Modeling Complex Air Traffic Management Systems

Neha Rungta
NASA Ames Research Center
neha.s.rungta@nasa.gov

Eric G Mercer
Brigham Young University
egm@cs.byu.edu

Franco Raimondi
Middlesex University
F.Raimondi@mdx.ac.uk

Bjorn C. Krantz
NASA Ames Research Center
bjorn.c.krantz@nasa.gov

Richard Stocker
Middlesex University
soxsheba@hotmail.com

Andrew Wallace
Brigham Young University
wallaceac@byu.edu

ABSTRACT

In this work, we propose the use of multi-agent system (MAS) models as the basis for predictive reasoning about various safety conditions and the performance of Air Traffic Management (ATM) Systems. To this end, we describe the engineering of a domain-specific MAS model that provides constructs for creating scenarios related to ATM systems and procedures; we then instantiate the constructs in the ATM model for different scenarios. As a case study we generate a model for a concept that provides the ability to maximize departure throughput at La Guardia airport (LGA) without impacting the flow of the arrival traffic; the model consists of approximately 1.5 hours real time flight data. During this time, between 130 and 150 airplanes are managed by four enroute controllers, three TRACON controllers, and one tower controller at LGA who is responsible for departures and arrivals. The planes are landing at approximately 36 to 40 planes an hour. A key contribution of this work is that the model can be extended to various air-traffic management scenarios and can serve as a template for engineering large-scale models in other domains.

1. INTRODUCTION

Multi-agent systems (MAS) offer design abstraction and modeling capabilities for systems involving both humans and automation, but to the best of our knowledge there is no work that provides support for creating models for a specific application domain. In recent years, there has been a lot of work in developing domain-specific languages (DSL) in context of programming languages and software engineering. DSLs provide developers constructs and functionality relevant to their domain-specific applications; there are, however, no domain specific MAS modeling frameworks. Several MAS models have been developed to model applications from different domains such as aviation, health care system, nuclear power plants, and others. But since the MAS architectures are designed as general-purpose frameworks, it is often challenging for experts in a specific domain to learn and use the MAS modeling frameworks for their applications. Although the practice of engineering domain-specific models is applicable to a wide variety of applications where humans interact with each other and automation, in this paper we consider problems related to the design of air traffic management systems and procedures for civil aviation.

The goal of several government agencies related to civil aviation in the US and Europe is to have a system-wide autonomous optimized airspace with a flexible, scalable, and resilient system to meet significant traffic growth while supporting ever changing operator roles and business-network models. This increase in automation is required for the projected growth in traffic demand including Unmanned Aerial Systems (UAS) in National Airspace (NAS) [3]. The National Research Council (NRC) report on "Autonomy Research for Civil Aviation" [6] points out,

> "current safety methods rely heavily on the human operator to mitigate system anomalies and unanticipated events".

There is, however, no formal or precise characterization of the decision making process or the actions of the human operators that lead to the safety of the current system. In light of this, it is very difficult to estimate the impact on the overall safety of the NAS with the introduction of new systems, automation, or procedures.

In this work, we propose the use of domain-specific multi-agent system (MAS) models as the basis for predictive reasoning about various safety conditions and the performance for Air Traffic Management (ATM) scenarios. We describe the engineering of a domain-specific MAS model that provides constructs for creating ATM scenarios. We then present how these constructs can be instantiated for specific tasks and scenarios. The instantiation process is described using a case study based on an evaluation of the Departure Sensitive Arrival Scheduling (DSAS) concept [9]. The DSAS concept provides the ability to maximize departure throughput at LaGuardia airport (LGA) without impacting the flow of the arrival traffic; each scenario in the DSAS evaluation consists of approximately 1.5 hours real time flight data. During this time, between 130 and 150 airplanes are managed by four enroute controllers, three TRACON controllers, and one tower controller at LGA who is responsible for departures and arrivals. The planes are landing at approximately 36 to 40 planes an hour. The case study demonstrates the potential utility of constructing domain-specific MAS models for ATM applications.

ACM acknowledges that this contribution was authored or co-authored by an employee, or contractor of the national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only. Permission to make digital or hard copies for personal or classroom use is granted. Copies must bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. To copy otherwise, distribute, republish, or post, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MiSE'16, May 16-17, 2016, Austin, TX, USA

© 2016 ACM. ISBN 978-1-4503-4164-6/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2896982.2896993

2. DOMAIN-SPECIFIC MODEL

In order to create a model, that allows for predictive reasoning about ATM systems, such that the model is extensible and reusable across different scenarios we were posed with two different choices:

1. Design and develop a new domain-specific MAS framework for creating and analyzing ATM models.

2. Use or extend an existing MAS framework for developing and checking ATM models.

The challenge with creating a new domain-specific MAS framework from scratch is that it takes a significant effort to develop such a framework. The required steps include defining a syntax for the language, formalizing the semantics, developing a compiler for the language, implementing the execution semantics so that we can run simulations, and finally implement algorithms that allow us to reason about interesting properties about the models. Each of these steps is non-trivial, making the process more than expensive in both time and resources. Hence, we decided to leverage an existing MAS framework, Brahms, to create and analyze ATM models. Brahms has been developed over the past 10 years and includes clear definitions of the language and semantics with a stable implementation.

The core contributions of defining a domain-specific ATM model is: (a) define constructs of the system that are familiar to the domain expert and (b) provide simulation and analysis data in a manner such that the domain expert can easily interpret and understand the results. In this section we first present a high-level overview of the Brahms language and its semantics; next we present how we create constructs for the ATM model in Brahms.

2.1 Overview of Brahms

Brahms is a simulation and development environment originally designed to model the contextual situated activity behavior of groups of people in a real world context [4, 15]. Brahms has now evolved to model humans, robots, automated systems, agents, and interactions between humans and automated systems. Brahms follows a rational agent approach: rational agents are autonomous, react to changes in circumstance and choose their actions based on their own agenda. Brahms is a beliefs, desires, and intentions (BDI) model that describes the goals the agent has and the choices it makes, all based on a certain set of beliefs.

A Brahms model contains a set of *Objects* and *Agents* that are used to model humans and automated systems in a real-world context. Brahms is able to represent artifacts, data, and concepts in the form of classes and objects. A geography model is used to locate agents and objects and provides an additional context to activities.

The key structural aspects of Brahms are:

1. *attributes*: properties of agents/objects/locations,

2. *activities*: actions an agent performs, typically advancing simulation time,

3. *beliefs*: each agent's own personal perception of all the *attributes* in the model,

4. *facts*: actual values of the *attributes*,

5. *detectables*: detection of *facts*, bringing facts into an agent's belief base and determining the response,

6. *workframes*: guarded plans which denote a sequence of events required to complete a task, including belief updates and actions,

7. *thoughtframes*: guarded plans for belief revisions made due to the situated context of the agent/object, and

8. *time*: forms the time-line output presented by Brahms to describe the simulation.

In this project, we use a Brahms Simulator implemented in Java that describes the semantics for executing a Brahms model [8]. The Brahms scheduler has a discrete clock. The scheduler can jump multiple time steps to move simulation faster than real time simulation.

Here we provide a brief overview of the Brahms semantics. For more details, please refer to papers on Brahms [4, 15, 8]. The Brahms scheduler initializes the system, the agents, objects, and geography. The scheduler first instantiates all the agents where they initialize their facts and beliefs. After initialization, the scheduler instructs all agents to process their thoughtframes. The agents check the guards to generate a set of active thoughtframes. Next, the scheduler selects the active thoughtframe with the highest priority. If, however, there is more than one active thoughtframe with the same priority, one is randomly selected from the set.

In the past, Brahms models have been created to analyze conditions that led to the Air France 447 accident [8], to review the sequence of events that led to the Überlingen collision [14], and to perform a comparative evaluation of operator workload between a single pilot and traditional two pilot concept [16]. Each of these models is a *one-off* exercise requiring a MAS modeling expert to create; they are not re-usable across different scenarios and do not share any modeling constructs.

2.2 Generic ATM Model

A key challenge in building models with a high enough fidelity to generate actionable predictions is to choose the right level of abstraction. The generic model, or framework, described here represents an abstraction level distilled from close work with a team of airspace design experts, controllers, cognitive scientists, human factor researchers, and pilots. The resulting model is both configurable and extensible to any specific air space or ATM protocol.

At a high-level the model leverages Brahms' object-oriented nature to create a type hierarchy of agents and objects that comprise the building blocks of an ATM system simulation. These can be roughly grouped into the following categories:

Geography represents physical locations in the real world and routes through those locations. This category provides flight plans for the navigation objects and the structure of the world for the controllers.

Navigation represents anything that moves in the airspace. This category provides a common set of trajectory based activities to determine movement through time between locations that includes altitude.

Controllers coordinate aircraft in the airspace. This category provides a common set of controller actions for interactions with the aircraft.

Clock that creates a universal view across the model of time advancing.

2.2.1 Geography

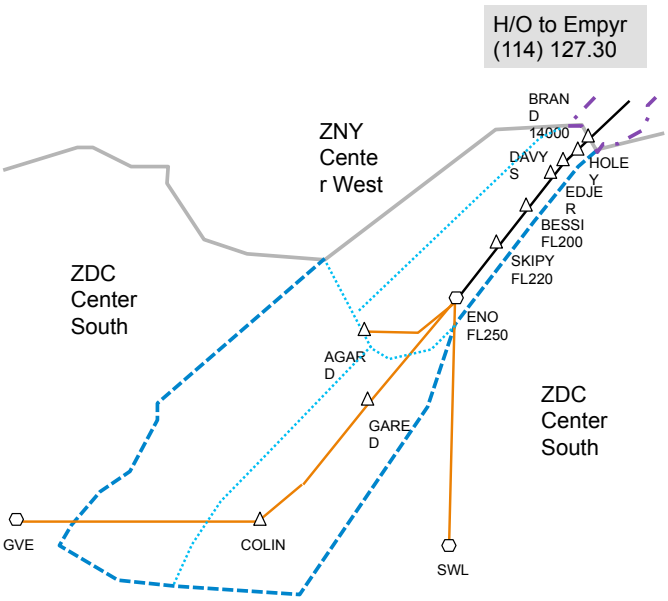

Figure 1: Example of a Sector (ZDC Center South) in the NY metroplex as defined in the DSAS HILT study setup

Geography elements in the ATM model are waypoints, flight segments, and flight plans; an example of the geography concept is illustrated in Fig. 1. The labeled points are waypoints and the lines connecting the waypoints (blue or gray) are flight plans. These define highways in air which aircraft follow through the airspace. The waypoints shown in Fig. 1[1] are part of the south sector of the airspace in the NY metroplex.

More specifically, waypoints are latitude and longitude coordinates of points on the earth shown below. An instance of a waypoint is made by creating a named object waypoint with appropriate coordinates in the ATM model.

A flight segment is a pair of waypoints that includes an ending altitude and speed. It reflects the intended trajectory a navigation object, e.g., an airplane, would follow through the segment. Similar to waypoints, instances of flight segments are created to reflect standard operating procedures (SOP)[2] in the airspace being modeled.

A flight plan is an ordered list of flight segments. The flight plan is the defined highway followed by a navigation object through the airspace. The framework is designed such that a modeler is able to make either a standard set of flight plans through the airspace or make flight plans specific to individual aircraft.

Normally in ATM, there is a concept of sectors that define a portion of the airspace managed by either a single controller or a group of controllers. Aircraft move through sectors along specific flight plans, and controllers hand-off aircraft at sector boundaries. The model presented here does not expressly model sectors directly. Rather it indirectly models sectors with boundary waypoints. The controller category is responsible for defining the boundaries that belong to any individual controller. Hand-off is handled via the flight segment and detected when the boundary waypoint is reached.

2.2.2 Automation

There are several systems being used in current day operations to improve the throughput of the NAS while maintaining the current levels of safety. One such automation is the Terminal Sequencing and Spacing (TSS) system which is currently being deployed across different regions in the US and is being used by controllers to assist with the complex task of scheduling and controlling aircraft during periods of congestion [18]. The goal of the TSS model is to create conflict-free schedules for arriving flights at meter fixes, runway thresholds, and any other merge points. The TSS generates a slot marker for each of the planes. A slot marker represents a 4-dimensional point (latitude, longitude, altitude, and time) in space that specifies where its respective plane needs to be in order to meet scheduling, sequencing, and miles in-trail (MIT)[3] spacing requirements.

2.2.3 Navigation

The navigation class updates the position of airplanes based on the current position, altitude, and airspeed using a Java activity. The class has workframes that activate on a regular frequency, determined by an attribute in the type, to call the external activity for the position update. Based on the position, the class has workframes that update or advance the flight plan. It uses the *Java externals* feature of Brahms to create Java activities implemented in the Java programming language. The choice to use Java for computing trajectory information means that it is possible to be as detailed or abstract as desired in modeling movement. As such, the model can be tuned, with little effort, to meet different modeling objectives.

The navigation class is specialized in two distinct ways: one for airplanes and the other for slot markers. The airplane specialization includes support for controller interactions. That support includes hand-offs between sectors and clearances for metering. Metering is a deviation from the specified airspeed in the flight plan; this change of speed is issued as a clearance by the controller. For example, a controller may issue a clearance for the airplane to decrease its airspeed by 20 knots in order to sequence the plane through a merge point and maintain MIT separation.

The slot marker specialization indicates where the plane needs to be in order to meet its *scheduled time of arrival* (STA) according to the TSS automation. The controller issues clearances in order to ensure that the plane is at the coordinates of its corresponding slot marker. The slot marker is placed in a specific position by the TSS automation, and it then follows the same flight plan as its associated airplane. As mentioned previously, a benefit to this model structure is that the airplane and slot marker use the same Java library to update their position. In essence a slot marker is a virtual airplane but it can make abrupt jumps when the TSS performs a change in the schedule. The airplane and slot types are used by instantiating specific instances of flights

[1]The figure was part of the airspace configuration setup provided to controllers in the DSAS HILT.

[2]A SOP specifies constraints for how a given class of aircraft should fly along a given route unless instructed otherwise by a controller.

[3]MIT is the number of miles required between aircrafts.

and slot markers for those flights. In this way, the modeler is able to build different traffic patterns using the model.

2.2.4 Controller

The controller is the most complex model in the system. It is responsible for managing traffic through a sector of the airspace including any sequencing that needs to take place in order for planes to maintain proper horizontal and vertical separation. More specifically, *Air Route Traffic Control Center* (ARTCC), also known as enroute centers, handle traffic during the cruise phase of flight. Each center has several en route sectors with aircrafts at altitudes between 10K to 37K feet altitude. Most aircraft in this phase of flight are simply transitioning through one section to another. Some aircraft are transitioning from cruise to descent; these aircraft are handed off to the appropriate TRACON approach controllers.

The TRACON handles a 40 mile airspace around an airport. TRACON controllers are responsible to transition arriving and departing aircraft with Tower controllers. Departure TRACON gives pilots headings, altitudes and speeds for the cruise phase of the flight, while approach TRACON controllers funnel and sequence aircraft approaching from different enroute sections into a single stream. The local controller in the tower is responsible for the aircraft landing and taking off on the runways. The local controller issues clearances to land and take-off and handle transitions with their respective TRACON controllers.

The controller model defines sectors by border waypoints and the controllers that own those points. For example, in Fig. 1, the blue dotted line defines a sector managed by a specific controller. The COLIN waypoint marks the hand-off waypoint. As a plane follows its flight plan, it moves through waypoints and, when it is approaching a hand-off waypoint, the controller initiates the hand-off with the owning controller.

The controller type keeps track of planes in its sector using a relation. At each hand-off the plane, or flight, is added to or removed from the relation. In this way, the controller is able to track all the planes in its airspace. Similar to the navigation class, the controller agents leverage Java external activities to check separations between planes. For each plane that it owns, at a regular frequency determined by an attribute in the type, it computes the ETAs of the planes to merge points, and issues clearances to individual flights to sequence through merge points. Similarly, once TSS has assigned an STA and put the slot marker in the appropriate position, the controller will check the distance between flights and slot markers issuing clearances as appropriate. As of writing, the controller is only able to meter flights (i.e., adjust airspeed) to accomplish sequencing. Vectoring flights (i.e, change trajectory), is a subject of future work. It should be noted that part of the controller monitoring process is reporting the loss of separation[4].

Similar to the other agents and objects, a modeler instantiates specific instances of the controller, including the requisite border waypoints and controllers, to define a specific airspace. Together with the instantiated flights and geography, the modeler constructs a scenario. The modeler does not have to define behavior in any of these instances; only the attributes are defined (i.e., given specific values).

[4]A loss of separation between aircraft occurs whenever the specified separation in controlled airspace is no longer valid.

2.2.5 Clock

The model has its own clock that tracks the time advancing during the simulation of the model. The clock internal to the model provides the flexibility to determine the granularity of time steps in the model. Sometimes when fine-grained actions are not relevant to the simulation, the clock can jump multiple time-steps at a given instance.

3. DSAS CONCEPT

Air traffic operations around the New York airspace are notoriously complex. High demand combined with a highly constrained airspace and uncertainty from factors such as weather, create a challenging operational environment and sometimes brittle system. Given the level of arrival demand and the priority given to arrival aircraft for runway use, departure aircraft are often required to sustain delays and await opportunities to depart as they arise. An extreme, though not entirely uncommon, result of this situation is that delays often grow on the departure side until the departure queue extends to the point of gridlock.

One strategy for addressing the departure problem is to work toward developing an arrival schedule that ensures departure release without reducing arrival capacity or the need for a reduction in arrival demand. Leveraging the capabilities of the TSS technology suite, Optimized Profile Descent (OPD) routings and procedures, and scheduling decision support, a human-in-the-loop (HITL) simulation was conducted in the summer of 2014 in the Airspace Operations Lab (AOL) at NASA Ames Research Center that examined the potential benefits, feasibility, and issues regarding Departure-Sensitive Arrival Scheduling (DSAS) at LaGuardia Airport (LGA) in New York [9].

Each scenario evaluated in the HILT of the DSAS study consists of approximately 1.5 hours real time of simulation. During this time, there are between 130 and 150 airplanes being managed by four enroute controllers, three TRACON controllers, and one tower controller at LGA who is responsible for departures and arrivals. The planes are landing at approximately 36 to 40 planes an hour.

We believe the airspace around the New York metroplex evaluated in the DSAS study serves as an ideal representative part of the NAS in order to create instances of the generic ATM model. The specific parts that we create are as follows:

1. The airspace in the NY metroplex and operations that includes the three major airports: JFK, Newark (EWR), and La Guardia (LGA). We will restrict the airport operations to LGA even though we will consider the entire NY metroplex airspace.

2. Traffic flow and runway configurations under standard operating conditions.

3. Air Traffic Controllers at the New York Air Route Traffic Control Center (New York ARTCC, New York Center, or ZNY), New York TRACON (N90), and the tower at LGA.

4. Supervisors and Traffic Management Coordinators at the respective Center, TRACON, and Towers.

5. Pilots flying the planes in the defined airspace.

6. Automation related to trajectory-based operations, e.g., TSS.

In Brahms, we were able to successfully model all the planes, the pilots of the planes, the different controllers, the interactions amongst the humans (controllers, pilots, and supervisors), and the interactions between human operators and automation by instantiating specific parts of the ATM general model. To create specific instances of the constructs in the model we use the data logged by the Multi-Aircraft Control System (MACS) tool; the MACS software was used to setup and run the simulations in DSAS HITLs. We use information about arrival traffic flow, flight plans, and departures from the MACS data logs to then automatically generate Brahms objects/agents for airplanes, the flight plans for the airplanes, the waypoints, the departures, the configuration of the sectors, and the various controllers directing the approaching traffic.

Waypoints are uniquely identified by their names. We automatically create all corresponding waypoints objects based on their latitude and longitude. We create flight segments such that they are specific to a plane; this is because the speed and altitude restrictions vary for aircraft based on their class. The flight plan is read from the MACS logs for each aircraft, the corresponding flight segments are first generated and then the Brahms object representing the flight plan is generated. Based on the information in the MACS logs we also generate the initial state of the aircraft as well. Note that we track traffic only in four enroute centers, the TRACON, and LGA; so we assign the initial latitude and longitude of a particular plane in the HITL at the time the plane enters this region of interest. This information is also gleaned from the MACS logs. The startTime of the aircraft indicates at what time in the simulation a particular plane appears in the region of interest to the model. Each aircraft is also assigned a flight plan. There are attributes for the starting latitude, longitude, speed and altitude of an aircraft. The currentFlightSegment attribute provides information about which flight segment the aircraft is currently flying on. The attributes such as latitude, longitude, speed, altitude, and the currentFlightSegment are updated as the plane starts flying. Consider the example shown below where the initial values of the plane JBU6365 are set based on when it enters the region of interest.

```
agent plane_JBU6365 memberof Airplane {
  initial_beliefs:
(current.latitude = 41.25311);
(current.longitude = 73.71723);
(current.myFlightPlan = flightPlan_JBU6365);
(current.startTime = 40);
(current.speed = 239.12);
(current.altitude = 8755.76);
        ..
```

A certain set of attributes are also generated from the MACS logs for the different enroute centers, TRACON, and Tower controller. These include the set of waypoints with the sector *managed* by the respective controller, the waypoint where the current controller performs the hand-off to controller managing an adjoining sector, as well as the identifier of the adjoining sector (current.handoffTo = ZNY_114).

The Center and TRACON controllers perform handoffs when planes cross sector boundaries. In the model, controllers maintain the required separation based on the specified rules. The rules were derived from discussions with retired controllers.

The Brahms constructs discussed above are all instances of the templates specified in the generic ATM model. We combine the auto-generated parts of the model (e.g., constructs that are different for each scenario such as traffic flow, airplanes, departures) that represent instances of the elements specified in the ATM model and the re-usable parts of the ATM model (e.g., how planes fly or how hand-offs are performed) to generate the final models.

4. DSAS SIMULATION

We were able to successfully simulate 1.5 hours of real traffic in the Brahms model using the auto-generated and static parts of the model. The simulation of the Brahms model takes approximately 3 to 4 minutes. The output of the simulation is visualized in a custom tool. The tool visualizes the flow of the arrival traffic, the departures, and the hand-off between the controllers. With this visualization we were able to validate that the simulation of the Brahms model, even at a high-level of abstraction, mimics the HITL scenarios. In this section we describe our observations of the model simulation.

Consider the snapshot of our visualization tool in Fig. 2 that is recreating one of the scenarios evaluated in the DSAS HITL. This image is best viewed in color. The triangle represents the waypoints around the LGA airspace, examples are FINSI, AWARE, FOLAM, DAVYS, HOLEY, and others. The dot followed by a call sign are identifiers for planes flying. The colors of the plane identifiers are indicative of which controller is managing the particular aircraft. When the plane transitions from one sector to another, the color of the call sign changes to that of the corresponding controller. All the planes shown in Fig. 2 are approaching the LGA airport that is indicated by the waypoint LGA22 (shown in the upper right quadrant of Fig. 2) where 22 is the runway configuration. Once an aircraft reaches LGA22, it is considered as landed.

As seen in this figure there are three streams of traffic approaching LGA, south, west, and north. First the south and west traffic merges around waypoint TYKES. This south/west merged traffic then merges with the traffic from the north sector at the MIRRA and GREKO waypoints. Here the controller models reason about whether the planes can have a potential loss of separation (less than 5 nautical miles in trail separation). A thick line shown in Fig. 2 between call sign JBU3635 and SWA1837 is indicative that if the controller does not take any action there will be loss of separation between these two planes. In our model, the controller slows down a plane (metering) to ensure that the planes continue to maintain the required separation. The visualization provides an explicit representation of the controllers' thought process.

All communications between the controllers or between a controller and plane is shown in the text box titled "Communication Window" on the left panel of Fig. 2. Note that communications are not restricted to voice communications. For example, hand-offs are performed by assigning a data block on the controllers' screen to another controller. When the other controller accepts the data block, the hand-off is

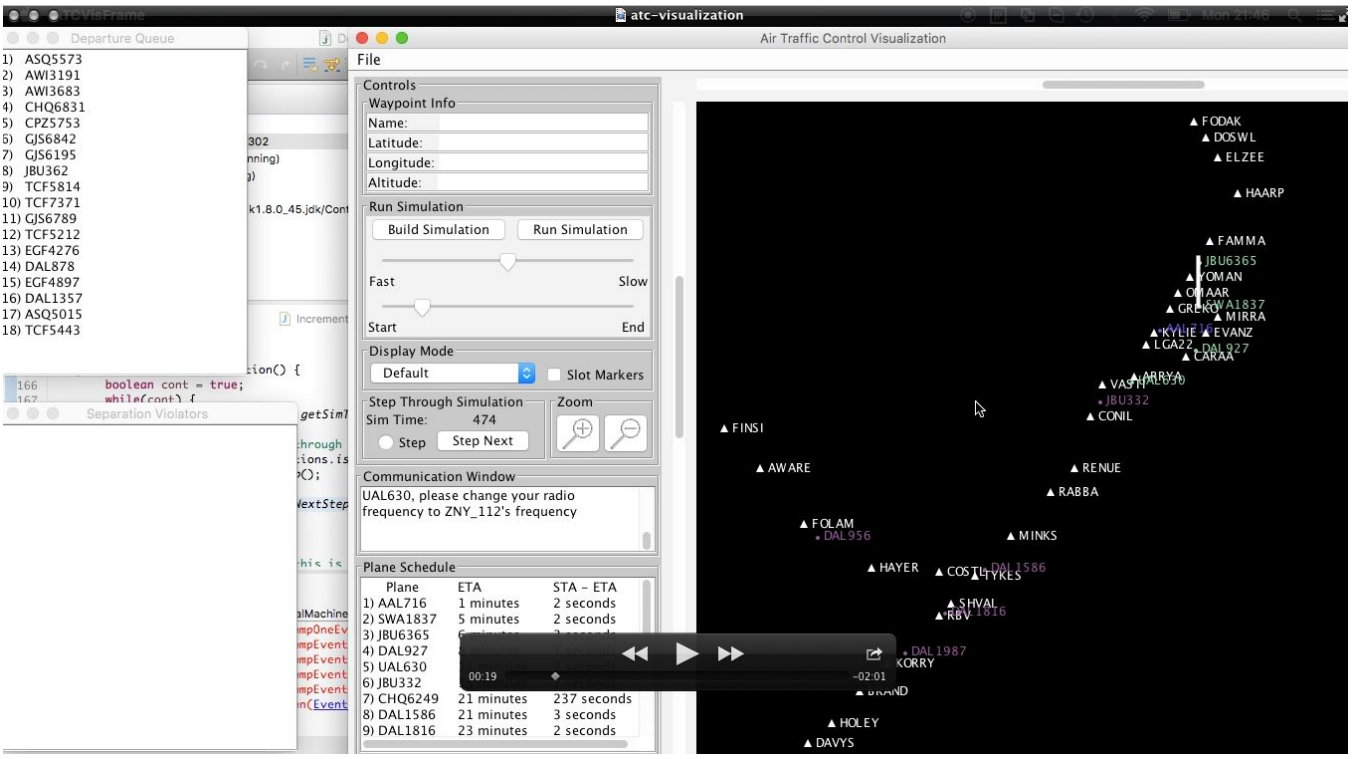

Figure 2: Visualization of the approach traffic in the NY metroplex airspace.

complete. The data communications are also presented in the communications window.

The panel below the communications window depicts the planes' schedule, its ETA and the difference between its STA and ETA. There are sliders which can be used to speed up or slow down the simulation to observe specific interactions. The Step Next button provides the ability to the user to step through the simulation. The Zoom functionality is also available in the tool. There are various display modes that were designed to mimic the display screen used by controllers and supervisors.

The visualization also presents a window titled "Departure Queue" and another titled "Separation Violators" shown in Fig. 2. The departure queue is as the name suggests: the list of airplanes waiting to depart from the LGA airport. In our simulation we delete the aircraft as soon as it takes off to mimic what was done in the DSAS HITLs. Though this is not a restriction of the simulation. The plane could have continued flying if we had defined departure TRACON controllers and other controllers that handle departing aircrafts. This demonstrates the flexibility of the ATM model. Elements and instances of controllers, traffic, and other agents can be added or removed based on the requirements of the simulation. The separation violators window presents any loss of separation between aircraft; when such a case does occur the relevant airplanes also blink to provide visual cues to the observer.

Even though the model and simulation of the DSAS model shown in Fig. 2 is at a high-level of abstraction compared to that of the HITL, we believe that a visualization tool such as this aids domain-experts understanding and interpreting the results of the simulation. This model and visualization, we believe, addresses, at least in part, a big challenge in any simulation-related modeling exercise. Often there are no good ways to map the results of the simulation back to its original application. The choice of our visualization and its layout was driven by the domain and display formats of existing systems in use.

The visualization and other information gathered during the simulation allows us to reason about the safety and performance of the system. If by increasing the traffic or changing the conditions of the scenario the number of loss of separation increases, then the coordination/communications activities occurring at the time can be used to determine the cause of reduced safety. Further, the model enables the measurement of cognitive workload impact of procedure changes on the controllers and planes. This measure can be used to guide the design of automated systems and procedures.

5. RELATED WORK

Multi-agent systems have been employed to model, simulate and analyse a range of scenarios, from banking systems to crowd management, auctions, trust issues and traffic simulations. A number of tools have been developed to support these activities. Examples of tools include, just to name a few: the Repast Agent Modelling Toolkit [12] that offers features to support simulation and analysis of results using external connectors; NetLogo [19], focusing mainly on ease of use and with support for educational activies; MASON, providing support for Java simulations and 3D visualization tools [10]; the Jason tool for AgentSpeak programs, enabling the use of the BDI paradigm in the modelling, simulation and verification of multi-agent systems; Brahms [5], as shown above, is a framework targeting mainly systems

involving both human and artificial agents.

In spite of the plethora of *generic* tools, there doesn't seem to be support for *domain-specific languages* (DSL) for multi-agent systems in the Software Engineering sense [11]. Indeed, while there is ample support in Software Engineering for DSL development and runtime support, it seems that the multi-agent community has focused mainly on the design phase, see for instance [17, 7].

The closest related work to our ATM model is perhaps [2], which employs the Prometheus methodology [13] to model an avionics scenario. Prometheus *models* scenarios, and it then generates JADE code [1]. Our approach, instead, aims at providing a measure of how certain choices impact the design of ATM technologies and procedures.

6. CONCLUSIONS

We propose the use of domain-specific MAS models as the basis for predictive reasoning about various safety conditions and the performance of systems and procedures in the domain of Air Traffic Management (ATM). In this work we describe the engineering of a domain-specific MAS model that provides constructs for creating scenarios related to ATM. We present how these constructs can be instantiated to implement specific tasks and procedures related to ATM. Here we describe instances of the ATM model for recreating the Departure Sensitive Arrival Scheduling (DSAS) concept evaluated in a HITL [9]. A key contribution of this work is that the model can be extended to various air-traffic management scenarios and can serve as a template for engineering large-scale models in other domains. We believe the process defined to create ATM models could be used to create models in other related domains for example autonomous cars in real world traffic and operators with drones.

7. REFERENCES

[1] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[2] J. M. Canino, J. García, J. M. Molina, and J. B. Portas. *A Multi-Agent Approach for Designing Next Generation of Air Traffic Systems*. INTECH Open Access Publisher, 2012.

[3] J. A. Cavolowsky, L. Quon, and P. Kopardekar. New directions NASA's airspace operations and safety program, 2014.

[4] W. Clancey, P. Sachs, M. Sierhuis, and R. Van Hoof. Brahms: Simulating practice for work systems design. *International Journal of Human-Computer Studies*, 49(6):831–865, 1998.

[5] W. J. Clancey, P. Sachs, M. Sierhuis, and R. Van Hoof. Brahms: Simulating practice for work systems design. *International Journal of Human-Computer Studies*, 49(6):831–865, 1998.

[6] N. R. C. Committee on Autonomy Research for Civil Aviation. Autonomy research for civil aviation: Towawrd a new era of flight, 2014.

[7] C. Hahn. A domain specific modeling language for multiagent systems. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 233–240. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[8] J. Hunter, F. Raimondi, N. Rungta, and R. Stocker. A synergistic and extensible framework for multi-agent system verification. In M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker, editors, *AAMAS*, pages 869–876. IFAAMAS, 2013.

[9] P. U. Lee, N. M. Smith, J. Homola, C. Brasil, N. Buckley, C. Cabrall, E. Chevalley, B. Parke, and H.-S. Yoo. Reducing departure delays in laguardia airport with departure-sensitive arrival spacing (dsas) operations. ATM 2015, NASA Ames Research Center, San Jose State University, NASA Ames Research Center, Moffett Field, CA, 2015.

[10] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, July 2005.

[11] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, Dec. 2005.

[12] M. J. North, N. T. Collier, and J. R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, 16(1):1–25, Jan. 2006.

[13] L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Agent-oriented software engineering III*, pages 174–185. Springer, 2003.

[14] N. Rungta, G. Brat, W. J. Clancey, C. Linde, F. Raimondi, C. Seah, and M. Shafto. Aviation safety: Modeling and analyzing complex interactions between humans and automated systems. In *Proceedings of the 3rd International Conference on Application and Theory of Automation in Command and Control Systems*, ATACCS '13, pages 27–37, New York, NY, USA, 2013. ACM.

[15] M. Sierhuis. *Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simulation language for work system analysis and design.* PhD thesis, Social Science and Informatics (SWI), University of Amsterdam, SIKS Dissertation Series No. 2001-10, Amsterdam, The Netherlands, 2001.

[16] R. Stocker, N. Rungta, E. Mercer, F. Raimondi, J. Holbrook, C. Cardoza, and M. Goodrich. An approach to quantify workload in a system of agents. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 1041–1050, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[17] A. Susi, A. Perini, J. Mylopoulos, and P. Gi. The tropos metamodel and its use. *Informatica*, 29(4), 2005.

[18] H. N. Swenson, J. Thipphavong, A. Sadovsky, L. Chen, C. Sullivan, and L. Martin. Design and evaluation of the terminal area precision scheduling and spacing system. ATM 2011, NASA Ames Research Center, NASA Ames Research Center, Moffett Field, CA, 2011.

[19] S. Tisue and U. Wilensky. Netlogo: A simple environment for modeling complexity. In *in International Conference on Complex Systems*, pages 16–21, 2004.