Poster: An Efficient Approach for Verifying Automobile Distributed Application Systems on Timing Property

Haitao Zhang[1], Guoqiang Li[2], Xiaohong Li[3], Zhuo Cheng[4,*], Jinyun Xue[5], Shaoying Liu[6]
[1]School of Information Science and Engineering, Lanzhou University, China
[2]School of Software, Shanghai Jiao Tong University, China
[3]School of Computer Science and Technology, Tianjin University, China
[4,5]International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University, China
[6]Faculty of Computer and Information Sciences, Hosei University, Tokyo, Japan
Email:htzhang@lzu.edu.cn

1 INTRODUCTION

OSEK/VDX standard [3] has now been widely adopted by many automotive manufacturers and research groups to develop a vehicle-mounted system. An OSEK/VDX vehicle-mounted system generally runs on several processors (e.g., the system shown in Figure 1 runs on two processor), and it consists of three components: OS, multi-tasking application and communication protocol. The OS locating at a processor manages an application and conducts tasks within the application to execute on a processor, especially a *deterministic* scheduler (static priority scheduling policy) is adopted by the OSEK/VDX OS to dispatch tasks. The applications are in charge of realizing functions and often interact with each other via the communication protocol such as controller area network (CAN). There are two complex execution characteristics in the OSEK/VDX vehicle-mounted systems: (*i*) tasks within an application concurrently execute on a processor under the scheduling of OSEK/VDX OS; (*ii*) applications simultaneously run on the different processors and communicate each other sometimes. Due to the concurrency of tasks and simultaneity between applications, how to exhaustively verify a developed OSEK/VDX distributed application system in which applications cooperatively complete a function based on the communication protocol has become a challenge for developers with the increasing development complexity.

Model checking [1] has attracted much attentions in the automobile industry because of its exhaustibility. As to apply existing model checkers to verify timing properties of OSEK/VDX distributed application systems, Libor et al. have proposed a method [4] based on UPPAAL [2]. In the method, concurrent `process` (timed automaton) is employed to simulate the communication protocol, tasks within applications and OSs locating at different processors, respectively. For example, the method will use nine concurrent `process`es to simulate the OSEK/VDX distributed application system shown in Figure 1. Although this method is in a position to verify OSEK/VDX distributed application systems, it is often not capable of dealing with a large-scale system that holds a number of multi-tasking applications running on the several processors because too many concurrent `process`es are used to simulate the target system.

In this paper, we propose an approach to make model checking more scalable in verifying large-scale OSEK/VDX distributed application systems. The key idea of our approach is to reduce the number of concurrent `process`es by means of a sequentialization technique. In our approach, for a multi-tasking application within an OSEK/VDX distributed application system, we translate the application into a sequential model and then use only one concurrent `process` to simulate the application. In addition, the OSEK/VDX OS is embedded in the sequential translation algorithm to perform the scheduling behaviours in order to further simplify the checking model. Compared with the existing method, there are two advantages in our approach, e.g., (*i*) checking model constructed in our approach holds fewer concurrent `process`es, and (*ii*) the checking model does not hold the OSEK/VDX OSs locating at different processors. For example, our approach makes the checking model of the system shown in Figure 1 hold only three `process`es, i.e., one process is for CAN, and two are for the applications CCP and BCP.

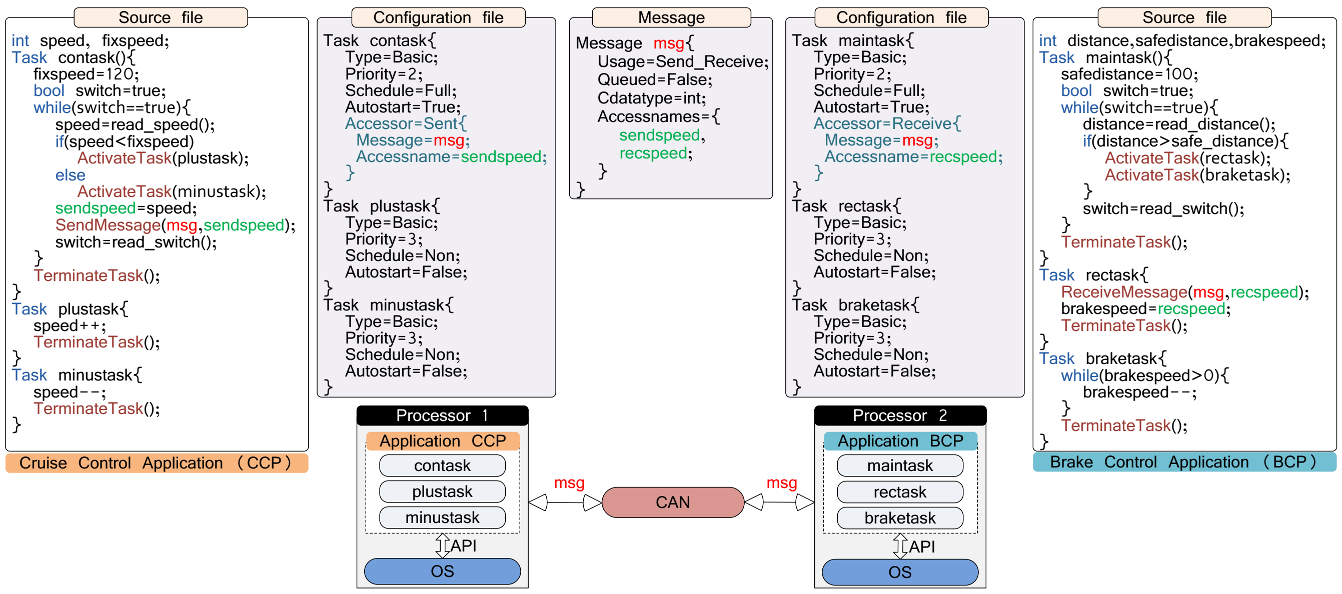
2 OUR APPROACH

The limitation in the existing method is mainly attributed to the fact that too many concurrent `process`es are included in the checking model. Our approach is characterized by its capability through translating an OSEK/VDX multi-tasking application into an equivalent sequential model. Based on the sequential model, we only need one concurrent `process` to simulate the multi-tasking application rather than multiple concurrent `process`es.

There are two problems that should be addressed when we translate an OSEK/VDX application into an equivalent sequential model, one is how to explicitly perform the scheduling behaviours of OSEK/VDX OS, and the other is how to compute the sequential model. In our approach, in order to explicitly perform the scheduling behaviours of the OSEK/VDX OS, we embed an OSEK/VDX OS model in the sequential algorithm for dispatching tasks and responding to

*Corresponding author, Email: zhuo_cheng@126.com

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194945



```
int speed, fixspeed;
Task contask(){
    fixspeed=120;
    bool switch=true;
    while(switch==true){
        speed=read_speed();
        if(speed<fixspeed)
            ActivateTask(plustask);
        else
            ActivateTask(minustask);
        sendspeed=speed;
        SendMessage(msg,sendspeed);
        switch=read_switch();
    }
    TerminateTask();
}
Task plustask{
    speed++;
    TerminateTask();
}
Task minustask{
    speed−−;
    TerminateTask();
}
```

```
Task contask{
    Type=Basic;
    Priority=2;
    Schedule=Full;
    Autostart=True;
    Accessor=Sent{
      Message=msg;
      Accessname=sendspeed;
    }
}
Task plustask{
    Type=Basic;
    Priority=3;
    Schedule=Non;
    Autostart=False;
}
Task minustask{
    Type=Basic;
    Priority=3;
    Schedule=Non;
    Autostart=False;
}
```

```
Message msg{
    Usage=Send_Receive;
    Queued=False;
    Cdatatype=int;
    Accessnames={
        sendspeed,
        recspeed;
    }
}
```

```
Task maintask{
    Type=Basic;
    Priority=2;
    Schedule=Full;
    Autostart=True;
    Accessor=Receive{
      Message=msg;
      Accessname=recspeed;
    }
}
Task rectask{
    Type=Basic;
    Priority=3;
    Schedule=Non;
    Autostart=False;
}
Task braketask{
    Type=Basic;
    Priority=3;
    Schedule=Non;
    Autostart=False;
}
```

```
int distance,safedistance,brakespeed;
Task maintask(){
    safedistance=100;
    bool switch=true;
    while(switch==true){
        distance=read_distance();
        if(distance>safe_distance){
            ActivateTask(rectask);
            ActivateTask(braketask);
        }
        switch=read_switch();
    }
    TerminateTask();
}
Task rectask{
    ReceiveMessage(msg,recspeed);
    brakespeed=recspeed;
    TerminateTask();
}
Task braketask{
    while(brakespeed>0){
        brakespeed−−;
    }
    TerminateTask();
}
```

Figure 1: Running example.

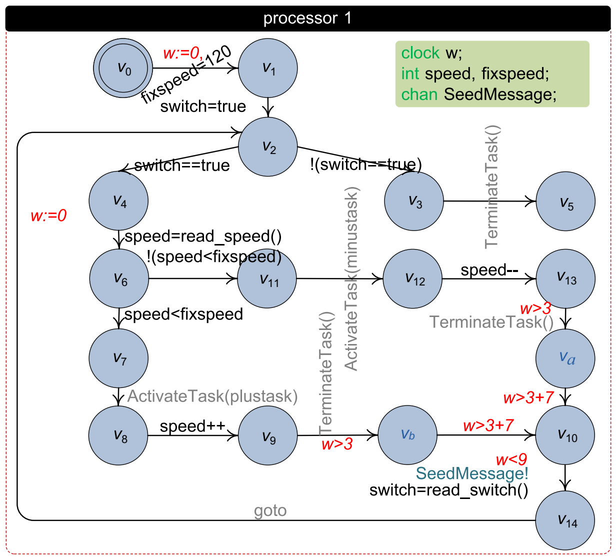


Figure 2: Sequential model for the application CCP.

the APIs invoked from tasks. In addition, to compute the sequential model of an OSEK/VDX application, an extended directed graph is employed to carry out sequential translation. In the translation process, the extended directed graph is used to execute the application in symbolic way and call embedded OS model to determine the running task when meeting an API. As shown in Figure 2, the application CCP included in the running example has been translated into a sequential model with one clock w, where the clock w is used to simulate timing constraints on instructions of tasks.

3 EVALUATION AND CONCLUSION

We have conducted a series of experiments to evaluate the efficiency and scalability of our approach. In the experiments, we firstly use our approach to translate all OSEK/VDX applications within an experimental distributed system into sequential models, and then employ model checker UPPAAL to perform the verification. The task number and processor number are the primary factors to influence the verification performance. We select the developed OSEK/VDX distrusted application systems with different task and processor numbers as benchmarks for the experiments. The experimental results indicate that our approach is not only capable of efficiently simplifying the checking models of OSEK/VDX distributed application systems, but also of making model checking competent in dealing with the OSEK/VDX distributed application systems with the industrial complexity.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation of China Grant Nos. 61602224 and 61472167 and 61732013, and also supported by JSPS KAKENHI Grant No. 26240008.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. E. Long. 1994. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 5 (1994), 1512–1542.
[2] G. Behrmann, A. David, and K. G. Larsen. 2004. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer Communication, and Software Systems.* 200–236.
[3] J. Lemieux. 2001. *Programming in the OSEK/VDX Environment.* CMP, Suite 200 Lawrence, KS 66046, USA.
[4] W. Libor, K. Jan, and H. Zdenek. 2009. Case study on distributed and fault tolerant system modeling based on timed automata. *Journal of Systems and Software* 82, 10 (2009), 1678–1694.