Formal Verification of a Gain Scheduling Control Scheme

Pablo Ordóñez, Andrew R. Mills, Tony Dodd
Automatic Control and Systems Engineering Department
University of Sheffield
S1 3JD Sheffield, United Kingdom
[paordonezaguileta1,a.r.mills,t.j.dodd]@sheffield.ac.uk

Jun Liu
Department of Applied Mathematics
University of Waterloo
Waterloo, ON, Canada N2L 3G1
j.liu@uwaterloo.ca

***Abstract*—Gain scheduling is a commonly used closed-loop control approach for safety critical non-linear systems, such as commercial gas turbine engines. It is preferred over more advanced control strategies due to a known route to certification. Nonetheless, the stability of the system is hard to prove analytically, and consequently, safety and airworthiness is achieved by burdensome extensive testing. Model checking can aid in bringing down development costs of such a control system and simultaneously improve safety by providing guarantees on properties of embedded control systems. Due to model-checking exhaustive verification capabilities, it has long been recognised that coverage and error-detection rate can be increased compared to traditional testing methods. However, the state-space explosion is still a major computational limitation when applying model-checking to verify dynamic system behaviour. A practical methodology to incrementally design and formally verify control system requirements for a gain scheduling scheme is demonstrated in this paper, overcoming the computational constraints traditionally imposed by model checking. In this manner, the gain-scheduled controller can be efficiently and safely generated with the aid of the model checker.**

I. Introduction

Gain scheduling is a commonly used control scheme for non-linear processes. It is appealing due to its simplicity compared to more advanced non-linear control methodologies. In safety-critical systems it is extensively used (e.g. commercial jet engines) and it is implemented in the form of embedded software [16, 13]. For safety reasons the software undergoes extensive verification and validation practices. Airworthiness certification requires evidence to show the correct behaviour of the system prior to operation, which can be done with gain scheduling but not with an adaptive system scheme [3]. However, current development and certification practices are prone to human error and requirements ambiguities [4, 9]. Demonstrating safety conformity for a gain scheduling controller is challenging from both design and implementation points of view. In this paper, for the first time, a gain scheduling scheme is formally designed and verified using model checking. The formal verification of control requirements is enabled by the proposed modelling methodology. The gain schedule is incrementally constructed using the model checker. The end result consists of a gain schedule with the minimum number of controller tunings to satisfy requirements.

The usual approach to embedded control is to design analogue controllers and digitize them for implementation in a computer-based system (Fig. 1). To guarantee safety and conformity with requirements extensive testing is performed. It is estimated that current testing activities amount to approximately 30% to 50% of the total cost of a software project [1]. It is therefore desirable to find a new approach to verification and validation.

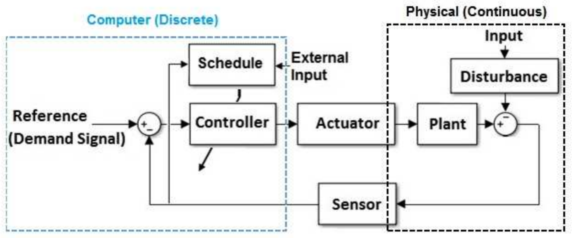


Fig. 1: Hybrid control system example. Continuous process regulated by a computer-based controller.

The development and certification processes of safety critical control systems can be improved by the incorporation of formal methods (e.g. model checking). Benefits include an increase in testing coverage, early error-detection, and requirements clarification [2, 7, 15]. However, their usage in industry (e.g. automotive, aerospace) is not a common practice [15] and has not yet been applied to gain scheduling control.

Formal methods are also used to synthesize controllers using requirements as a formal input, known as correct-by-design approach. The synthesized controller is one of symbolic nature - e.g. a state machine [5, 17]. However, so far the correct-by-design approach does not contemplate common controller structures (e.g. PID). It is highly desirable to enable common modelling practices for control systems in a model checking environment thus allowing control engineers to exploit the benefits of model checking.

To design and validate a controller the most common requirements are [12]:

1) Settling time.
2) Overshoot.

3) Rise time.
4) Steady State error.

Current model checking tools are not aimed to address such type of requirements despite these being the most common in control design. It is desirable to increase the range of requirements that can be formally verified [15]. Potential benefits of doing so include automatic test case generation, correct-by-design approach for software generation, and the early detection of requirements collision. However, challenges need to be addressed to make this a reality. Floating-point arithmetic is not fully supported in model checking which creates a challenge when modelling feedback controllers [18, 10].

In this paper a novel methodology is proposed to systematically generate a control schedule using model checking. An abstraction and modelling framework is proposed so that common control requirements can be verified using model checking. The framework is then used to solve a gain scheduling design problem. Starting from one controller tuning for an arbitrary operating region, the schedule is incrementally constructed using the model checker to verify the requirements compliance in all the operating space. In this manner control system requirements can be formally verified with a push-button approach.

The rest of this paper is structured as follows. Section II presents the problem formulation and the abstraction methodology. Section III presents the required elements to implement the abstraction in a model checker environment. Section IV presents a case study to show the applicability of the methodology. Finally, Section V presents conclusions.

II. Problem Formulation

The problem to address can be stated as follows: to guarantee that a non-linear process controlled by an embedded gain scheduling controller is safe and meets design requirements.

Usually the design methodology to address this problem is [8, 14]:

1) Partition the operating space.
2) Obtain linear models of the partitions.
3) Tune a controller for each partition.
4) Extensive testing to verify and validate requirements.

For verification and validation activities, current software engineering practices include unit testing, integration testing, and acceptance testing [11]. Model checking is an exhaustive verification technique and by using it during design and verification phases benefits can be obtained (e.g. test case generation, and coverage increase).

When modelling a system such as Fig. 1 the continuous part is usually represented using Ordinary Differential Equations (ODE). From a programming point of view it is easier to work with the discrete version of the system because no integration routine is needed which is computationally expensive. Also, the controller is implemented in a discrete manner using difference equations. A discrete representation of the system is thus convenient for both modelling and implementation purposes.

For the purpose of this paper Single-Input Single-Output (SISO) Linear Time Invariant (LTI) models are selected for modelling the system. A similar case can be made using Multiple-Input Multiple-Output (MIMO) models. An abstraction technique is proposed so the gain scheduling problem can be addressed using SISO LTI models within a model checker environment.

A. Discrete SISO LTI Models

In order to model the system and the controller in the gain scheduling problem formulated in Section II, discrete SISO LTI models have been selected. Discrete SISO LTI models are described by an auto-regressive with exogenous input (ARX) model:

$$\frac{Y(z^{-1})}{U(z^{-1})} = \frac{b_1 z^{-1-n} + b_2 z^{-2-n} + ... + b_{n_b} z^{-n_b-n}}{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_{n_a} z^{-n_a}} \qquad (1)$$

where the output of the system is *Y*, the input is *U* and the system response delay to the input is *n*. The order of the system is determined by the number of coefficients *a* ($\mathrm{n_a}$) and *b* ($\mathrm{n_b}$). The output calculation is therefore the weighted sum of previous input and output values:

$$Y(k) = \sum_{i=1}^{n_a} a_i Y(k-i) + \sum_{i=1}^{n_b} b_i U(k-i-n) \qquad (2)$$

Inputs, outputs, and coefficients are real numbers which are best represented by floating point variables. The use of floating point data-type is currently very limited for model checking and rounding does not provide good results due to the fractional part containing dynamics information. To overcome this limitation a scaling approach which uses integer-type computations only to simulate discrete SISO LTI models is proposed.

B. Abstraction

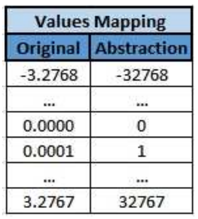
Floating point arithmetic is not supported by most tools or has limited use (e.g. clock variables with set and reset operations). To overcome this limitation and to be able to recover the system's dynamics using integer only data a scaling approach in combination with a fixed-point representation is proposed [6]. There are two main components:

1) Coefficients *a* and *b* representation.
2) Input-Output representation.

To address item 1, coefficients are scaled and rounded up by a fixed gain value which allows to recover 4 decimal places. The weighted sum (2) is performed in this scaled manner and in the end the scaling effect is removed by dividing the result by the same gain value.

To address item 2 a fixed point approach is followed. A representation is constructed over the 5 digits available in a 16-bit signed integer. If the input-output relationship is normalized then a 1 digit with 4 decimal places representation is generated (Fig. 2). In this manner the input-output values are scaled-up. The selected gain to do so depends on the needed resolution.

The abstraction methodology allows to recover the system's dynamics in a scaled-up fashion. The resolution will depend on



Values Mapping	
Original	Abstraction
-3.2768	-32768
...	...
0.0000	0
0.0001	1
...	...
3.2767	32767

Fig. 2: Mapping between original floating point values and fixed point integer representation.

the selected gains to do the scaling and numeric representation. Due to scaling and rounding operations there is error during the calculations. To measure the inaccuracy of the abstraction, a comparison with the original system (no data type restriction) needs to be performed.

As an example, Fig. 3 shows the comparison of the open loop response of % generated thrust for two operating points in a jet engine model (further details in Section IV). Each operating point has a particular dynamic for which a linear model is generated. Afterwards the abstraction is generated and the same input is fed into it.

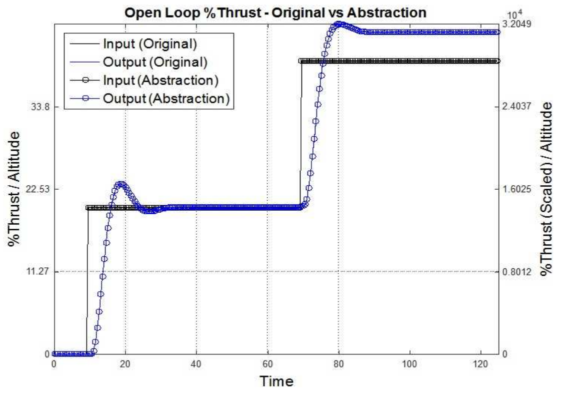


Fig. 3: Open loop response for two operating points. Original model vs abstractions.

To quantify the amount of error in the abstraction, overshoot is compared in both operating points (Table I). The abstraction consisted of a 4 decimal digits mapping criteria, a 4 digits fixed-point arithmetic. Differences are less than 0.1% and as shown in Fig. 3 the abstraction allows to recover the system's dynamics. A comparison must be performed to measure differences and compensate when using the abstraction for design and verification purposes. Once the abstraction has been generated it needs to be implemented in a model checking environment so the system can be formally verified. The following section covers how to implement the abstraction in a model checking environment.

III. Modelling Within a Model Checker Environment

To simulate the generic system in Fig. 1 and verify control requirements in a gain scheduling control scheme (Section I) within a model checker environment, a set of automata are proposed (Fig. 4). Using the abstraction technique presented in Section II-B scaled-discrete SISO LTI models are then implemented in the model checker.

The automata design is driven by both the gain scheduling problem and the control requirements to be verified (1-4 from Section I). In this manner the requirements verification will be performed using the model checker formulae query language using a push-button approach.

A. Model Checking Automata

Model checking tools can be classified by their modelling language (Java, C, PROMELA, LOTUS, etc.), properties language (LTL, CTL, PCTL, assertions, etc.), and the nature of the system they are intended to verify (probabilistic, plain, hybrid, real-time, timed, etc.). Within real-time systems, the model checker UPPAAL is designed to model systems as networks of timed-automata with integer variables, structured data types, clocks, and channel synchronization. UPPAAL offers a modelling environment which gives the user freedom to program tailor-made functionalities for the timed-automata. For this reason it was selected to implement the abstraction and perform the formal verification.

To verify if the system complies with requirements the model checker uses properties. There are 3 types of properties available in UPPAAL:

1) Reachability: It is possible to reach a system state.
2) Safety: Something can never happen.
3) Liveness: Something will eventually happen.

System modelling is strongly driven by the type of available properties and requirements to be verified. Therefore a *design for verifiability* approach is taken. In order to verify requirements 1-4 from Section I the automata have to be designed so that a control requirement can be translated into a property. The following automata are proposed to achieve this translation. The *Plant* automata generates the process output and monitors it. Because requirements are related to the process output it is in this automata where requirements are portrayed so they can be verified. The *Controller* automata is in charge of generating the control action. The *Observer* automata synchronizes the correct execution of events between the *Controller* and *Plant* automata. In more detail:

- *Observer*: Automata in charge of synchronising the controller and plant execution. This automata monitors controller and plant outputs to determine transitions. It is in charge of deciding which controller configuration to select, when to trigger an event, and when to stop the verification process.
- *Plant*: Automata in charge of simulating the process under control. When the plant is required to generate a new output a transition is triggered from the *Settled* to the *Transient* state. Both *Rise Time* and *Overshoot* states are included to perform the verification of such requirements.
- *Controller*: Automata in charge of simulating the gain scheduled PID-type controller. When the controller is

TABLE I: Original System vs Abstraction Differences.

Operating Point	Overshoot % - Original	Overshoot % - Abstraction	Error %
1	16.3163	16.3996	0.0833
2	4.5508	4.5426	0.0081

required to generate a new output a transition is triggered from the *Settled* state to the *Processing* state. Once the control signal has reached an equilibrium with the plant the *Settled* state is reached.

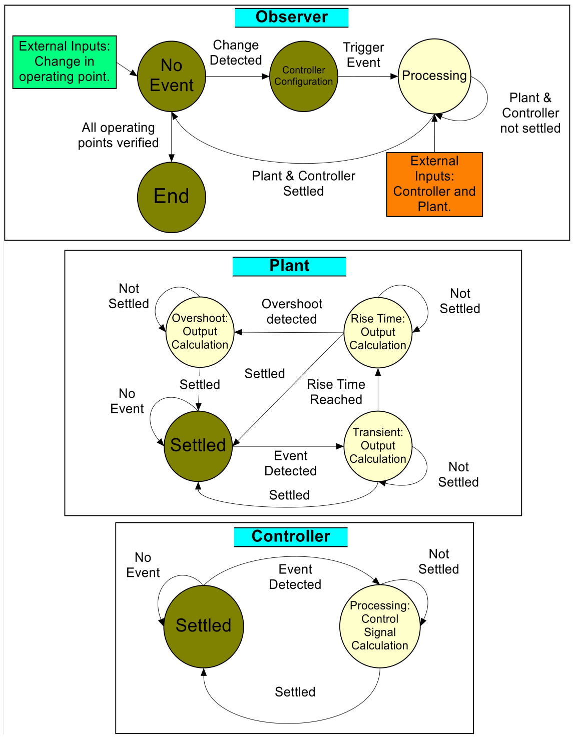

Observer
External Inputs: Change in operating point.
No Event
Change Detected
Controller Configuration
Trigger Event
Processing
Plant & Controller not settled
All operating points verified
End
Plant & Controller Settled
External Inputs: Controller and Plant.
Plant
Not Settled
Overshoot: Output Calculation
Overshoot detected
Rise Time: Output Calculation
Not Settled
Settled
Settled
Rise Time Reached
No Event
Settled
Event Detected
Transient: Output Calculation
Not Settled
Settled
Controller
No Event
Event Detected
Not Settled
Settled
Processing: Control Signal Calculation
Settled

Fig. 4: Proposed automata to simulate the generic control system, Fig. 1.

By using properties to verify requirements the model checker can return a *witness* or a *counter example* trace. A *witness* trace contains the actions that lead to a requirement being fulfilled and a *counter example* trace contains the actions that lead to a requirement not being fulfilled. In this manner the designer can obtain information about the system from the model checker. To verify control requirements (Section I) a *reachability* property will be used so that a *witness* trace can be obtained as feedback.

B. Gain Schedule Verification Methodology

Once the control system is implemented in the model checker environment using the abstraction methodology (Section II-B) and timed automata (Section III-A), control system requirements can be formally verified. The following algorithm shows the steps to formally design and verify a gain schedule control scheme.

The objective of this design process is to generate a control schedule with the minimum necessary control tunings in order to meet requirements for all operating regions. The schedule design is incremental. After the operating space has been split, an operating region is arbitrarily selected and the controller is tuned to meet requirements in that region. The model checker is then used to verify if requirements are met for all the other regions as well. If not, another region where requirements are not met is selected and the process is repeated. The number of available tunings increases and this also allows the model checker to use those tunings as options in order to meet requirements. The outcome after applying Algorithm 1 will be a control schedule which meets design requirements and has been formally verified.

Algorithm 1: Control Schedule Design and Verification Procedure.

```
Input  : Non-linear model, Performance Requirements.
Output: Control Schedule.
1 Partition the operating space into $M$ regions;
2 Obtain a linear model for each of the $M$ operating
  regions (2);
3 Use classical control methods and design a controller for
  operating region 1;
4 Use the abstract methodology (Section II-B) and
  implement the abstraction in the model checker;
5 Use the model checker to verify requirements for all $M$
  operating regions using the available control tunings;
6 If requirements are met for all regions, cross-check in the
  original model;
7 If not met, design a controller for the operating region
  which does not meet requirements;
8 Update model abstraction with the new designed
  controller tuning;
9 Go back to step 5, repeat;
```

IV. Case Study

A. Thrust Control System

Consider a commercial jet-engine, thrust is regulated using a PID controller with a gain scheduling scheme. The process dynamics will vary depending on the operating point: factors such as altitude and temperature generate a non-linear behaviour [16].

Fig. 5 shows the behaviour of the control system. There are five operating regions ($M = 5$). The objective is to design a control schedule to meet a given set of performance requirements for all operating regions.

B. Requirements

The system must comply with certain performance requirements. The following illustrative requirements will be

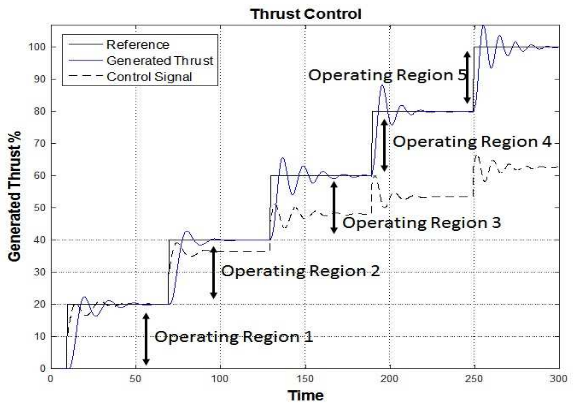


Fig. 5: Thrust control consisting of 5 operating regions. Each region has a particular dynamic behaviour.

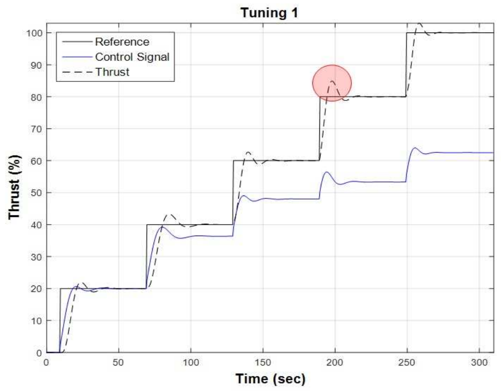


Fig. 6: Results after the first iteration. Region 4 fails to meet overshoot requirement.

verified. These requirements are specifically chosen so that the methodology in Section III-B is demonstrated. They apply for all operating regions:

1) Overshoot % (OS) $\leq$ 10%.
2) Settling Time (ST) $\leq$ 40 seconds.
3) Rise Time (RT) $\leq$ 15 seconds.
4) Steady state error % (SSE) $\leq$ 1%.

The requirements verification is thus performed using a push-button approach by querying the model checker using a *reachability* property - $E<>$:

$$E <> Observer.End \quad \text{and} \quad Plant.OS \leq 10\% \quad \text{and} \quad Plant.ST \leq 40 \quad \text{(seconds) and} \quad Plant.RT \leq 15 \quad \text{(seconds) and} \quad Plant.SSE \leq 1\% \tag{3}$$

Equation (3) shows the verification of requirements 1-4 for a single operating region using the proposed automata (Fig. 4). The query can be read as: *there exists a path where the observer has reached a final state, overshoot is less than or equal to the specification, settling time is less than or equal to the specification, rise time is less than or equal to the specification, and steady state error is less than or equal to the specification.*

C. Verification Results and Discussion

Algorithm 1 is applied to the problem formulated in Section IV-A. Initially the operating space is split into 5 regions. After verifying requirements for all regions with the controller designed for region 1 it is found that region 4 does not meet requirements. The overshoot verification fails because the overshoot state is visited when in that region. The overshoot value is 19.9% which was confirmed using the original model.

A second controller tuning is designed around region 4. The verification is run again and after two iterations the model checker returns a schedule where all requirements are met across all regions, i.e. only two controller tunings are needed to meet requirements across all regions. A controller tuning is selected for each region:

- Region 1: Tuning 1 - Initial tuning.
- Region 2: Tuning 1.
- Region 3: Tuning 1.
- Region 4: Tuning 2 - Designed after 1 iteration.
- Region 5: Tuning 1.

Fig. 7 shows the comparison between the initial controller tuning applied in all regions and the final schedule consisting of 2 controller tunings. As expected, because the only region where tunings differ is region 4, it can be observed that tuning 1 has a higher overshoot in region 4, where it originally failed to meet requirements.

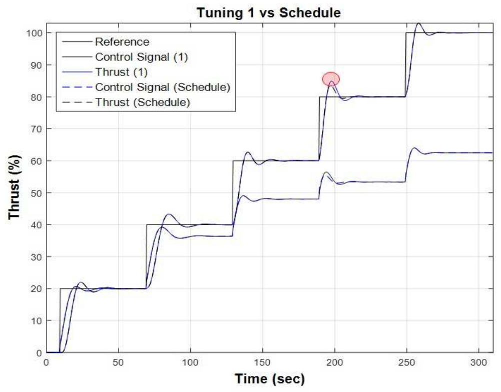


Fig. 7: Final schedule consisting of 2 controller tunings versus initial tuning from iteration 1. The final schedule consists of 2 different tunings for 5 operating regions.

Current software design practices rely on a trial and error approach. Verification and validation use requirement-driven testing and complement it with corner cases (e.g. worst case scenarios). The use of model checking enables a push-button approach to verify several requirements at the same time. Reasoning about the control system in an automated manner saves time compared to trial and error which allows to explore more cases during verification. The formality of model checking makes the design more robust against human errors.

Modelling in the model checker is driven by problem formulation. The design has to consider the type of requirements to be verified, generating an ad hoc solution. In this manner more benefits can be exploited from the model checker. Nonetheless results in the model checker have to be cross-verified in the original model. Results show that the methodology is accurate to reason about the original system using the abstraction. The use of both tools in combination to solve the problem shows the benefits of the methodology.

V. Conclusions

For the first time a methodology to formally verify a gain scheduling control system is proposed. The type of requirements which were formally verified include overshoot, settling time, rise time, and steady-state error. The methodology enables the use of model checking to aid during the design and verification phases of a gain scheduling control system. A model abstraction is generated by the means of a scaling fixed-point approach which uses integer data-type only to overcome the data-type limitation in model checking. The abstraction allows to recover system dynamics and model feedback control systems without using floating-point data. The methodology also enables the use of a typical control system model such as discrete SISO LTI in a model checker. This makes the transition to a model checking environment more understandable for the designer.

References

[1] C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking*. MIT press, 2008.

[2] M. Bennion and I. Habli. A candid industrial evaluation of formal software verification using model checking. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 175–184. ACM, 2014.

[3] S. Bhattacharyya, D. Cofer, D. Musliner, J. Mueller, and E. Engstrom. Certification considerations for adaptive systems. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 270–279. IEEE, 2015.

[4] J. Fan, J. Jiao, W. Wu, and T. Zhao. A model-checking oriented modeling method for safety critical system. In *Reliability Systems Engineering (ICRSE), 2015 First International Conference on*, pages 1–6. IEEE, 2015.

[5] A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2010.

[6] R. Gordon. A calculated look at fixed-point arithmetic. *Embedded Systems Programming*, 11(4):72–79, 1998.

[7] N. Y. Jeppu, Y. Jeppu, and N. Murthy. Arguing formally about flight control laws. In *Industrial Instrumentation and Control (ICIC), 2015 International Conference on*, pages 378–383. IEEE, 2015.

[8] D. J. Leith and W. E. Leithead. Survey of gain-scheduling analysis and design. *International journal of control*, 73(11):1001–1025, 2000.

[9] J. Markovski. An integrated systems engineering framework for supervisor synthesis, verification, and performance evaluation. In *Control Conference (ECC), 2013 European*, pages 650–657. IEEE, 2013.

[10] L. K. Martin, M. Schatalov, M. Hagner, U. Goltz, and O. Maibaum. A methodology for model-based development and automated verification of software for aerospace systems. In *Aerospace Conference, 2013 IEEE*, pages 1–19. IEEE, 2013.

[11] N. M. A. Munassar and A. Govardhan. A comparison between five models of software engineering. *IJCSI*, 5:95–101, 2010.

[12] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron. The rosace case study: From simulink specification to multi/many-core execution. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 309–318. IEEE, 2014.

[13] M. Pakmehr, N. Fitzgerald, E. M. Feron, J. S. Shamma, and A. Behbahani. Gain scheduled control of gas turbine engines: Stability and verification. *Journal of engineering for Gas turbines and power*, 136(3):031201, 2014.

[14] W. J. Rugh and J. S. Shamma. Research on gain scheduling. *Automatica*, 36(10):1401–1425, 2000.

[15] D. Sexton, P. Gilhead, and R. Quadir. Practical experiences of using formal requirements and their role in an overall work-flow. *System Safety Conference incorporating the Cyber Security Conference 2013, 8th IET International*, 2013.

[16] H. A. Spang III and H. Brown. Control of jet engines. *Control Engineering Practice*, 7(9):1043–1059, 1999.

[17] P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *Automatic Control, IEEE Transactions on*, 51(12):1862–1877, 2006.

[18] K. F. J. N. Valkonen, Janne; Björkman. Model checking methodology for verification of safety logics. *SIAS 2010 - The 6th International Conference on Safety of Industrial Automated Systems*, 2010.