Timed Discrete-Event Simulation of Aviation Scenarios

Hai Nguyen Van, Frédéric Boulanger, Burkhart Wolff

Timed Discrete-Event Simulation of Aviation Scenarios

Hai Nguyen Van*, Frédéric Boulanger**, Burkhart Wolff**

Université Paris-Saclay, CNRS, LRI, 91405 Orsay, France; *ORCID 0000-0002-0585-1651 ; **firstname.lastname@lri.fr

https://www.sne-journal.org/sne-volumes/volume-30/sne-301-articles-1/timed-discrete-event-simulation-of-aviation-scenarios

Abstract. Ensuring systems behave as they are expected is unavoidable in the context of critical environments. In the aviation industry, certification standards provide rules and protocols to ensure correct maneuvers with respect to logical or timed events. These are targeted to computer-intensive systems as well as to human flight crews. In this setting, we are interested in the modeling and simulation of event-driven and time-driven behaviors at a high level. This study focuses on the TESL language [9] that provides a logical framework for timed behaviors with monitoring and testing features. In particular, we model various aviation scenarios and focus our study on fault monitoring.

Introduction

In past years, an increase in modeling and simulation in industry has emerged to assist engineers and designers of various process levels. In a broader way, this has been ensured by the emergence of Model-Based Design that allows the differentiation of stages and components composing large systems. These large systems consist of modeled components of various nature and form a multi-paradigm environment where each part is modeled with its own semantics of execution: this is commonly called *heterogeneous modeling* [23]. For instance in control systems, mode switches can be modeled with finite-state machines, and sensor data processors with dataflow models. Recent advances have proved that these submodels could be unified to form a *supermodel*. Figure 1 highlights this idea where each submodel is described by a different paradigm, then they are coordinated as a supermodel.

Complementary to modeling, the increased demand of automatic validation relates to the critically-large models where sole human analysis no longer suffices. The addition of mathematics and logics to the understanding of modeling and computing problems at a larger scale is named *formal methods*. The problem is two-fold. On one side, heterogeneous modeling raises the question of the adaptability of paradigms. Indeed, each modeling paradigm comes with a specific model of computation detailing a precise semantics of execution for each submodel. On the other side, validation demands a unified environment for safety property verification or test generation for these various paradigms. In the last decades, several multi-paradigm frameworks have appeared and attempt to address these issues, *e.g.*, Ptolemy II [32, 13], ModHel'X [8], BCool [37].

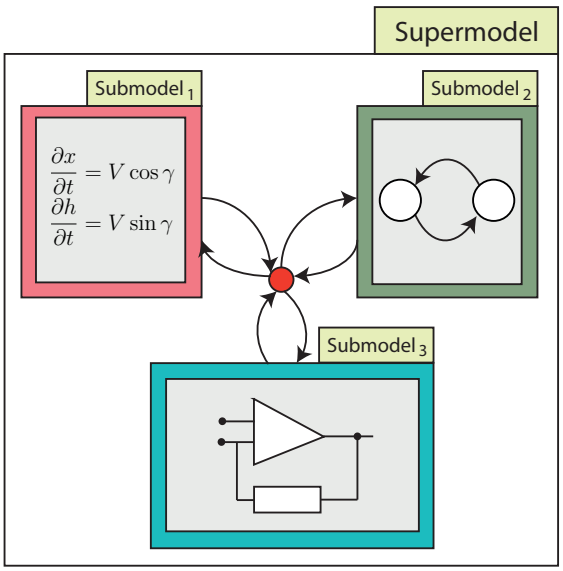


Figure 1: A supermodel for heterogeneous submodels

Our study focuses on TESL [9] which has been introduced as the inner language of the ModHel'X framework. Similarly to intermediate programming languages in compiler theory, models in ModHel'X are coordinated using TESL. Indeed, it is a specification language that describes discrete-events with time annotations (*tags*). This flavor of chronometric time is nec-

essary to compose modeled systems where events are described with respect to chronometric time durations, instead of logical time as would be found in temporal logics [31].

In this paper, we highlight the application of this language to aeronautical systems. We are interested in modeling standard scenarios used by common aircrafts, and validated by airworthiness authorities (ICAO Annex 8 [1], EASA AIR OPS [2]). Our goal is to exhibit a unified modeling framework suited for validation and verification by means of:

- multi-level model specification;
- generation of execution traces for simulation purposes;
- real-time testing and system monitoring.

In the heterogeneous context, our framework allows to abstract from programming details in order to reason on high-level behaviors. We believe this is particularly useful for aviation-related systems. Indeed, software in current airborne systems are usually certified for the highest Design Assurance Level as defined by RTCA/DO-178C [14, 33]. This certification is known to provide guidance for software life-cycle processes, and emphasizes on verifying the relation between high-level requirements and low-level implementation. The testing topic of this certification is especially focused on requirements. Our work precisely targets this problem by providing a specification-based testing/monitoring framework. For these reasons, we believe our study addresses several current Flight Control Systems (FCS):

Multi-pilot aircrafts. The Airbus A320 aircraft family is a well-known system made of fault-tolerant components based on redundancy and dissimilarity. Traverse [36, 35] reported that the aircraft primary control surface computers were designed by different design analysts on independent architectures. This case of heterogeneity clearly exhibits how a multi-paradigm environment needs to be unified in order to be validated at a higher-level.

Unmanned aircrafts. Furthermore, recent advances in unmanned aircrafts [39, 15] make the topic of verification and validation even more crucial due to the need of safe, reliable and fully automated software-intensive systems [19, 24] where, accordingly, design and test procedures tend to be fully automated.

1 The TESL Language

The core language of our study is the TESL language. It is inspired from the CCSL language [25, 7] and the Tagged Signal Model [21]. It lies at the heart of the ModHel'X modeling and simulation framework and serves as an intermediate representation for simulation solvers. In our setting, events are described and specified by *clocks*. A clock that *ticks* means that the associated event is occurring. Theses entities are ruled by three kinds of modality:

- *Event-driven implications.* An occurring event can trigger another one: "If clock K_1 ticks, then clock K_2 will tick under conditions".
- *Time-driven implications.* An occurring event triggers another one after a chronometric time delay measured on the time scale of a specific clock. Remark that this delay is a duration expressed as a difference between two tags, and not as a number of ticks.
- *Tag relations.* By default, clocks live in independent *time islands*. The purpose of tag relations is to link these different time scales, *e.g.*, time expressed in seconds and minutes admits an arithmetic relation stating that time flows 60 times as fast in seconds than in minutes. This does not mean that the seconds clock ticks 60 times more than the minutes clock, but simply that their tag annotation satisfies this arithmetic relation.

To provide a glimpse of the core features of TESL, we present a brief grammar of the language:

- `tag relation` $\lfloor K_1, K_2 \rfloor \in R$
  The time frames of clocks K_1 and K_2 are related by the arithmetic relation R.
- K_{evt} `sporadic` τ `on` K_{meas}
  Some event will occur on clock K_{evt} at timestamp τ measured on clock K_{meas}.
- K_{master} `implies` K_{slave}
  At every instant, if K_{master} ticks, then K_{slave} instantaneously ticks.
- K_{master} `sustained from` K_{begin} `to` K_{end} `implies` K_{slave}
  In the interval between a tick on clock K_{begin} and a tick on clock K_{end}, K_{master} implies K_{slave} as previously (scoped implication).

- K_{master} `time delayed by` τ `on` K_{meas} `implies` K_{slave}
  Whenever the master clock K_{master} ticks, the time tag on the measuring clock K_{meas} is measured and delayed by duration τ to yield the date of a future instant at which clock K_{slave} will tick.
- K_1 `strictly precedes` K_2
  Any event occurring on K_2 is preceded in the strict past by a distinct event occurring on K_1.

2 The Takeoff Scenario

Figure 2: Airspeed indicator and altimeter (courtesy of Laminar Research)

To illustrate how time scales can be constructed in TESL, we are interested in modeling the takeoff procedure of a small single engine aircraft with basic parameters: time, airspeed and altitude. We assume usual atmospheric conditions and specifically chose to model performance parameters extracted from the Cessna 172 aircraft [10, 22]. The rotation speed V_{R} specifies when the pilot should move the pitch control backwards to generate lift.

Speed	Description
$V_{\text{R}} = 55$ kt	Rotation speed

Table 1: Extract of V-speeds of the Cessna 172

In the next subsections, we first introduce an execution trace to provide intuitions for our case study. Then, we will exhibit the TESL specification for this scenario.

2.1 Clocks and execution traces

To define the basic quantities and events in which we are interested, we define *clocks* that describe the timeline of events. These are embedded with time tag annotations that rule how quantities and units are related. Each clock denotes a quantity with a specific unit:

```
// Declaring quantities and units
rational-clock time-S          // in [s]
rational-clock time-MIN        // in [min]
rational-clock speed-MPS       // in [m.s^-1]
rational-clock speed-KT        // in [kt]
rational-clock altitude-FT     // in [ft]
```

Moreover, the specification is augmented with three clocks for the events of our interest:

- `VR-reach`: speed reaches V_R;
- `liftoff`: the aircraft is airborne;
- `flaps-retract`: flaps are retracted.

The TESL language is a specification language that allows to describe traces. Figure 3 depicts a minimal execution trace with three instants. At the first instant, time is just 0 s. Then at the second instant, speed has reached $V_R = 55$ kt at 12.2 s. Hence, clock `VR-reach` is triggered, and so is `liftoff` consequently: they are said to be *ticking synchronously*. Then at the third instant, `flaps-retract` ticks at the altitude of 400 ft at 27.2 s.

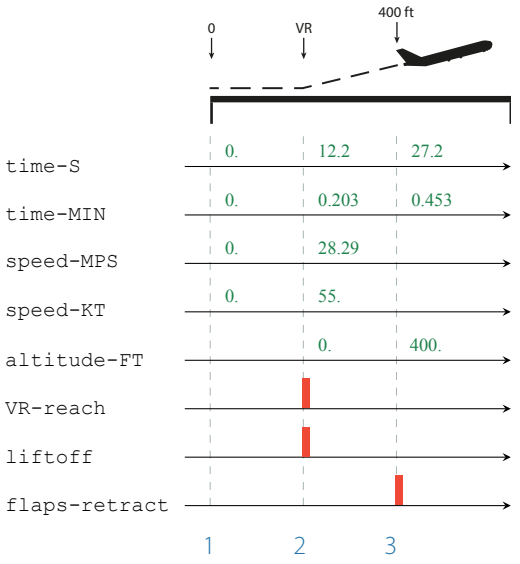


Figure 3: Execution trace when performing takeoff

Remark. The above trace depicts a minimal run. It illustrates an observation. Other instants may exist in between those mentioned, where none of our events occur. These instants are simply not "observed".

2.2 Causality and Timestamp Relations

Units In the next paragraphs, we use TESL to describe the above potential behavior.

```
// Unit conversion between [s] and [min]
tag relation time-S = 60.0 * time-MIN
// ... and between [m.s^-1] and [kt]
tag relation speed-KT = <3600/1852> * speed-MPS
```

Clocks `time-S` and `time-MIN` respectively express time given in seconds and minutes. Clocks `speed-MPS` and `speed-KT` respectively denote speeds in meters per second and knots (as given by the airspeed indicator). We use tag relations to describe unit conversions between such quantities. In particular, tags on clocks `time-S` and `time-MIN` shall satisfy the arithmetic relation that one minute is equivalent to sixty seconds. Likewise, one knot is equivalent to $\frac{3600}{1852}\,\mathrm{m}\,.\,\mathrm{sec}^{-1}$. In our context, the notion of time lies under tag annotations. Compared to temporal logics where time is purely logical, we precisely capture chronometric durations.

Acceleration and liftoff Tag relations also allow to describe how quantities are related and can define the acceleration profile of the modeled aircraft. Here we consider the uniform acceleration of a light aircraft gaining speed at $4.5\ \mathrm{kt}\,.\,\mathrm{sec}^{-1}$:

```
tag relation speed-KT = 4.5 * time-S
```

Remark. The permissive nature of the language also allows to leave this unspecified. We could have also required to design a general-purpose specification independently from the physical profile.

Whenever speed reaches V_R at 55 kt, the event `VR-reach` is triggered, and instantaneously triggers `liftoff` indicating that the aircraft is airborne.

```
VR-reach sporadic 55.0 on speed-KT
VR-reach implies liftoff
```

Flaps retraction To quickly reach the desired altitude, the pilot controls the pitch (longitudinal axis) to maintain the airspeed at fixed value V_y while climbing. This approximately corresponds to a vertical speed of $1200\ \mathrm{ft}\,.\,\mathrm{min}^{-1}$. The relation is written:

```
tag relation altitude-FT =
                1200.0 * time-MIN + -244
```

Finally, the aircraft reaches the altitude for flaps retraction at 400 ft.

```
liftoff time delayed by 400. on altitude-FT
        implies flaps-retract
```

3 The Autobrake System

In this section, we explore more complex specifications by employing a mix of event and time-triggered events, and the usage of sequential and asynchronous operators. We are interested in modeling the takeoff procedure of a transport-category aircraft with takeoff rejection components such as the Airbus A320 [3, 20]. As said earlier, decision, rotation and lift off do not necessarily coincide (depending on the aircraft performance category). As a matter of fact, the manufacturer distinguishes and specifies the following speed thresholds as illustrated in Table 2.

Speed	Description
$V_1 = 118$ kt	Decision speed
$V_R = 126$ kt	Rotation speed

Table 2: Extract of V-speeds of the A320 under given physical assumptions [4]

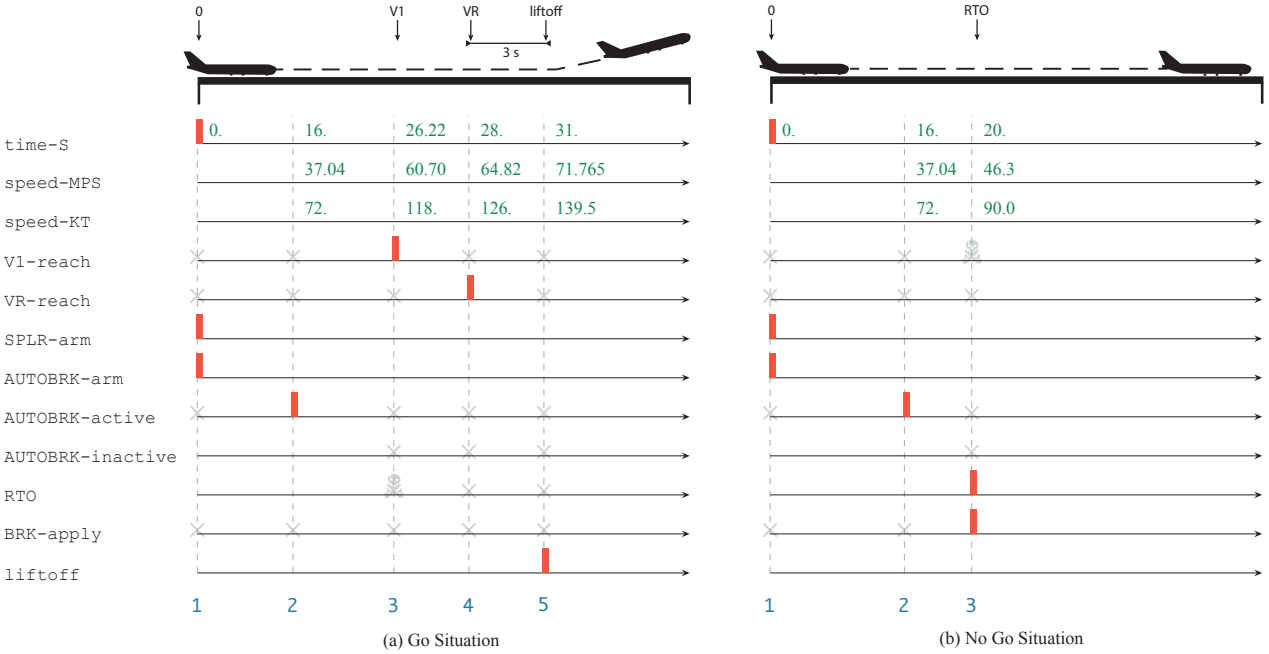
As depicted in Figure 4, decision speed V_1 defines the speed limit at which the pilot in command is allowed to reject takeoff (RTO). Then V_R is the rotation speed as previously described, and lift off occurs 3 s after. Should the pilot decide to reject after V_1, the aircraft would brake on a too short remaining runway, and hence overrun.

3.1 Acceleration and Speed Thresholds

We define speed thresholds as previously and also add a precedence constraint. This emphasizes on the fact that reaching V_R must have been preceded by reaching V_1 prior.

```
V1-reach strictly precedes VR-reach
V1-reach sporadic 118.0 on speed-KT
VR-reach sporadic 126.0 on speed-KT
```

Again, to define a specification independently from the aircraft performance, the precedence operator is



(a) Go Situation (b) No Go Situation

Figure 4: Execution trace when rejecting takeoff

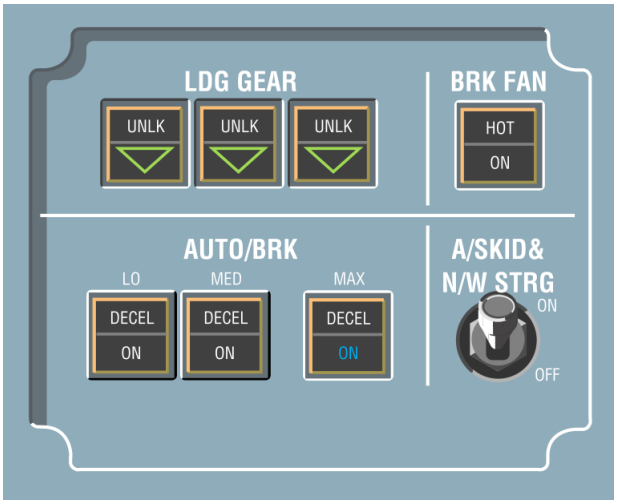


Figure 5: Autobrake command switch on panel board (extracted from [5])

enough and the two following lines could have been ignored. Finally, the aircraft is airborne 3 s after rotation speed has been reached.

```
VR-reach time delayed by 3. on time-S
                              implies liftoff
```

3.2 Rejecting Takeoff with Autobrake

One of the aircraft braking systems is named *Autobrake* [3], and opposes to manual pedal braking. Its usage is preferable as:

- The number of brake pressure is minimized, reducing brake wear;
- A symmetrical brake pressure is applied ensuring an equal braking effect on gear wheels, especially on wet runways.

The system is activated whenever the following conditions are met:

1. Ground spoilers ARMED
2. Auto brake ARMED
3. Speed exceeds 72 kt
4. Accelerate-stop with THRUST on IDLE

These above conditions are written as:

```
SPLR-arm strictly precedes AUTOBRK-active
AUTOBRK-arm strictly precedes AUTOBRK-active
AUTOBRK-active sporadic 72.0 on speed-KT
```

Finally in the event of takeoff rejection (RTO), the system applies brakes if autobrakes were in an active state.

```
RTO sustained from AUTOBRK-active
    to AUTOBRK-inactive implies BRK-apply
```

Event Death Killing a clock allows to prevent it from ticking, *i.e.* from having further event occurrences. This mechanism can be used to describe different forms of race conditions. Likewise, we can specify that reaching V_1 will prevent the event of rejecting takeoff. Conversely, applying brakes prevents from reaching V_1.

```
V1-reach kills RTO
BRK-apply kills V1-reach
```

3.3 Simulation

Two situations that satisfy the above specification are depicted in Figure 4. The first execution trace in Figure 4a shows the Go situation where the pilot normally proceeds to aircraft takeoff. In the first instant, the pilot arms ground spoilers and autobrake. Consequently, autobrake is activated when speed exceeds 72 kt. There is no takeoff rejection (clock `RTO`), and the aircraft keeps on accelerating by reaching V1 and VR speeds until being airborne (clock `liftoff`) after a delay of 3 s.

On the other side, a No Go situation is illustrated in Figure 4b where speed has exceeded 72 kt but takeoff rejection has been declared at 20 s when the speed is 90 kt, which immediately triggers brakes. This prevents from reaching V1 and consequently VR and aircraft liftoff.

4 Towards Hybrid Systems: Accelerate-Stop Distance

Our language also addresses the modeling of hybrid systems. This is exhibited by the ability to define differential equations in tag relations. In our case study, we can refine our specification to take into account differential quantities. In the case of a takeoff rejection, it is necessary to ensure that the two-stage acceleration-deceleration ensures that the aircraft does not overrun, and remains within the limits of the runway. To provide a good abstraction level to the reader, we chose to deliberately simplify the definition of the Accelerate-Stop Distance and not take into account the recognition and decision time as specified by airworthiness authorities ([2], CAT.POL.A.205 Take-off).

Acceleration To compute these distances, we need to express instantaneous distance with respect to current time and speed. If we denote x as the distance of the running aircraft, provided time t and speed v, we have

$$\mathrm{d}x = v.\mathrm{d}t$$

In our setting, this is straightforwardly expressed as

```
tag relation (d distance-M)
                  = speed-MPS * (d time-S)
```

where `distance-M` is the quantity denoting the run distance during acceleration phase.

Deceleration As done previously, we need to define how deceleration is expressed and accordingly the run distance. To proceed so, we will define new clocks for this deceleration stage: `speed-MPS-DECEL`, `speed-KT-DECEL` and `distance-M-DECEL`. Likewise, to keep our model simple enough but still relevant, we will assume a uniform deceleration of $-3\,\mathrm{kt}.\mathrm{s}^{-1}$. In TESL, this would be expressible as a linear tag relation as previously, or equivalently with a differential equation between speed and time:

```
tag relation (d speed-KT-DECEL)
                        = -3.0 * (d time-S)
```

Finally, distance during deceleration relates to the following tag relation:

```
tag relation (d distance-M-DECEL)
             = speed-MPS-DECEL * (d time-S)
```

To run our specification with a concrete example, let us assume that takeoff rejection has been declared at 20 s. Figure 6 illustrates this process where the aircraft has approximately reached 520 m at 20 s. Finally, the aircraft reaches speed zero at 50 s with a final run distance of approximately 1159 m.

Remark. The precision of the differential calculus lies in the ODE solver in use. In this example, we

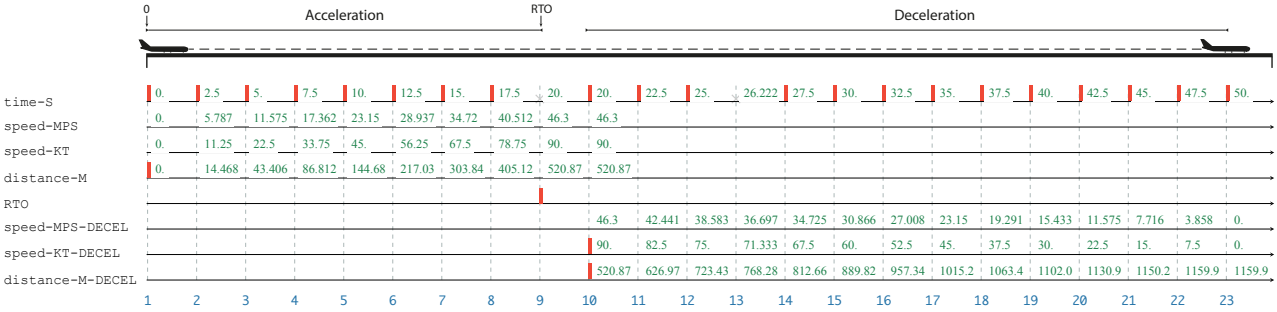


Figure 6: Execution trace when performing accelerate-stop

used the simple first-order forward Euler with an integration step of 2.5 s. Our language is agnostic to this design choice and potentially admits any other relevant ODE solver. It aims at bridging gaps between different paradigms generally found for modeling complex systems.

5 Formal methods

5.1 Testing and Monitoring

In the previous sections, we have observed that the TESL language specifies execution traces. These can be constructively generated with a solver named Heron [27]. It is a multicore-aware solver made of approximately 4,000 lines of Standard ML [26] code compiled with MPL [40]/MLton [38]. It solves TESL specifications by exhaustively constructing execution traces as illustrated in the previous paragraphs. In particular, this allows to exhaustively test and monitor systems to the extent of the system observation interfaces. Our solver fetches observations provided by an external driver and filters out irrelevant execution trace branches. By keeping the only satisfying runs of the specification combined with those compliant with the observation of the system, the solver keeps exploring exhaustive possibilities. Whenever the solver faces an unwanted behavior, it will finally filter out all branches and remain in an inconsistent state, meaning that the system has produced a violating behavior.

For this purpose, it is possible to suggest a scenario to the solver and request a simulation trace (if it exists). From the specification of the Autobrake in Section 3 and its corresponding satisfying execution traces in Figure 4, we can specify an additional directive to the solver and request a run where the clock `RTO` is the *only* one allowed to tick at instant 3. This is written

```
@scenario strict 3 RTO
```

The Heron solver will find no possibly satisfying run as it is not possible for `RTO` to tick alone without `BRK-apply` to tick as well. After successfully generating 2 instants, it will eventually fail and output the following:

```
##### Solve [1] #####
 -> Consistent premodels: 1
 -> Step solving time measured: 0.005 s
##### Solve [2] #####
 -> Consistent premodels: 1
 -> Step solving time measured: 0.016 s
##### Solve [3] #####
 -> Consistent premodels: 0
 -> Step solving time measured: 0.003 s
### ERROR: No further state found.
```

The solver, its source code and all mentioned examples in this paper are provided at `github.com/heron-solver/heron`.

5.2 Formalized Semantics

In an effort to fully validate our language and its logical foundations, a fragment of the TESL language, which consists of core formulae, has been proved to enjoy good and formal properties ensuring its well-foundedness. It has been formalized with two kinds of logical semantics providing an accurate meaning of language terms and how they are supposed to behave:

- a *denotational semantics* [28] mathematically describes the set of execution traces denoted by the language;
- an *operational semantics* [27] describes how the languages behaves/executes to generate traces.

These two semantics have been proved to be equivalent in the Isabelle/HOL proof assistant [29, 30]. This ensures that the language qualifies for two key properties:

compositionality The semantic composition of two models yields the semantics of the supermodel. This is notably emphasized by the property of *stuttering-invariance* which shows that the addition of observation instants does not "break" a run and preserves specification satisfiability.

executability The specification is constructive and allows the derivation of execution traces. This allows for trace generation for testing and simulation purposes.

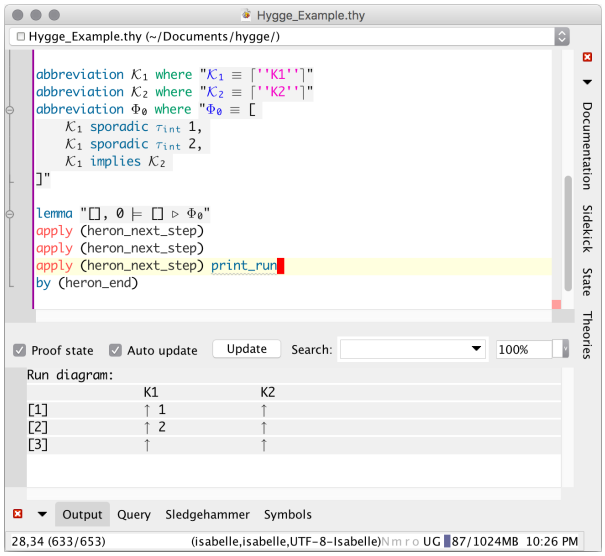


Figure 7: Executing the operational semantics in the Isabelle/HOL proof assistant

6 Related Work

The aim of our study and its purpose to the aviation community is similar to the goal of the real-time on-board Fault Detection and Diagnosis introduced by Goupil *et al.* [16] but differs in the scope of their study. Their study focuses on numerical aspects of signals and emphasizes on the link between academic and industrial R&D. Our research work aims at bringing new knowledge regarding timed aspects of discrete-event models.

Chhaya *et al.* [11] introduced the Aviation Scenario Definition Language (ASDL) which provides a Domain-Specific Language for scenarios of the same kind of our study. They provide an extensive framework to design aviation-related procedures. On the mathematical side, specifications consider only logical time and are translated into finite-state machines which employ traditional model-checking for verification purposes. Compared to our approach, our language aims at remaining a general-purpose and multi-level coordination framework, combined with the ability of specifying constraints containing chronometric time and physical quantities.

On a system-oriented level, Lustre/SCADE [17] is a well-known asset for the development of critical embedded systems. It has been qualified for DO-178B and is used for specifying flight control systems onboard the Airbus A340-600 and A380 [6]. Lustre/SCADE considers a unique and global driving clock for all specified components, whereas our framework allows to specify independent time islands where no global clock may exist. Moreover, time in SCADE is purely logical and does not consider chronometric time. In the problem we attempt to address, it may be necessary to consider the latter while designing and specifying real-time systems. Indeed, time delay constraints are crucial for closed loop controls of FCSs as time lags may exist due to computing, latency or storage [6].

Dealing with test oracles [12], our framework rightfully determines correctness on the outputs of a system. Indeed, the previously mentioned semantics allows to exhaustively generate execution traces that are correct with respect to specifications. The test oracle consists in filtering out generated branches which no longer satisfy the current outputs. The test oracle detects a violation whenever all possibly-satisfying branches have been filtered out.

7 Future Work

The permissive nature of the language allows to leave time relations between different quantities unspecified by default. Time related clocks can be gathered and are said to live in *time islands*. Leaving them unrelated means that they live independently. This feature is particularly interesting for *distributed computing* as it is not always possible to determine how time flows in one computing unit relatively to another. Yet, it has to be made sure that any computation is completed eventually. A similar synchronization mechanism can be found, for instance, in the Airbus A380 [34].

8 Conclusion

Our study highlights the TESL specification language as a unified environment for modeling and validation along the different stages of (1) designing models, (2) running their simulation, and (3) monitoring their runtime-compliance. We have presented a case study of fundamental operational scenarios found in the aviation industry, with high-level models addressing large-scale systems. In particular, our language and its associated high-level specifications are agnostic to concrete hardware implementations, providing a suitable framework for testing and monitoring systems similarly to black-box testing. We believe our framework also addresses the current trend for distributed computing [18] which is increasingly finding its way in critical embedded systems.

References

[1] *Annex 8 - Airworthiness of Aircraft*. International Civil Aviation Organization, 2018.

[2] *Easy Access Rules for Air Operations (Regulation (EU) No 965/2012)*. European Union Aviation Safety Agency, 2019.

[3] Airbus. *A318/A319/A320/A321 Flight Crew Training Manual*, 2002.

[4] Airbus. *A318/A319/A320/A321 Performance Training Manual*, 2006.

[5] Airbus. *Airbus A320 - Front Panel*, 2007.

[6] Ameur Y. A, Boniol F, and Wiels V. Toward a wider use of formal methods for aerospace systems design and verification. *International Journal on Software Tools for Technology Transfer 12* (2009), 1–7.

[7] André C. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Research Report RR-6925, INRIA, 2009.

[8] Boulanger F, Hardebolle C, Jacquet C, and Marcadet D. Semantic adaptation for models of computation. In *2011 Eleventh International Conference on Application of Concurrency to System Design* (June 2011), pp. 153–162.

[9] Boulanger F, Jacquet C, Hardebolle C, and Prodan I. TESL: a language for reconciling heterogeneous execution traces. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on* (Lausanne, Switzerland, Oct 2014), pp. 114–123.

[10] Cessna Aircraft Company. *Cessna Skyhawk Information Manual*.

[11] Chhaya B, Jafer S, and Durak U. Formal verification of simulation scenarios in aviation scenario definition language (asdl). *Aerospace 5*, 1 (2018).

[12] Durrieu G, Waeselynck H, and Wiels V. Leto - a Lustre-based test oracle for airbus critical systems. In *Formal Methods for Industrial Critical Systems* (Berlin, Heidelberg, 2009), D. Cofer and A. Fantechi, Eds., Springer Berlin Heidelberg, pp. 7–22.

[13] Eker J, Janneck J. W, Lee E. A, Liu J, Liu X, Ludvig J, Neuendorffer S, Sachs S, and Xiong Y. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE 91*, 1 (Jan 2003), 127–144.

[14] Ferrell T. K, and Ferrell U. D. RTCA DO-178C/EUROCAE ED-12C. In *Digital Avionics Handbook*. Taylor & Francis, 2017, pp. 16–1.

[15] Garcia R, and Barnes L. Multi-UAV simulator utilizing X-Plane. *Journal of Intelligent and Robotic Systems 57*, 1 (Oct 2009), 393.

[16] Goupil P, Dayre R, and Brot P. From theory to flight tests: Airbus flight control system TRL5 achievements. *IFAC Proceedings Volumes 47*, 3 (2014), 10562 – 10567. 19th IFAC World Congress.

[17] Halbwachs N, Caspi P, Raymond P, and Pilaud D. The synchronous data flow programming language lustre. *Proceedings of the IEEE 79*, 9 (Sep. 1991), 1305–1320.

[18] Hildenbrand Y. ED-247 (VISTAS) gateway for hybrid test systems. In *SAE Technical Paper* (10 2018), SAE International.

[19] Jung D, and Tsiotras P. Modeling and hardware-in-the-loop simulation for a small unmanned aerial vehicle. In *AIAA Infotech@ Aerospace 2007 Conference and Exhibit* (2007), p. 2768.

[20] Ladkin P. B. Analysis of a technical description of the Airbus A320 braking system. *High Integrity Systems 1* (1995), 331–350.

[21] Lee E. A, and Sangiovanni-Vincentelli A. L. The tagged signal model a preliminary version of a denotational framework for comparing models of computation. Tech. Rep. UCB/ERL M96/33, EECS Department, University of California, Berkeley, 1996.

[22] Linton J. O. The physics of flight: I. fixed and rotating wings. *Physics Education 42*, 4 (2007), 351.

[23] Liu X, Liu J, Eker J, and Lee E. A. *Heterogeneous Modeling and Design of Control Systems*. Wiley-Blackwell, 2005, ch. 7, pp. 105–122.

[24] Livadas C, Lygeros J, and Lynch N. A. High-level modeling and analysis of the traffic alert and collision avoidance system (TCAS). *Proceedings of the IEEE 88*, 7 (July 2000), 926–948.

[25] Mallet F. Clock constraint specification language: specifying clock constraints with UML/Marte. *Innovations in Systems and Software Engineering 4*, 3 (2008), 309–314.

[26] Milner R, Tofte M, and Macqueen D. *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA, 1997.

[27] Nguyen Van H, Balabonski T, Boulanger F, Keller C, Valiron B, and Wolff B. A symbolic operational semantics for TESL - with an application to heterogeneous system testing. In *Formal Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings* (2017), pp. 318–334.

[28] Nguyen Van H, Boulanger F, and Wolff B. A formal development of a polychronous polytimed coordination language. *Archive of Formal Proofs 2019* (2019).

[29] Nipkow T, and Klein G. *Concrete Semantics: With Isabelle/HOL*. Springer Publishing Company, Incorporated, 2014.

[30] Nipkow T, Wenzel M, and Paulson L. C. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[31] Pnueli A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* (Oct 1977), pp. 46–57.

[32] Ptolemaeus C, Ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[33] RTCA DO-178C. *Software Considerations in Airborne Systems and Equipment Certification*. RTCA Inc., Washington, D.C., 1992. This document is also known as EUROCAE ED-12C in Europe.

[34] Sánchez-Puebla M. A, and Carretero J. A new approach for distributed computing in avionics systems. In *Proceedings of the 1st international symposium on Information and communication technologies* (2003), Trinity College Dublin, pp. 579–584.

[35] Spitzer C, Ferrell U, and Ferrell T, Eds. *Digital Avionics Handbook*. Electrical engineering handbook series. Boca Raton: CRC Press, 2015.

[36] Traverse P. Airbus electrical flight controls: A family of fault-tolerant systems. In *Digital Avionics Handbook*. Taylor & Francis, 2015, pp. 31–1.

[37] Vara Larsen M. E, Deantoni J, Combemale B, and Mallet F. A behavioral coordination operator language (BCOoL). In *18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015)* (Aug. 2015).

[38] Weeks S. Whole-program compilation in MLton. In *Proceedings of the 2006 Workshop on ML* (New York, NY, USA, 2006), ML '06, ACM, pp. 1–1.

[39] Weibel R. E, and Hansman R. J. Safety considerations for operation of unmanned aerial vehicles in the national airspace system. Tech. rep., 2006.

[40] Westrick S, Yadav R, Fluet M, and Acar U. A. Disentanglement in nested-parallel programs. *Proc. ACM Program. Lang. 4*, POPL (Dec. 2019).