), the ATFM in the UK was upgraded to handle increasing capacity of air traffic by developing the Central Control Function for generating and manipulating the sequence of flights inbound to a major airport complex such as Heathrow and Gatwick. The automated support was provided by a number of systems including National Airspace System and Airport Data Information System. Intelligent models are claimed for ATFM as presented in (Weigang et al. 2010). A protocol for aircraft conflict identification and resolution is proposed focussing on communication which is limited (Hwang et al. 2004). Few other protocols for aircraft identification and conflict resolution are proposed in which communication range of an aircraft is finite (Hwang et al. 2003; Hwang and Tomlin 2002). In this work, the communication topology among aircrafts is modelled and represented by communication graph. A protocol for multiple-aircraft conflict resolution is developed to show the safety of the protocol and validated through simulations with a dynamic aircraft model. An automated approach for modelling and analysis of complex Air Traffic Organization (ATO) is introduced in (Sharpanskykh 2009). The model addresses all the important structural and behavioural aspects of an ATO illustrated by a case study considering movement of aircraft on the ground with safety measures. A predictable system is developed to choose an optimal path by minimizing fuel consumption and delay time rather than using predefined flight schedules (Bousson 2003), (Kahne and Frolow 1996). The performance of

conflict resolution is presented in (Kuchar and Yang 2000). Aircrafts departure procedure is given using activity diagram by focussing on requirements analysis (Amy et al. 2002). Agent-based organizational modelling approaches for integrated and systematic evaluation of material are presented in (Sharpanskykh et al. 2015; Sharpanskykh and Haest 2015; Stroeve et al. 2011) in which immaterial characteristics of socio-technical organizations in safety culture is analyzed. Results are presented of a model for safety occurrence reporting at an air navigation service provider. The results are fairly good in which an impact of social interaction and coordination among employees are explored to ensure safety regulations at ground level. The model is based on a case study which is partially simulated and validated at a real airline ground services. A systematic development of an unambiguous model of ATC system with its interactions with pilots is claimed in (Netjasov et al. 2010) using Stochastic and Dynamic Coloured Petri Nets (SDCPN). The SDCPN model is demonstrated for evaluation for a historical en-route and mid-air collisions. Satellite-based communication systems have been suggested in current advances to consider free flight concept for future ATC systems (Hu et al. 2002). In most of the existing work, the safety criteria are defined and developed by testing or simulation which has various disadvantages. For example, the testing or simulation is lacking in verifying the correctness of complex and safety critical systems. Moreover, the number of test cases or simulations increases exponentially to provide a required level of confidence due to complexity of such systems. Further, when a modification to the system is required, regression testing is needed to perform which suggests that the complete set of simulations must be re-conducted. The most important is that the performance promises obtained through simulations are not absolutely correct even if they do provide a conditional proof that the system is correct under certain assumptions. Other similar work is found in (Heffernan et al. 2014) (Jamal and Zafar 2007a, 2007b) (Moertl et al. 2003; Nguyen-Duc et al. 2003) (Yousaf et al. 2010; Zafar 2009).

CAS literature review using formal methods

Although there exists some work on modelling of multi-gents systems using formal techniques but it requires further investigation to apply formal methods in conjunction with multi-agent methodologies and complex networks to model the CAS as discussed in subsection "Modelling complex adaptive systems". Architecture Description Language and Unified Modeling Language based framework of multi-agent systems have been proposed in which service agents are used for agent communication that accepts Knowledge Query Manipulation Language (Park and Sugumaran 2005). In another work (Martin et al. 1999), an agent framework is proposed named as open agent architecture which has used Interagent Communication Language using a facilitator agent. Dynamic linear temporal logic (DLTL) is used for describing and verifying properties of agent communication systems (Giordano et al. 2007). In another research, an approach based on process algebra is used for agents based communication to establish a link between mathematical models and biological systems (Sumpter and Blanchard 2001). Petri-nets are used for modelling of railway interlocking components to ensure safety and deadlock free requirements (Giua and Seatzu 2008). Formal methods in terms of Z-notation and X-machine are used for the formal specification of multi-agent systems with a dynamic behaviour and structures (Ali et al. 2012). In another research work (Humphrey 2012),

an interesting technique is applied to verify that an existing procedure satisfies the specifications.

Conclusions

Complex Adaptive Systems are dynamic networks with many agents which are constantly acting and reacting in response to the other agents. The control of CAS is decentralized which needs to have a coherent behaviour which is achieved by a large number of decisions made by many agents in competition and cooperation among themselves. CAS can be characterized by various critical properties including emergence, co-evolution, sub-optimality, variety, connectivity, iteration, self organisation and nesting. By considering all such characteristics, we can find many CAS all around us. The development of CAS can be done by the combination of both the mathematical modelling and computer simulation techniques. The CAS are commonly investigated by agent-based methodologies and complex networks which are being widely appreciated in the natural and social sciences. Both of these methodologies are based on simulation techniques for verification of the systems. The simulation techniques are lacking of in verifying correctness of the systems due to exponential increase of test cases to achieve the required level of confidence. Formal methods help to ensure that the models developed are correct and reliable to overcome the disadvantages of simulation. That is why a formal approach in term of VDM-SL is used in modelling of the system underhand to provide an exhaustive support for verification of the take-off procedure of the ATC system.

Although there exist various computer models of ATC system but the safety and efficiency of the system are required to be addressed with further details. For example, there is some good work on modelling of ATC systems as reported in the related work however it needs to apply rigorous verifiable mathematical approaches to address the next generation automated systems achieving the required level of safety and efficiency. The work of Koeners et al. was found interesting (Koeners et al. 2011) in which planning function is developed for taxiing operation to address the uncertainties based on the real data simulating scenarios-based approach. This work was a good starting point for us to expedite the take-off procedure at the airport. Most of the work on modelling of ATC system is based on simulation techniques which have various limitations. To overcome the weaknesses of simulation, formal validation and verification are processes that help to ensure that the computer models are correct and reliable. That is why VDM-SL was used in this research which is a formal specification language and it provided an exhaustive support for validation and verification of the algorithm.

This work is based on our ongoing project on modelling of ATC system using formal methods (Zafar and Araki 2003; Zafar 2006; Zafar et al. 2012). The ATC system is a complex one, only part of it that is a procedure from taxiing to take-off is described in this paper using VDM-SL by focussing both on the safety and efficiency of the air traffic flow management. The model is developed by a series of refinements following the stepwise development approach. The airport surface is represented by the graph relation and then transformed to a static model in VDM-SL. The state space of the system is described by identifying and linking the ATC components. Before description of the model, a clear scope of the problem and set of assumptions were defined.

The safety is achieved by defining properties in terms of invariants available in VDM-SL over the data in the static model. The safety in dynamic model is provided by defining pre and post conditions over the operations for manipulating critical information to prevent any unwanted situation. The model is analysed and validated by testing and animation facilities available in the VDM-SL toolbox. The efficiency in the system is claimed because the model is assumed for the next generation ATC system which will expedite the take-off procedure. In addition, the model is based on graph theory which is used for defining the optimal routes in the dynamic part of the model. The model in graph theory can easily be extended to automate the algorithm increasing significance of the approach because automata are the special types of graphs. Finally, the constraints are put to ensure minimal queue size of aircrafts enhancing efficiency of the system.

In modelling of the ATC system, various benefits for applications of formal methods in a safety critical system were observed. For example, decomposition of system into its components provided us a complete characterization at a detailed level of specification. After the component-level analysis, compositional approach enabled us to give reasoning about the components and subsequently the entire system. The model is near to implementation while guaranteeing the transformation of syntax and semantics rules.

VDM-SL is selected because of its detailed expressive power for description of systems where validation, testing, visualization and animation are required. The VDM Tools allowed us developing precise model by checking automatic consistency and completeness of the requirements. The execution facility supported us for testing at the analysis and design level. The interpreter enables us for interactive debugging of the ATC model through various facilities.

Abbreviations

ATC: air traffic control; VDM-SL: Vienna Development Method Specification Language; ATFM: Air traffic flow management.

Authors' information

Nazir A. Zafar was born in 1969 in Pakistan. He received his M.Sc. (Math. in 1991), M. Phil (Math. in 1993), and M.Sc. (Nucl. Engg. in 1994) from Quaid-i-Azam University, Pakistan. He was awarded PhD degree in computer science from Kyushu University, Japan in 2004. He has served at various universities and well-reputed scientific organizations in Pakistan. For example, he has worked (2010-14) as an Associate Professor at the College of Computer Sciences and Information Technology (CCSIT), King Faisal University (KFU), Al Ahsa, Saudi Arabia. He has also worked (2007–10) as Dean/Professor of Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan. Currently, he is working as a Professor & Head at the Department of Computer Science, COMSATS Sahiwal, Pakistan. His current research interests are modelling of systems using formal approaches, integration of approaches, safety critical systems. He is an active member of Pakistan Mathematical Society. He has contributed for scientific and technical committees including organizing conferences and curriculum development in the capacity of a member as well as chairman.

Acknowledgements

The author greatly acknowledges COMSATS Institute of Information Technology, Sahiwal Campus for providing an excellent environment and resources at the Department of Computer Science to conduct research related activities.

Competing interests

The authors declare that they have no competing interests.

Received: 17 September 2015 Accepted: 7 January 2016
Published online: 15 February 2016

References

Ali G, Khan S, Zafar NA, Ahmad F (2012) Formal modeling towards a dynamic organization of multi-agent systems using communicating X-machine and Z-notation. Indian J Sci Technol 5(7):2972–2977

Alves DP, Weigang L, Souza BB (2008) Reinforcement learning to support meta-level control in air traffic management. Reinforcement Learning: theory and applications. ARS Publishing, pp 357–72
Amy S, Philip JS, Charles B (2002) Ramp Control issues in the design of a Surface Management System. Cognitive Systems Engineering Laboratory. The Ohio State University, Columbus
Armano G, Javarone MA (2013) Clustering datasets by complex networks analysis. Complex adaptive systems modeling (CASM), 1(5)
Boulaire F, Utting M, Drogemuller R (2015) Dynamic agent composition for large-scale agent-based models. Complex adaptive systems modeling (CASM), 3(1)
Bousson K (2003) Waypoint-constrained free flight collision avoidance. Proceedings of the SAE Advances in Aviation Safety Conference, 2003
Debbache NE (2001) Toward a new organization for air traffic control. Aircr Eng Aerosp Technol 73(6):561–567
Erzberger H, Heere K (2009) Algorithm and operational concept for resolving short range conflicts. J Aerosp Eng 224:225–243
Erzberger H (2006) Automated conflict resolution for air traffic control. Proceedings of the 25th International Congress of the Aeronautical Sciences, 2006
Fitzgerald J, Larsen PG (2009) Modelling Systems: practical tools and techniques in software development, 2nd Edn, Cambridge University Press, 2009
Garcia J, Berlanga A, Molina JM, Besada JA, Casar JR (2002) Planning techniques for airport ground operations. Proceedings The 21st Digital Avionics Systems Conference, 2002
Giordano L, Martelli A, Schwind C (2007) Specifying and verifying interaction protocols in a temporal action logic. J Appl Log 5(2):214–234
Giua A, Seatzu C (2008) Modeling and supervisory control of railway networks using petri nets. IEEE Trans Autom Sci Eng 5(3):431–445
Hall A (1999) Using formal methods to develop an ATC information system. Industrial-strength formal methods in practice: formal approaches to Computing and Information Technology (FACIT), Springer, Heidelberg, pp 207–229
Hanh TT, Hung D (2007) Verification of an air traffic control system with probabilistic real-time model checking. UNU-IIST, Report No. 355
Heffernan D, MacNamee C, Fogarty P (2014) Runtime verification monitoring for automotive embedded systems using the ISO 26262 functional safety standard as a guide for the definition of the monitored properties. IET Softw 8(5):193–203
Holland John H (2006) Studying complex adaptive systems. J Syst Sci Complex 19(1):1–8
Hu J, Prandini M, Sastry S (2002) Optimal coordinated maneuvers for three-dimensional aircraft conflict resolution. J Guid Control Dyn 25(5):888
Humphrey L (2012) Model checking UAV mission plans. In: Proceedings of AIAA Conference on Modeling and Simulation Technologies, 2012
Hwang I, Tomlin C (2002) Protocol-based Conflict Resolution for Finite Information Horizon. Proceedings of the AACC American Control Conference, IEEE Publ., Piscataway, 2002
Hwang I, Hwang J, Tomlin C (2003) Flight-mode-based aircraft conflict detection using a residual-mean interacting multiple model algorithm. Proceedings of the AIAA Guidance Navigation and Control Conference, 2003
Hwang I, Balakrishnan H, Roy K, Tomlin C (2004) Target tracking and identity management in clutter for air traffic control. Proceedings of the AACC American Control Conference, 2004
Jamal M, Zafar NA (2007a) Formal model of computer-based air traffic control system using Z notation. Proceedings of 17th International Conference on Computer Theory and Applications, 2007
Jamal M, Zafar NA (2007b) Requirements analysis of air traffic control system using formal methods. Proceedings of IEEE International Conference on Information and Emerging Technologies, pp 216–22
Kahne S, Frolow I (1996) Air traffic management: evolution with technology. IEEE Control Systems Magazine, 16 (4)
Koeners GJ, Stout EP, Rademaker RM (2011) Improving taxi traffic flow by real-time runway sequence optimization using dynamic taxi route planning. 30th IEEE/AIAA Digital Avionics Systems Conference, 2011
Kuchar JK, Yang LC (2000) A review of conflict detection and resolution modeling methods. IEEE Trans Intell Transp Syst 1(4):179–189
Kwiatkowska M, Norman G, Sproston J, Wang F (2004) Symbolic model checking for probabilistic timed automata. Joint Conference on formal modeling and analysis of timed systems and formal techniques in real-time and fault tolerant systems, LNCS, Springer, 3253, pp. 293–08
Marshall W, Joseph WI (1992) Airport Movement Area Safety System. IEEE proceedings of Digital Avionics Systems Conference, pp 549–552
Martin DL, Cheyer AJ, Moran DB (1999) The open agent architecture: a framework for building distributed software systems. Appl Artificial Intelligence 13(1–2):91–128
Medina M, Sherry L, Feary M (2010) Automation for task analysis of next generation air traffic management systems. Transp Res Part C 18:921–929
Michael C, Steven S (2012) Managing gate and ramp operations to reduce delay, fuel burn, and costs. Integrated communications, Navigation and Surveillance Conference (ICNS), 2012
Moertl PM, Atkins S, Hitt JM, Brinton C, Walton DH (2003) Factors for predicting airport surface characteristics and prediction accuracy of the Surface Management System. IEEE International Conference on Systems, Man and Cybernetics, 2003
Moulding MR, Smith LC (1992) Formalizing a CORE requirements model in the air traffic control domain, the Future. IEE Colloquium on software in Air Traffic Control Systems, 1992
Netjasov F, Vidosavljevic A, Tosic V, Everdij M, Blom H (2010) Stochastically and dynamically colored petri-net model of ACAS operations. 4th International Conference on Research in Air Transportation, 2010
Nguyen-Duc M, Briot JP, Drogoul A, Duong V (2003) An application of multi-agent coordination techniques in air traffic management. Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, pp 622–28

North MJ (2014) A theoretical formalism for analyzing agent-based models. Complex Adaptive Systems Modeling (CASM), 2(3)
Park S, Sugumaran V (2005) Designing multi-agent systems: a framework and application. Expert Syst Appl 28(2):259–271
Rademaker R, Koeners GJM (2011) Analyze possible benefits of real-time taxi flow Optimization using actual data. 30th IEEE/AIAA Digital Avionics Systems Conference, 2011
SCSK Corporation, VDM Tools, Language Manual, Version 9.0.2, 2013
SCSK Corporation, VDM Tools, User Manual, Version 9.0.2, 2013
Sharpanskykh A (2009) Agent-based modelling and analysis of air traffic organisations. Intelligent systems for knowledge management, Springer, Vol. 252, pp 251–274
Sharpanskykh A, Haest R (2015) An agent-based model to study effects of team processes on compliance with safety regulations at an airline ground service organization. Proceedings of the 18th Conference on Principles and Practice of Multi-Agent Systems (PRIMA), 2015
Sharpanskykh A, Lindenberger T, Blom H (2015) Agent-based modeling and analysis of coordination mechanisms in air traffic management. The Eleventh Conference of the European Social Simulation Association (ESSA), 2015
Shorrock ST, Kirwan B (2002) Development and application of a human error identification tool for air traffic control. Appl Ergon 33(4):319–336
Simcox LN (1989) The application of Z to the specification of air traffic control systems: 1. Memorandum No. 4280, RSRE, Ministry of Defense, Malvem, April 1989
Stroeve S, Sharpanskykh A, Kirwan B (2011) Agent-based organizational modelling for analysis of safety culture at an air navigation service provider. Reliability Engineering & System Safety 96, pp. 515-533
Sumpter DJT, Blanchard GB (2001) Ants and agents: a process algebra approach to modelling ant colony behavior. Math Biol 63(5):951–980
Weigang L, Dib MVP, Alves DP, Crespo AMF (2010) Intelligent computing methods in air traffic flow management. Transportation Res Part C Emerg Technol 18(5):781–793
Yang LC, Kuchar JK (1997) Prototype conflict alerting system for free flight. J Guidance, Control, Dynamics, 20 (4)
Yousaf S, Zafar NA, Khan SA (2010) Formal analysis of departure procedure of air traffic control system. 2nd International Conference on Software Technology and Engineering, 2010
Yousaf S, Khan SA, Zafar NA, Ahmad F, Khan MA (2012) Formal analysis of arrival Procedure of air traffic control system. Life Sci J 9 (4)
Zafar NA (2009) Formal specification and validation of railway network components using Z notation. IET Softw 3(4):312–320
Zafar NA, Khan SA, Araki K (2012) Towards the safety properties of moving block railway interlocking system. Int J Innovative Computing, Inf Control 8(8):5677–5690
Zafar NA (2006) Modeling and formal specification of automated train control system using Z notation. IEEE Multi-topic Conference, pp 438–43
Zafar NA (2014) Safety control management at airport taxiing to take-off procedure. Arabian J for Sci Eng (AJSE), Springer, 39, pp 6137–48
Zafar NA, Araki K (2003) Formalizing moving block railway interlocking system for directed network. Research reports on Information Science and Electrical Engineering, Kyushu University, 8(2), pp 109–14