Wind Turbine System : An Industrial Case Study in Formal Modeling and Verification

Jagadish Suryadevara[1], Gaetana Sapienza[2],
Cristina Seceleanu[1], Tiberiu Seceleanu[2], Stein-Erik Ellevseth[2], and Paul Pettersson[1]

[1] Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden.
{jagadish.suryadevara,cristina.seceleanu,paul.pettersson}@mdh.se
[2] ABB Corporate Research.
{gaetana.sapienza,tiberiu.seceleanu}@se.abb.com,
stein-erik.ellevseth@no.abb.com

Abstract. In the development of embedded systems, the formal analysis of system artifacts, such as structural and behavioral models, helps the system engineers to understand the overall functional and timing behavior of the system. In this case study paper, we present our experience in applying formal *verification and validation* (V&V) techniques, we had earlier proposed, for an industrial *wind turbine system* (WTS). We demonstrate the complementary benefits of formal verification in the context of existing V&V practices largely based on *simulation* and *testing*. We also discuss some modeling *trade-offs* and challenges we have identified with the case-study, which are worth being emphasized. One issue is related, for instance, to the *expressiveness* of the system artifacts, in view of the known limitations of rigorous verification, e.g. *model-checking*, of industrial systems.

Keywords: Industrial Case-Study, Wind Turbine System, MARTE/CCSL, EAST-ADL, Verification, Model Checking, UPPAAL

1 Introduction

The increasing complexity and criticality of real-time embedded systems (RTES), in domains such as industrial automation, automotive and avionics, stresses the need for applying systematic design phases, combined with rigorous *verification and validation* (V&V) techniques, during system development [3]. A well-defined design process with necessary tool support leads to ensuring system *predictability*, w.r.t intended functional and timing behavior. Nevertheless, meeting such a clear objective has several challenges. One of pre-requisites is well-defined system artifacts representing system *structure* as well as *behavior* with *reactive*, *continuous*, *discrete*, and *real-time* features, or a combination thereof, at suitable *levels-of-abstraction*. For complex industrial systems, the above design by-products, while necessary, may lead to additional issues such as ensuring *traceability*, *analyzability* as well as *reusability* of the system artifacts. In this context, model-based development approaches, which enable continuous V&V throughout the development process, have become a feasible solution to tackle some of the challenges. However, formal verification techniques such as *model checking*, while

useful for the exhaustive analysis of system behavior, are challenging to apply for complex system models. A related issue is choosing a suitable level of *granularity* and *expressiveness* for system artifacts, given the well-known limitations of model-checking, such as the *state-space explosion* problem. In this paper, we address some of these challenges in the context of applying modeling and formal verification techniques using a *wind turbine system* case-study, a complex industrial RTES.

The Unified Modeling Language (UML) provides a modeling profile called MARTE (Modeling and Analysis of Real-Time and Embedded systems) [7] to support the *performance* and *schedulability* analysis of system models. MARTE also includes CCSL – a time model and a *clock constraint specification language* [1] for specifying logical and chronometric constraints for system models. On the other hand, EAST-ADL [2], an emerging standard for automotive systems, provides an integrated model-based development for RTES, through well-defined phases, as well as support for *traceability*. Recently, EAST-ADL has been integrated with *model-checking* support for component-based designs, e.g. the ViTAL tool [4] based on the timed automata technology for verification [5,11,10].

In this paper, we target the verification of functionality and timing behavior of a *wind turbine* system developed in the context of the iFEST (industrial Framework for Embedded Systems Tools), a ARTEMIS project. In Section 2.2, we overview a simplified version of the *wind turbine system* (WTS), and describe its functionality and timing behavior. Rest of the paper is organized as follows: In Section 3, we briefly recall CCSL and timed automata. In Section 4, we describe a modeling methodology for the WTS to enable verification using *model checking*. The analysis results of simulating, as well as model checking the WTS model are presented in Section 5. In Section 6, we discuss our experience with the case study with respect to the challenges and limitations in applying formal techniques to complex industrial systems. We conclude the paper in Section 7.

2 Windturbine System (WTS) : An overview

Wind energy sources are fast-growing and in line with the technological advancement. Modern wind turbine systems require sophisticated and effective control functionalities in order to fulfill *performance*, *safety*, and *maintainability* requirements. The main purpose of a wind turbine system is to convert the rotational mechanical energy of the rotor blades (i.e. mechanical components of a wind turbine) caused by the wind into electrical energy to be redistributed via a power grid. Given the system's complexity, the iFEST (industrial Framework for Embedded Systems Tools) project[3] aims at providing a model-based approach for system development, to ensure the system *predictability* w.r.t the specified functional and timing behavior.

2.1 Development Process and Environment

In the iFEST project, we have carried out the system development by adopting the V-model based software development approach, as follows:

[3] http://www.artemis-ifest.eu/

During *Requirement and Analysis* phase, we have documented the WTS requirements, both functional and extra-functional including timing behavior. For the *Design* phase, we have combined component- and model-based approaches, keeping in view the overall system *analyzability* and *reusability* requirements. During the *Implementation* phase, we have applied automatic code generation technologies.Subsequently, the implemented system, a combined FPGA and CPU solution, has been deployed on a heterogenous hardware platform (XilinX ZynQ 7000 product family). For the *Verification and Validation* (V&V), we have used model-based techniques as follows: (i) simulation of the WTS functionality using Simulink and related toolboxes, and (ii) automatic model-based test-case generation with MaTeLo tool. However, the above techniques are not sufficient to ensure system *predictability* w.r.t to all possible system executions, hence formal verification is desirable to complement the current analysis methods. To address the above open issue, in this paper, we present a verification technique towards enhanced system validation. And, our contributions are as below:

- As enhanced system validation, we apply verification technique to establish system properties, (partially) based on simulation results of Simulink-based system models.
- We are able to verify safety requirements that involve timing behavior (e.g. "the wind turbine moves to `Park` mode, within 30s of detecting that the wind speed has crossed the upper limit of 20m/sec").

2.2 The Wind Turbine System Model

The wind turbine system is modeled as a *closed-loop* control system, as shown in Figure 1. The key components are the *Plant* and the *Controller* subsystems. The *Controller* dynamically regulates the rotor blades of the *wind turbine* w.r.t the specified wind profile, to maximize the generation of electrical energy and also to avoid damage to the plant in case of turbulent wind scenarios. It automatically changes the *Controller Output* signal to regulate the plant, based on the wind and the plant's actual conditions, which are received by the *Controller* via the *Sensor Input* signals. The *Wind Profile* and the *Resistive Load* are used to simulate and test the behavior of the plant and the controller, under specific wind and resistive load conditions. Further details of the plant and controller subsystems are described below.

2.2.1 Plant model. As shown in Figure 2 (in Section 4), it consists of three main components; *Servo*, *Rotor*, and *Generator*. The pitch of the turbine, determined by the *Controller* (described below), is actuated by the *Servo*. The *Rotor* produces the required *torque* to maximize the angular speed of the *Generator* (which produces the final voltage), based on the pitch value as well as the current wind speed (we assume a fixed *resistive load*). The *Rotor* optimizes the produced torque value based on the current angular speed of the *Generator*.

2.2.2 Controller model. As shown in Figure 3 (in Section 4), it consists of four main components: the *Filter*, the *Main Controller*, the *Pitch Controller*, and the *Park*

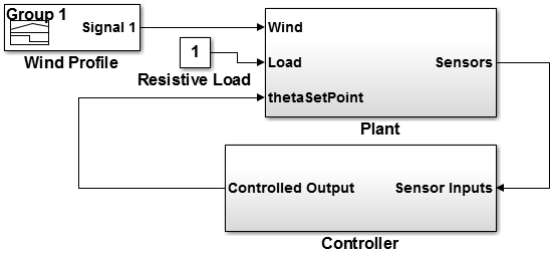

Fig. 1. Wind Turbine System Model

and Brake Controller. The *Filter Subsystem* is responsible for transducing, filtering and scaling the wind signal and plant signal (for instance the rotational speed of the turbine), which are used by the *Main Controller* and the *Pitch Controller*. Based on the inputs received through the *Filter*, the *Main Controller* directs the overall control. It oversees the performance and operations of the turbine in order to maximize the energy production and prevent any damage to the plant. Based on the wind and plant state, the controller determines the operational mode (i.e. park, start-up, generating, or brake) of the turbine. The *Pitch Control* calculates the proper pitch i.e. angle to steer the rotor blades when the turbine starts up or generates power. The *Pitch and Brake* controller determines if the turbine needs to brake or park, to ensure the safety of the wind turbine, for instance, during wind turbulances.

3 Preliminaries

In this section, we present an overview of the preliminaries needed for modeling of the wind turbine system. We have used EAST-ADL modeling framework for structural modeling of the WTS. The timed causality behavior of the system is specified using CCSL. To provide the verification using the UPPAAL, a model checking tool, we have develop the timed automata based semantic models of the system, based on the corresponding EAST-ADL models and the CCSL specifications.

3.1 EAST-ADL

The modeling process in EAST-ADL framework, developed in the context of the EAST-EEA project, is structured into different abstraction levels such as *feature level*, *analysis level*, *design level* etc. At both analysis and design levels, the system is described by a *FunctionalArchitecture* that consists of a number of inter-connected *FunctionPrototypes* (instantiation of *FunctionType* components). FunctionProtoype components are either *event-* or *time*-triggered. The execution semantics of the EAST-ADL components is as follows; components interact through single buffer, *rewritable*, *non-consumable* ports, and execute in *read-execute-write* phases in *run-to-completion* fashion. The detailed timing behavior as well as timing constraints for an EAST-ADL model can be specified using TADL2, the Timing Augmented Description Language (ver 2), currently being integrated with EAST-ADL framework [8]. In related works, we have proposed verification techniques for TADL2-based EAST-ADL models [5,11,10].

3.2 CCSL

CCSL is used to specify the constraints imposed on the logical *clocks* (activation conditions) of a model. A CCSL clock is defined as a sequence of *instants* (event occurrences). CCSL constraints are of three kinds: (i) *Synchronous* constraints rely on the notion of *coincidence*. For example, the constraint "`a coincidesWith b`", denoted by `a` $\boxed{=}$ `b`, specifies that each instant of `a` coincides with the corresponding instant of `b`. Another example of a synchronous constraint is "`a isPeriodicOn b period n`" , which specifies the subclock `a` whose 'ticks' correspond to every n^{th} 'tick' of b. (ii) *Asynchronous* constraints are based on instant *precedence*; the constraint "`a isFasterThan b`" (denoted by `a` $\boxed{\preccurlyeq}$ `b`) specifies that clock `a` is (non-strictly) faster than clock `b`. (iii) *Mixed* constraints combine *coincidence* and *precedence*; the constraint "`c = a delayedFor n on b`" specifies that `c` 'ticks' synchronously with the n^{th} 'tick' of `b` following `a` 'tick'.

3.3 Timed Automata

A timed automaton is a tuple $< L, l_0, C, A, E, I >$, where L is a set of $locations$, $l_0 \in L$ is the initial location, C is the set of clocks, A is the set of actions, synchronization actions and the internal τ-action, $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard, a set of clocks to be reset, and $I : L \rightarrow B(C)$ assigns clock $invariants$ to locations. A location can be marked *urgent* (u) or *committed* (c) to indicate that the time is not allowed to progress in the specified location(s), the latter being a stricter form indicating further that the next transition can only be taken from the corresponding location(s) only. Also, synchronization between two automata is modeled via $channels$ (e.g., **x**! and **x**?) with rendezvous or broadcast semantics.

The UPPAAL model-checker extends the timed automata language with a number of features such as global and local (bounded) integer variables, arithmetic operations, arrays, and a C-like programming language. The tool consists of three parts: a graphical editor for modeling timed automata, a simulator for trace generation, and a verifier for the verification of a system modeled as a network of timed automata. A subset of CTL (computation tree logic) is used as the input language for the verifier. For further details, we refer to UPPAAL tutorial [6].

4 WTS: Formal Specification and Modeling

In this section, we present a formal specification and modeling approach for WTS, an *aposteriori* modeling technique, that is, the specification and modeling artifacts are based on existing design artifacts such as Simulink models, requirements documents etc. However, we apply an *abstraction* strategy to obtain the corresponding real-time semantic models that represent the system functionality as well as the timing behavior. Further, the strategy attempts to preserve the models' *tractability* to make the exhaustive verification feasible. The overall modeling strategy, based on design principles such as *separation-of-concerns* and *correctness-by-construction*, captures the underlying *model-of-computation* and the execution behavior of the WTS. Below, we outline some generic views/assumptions on which we base our formal modeling:

- ***Plant models and Instantaneous executions***. A *Plant* model represents physical devices such as *sensors* and *actuators* with the corresponding *model-of-computation* based on *reactivity* and *instantaneity*.
- ***Controller models and Timed executions***. *Controllers* contain software components based on timed *model-of-computation*, with explicit timing aspects, such as *delay*, *execution time*, *end-to-end deadline* etc, to be formally specified and modeled.
- ***Time and event triggering***. The activation or triggering of RTES components is generally based on specific *time* or *event*[4] occurrences. *Plant* components are event-triggered (i.e. in response to occurrence of input data), whereas *controller* components are time- or event-triggered, this primarily being a design-choice.
- ***Run-to-completion***. RTES components execute in *run-to-completion* steps, that is, in terms of *read-execute-write* cycles.
- ***Data and value semantics***. Due to the associated *models-of-computation*, as described above, a data entity at a given 'instant', in the *Plant* or *Controller*, may correspond to two distinct value instants.
- ***Real-time features***. The structural and behavioral models of RTES often fail to model real-time features, such as *urgency*, *priority*, and *synchrony* (explained later) w.r.t to the underlying execution model.
- ***Environment modeling***. An *environment* is external to the system, representing physical parameters such as *temperature*, *pressure*, *wind speed* etc. To support formal verification, a modeling strategy based on *non-determinism* as well as the properties to be verified, is needed.

To obtain an *expressive* and *verifiable* semantic model of the WTS, we employ a component-based modeling approach, based on real-time formalisms such as CCSL and timed automata.The overall modeling approach is as follows:

- Data and event representations are made based on the structural models.
- The timed causality behavior of the system components, w.r.t the associated *model-of-computation*, is formally specified using CCSL constraints.
- The functional behavior of the components is modeled using an abstract finite-state-machine notation, and transformed into timed automata.
- The CCSL constraints are transformed into timed automata, and composed using the notion of *synchronization product* (described later).

Finally, a real-time semantic model of the overall system is obtained as a network (i.e., a *parallel composition*) of timed automata described above.

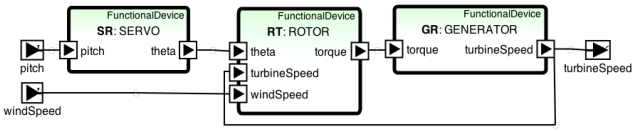

Fig. 2. Structural modeling: a *plant* model for the WTS.

[4] While time is also an 'event', we differentiate this in this paper explicitly.

In Figure 2, we present the structural model of *plant* and *controller* for the WTS (based on the corresponding Simulink models), using the EAST-ADL modeling framework (in MetaEdit+[5]). The main components of the plant, that is SERVO, ROTOR, and GENERATOR are modeled as `FunctionalDevice` prototypes in EAST-ADL. In Figure 3, we present the structural model of the Controller. It models the three sub-controllers MainControl, PitchRegulation, and ParkBrake, modeled as `AnalysisFunctionTypes`. For further details of the functionality of these components, we refer to Section 2.2. We demonstrate the overall modeling approach for WTS, using the ROTOR and the MainControl components, below. We will also discuss some related modeling issues.

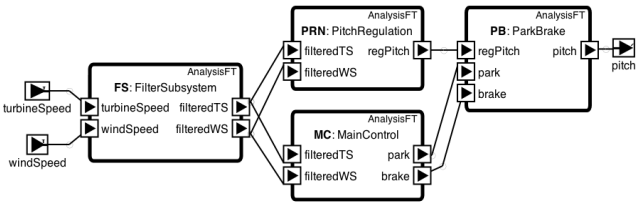

Fig. 3. Structural modeling: a *controller* model of the WTS.

4.1 Data and Events

As shown in Figure 2, the ROTOR prototype, denoted by RT, receives input *pitch* (`theta`), *turbine speed* (`omega`), and *wind speed* (`ws`) and produces the corresponding `torque` value as the output. Hence, we define the local variables $theta_l$, $omega_l$, and ws_l and the corresponding global variables ws_g, $omega_g$, $theta_g$. The local variables are updated at the activation of the RT using the corresponding global values. This is consistent with the data semantics discussed previously.

```
1 CCSLclock RT_in;          // read (input) instants
  CCSLclock RT_out;          // write (output) instants
3 CCSLclock RT_omega;        // activation (trigger) instants

5 CCSLconstraint
  RT_omega = RT_in;     //RT_omega coincidesWith RT_in
7 RT_in = RT_out;        //RT_in coincidesWith RT_out
```

Listing 1.1. CCSL specification of ROTOR component.

4.2 Specification of timed causality behavior

The timed causality behavior of real-time components, w.r.t the corresponding *model-of-computation*, can be specified precisely using CCSL logical clocks and CCSL constraints. We use CCSL (logical) clocks to represent events corresponding to 'read', 'execute', and 'write' instants of a component. In Listing 1.1 and Listing 1.2, we present the CCSL specification of ROTOR (RT) and MainControl (MC)

[5] www.metacase.com

Table 1. Timing attributes of *Controller* components.

Component	Period (ms)	Min Execution Time (ms)	Max Execution Time (ms)
MainControl	100	10	15
PitchRegulation	50	35	45
ParkBrake	50	15	20
Filter	–	20	25

prototypes, respectively. The constraints specify the timed causality behavior of the components w.r.t to the corresponding *model-of-computation*. For instance, the CCSL constraints for RT specify the *reactivity* and *instantaneity* behavior of RT execution within the *Plant* model. On the other hand, the CCSL constraints for MC specify the time-triggered behavior of the *controller* execution. The timing attributes of the controller components are given in Table 1. The CCSL specifications provide a basis for constructing real-time semantic models e.g. timed automata based models, as well as *observers* to establish the system properties, as presented later in this section.

```
1 CCSLclock MC_in          //read (input) instants
  CCSLclock MC_out         //write (output) instants
3
  CCSLconstraint
5 MC_in delayedFor 10 on SysClk ≼ MC_out   //Minimum execution time
  MC_out ≼ MC_in delayedFor 15 on SysClk   //Maximum execution time
7 MC_in isPeriodicWith period 100 on SysClk   //Time triggering
```

Listing 1.2. CCSL specification of MainControl component.

4.3 Modeling functional behavior of real-time components

In Figure 4, we present the behavior modeling for the MainControl protoype (based on the corresponding Simulink model). The behavior is specified using a *finite-state-machine* (FSM) notation. It represents the overall system behavior (stateful) in terms of control states PARK, START, BRAKE, and GENERATE. The states represent the operational *modes* of the WTS, based on the *wind speed* and the *turbine speed*; the mode transitions corresponding to mode-change behavior are triggered by boolean conditions (guards) *g1*, *g2*, .. etc. Further, we simply annotate the behavior model to denote the execution semantics such as *run-to-completion* (R-T-C) and *history* (denoted by the control node H). The functionality of other components in the WTS are stateless computations, that is partial functions between input and corresponding output values, for instance as represented by the *writeTorque*() function of the ROTOR.

4.4 Formal modeling of *Plant* components

In this subsection, we present formal modeling approach, based on CCSL, for the plant components of the WTS. We had earlier proposed, in a previous work [10],

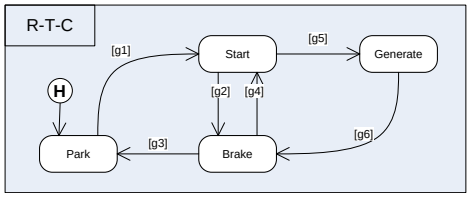

Fig. 4. Functional behavior of the MainControl component.

transformation of CCSL constraints into timed automata. The transformations can be used to derive timed automata based models that represent the timed causality behavior of the system. For instance, in Figure 5.(a) and 5.(b), we present the timed automata semantics of CCSL constraints that specify the timed causality behavior of ROTOR executions (see Listing 1.1), using events `RT_in` and `RT_out` representing component activation and termination respectively. Note that an event e.g. `RT_in` is modeled using synchronization channels i.e. send/receive signals `RT_in!` and `RT_out!`. Also note that the *synchronous* occurrence of event signals, e.g. `GR_out?` and `RT_in!` in Figure 5.(a), is specified using *committed* locations. A *committed* location indicates that the corresponding enabled transitions from the location are 'forced' before time can progress. This facilitates precise modeling of overall timing behavior of the system.

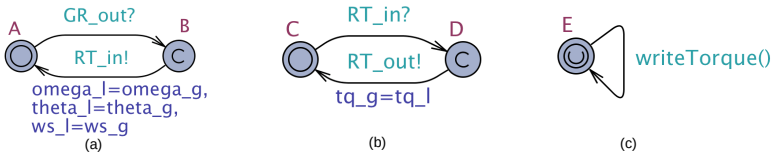

Fig. 5. Timed automata modeling: (a) `RT_omega` = `RT_in` (b) `RT_in` = `RT_out` (c) Computation RT.

The above automata can be composed, using the notion of *synchronization product* (based on common labels or synchronization signals), as shown in Figure 6.(a). For instance, locations B and C in the automata in Figure 5.(a) and Figure 5.(b) respectively, are mapped to the location BC in Figure 6.(a), due to the synchronization of signals `RT_in`! and `RT_in`?.

Fig. 6. Timed automata model for (event-triggered) ROTOR.

It can be noted that the composed location 'BD' is not possible in the synchronized product automaton, as the location is *non-reachable* due to the synchronization at B and C, leading to location AD (i.e. location A, and D simultaneously in resp. automata) in the synchronized product, instead. Further, as shown in Figure 6.(a), we

associate the transitions corresponding to component activation, with data updates and the corresponding computation; the `RT_in` event denotes input as well as execution of the corresponding functionality, during a transition from location BC to location AD. However, to make the overall automata model of the WTS system *tractable* (time-wise), and hence formally verifiable, we need to relax the notion of *instantaneity* for the automata models of the *Plant* components. This can be done by introducing a minimum time delay for each component, if not specified already. This is done by assigning a timing *invariant*, the delay of one time unit, for instance at location AD in Fig.6.(b).

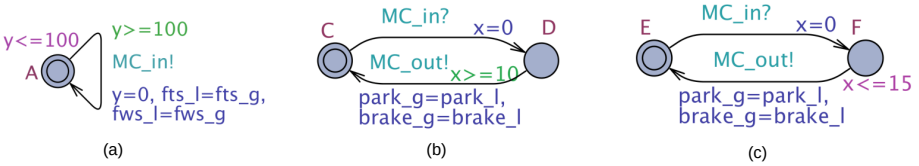

Fig. 7. Semantic modeling: (a) Periodic triggering (b) Min. exec. time (c) Max. exec. time

4.5 Formal modeling of *Controller* components

In this subsection, we describe the timed automata modeling of the *Controller* components for the WTS. In Figure. 7, we present the timed automata semantics of the CCSL constraints (Listing 1.2) that specify the time-triggered execution behavior of the MainControl (MC) prototype. We have composed these automata, as shown in Figure 8.(a), based on the notion of *synchronization product* (as described in the previous subsection). This consists of following steps; we have composed the automata in Figure 7.(b) and 7.(c), and then finally with the automaton in Figure 7.(a) (note the invariant $y \leq 100$ at every location in the product automaton).

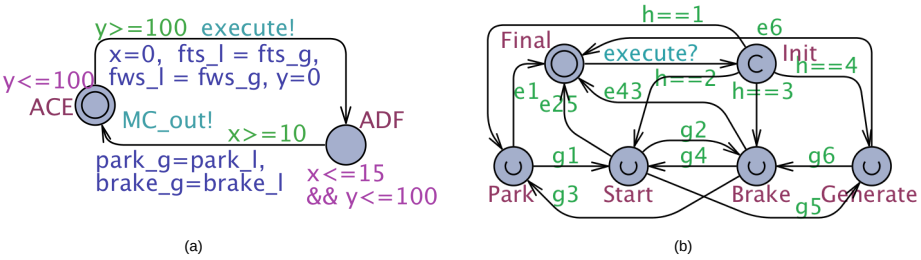

Fig. 8. Timed automata modeling of MainControl: (a) time-triggering (b) functional behavior.

As shown in Fig.8.(b), we have also transformed the behavior (functional) model of the MainControl (Fig.4) component into corresponding timed automaton, following the mapping techniques proposed previously [9]. We briefly outline the transformation as follows; we have mapped the control states to automaton locations. Further, using additional locations *Init* and *Final* and the history variable '*h*', we have modeled the execution semantics, that is, *run-to-completion*, and preserving the *history*. For model

readability, we have not shown the data updates for the transitions; also, the boolean guards of the form 'eij' correspond to actual expression ($\neg$**g**i && $\neg$**g**j). It can be noted that all the locations of the transformed automaton are marked 'urgent' indicating the behavior model does not consume time, which has been separately modeled using the timed causality model discussed above. Finally, we 'connect' the transformed behavior model of the MainControl prototype, as described above, with the automata model of the corresponding timing behavior (Fig.8.(b)), using synchronization channel 'execute'.

4.6 Modeling the WTS system

Following the modeling strategy presented in the previous subsections, we can obtain the timed automata models for all the WTS components, and form a network (parallel composition) of these automata to obtain a timed automata based semantic model for the complete system. However, some issues exist as discussed below:

Modeling the Environment: The plant model described previously, models the components such as sensors and actuators constituting an *environment* model for the WTS *controller*. However, this is not sufficient to obtain a 'closed' model of the system that is necessary to enable exhaustive verification of the WTS model. For instance, modeling external parameters such as WindSpeed, while necessary, is not feasible using timed automata. In view of this, as well as the hybrid nature of the plant components e.g. ROTOR, GENERATOR etc, we choose to integrate the simulation data of the corresponding Simulink models, to construct the partial functions that represent the computations of the components.

Modeling 'observer' automata: The formal specification of complex properties of the system, while possible using CTL (the property specification formalism of UPPAAL), may not be directly verifiable. Instead, these can be intuitively modeled as *observer* automata, (parallel) composed with the main system model, and can be efficiently verified.

5 WTS Analysis

In this section, we present both simulation as well as the verification results for the WTS, and their correlation in verifying functional and *safety-critical* properties w.r.t the overall timing behavior of the system.

5.1 Simulation

The main purpose of simulating the WTS, using the MathWorks Simulink and StateFlow[6], is to analyze the system behavior under normal operating conditions, and to validate the system (in particular the *Controller*) when the *wind speed* exceeds the

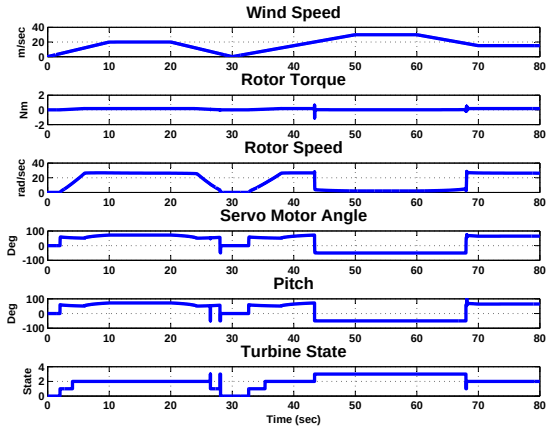

Fig. 9. Simulation Results

designed limit. The simulation results are presented in Figure 9.

The simulation time is step-wise incremented from 0 up to 80 sec., with a fixed sample time equal to 1 msec. For the simulation, a specific wind speed profile has been created. According to this, the system is simulated for normal operating limits, i.e., 5 - 20 m/sec up to 30 sec, then up to 30m/sec above 43 sec. The simulation results are analyzed w.r.t the turbine control states representing the operational modes (i.e. 0:park, 1:start, 2:generate, 3:brake).

While the simulation provides rich data representing the computation and control of the WTS w.r.t complex environment behavior, system *properties* however can not be established without analyzing the data. In the next subsection, we present a verification technique to 'exhaustively' analyze the simulation data, w.r.t the overall system timing and causality behavior, towards establishing the system properties. Below, we describe some verification results for the WTS system.

5.2 Verification

For WTS, a formal modeling of the corresponding *plant* and the *environment* parameters is not possible. Hence, we use simulation data and construct partial functions (input to output values) that represent the computations of the *plant* components, for instance ROTOR. Also, we use simulation values corresponding to the environment parameters e.g. *wind-profile* of the WTS. In the next section, we will discuss some aspects about the construction of the relevant partial functions.

Verification of functional properties: Verification of functional properties gives insight into the overall system (architectural) design. For instance, in the WTS case, it is useful to verify the following property: "if the *wind speed* is within the prescribed limits, the

[6] http://www.mathworks.se/products/stateflow/

controller eventually moves to `Generate` mode". The property can be formulated as a liveness property or `leads_to` property (denoted by $\rightsquigarrow$, implemented as `-->` in UPPAAL), as below.

$$(\texttt{ws>=5 \&\& ws<=20}) \texttt{ --> state==2} \tag{1}$$

Verification of safety-critical properties: One of the safety-critical requirements for the WTS is to fulfill the following property: "the wind turbine moves to `Park` mode, within 30s from detecting that the wind speed has crossed the upper limit of 20m/sec". To verify the property (w.r.t to simulation data), we construct an *observer* automata for the property as shown in Fig.10, compose the observer with the system model, and verify that the corresponding *invariant*, the Property (2), holds for the composed model. Note that the *urgent* channel 'U!' forces the transition from location B to A without any further delay, when the corresponding transition is enabled.

$$A\Box \text{ obs.B implies } x <= 30 \tag{2}$$

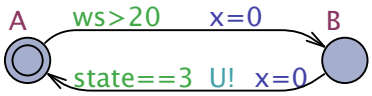

Fig. 10. An observer automata to verify the safety-property: `A[] obs.B implies x<=30`

Verifying reachability properties: We can verify *reachability* of specific control states or computation. For instance, using the Property 3, a *reachability* property, we can verify that the control state 'Park' (Figure 8.(b)) has been reached (at least once) during the simulation of the WTS. While this may be easily validated using the simulation trace, we can use similar properties to verify specific 'error' states e.g. by extending the behavior model with special 'locations' that are reached if the corresponding 'error' is detected. The presence of these error locations in the simulation data can then be 'exhaustively' verified.

$$\exists <> \text{ MC.Park} \tag{3}$$

Verifying deadlock-freeness: Using the Property 4, we can verify that the system is *deadlock-free*, w.r.t overall timed causality behavior of the WTS, as modeled by the corresponding timed automata model. The property is an important validation of the system, which can not be achieved using simulation only, as the corresponding Simulink model does not represent the timing behavior of the system explicitly. Also, the property, when satisfied, verifies the correctness (i.e. *consistency*) of the timing attributes (Table 1) associated with the system (architectural) design.

$$A\Box \texttt{(not deadlock)} \tag{4}$$

6 Discussion and Lessons-learned

In this paper, we have presented a formal modeling and verification approach for an industrial system, namely a *wind turbine system*. The main goal of the work has been to provide formal verification as a complementary analysis method to existing validation techniques based primarily on simulation. We have successfully addressed the following challenges:

- *Abstract but expressive system models*: Using real-time formalisms such as CCSL and timed automata, we were able to construct intuitive system models amenable for exhaustive verification (w.r.t to timing). With the separation of timing and functional modeling, the technique is scalable to complex system models.

- *Verification as complementary analysis to simulation*: The verification is based on 'exhaustively' analyzing the simulation data w.r.t the timing behavior of the system. While verification models are expressive in terms of system structure and precise timing behavior, simulation models are suitable to specify *plant* and the *environment*, e.g. 'wind profile' modeling in the case of WTS simulation. Thus, the verification approach provides an enhanced simulation-based validation.

However, some limitations of our approach do exist. The exhaustiveness of the verification is limited to partial functions constructed using specific instance(s) of simulation. Hence, the approach may be similar to testing-based analysis (albeit model-based). Hence, we need strategies, e.g. choosing suitable simulation *step* and data profiles, to generate simulation data w.r.t the system properties to be verified. Further, it may be noted that the simulation-extended verification approach presented above may be suitable for data-intensive control systems (e.g. hybrid systems), such as the *wind turbine system* case study presented in the paper. On the other hand, control-intensive systems may be exhaustively modeled and verifiable using *model-checking* independent of simulation.

7 Conclusion

In this paper, we have presented a formal modeling and verification approach for an industrial case-study, namely an example *wind turbine system*. The architectural and behavioral modeling, partially based on the existing system artifacts such as Simulink-models, additionally captures precise timing behavior of the system. The modeling approach, based on the real-time formalisms such as CCSL and timed automata, also integrates simulation data to model *plant* and *environment* behavior. Based on this, the proposed verification technique using *model-checking*, enhances the simulation-based system *validation*. Besides verifying functional properties that validate correctness of the system design, *safety-critical* properties w.r.t the overall system timing behavior can also be verified. This is clearly an important analysis step forward within existing validation approaches for industrial applications. Thus the paper addresses V&V challenges in the industrial context, by combining both simulation and verification techniques, paving the way towards scalable application of *model-checking* for an enhanced

validation process. As future work, we intend to investigate *requirement*-driven strategies to derive the simulation criteria for generating relevant partial functions. This leads to enhanced validation process that can verify useful classes of system properties.

Acknowledgment

This work was partially funded by Swedish Research Council (project ARROWS) and Mälardalen University (Sweden).

References

1. André, C., Mallet, F., de Simone, R.: Modeling Time(s). In: Models'07. LNCS, vol. 4735, pp. 559–573. Springer (2007)
2. ATESST (Advancing Traffic Efficiency through Software Technology): East-ADL2 specification (March 2008), http://www.atesst.org, 2008-03-20
3. Bouyssounouse, B., Sifakis, J.: Embedded Systems Design: The ARTIST Roadmap for Research and Development (Lecture Notes in Computer Science). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
4. Enoiu, E.P., Marinescu, R., Seceleanu, C., Pettersson, P.: Vital : A verification tool for east-adl models using uppaal port. In: ICECCS'12 (July 2012)
5. Goknil, A., Suryadevara, J., Peraldi-Frati, M.A., Mallet, F.: Analysis Support for TADL2 Timing Constraints on EAST-ADL Models. In: ECSA 2013 : 7th European Conference on Software Architecture. p. 10 pages. LNCS, Montpellier, France (Jul 2013)
6. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. Int. Journal on Software Tools for Technology Transfer 1(1–2), 134–152 (Oct 1997)
7. OMG: UML Profile for MARTE, v1.0. Object Management Group (November 2009), formal/2009-11-02
8. Peraldi-Frati, M.A., Goknil, A., DeAntoni, J., Nordlander, J.: A Timing Model for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language V2. In: ICECCS 2012. pp. 230–239 (2012)
9. Slutej, D., Håkansson, J., Suryadevara, J., Seceleanu, C., Pettersson, P.: Analyzing a pattern-based model of a real-time turntable system. In: Jens Happe, B.Z. (ed.) 6th International Workshop on Formal Engineering approaches to Software Components and Architectures(FESCA), ETAPS'09, York, UK, March. pp. 161–178. Electronic Notes in Theoretical Computer Science (ENTCS), Vol 253, Elsevier (September 2009)
10. Suryadevara, J., Seceleanu, C., Mallet, F., Pettersson, P.: Verifying MARTE/CCSL mode behaviors using UPPAAL. In: 11th International Conference on Software Engineering and Formal Methods (SEFM 2013) (September 2013)
11. Suryadevara, J.: Validating EAST-ADL timing constraints using UPPAAL. In: 39th Euromicro Conference on Software Engineering and Advanced Applications SEAA 2013) (September 2013)