Validating the Design Model of an Autonomous Truck System

Jagadish Suryadevara[*]
Mälardalen Real-Time Research Centre
Västerås, Sweden

Paul Pettersson[†]
Mälardalen Real-Time Research Centre
Västerås, Sweden

Cristina Seceleanu[‡]
Mälardalen Real-Time Research Centre
Västerås, Sweden

ABSTRACT

Model driven approaches have become effective solutions for the development of embedded systems. In particular, models across various abstraction layers, e.g., application, design, and implementation, provide the opportunity for applying different analysis techniques appropriate at various phases of system development. In this paper, we informally show how to validate the design model of an *Autonomous Truck* embedded system, by comparing its trajectories with the trajectories of the corresponding application model. In the comparison, we also correlate the corresponding time scales of the two different models. The autonomous truck system is designed in the integrated modeling environment of SaveIDE. The system's functional and timing requirements verification is carried out on the truck's design model. Our work can be regarded as a preliminary step towards developing a general solution to the problem of bridging the gap between application and design models of embedded systems.

1. INTRODUCTION

To achieve *predictability* throughout the development of embedded systems (ES), the designer needs to employ a design framework equipped with analysis methods and tools that can be applied at various levels of abstraction. Usually, embedded system designers deal with different kinds of requirements. *Functional* requirements specify the expected services, functionality, and features, independent of the implementation. *Timing* requirements translate into meeting deadlines at run-time. Hence, in the presence of the external environment, the verification of functional correctness alone is not sufficient for ES. The development processes for ES need to integrate the timing aspects and related analysis techniques through all development phases, starting as early as possible.

Model-driven approaches such as UML/MARTE [8] are intended for the specification, design, and verification / validation stages of the ES development. The MARTE profile adds capabilities to UML for schedulability, performance and timing analysis of models. Although such capabilities are invaluable to obtaining a mature ES development process for predictability, they often need to be related to higher-level ES models that use a different specification paradigm, e.g., a continuous representation of time rather than an implicit or discrete one. For example, the timing aspects of higher level system artifacts, e.g., requirements and specification, may be represented in a dense-time domain, whereas those of design, implementation phases may involve a discrete time representation, deemed more suitable and closer to the actual platform. Consequently, techniques for relating various modeling paradigms and associated time scales are needed. The goal of employing such techniques would be to ensure the correctness of the design model with respect to the application model, despite the different paradigms used for representing their behaviors.

In this paper, we show how to validate the design model of an *Autonomous Truck* example system against its application model, by comparing the runs/trajectories of both models. To accomplish this, we describe both application and design models first, and informally present their underlying semantics, respectively. Next, we exemplify a "run" for each model, respectively, by outlining corresponding sets of representative trajectories (sequences of observable states and associated transitions) of the application and design models. The timing aspects of both runs are also apparent in the respective trajectories. For creating the truck's design model, we use the development environment of SaveIDE [9], an integrated design environment for ES. SaveIDE is developed as part of the PROGRESS project [1] for component-based development of predictable ES for the vehicular domain.

The remainder of the paper is organized as follows. In Section 2, we present the case study details including the high level application behavior and timing requirements. This section also includes an overview of the design framework used for the case study. In Section 3, we describe the application and design models and their behavior, respectively. We show how to carry out the validation of the truck's design model against its application model, in Section 4. Here, we also include the verification of timing requirements. Selected related work is presented in Section 5, whereas Section 6 concludes the paper.

[*]Email:jagadish.suryadevara@mdh.se
[†]Email:paul.pettersson@mdh.se
[‡]Email:cristina.seceleanu@mdh.se

2. CASE STUDY: AUTONOMOUS TRUCK

The case study is part of a demonstrator project that serves for show-casing the research results of the PROGRESS project, and also for the validation of the related integrated development environment, SaveIDE. The case study consists of an autonomous truck that moves along a specified path (Figure 1), according to a well-specified application behavior.

2.1 Application

In a nutshell, the truck application has three operational modes:

- *FOLLOW* : the truck simply follows the straight path (the black thick line in Figure 1) using its light sensors. When the end of the path is detected, it changes to *Turn* mode.
- *TURN* : the truck turns right, until the specified amount of time expires, after which it changes to *Find* mode.
- *FIND* : the truck searches for the original path. When the path is detected, the truck returns to the *Follow* mode again.

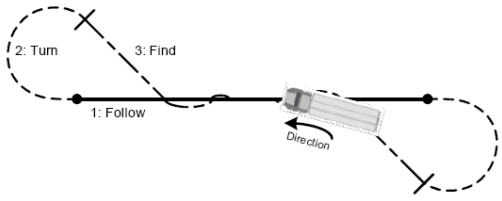

Figure 1: Path of the Truck movement

2.2 Timing Requirements

In addition to the general application requirements (e.g., functional), one can specify some timing requirements to be verified on the design model. Due to the difference in the time scales of the application (associated with dense time) and design models (associated with discrete time), a certain latency in the design (or implementation) should be allowed. For example, event detection and mode change may occur instantaneously in the application model. However, in the corresponding design model, there is usually some latency in the event detection and associated mode change.

The timing requirements of the truck application are specified in Table 1 below:

Timing Requirement	Constraint
Latency: event detection : *e_0_l*	≤10 t.u.
Latency: event detection : *line_detected*	≤10 t.u.
Latency: mode change	≤10 t.u.
End-to-end timing	≤5 t.u.

Table 1: Timing requirements of the Autonomous Truck application model.

For simplicity, we assume the same time scale in both application and design models, measured in time units t.u. (e.g., *ms*). In order to guarantee the timing correctness, one needs to verify that the corresponding design model satisfies the application requirements.

2.3 Design Framework

The design model of the autonomous truck has been created in the ProCom framework [4], a component-based design environment for ES (in particular vehicular domain). The component model ProCom, a successor of SaveCCM (SaveComp Component Model [2]), is a two-layer modeling entity. ProCom is developed to address the particularities of the ES domain, including resource limitations and requirements on safety and timing. In order to meet the different concerns that exist at different levels of granularity, spanning the overall architecture of a distributed embedded system up to the details of low-level control functionality, ProCom is organized in two distinct, but related, layers: ProSys and ProSave. Besides the difference in granularity, the layers differ in terms of architectural style and communication paradigm. In the top layer ProSys, a system is modeled as a collection of communicating *subsystems* that execute concurrently, and communicate by asynchronous messages sent and received at typed output and input *message ports*. Contrasting this, the lower lever, ProSave, consists of passive units, and it is based on a pipe-and-filter architectural style, with an explicit separation between data and control flows. The former is captured by *data ports*, where data of a given type can be written or read, and the latter by *trigger ports* that control the activation of components.

In our case study, we use a subset of ProSave that is common to SaveCCM too. Consequently, the design can be developed by using SaveIDE, an available integrated environment for SaveCCM designs. Figure 2 shows the autonomous truck design, in the SaveIDE development environment. We skip the formal behavior modeling of the components, as a network of timed automata [3] models (available through the UPPAAL [6] tool plug-in in SaveIDE). The schematic view of the design is presented in figure 4.

3. APPLICATION AND DESIGN MODELS OF THE AUTONOMOUS TRUCK

In this section, we define the application and design models of the Autonomous Truck system. The syntax and the informal semantics of these models are also described.

3.1 Application Model: Syntax & Semantics

Figure 3 presents the application model of the Autonomous Truck system. The location *Turn* is associated with a time out duration of n time units. We define the application model as the tuple $\langle L, Init, A, E, M\rangle$, where

- L is the set of locations: $\{Init, Follow, Turn, Find\}$;
- A is the set of events:

  $$\{start, end\_of\_line\,(e\_o\_l), line\_detected, tm\},$$

  where tm is a timer event;
- $E \subseteq L \times A \times L$ is the set of edges between locations;
- $M : L \rightarrow \{\perp\} \cup \mathbb{N}$ is a function that associates timeouts to locations, where $\perp$ is a special symbol indicating that no timeout is associated with the location.

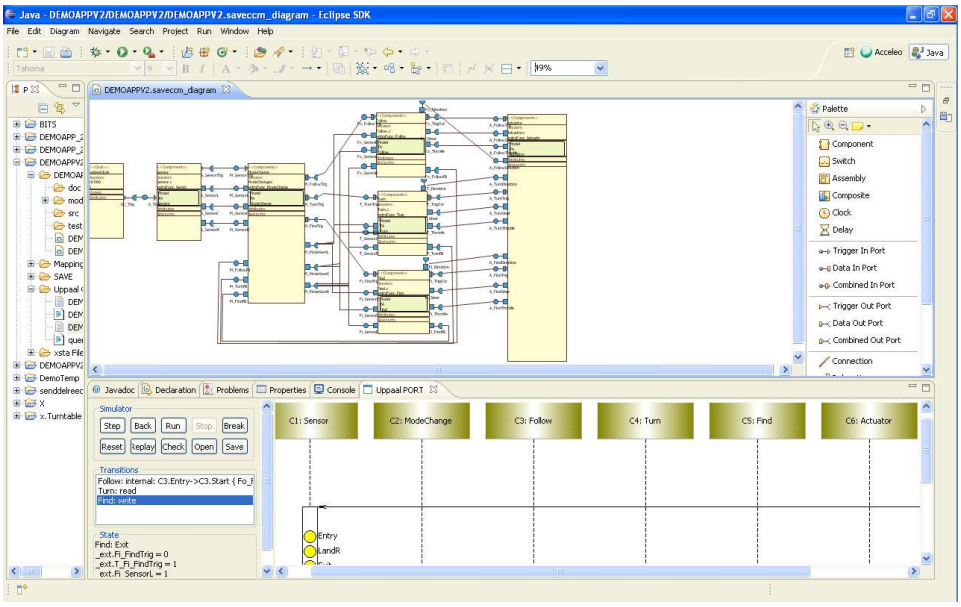

Figure 2: The design of the Autonomous Truck in SaveIDE.

The runs, that is, the trajectories of the application model are defined over the semantic representation of the application model, the underlying transition system, consisting of states and transitions between states. A state in the transition system is given by the application model's current location and current value of the timeout duration, if defined. A run of the application model in Figure 3 begins in the *Init* location. When the *start* event is detected, the location *Follow* is entered. This location is exited and consequently *Turn* is entered, when *end_of_line (e_o_l)* is detected. At location *Turn*, n time units, called *ticks*, are consumed before the timer event *tm* occurs. When *tm* occurs, the location *Turn* is exited and *Find* is entered after taking the corresponding edge. Next, the location *Find* is exited, and *Follow* entered, as a result of the event *line_detected* occuring. Once in location *Follow* again, the trajectory continues as described above.

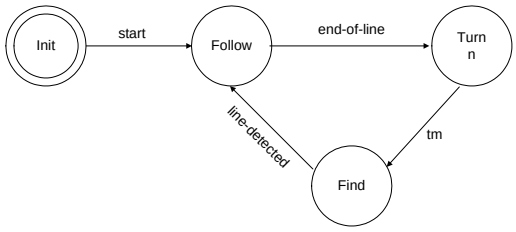

Figure 3: The Truck application model.

3.2 Design Model: Syntax & Semantics

We define the design model of the *Autonomous Truck* as a tuple $\langle \mathbb{C}, \rightarrow, \delta \rangle$, where

- $\mathbb{C}$ is the set of components: { SystemClock, Sensor, Controller, Follow, Turn, Find, Actuator } (Figure 4);
  - I is the boolean expression over input variables, which triggers the component execution;
  - $\beta \in \mathbb{N}$ is the execution time of a component;
- $\rightarrow$ denotes component connections (i.e., dataflow between components). $C_i \rightarrow C_j$ implies that an output variable of component C_i is mapped to an input variable of component C_j. If $i = j$, the output of the component is mapped to one of its own input variables, meaning that the component can trigger itself.
- $\delta \in \mathbb{N}$ is the discrete time unit (i.e., the increment of time used for sampling the continuous time) in the model. If the periodicity of a clock component is 'p', then $\delta \ll p$ and $p \leq n\delta$, for some $n \in \mathbb{N}$.

Let FB_{fo}, FB_{tu}, and FB_{fi} denote the feedback, through data ports, from components Follow, Turn, and Find, respectively, to the Controller component. These values indicate the completion of the Follow, Turn, and Find modes, respectively, in the execution of the design model.

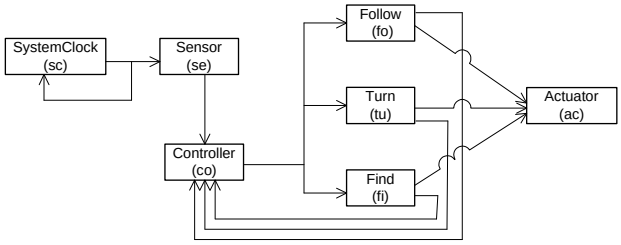

Figure 4: Schematic view of the ProSave design model of the Autonomous Truck.

The runs, or trajectories, of the design model are defined over the underlying transition system. For the design model, a state in the transition system is characterized by the set of currently executing (*active*) components, and the current valuations of their variables and their remaining execution times. A component that is not active may be either *inactive*, or *ready* to begin its execution. In Figure 4, a run of

the design model begins in the initial state where only the clock components are ready. The executions of all active components progress simultaneously, in discrete steps of δ time units. When the execution time of a component expires, the latter immediately sends its output and triggers other components that are inactive, but connected to the component, after which it becomes inactive. This step is repeated for all active components whose execution time has expired. Then, the newly triggered components, that is, the ready components, if any, begin their execution. The time in the design model progresses discretely only when there exists no active component and no ready component.

An intuitive view of the underlying transition system is given in Figure 5. The abstract states of the transition system are TimePassing (TP), CompDone(CD), and CompStarts(CS). The transition labels *idling*, $comp\_finish_i$, and $comp\_ready_i$ are associated to the the i_th component, as follows: *idling* in case nothing else happens with the component, but the time progresses in discrete steps, $comp\_finish_i$ in case the component's execution time has expired, and $comp\_ready_i$ in case it is ready to begin execution.

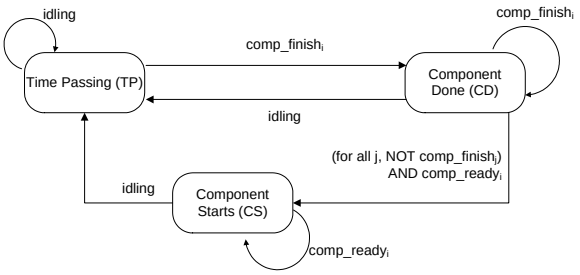

Figure 5: An intuitive view of the underlying state transition system of the design model.

4. DESIGN MODEL VALIDATION

Here, we show how to validate the ProSave-based design model of the truck case study, presented in Figure 2. Although a component in our design model can trigger itself, the ProSave components used in the case study are not allowed any self-triggering. However, the more liberal definition of a design model component element lets us describe the clock components of the ProSave model, too, since the clocks actually trigger themselves (see Figure 4).

To validate the ProSave design model against the application model of Figure 3, we resort to comparing the generated trajectories of the two models. First, we consider the set of trajectories of the application model. For simplicity, this set is exemplified by a representative trajectory given below:

$$\langle Init, \perp\rangle \xrightarrow{start} \langle Follow, \perp\rangle \xrightarrow{end-of-line} \langle Turn, n\rangle \xrightarrow{(n-1)ticks} \langle Turn, 1\rangle \xrightarrow{tm\&tick} \langle Find, \perp\rangle \xrightarrow{line-detected} \langle Follow, \perp\rangle \cdots$$

Now, we consider the set of trajectories that characterize the design model of Figure 4. Again for simplicity, this set is exemplified by the following trajectory, in its simplest form, obtained by hiding all the internal transitions, as they are not externally observable:

$$\begin{aligned}
&S_0 : \langle\{\}, I_1, -\rangle \rightarrow \\
&S_1 : \langle\{sc\}, -, \beta_{sc} = 4\rangle \xrightarrow{5*TP} \\
&S_2 : \langle\{sc, fo\}, -, \beta_{sc} = 2, \beta_{fo} = 1\rangle \xrightarrow{TP} \\
&S_3 : \langle\{sc, ac\}, -, \beta_{sc} = 1, \beta_{ac} = 1\rangle \xrightarrow{3*TP} \\
&S_4 : \langle\{sc, fo\}, -, \beta_{sc} = 2, \beta_{fo} = 1\rangle \xrightarrow{TP} \\
&S_5 : \langle\{sc\}, I_7, FBfo = true, \beta_{sc} = 1\rangle \xrightarrow{3*TP} \\
&S_6 : \langle\{sc, tu\}, -, \beta_{sc} = 2, \beta_{tu} = 1\rangle \xrightarrow{TP} \\
&S_7 : \langle\{sc, ac\}, -, \beta_{sc} = 1, \beta_{ac} = 1\rangle \xrightarrow{3*TP} \\
&S_8 : \langle\{sc, tu\}, -, \beta_{sc} = 2, \beta_{tu} = 1\rangle \xrightarrow{TP} \\
&S_9 : \langle\{sc\}, I_7, FBtu = true, \beta_1 = 1\rangle \xrightarrow{3*TP} \\
&S_{10} : \langle\{sc, fi\}, -, \beta_{sc} = 2, \beta_{fi} = 1\rangle \xrightarrow{TP} \\
&S_{11} : \langle\{sc, ac\}, -, \beta_{sc} = 1, \beta_{ac} = 1\rangle \xrightarrow{3*TP} \\
&S_{12} : \langle\{sc, fi\}, -, \beta_{sc} = 2, \beta_{fi} = 1\rangle \xrightarrow{TP} \\
&S_{13} : \langle\{sc\}, I_7, FBfi = true, \beta_{sc} = 1\rangle \xrightarrow{3*TP} \\
&S_{14} : \langle\{sc, fo\}, -, \beta_{sc} = 2, \beta_{fo} = 1\rangle \xrightarrow{TP} \cdots
\end{aligned}$$

By comparing the above application and design trajectories, it can be shown that the specified functionality of the application model is satisfied by the corresponding design model (note that the conditions $FB_{fo} ==$ true, $FB_{tu} ==$ true, $FB_{fi} ==$ true indicate the completion of the operational modes *Follow*, *Turn*, *Find*, respectively). From the design trajectory, we can derive the following timing information:

Timing aspect	**Time units (t.u.)**
Event detection: *e_0_l*	Max: 6 t.u., Min: 3 t.u.
Event detection: *line_detected*	Max: 6 t.u., Min: 3 t.u.
Mode (location) change	4 t.u.
End-to-end timing	$\leq$5 t.u.

Table 2: Timing aspects in the Truck design model.

By inspecting Table 1 and Table 2, it follows that the design model satisfies the timing requirements specified on the application model.

5. RELATED WORK

Here, we briefly mention some of the related work that is relevant to our validation problem. Sifakis et.al. present a methodology for building timed models of real-time systems by adding time constraints to corresponding application software [10]. The timed models of the application obtained in this manner can be analyzed by using timing analysis techniques, to check relevant real-time properties. In comparison, our trajectory comparison technique seems more restricted by the underlying timing assumptions, yet simpler. Mallet et al. [7] show the expressiveness of MARTE for event-triggered and time-triggered communication, which can be employed for validating models using these paradigms. Paving the way towards proving correctness of implementations, Krčál et al. propose an alternative, discretized semantics of timed automata [5], which gives rise to a natural notion of digitization for timed languages.

6. CONCLUSIONS

In this paper, we have presented a case-study on validating the design model of an autonomous truck ES, against higher-

level requirements including both functionality and timing requirements. We validate the design model by comparing its trajectories to those of the truck application model. The trajectories of the two models are expressed over the corresponding underlying transition systems, respectively. To achieve our goal, we have first described the syntax and informal semantics of both models, used to generate the trajectories, after which we have proceeded to the latter's comparison. The timing requirements specified in the application model are verified in the design model. Our work can be seen as a first step towards addressing an important gap in the current state-of-art of model-driven engineering for embedded systems: relating implementations to their specifications for proving the correctness of the former with respect to the latter. We intend to extend the case-study driven, informal approach presented in this paper, to a detailed, general formal technique that would encompass a larger class of ES models. Further, the application model can be integrated with the industry-targeted standard approaches, such as UML/MARTE profile for specification of timing requirements. Such a step could also facilitate integrating the existing MARTE-based analysis approaches into our development process.

7. REFERENCES

[1] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The SAVE approach to component-based development of vehicular systems. *Journal of Systems and Software*, 80(5):655–667, May 2007.

[2] M. Åkerholm, J. Carlson, J. Håkansson, H. Hansson, M. Nolin, T. Nolte, and P. Pettersson. The SaveCCM language reference manual. Technical report, January 2007.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] T. Bures, J. Carlson, I. Crnkovic, S. Sentilles, and A. Vulgarakis. Procom - the progress component model reference manual, version 1.0. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-230/2008-1-SE, Mälardalen University, June 2008.

[5] P. Krčál, L. Mokrushin, P. Thiagarajan, and W. Yi. Timed vs time triggered automata. In P. Gardner and N. Yoshida, editors, *Proc. of CONCUR'04.*, number 3170 in Lecture Notes in Computer Science, pages 340–354. Springer–Verlag, 2004.

[6] K. Larsen, P. Pettersson, and Y. Wang. Uppaal in a nutshell. *Int. J. on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[7] F. Mallet, R. de Simone, and L. Rioux. Event-triggered vs. time-triggered communications with UML Marte. pages 154–159.

[8] OMG. Unified modeling language (uml) profile for modeling and analysis of real-time and embedded systems (marte). In *Document ptc/07-08-04*. OMG, 2007.

[9] S. Sentilles, A. Pettersson, D. Nyström, T. Nolte, P. Pettersson, and I. Crnkovic. Save-ide - a tool for design, analysis and implementation of component-based embedded systems. In *Proceedings of the Research Demo Track of the 31st International Conference on Software Engineering (ICSE 2009)*, May 2009.

[10] J. Sifakis, S. Tripakis, A. Member, and S. Yovine. Building models of real-time systems from application software. In *Proceedings of the IEEE Special issue on modeling and design of embedded*, pages 100–111. IEEE, 2003.