A Formal Analysis of A Car Periphery Supervision System*

Biniam Gebremichael[1],
Tomas Krilavičius[2], Yaroslav S. Usenko[2]

[1]Nijmegen Institute for Computing and Information Sciences
University of Nijmegen,
P.O. Box 9010, 6500 GL, Nijmegen, The Netherlands,
`biniam at cs.kun.nl`
[2]Formal Methods and Tools, University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands,
`[t.krilavicius,usenko] at utwente.nl`

June 7, 2004

Abstract

This paper presents a formal model of the real-time service allocation unit for the Car Periphery Supervision (CPS) system—a case study proposed by Robert Bosch GmbH in the context of the EU IST project AMETIST. The CPS system is a hybrid system, which is modeled in terms of timed automata. It is done by splitting the values of nonlinear continuous variables into finite set of regions and over-approximating the constraints on continuous variables into clock constraints. Safety properties of the timed model have been verified using UPPAAL. This is a sufficient condition for validating the corresponding safety properties of the initial hybrid system. The difference in time scale between the CPS components have also been taken care of by over-approximating the timed model using the convex-hull over-approximation feature available in UPPAAL.

1 Introduction

A number of modeling and verification tools for real-time and hybrid systems have been developed. For instance, the tools based on the theory of *timed automata*, [1], such as KRONOS by [4] and UPPAAL by [3]; and the theory of *hybrid automata*, [9], such as HyTECH by [11]. Recent development in devising clever computational procedures have improved the ability of verification tools to handle industrial-size problems automatically. Yet many essential problems,

*This work was supported by the European Community Project IST-2001-35304 AMETIST `http://ametist.cs.utwente.nl`.

specifically in the area of hybrid systems, remain unsolvable with these techniques.

Two major problems are (1) the lack of adequate abstraction concepts for modeling large systems, [2], (2) the exploding consumption of computing resources by the verification algorithms, [5].

The present paper addresses the first issue and provides evidence that correct abstraction and appropriate over-approximation techniques in modeling of large systems leads to verifiable models from which one can infer properties of the original model.

This is illustrated by the real-time service allocation case study for Car Periphery Supervision (CPS) system (cf. [13]) proposed by Robert Bosch GmbH in the context of the EU IST project AMETIST. The CPS system is a hybrid system that interacts with continuous environment via a discrete controller. Moreover, CPS safety properties are parametrized with free variables, which should be determined in order to prove the safety of the system. The CPS model presented in this paper is a timed automata based abstraction of the system constructed manually by dividing the environment to a finite set of regions.

Related Work

Verifying whether a hybrid system H satisfies a property P can turn out to be undecidable for most cases. Appropriate abstraction can extract a finite discrete system F from H by partitioning the state space of H into finite number of regions. The survey by [2] aims in finding a class of hybrid systems which can be abstracted into F and whose verification against a property P is decidable. The survey has shown that, proving that F satisfies P is equivalent or sufficient for proving that H satisfies P.

The approach used in the present paper is similar to the one of [10], where, nonlinear continuous variables of the original hybrid system are over-approximated to define a time-constrained automaton, and the approximated automaton will satisfy strictly fewer safety properties.

Another problem that leads to the state-space blowup in timed automata is the time scale difference between the different components of the system. This is often the case when an embedded system interacts with its environment (cf. [12]). One way to solve such problem is to use the convex-hull over-approximation method of [7] when verifying invariant properties.

Outline

The organization of the paper is as follows: Section 2 presents an informal description of the CPS system. Next, in Section 3, the CPS system is formally modeled in a way that the desired properties are fully preserved and the model of the system is abstracted to allow verification. Section 4 presents the properties and the verification results. Finally Section 5 concludes the paper and lists some directions for future work.

An abstract of this paper appeared as [6]. The complete Uppaal model of the CPS system is available via the AMETIST project web-page.

2 Car Periphery Supervision System

Car Periphery Supervision (CPS) refers to the functionality and technology for obtaining information about the environment of a car. Applications like parking assistance, pre-crash detection and blind spot supervision depend on CPS for the basic operation and information sharing. Short Range Radar (SRR) sensors are mounted in-front of the car, and they scan the environment for nearby objects. The data collected by the sensors is sent to the computing part of CPS known as the Electronic Control Unit (ECU). The ECU processes the data and invoked applications based on the data. The structure of CPS and its environment is depicted in Fig. 1. A detailed description of the CPS system is given in [13], [14] and [15].

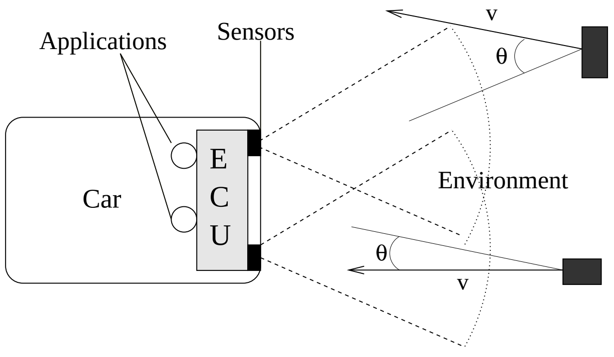

Figure 1: The CPS system and its environment

The CPS system and its environment can be viewed as a system consisting of three parts, namely:

1. **Environment:** the environment of the CPS system is a dynamic system with moving and stationary objects in-front of the car. Object velocity and distance are important characteristics that determine the behavior of the CPS system in a continuous manner. There are several restrictive assumptions that apply to the environment in which the CPS system is supposed to operate. This is done to make the system more tractable.

2. **Sensors:** the Sensors are the interfaces through which the CPS is informed about the behavior of the environment. Sensors operate in discrete time, which can either be periodic or event-driven. The sensor component includes not only the equipment for sending and receiving radar signals but also a processor for basic data processing and control. Typically, the sensors that are situated in-front of the car return the distance to a nearest object (from their perspective) approaching the car.

3. **ECU:** the Electronic Control Unit is a board computer that performs a collection of tasks running on top of the OSEK operating system[1]. These tasks are used to control the operation of the sensors, and to deliver accurate and on-time information about the environment to the applications.

[1] `http://www.osek-vdx.org/`

3 Formal Modeling

Each of the three parts of the CPS system are operating in different modes. The environment is a dynamic and *continuously* changing system. The sensors operate on a *discrete* time scale, while ECU tasks are *real-time* tasks. An appropriate model for this system would be a hybrid automaton. However, proving correctness of a system using hybrid automata is difficult, if at all possible.

Another approach is to abstract from the unnecessary details of the model and transform it into a timed automata model, while preserving the desired properties. In this section, the CPS system is modeled in timed automata, and relevant properties are verified using UPPAAL. This model consists of six timed automata that are put in parallel. The structure of the system is shown in Fig. 2. The boxes represent the timed automata, the thin arrows represent

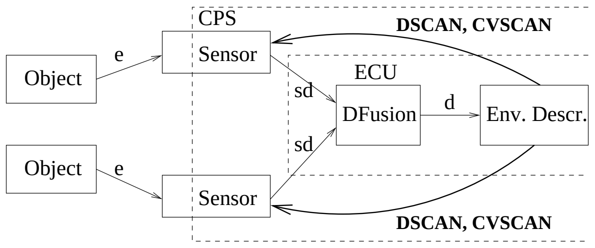

Figure 2: Decomposition of CPS in timed automata.

the communication via shared variables, and the thick arrows represent the multiparty action synchronization.

3.1 CPS Environment

The environment is a collection of several objects moving with different velocities at different distances in-front of the car. An object i in the environment has the approaching velocity v_i and the angle θ_i relative to the direction of the car's movement (see Fig. 1).

Relative Velocity

Let $\mathbb{T} = \mathbb{R}^+$ denote the time domain and let d_i denote the distance to an object i from the middle of the front of the car. Then the relative velocity of an object is defined as the function $\dot{d_i} : \mathbb{T} \to \mathbb{R}$ and it is given by the difference between the velocity of the object and the velocity of the car (v_{car}):

$$\dot{d_i} = v_i \cos(\theta_i) - v_{car}$$

The CPS system only tracks the objects that are close enough to the car. The distances to the remote objects and their velocities do not affect the behavior of the CPS system.

Regions

As will be described in more detail in Section 3.2, the sensors in the CPS system scan for a nearest object, and return the distance to it. This can be given as:

$$d(t) = \min_{\forall i}(d_i(t))$$

The area in-front of the car is divided into twelve regions (see [13] for details). These regions are ordered in descending order and numbered from -1 to 9 (see Table 3.1).

Region name	Region number (e)
FAR	`-1`
PreCV	`0`
RG_i	`1...8`
PreCrash	`9`

Table 1: CPS regions

Assumptions

The environment that can be handled by the limited capacity of the sensors and other components of the CPS is rather small. The following three criteria restrict the behavior of the environment that the CPS system can interact with.

1. An object can approach the car with a relative velocity in the range from 13m/s to 56m/s.
2. Only one object is present in the environment.
3. The CPS system will be externally reinitialized when an object reaches the last region. In other words, the system terminates after the crash.

Environment behaviors outside the above assumptions have rare occurrence. Extra recovery treatments as in [13], which are not included in the current model, could be used to cope with such behaviors.

Timed Automaton of the Environment

Figure 3 shows a timed automaton model of the CPS environment, which is based on the allowed relative velocity defined above. The automaton has four locations which correspond to the regions names in Table 3.1. Initially objects are far from the car (in location `FAR` and `e:=-1`). If an object comes closer to the car it will reach the location `PreCV`.

The constants `zpreCV_max` and `zpreCV_min` are time bounds on the transition of the environment from `PreCV` to $\mathtt{RG}_0$. These time bounds are calculated from velocity bounds and the length of the `PreCV` region. Other time bounds shown in Fig. 3 are calculated in a similar manner. This method of substituting the maximal and minimal velocity constraints into clock constraints is done in accordance with the method of [10].

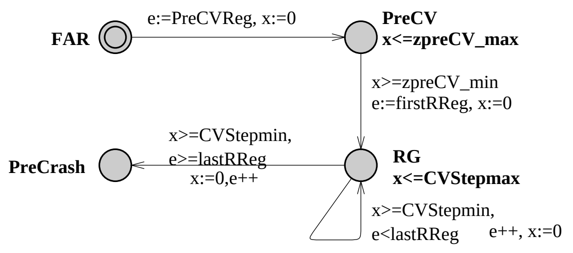

Figure 3: CPS environment template

An object has to cross all the regions up-to the `PreCrash` region. ($\texttt{e} \geq \texttt{lastRReg}$) before it can go into `PreCrash`. As soon as the object is in `PreCrash`, an immediate action has to be taken by the ECU. The clock `x` is reset to measure the amount of time that the object spends in the `PreCrash` location before the ECU does something about it. Section 4 lists important invariants that have to be satisfied when an object reaches the `PreCrash` location.

3.2 Sensor

A sensor in the CPS system scans the environment for a nearest object in-front of the car, and returns the region value of the scanned object. A sensor has two modes of operation: `DSCAN` and `CVSCAN`, and one `IDLE` mode. These modes of operation are used as locations in the timed automaton of the sensor depicted in Fig. 4.

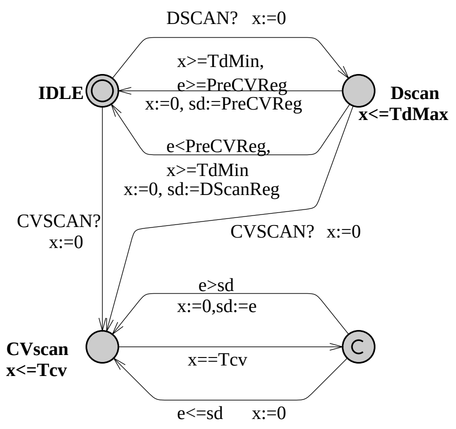

Figure 4: Sensor Template

The sensor is initially in location `IDLE` and waits for a command from the ECU. If it receives a `DSCAN` command, it conducts a long range scan and, as soon as it finds an object in `PreCV`, the sensor's distance reading `sd` is updated:

$$sd_i(t) = e_i(t),$$

where t is the time when the ith sensor scans the environment from its own perspective (e_i). For a `CVSCAN` command, the sensor produces a maximum of eight readings, one for each region $\mathtt{RG}_i$. It scans the environment every `Tcv` time units, and sd_i is updated in a similar manner.

The CPS system usually has several sensors placed in-front of the car. In this paper only two sensors are considered. It is possible that the two sensors return different readings. This may be a result of the fact that they are situated at different positions, or the object may have an irregular form, or simply there are several objects in-front of the car. Thus, in general, there need to be no correlation between the readings of the two sensors (even though a visibility analysis shows that the difference is rather small).

3.3 Electronic Control Unit (ECU)

ECU stands for the collection of tasks running on a single processor. According to [13], the ECU executes more than two tasks. Most of these tasks are sequential and run in a predictable manner. Thus, they can easily be grouped into two tasks without affecting their behavior. These combined tasks are called DFusion (sensor fusion) and EnvDescription (environment description).

3.3.1 Sensor Fusion

DFusion is a part of the ECU which receives individual distance values from all sensors and sends a "combined" single value to the remaining tasks. Combining several readings (sd_is) into one reading (d) can be done by the triangulation function as suggested by [13]. The maximum function, instead of triangulation, is used here to compute the final value, since our model deals with one-dimensional value of the sd_is only. DFusion is a periodic task and in every jth step it computes $d(j)$ as follows:

$$d(j) = \max_{\forall i=\{1,2\}} (sd_i(j))$$

The variable $d(j)$ is a shared variable which is also readable by EnvDescription. Figure 5 shows the timed automaton of DFusion.

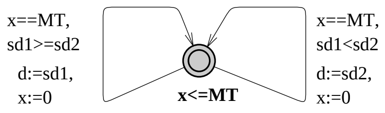

Figure 5: Sensor Fusion

3.3.2 Environment Description

EnvDescription is a part of ECU that receives environment data from DFusion and maintains accurate information about the environment. While doing so, EnvDescription controls the mode of operation of the sensors as well. The sensor-controlling part is defined in [13] as the "situation analysis" task. In the present paper it is combined with the EnvDescription task to avoid the state-space explosion.

The timed automaton of EnvDescription is shown in Fig. 6. Initially En-

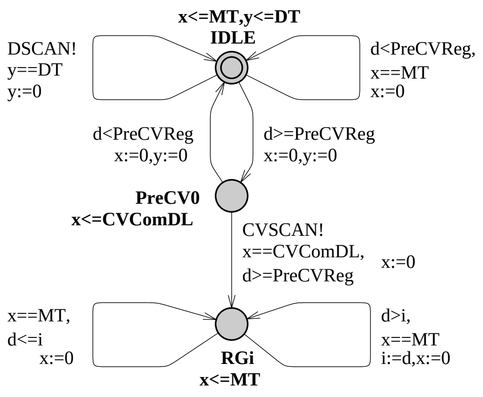

Figure 6: Environment Description (ED)

vDescription is in location `IDLE`, from which it periodically reads the value of `d` and broadcasts the `DSCAN` command to the sensors. If the value of `d` shows that an object is present in the `PreCV` region, then the automaton jumps to location (`PreCV0`), from which it sends the `CVSCAN` command. There is a time delay of `CVComDL` time units before this command happens. Once the sensors receive the `CVSCAN` command, the EnvDescription counts the number of $\texttt{RG}_i$ regions by reading `d` computed by DFusion.

4 Verification

4.1 Requirements

The primary goal of the CPS system is to provide accurate information about the environment of the car to the applications such as airbag inflation, parking assistance, pre-crash detection and others. The accuracy of the information provided by ECU is measured by the time delay between a change in the environment and the knowledge of the ECU about this change. The properties are stated below as parametrized temporal logic formulas, and they were verified using Uppaal on a workstation with a 512MHz CPU and 256Mb of RAM. The time required to verify these properties was drastically reduced using the convex-hull approximation feature of Uppaal to less than a minute.

Property 1 (P1): EnvDescription has an accurate information about the object in the collision course. This property is modeled in the following way: the EnvDescription's information `ED.i` about the region of the object should not deviate too much from the environment's information `e1` about the same object. The goal is to find the maximal difference `Q` between these two values (the minimal Q for which the formula is satisfied).

```
A[] (e1-ED.i <= Q and e2-ED.i <= Q)
```

Property 2 (P2): When an object reaches the pre-crash region (the environment automaton is in location `ENV1.PreCrash`), EnvDescription knows about this (`ED.i == lastRReg+1`) within a few time units (`ENV1.x > P`). Here `ENV1.x` represents the time after the environment automaton moved into location `ENV1.PreCrash`, and we are interested in finding the minimal value of `P`.

```
A[] ((ENV1.PreCrash and ENV1.x > P)
     imply (ED.i == lastRReg+1))
```

Property 3 (P3): The ECU avoids false alarm. EnvDescription never reports advancement of an object toward the car before the object (the environment) actually does so.

```
A[]  (ED.i <= e1 or ED.i <= e2)
```

Property 4 (P4): The system is deadlock free.

```
A[] (not deadlock)
```

4.2 Results

The CPS model satisfies properties `P1` for $Q \geq 3$, `P2` for $P \geq 5$, `P3` and `P4`. Note that Q can also be computed from P as $Q = \lceil P/CVStepmin \rceil$, since the difference in the number of regions can also be expressed as difference in time.

In the above model DFusion and EnvDescription run in arbitrary order, no prior scheduling is assumed. It is possible, however, to schedule the execution in such a way that, DFusion computes `d` and EnvDescription updates its counter immediately. Under such synchronization EnvDescription will update its information faster. Figure 7 shows the model of EnvDescription using this alternative schedule. The model of the new DFusion is not shown here, but it is similar to the one in Fig. 5 except that the new model sends a synchronization signal (`ND!`) to the EnvDescription as soon as a new value of `d` is computed. After the synchronization with DFusion, EnvDescription makes its decision without a delay (see the urgent locations in Fig. 7). For this setting the properties P1 for $Q \geq 2$ and P2 for $P \geq 3$ are satisfied.

In both cases P is equal to the time needed for an information to propagate from the environment to EnvDescription. That is the sum of the time spent by the sensor (Tcv), DFusion and EnvDescription.

$$P = Tcv + MT + MT$$

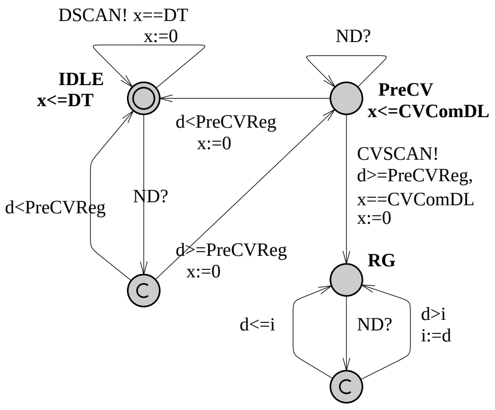

Figure 7: Modified Environment Description

But in the scheduled model, both EnvDescription and DFusion need only one MT to update their information, and propagation time is reduced to $P = Tcv + MT$. These two cases show the worst and best cases for determining the value of P. In general, however, P is equal to $Tcv + MT$ plus the overhead associated with scheduling of the ECU tasks and the time spent by each task. When OSEK operating system, is used to schedule the ECU tasks, and if $OSEK_{SchTime}(T_i)$ is the time delay due to scheduling and running the tasks T_i, then the value of P is

$$P = Tcv + MT + OSEK_{SchTime}(T_i).$$

5 Conclusion and Future Work

The car periphery supervision is a hybrid system. Verifying properties for hybrid systems is undecidable in general. However, the continuous variables of the model, the environment in this case, can be discretized to a finite block of regions in order to make verification of the properties possible.

The different time scale between the environment and CPS components have resulted in a state-space blow up. The convex-hull over-approximation technique of Uppaal was used to verify the safety properties of the system. Another approach could be the exact acceleration method of [8].

The assumptions made on the environment of CPS are too restrictive. Allowing only one object and having non decreasing `e` in the regions can be omitted by introducing a recovery mode in case when more than one object appears in the `RG` regions. As described in [13], the recovery mode is a third mode of sensor operation, which scans for follow-up objects when a nearest object disappears from the scene. This scenario may happen when one object in the collision course changes its trajectory and disappears; and later on, another object, which was close to the first one, enters a collision course. Adding recovery method to the model for such a scenario would be an interesting future work.

Acknowledgments

The authors would like to thank (in the reverse-alphabetic order) Frits Vaandrager, Stefan Kowalewski Holger Hermanns, Marko Auerswald and the members of the Sensor Group at Robert Bosch GmbH for their valuable comments and reviews.

References

[1] R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.

[2] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88:971–984, 2000.

[3] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a tool suite for automatic verification of real–time systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, pages 232–243. Springer, 1995.

[4] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: A model-checking tool for real-time systems. In A.J. Hu and M.Y. Vardi, editors, *Proc. 10th CAV'98*, LNCS 1427, pages 546–550, Vancouver, Canada, 1998. Springer.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *I&C*, 98(2):142–170, June 1992.

[6] B. Gebremichael, H. Hermanns, T. Krilavičius, and Y.S. Usenko. Hybrid modeling of a vehicle surveillance system with real-time data processing. In *Proc. Int. Conf. on Dynamical Systems Modeling and Stability Investigation*, page 419, Kyiv, Ukraine, May 2003.

[7] N. Halbwachs, Y. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Proc. 1st. SAS'94*, LNCS 864, pages 223–237, Namur, Belgium, September 1994. Springer.

[8] M. Hendriks and K.G. Larsen. Exact acceleration of real-time model checking. *ENTCS*, 65(6): , April 2002.

[9] T.A. Henzinger. The theory of hybrid automata. In *Proc. 11th LICS'96*, pages 278–292, New Brunswick, New Jersey, USA, 1996.

[10] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Proc. 7th CAV'95*, volume 939, pages 225–238. Springer, 1995.

[11] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Int. Journal on Software Tools for Technology Transfer*, 1:110–122, 1997.

[12] T.K. Iversen, K.J. Kristoffersen, K.G. Larsen, M. Laursen, R.G. Madsen, S.K. Mortensen, P. Pettersson, and C.B. Thomasen. Model-checking real-time control programs — Verifying LEGO mindstorms systems using Uppaal. In *IEEE Euromicro Conf. on Real-Time Systems*, pages 147–155, 2000.

[13] S. Kowalewski and M. Rittel. IST Project AMETIST: Real-time allocation for car periphery supervision. Technical report, Robert Bosch GmbH, 2003. Preliminary Description (Deliverable No. 3.1.3) see `http://ametist.cs.utwente.nl/`.

[14] R. Moritz. Pre-crash sensing - functional. Technical report, Robert Bosch GmbH, 2000.

[15] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick. A case study in applying a product line approach for car periphery supervision system. Technical report, Robert Bosch GmbH, 2001.