TimePF: A Tool for Modeling and Verifying Timing Requirements Based on Problem Frames

Yuanyang Wang, Xiaohong Chen[✉], and Ling Yin

Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
xhchen@sei.ecnu.edu.cn

Abstract. As the key element of embedded systems, the timing requirements are becoming more and more important. An increasing number of systems need strict time constraints especially some safety critical systems such as high-speed railway systems. We have proposed an approach for modeling and verifying timing requirements [1–3]. By combining the Problem Frames (PF) approach [4] and CCSL (Clock Constraint Specification Language) [5], it can model timing requirements from the perspective of environment and verify them with NuSMV. To support this approach, we develop a supporting tool (named TimePF) by extending the DPTool [6]. TimePF is a graphical tool which provides various modeling and verifying techniques for timing requirements. This paper presents its architecture and implementation, and gives an illustrating example to show how to use it following the modeling and verifying process.

1 Architecture and Implementation

1.1 Architecture

Fig.1 shows the architecture of TimePF. It includes 3 layers, i.e., data layer, function layer and interface layer. The function layer models and verifies timing requirements. The data layer provides data for function layer. And the interface layer is responsible for the interactions with users. We will introduce the function layer in detail. It includes modeling module and verifying module. The modeling module includes 2 sub-modules, i.e., data processing module and graphical operation module.

Modeling Module

- Data processing sub-module

This module accepts the inputs in terms of a problem diagram and scenario graphs. All the problem domains in problem diagram will be traversed to find interactions related to each problem domain. Meanwhile, all the paths in scenario diagram will be traversed to find the possible temporal relations among interactions.

- Graphical operation sub-module

This module edits graphic elements on the interface. It adds a clock for each interaction. All the clocks and clock relations form a clock diagram. In addition, this module defines composite clocks and additional constraints. These new added constraints will be checked to ensure that there are no conflicts among the new constraints and existing ones.

DOI: 10.1007/978-3-662-48634-4_11

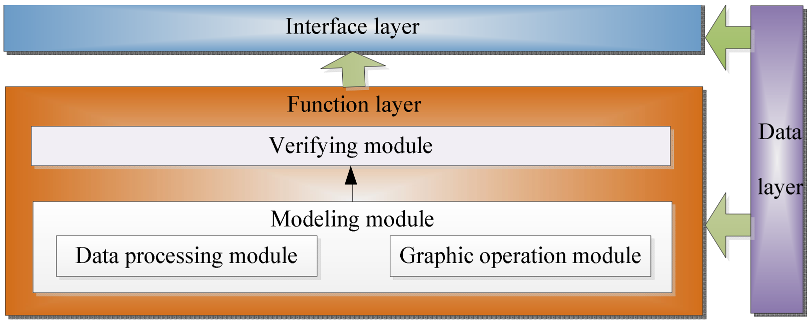
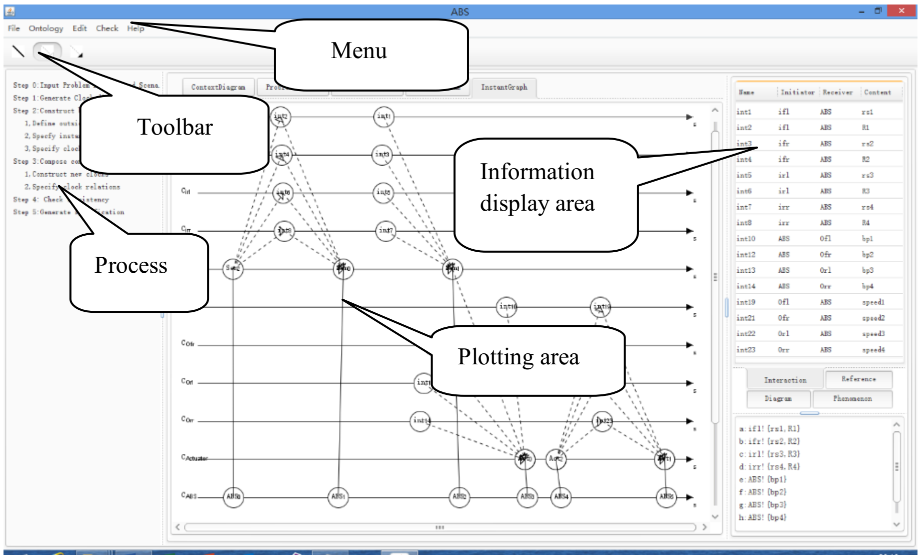


Fig. 1. Architecture of TimePF

Verifying Module
The verifying module transforms the CCSL description from the model into NuSMV description, and verifies the consistency of timing specification. The transformation follows the transforming rules. It also defines the consistency properties with CTL, and verifies these properties by calling NuSMV.

1.2 Implementation

The TimePF is implemented in Java. Fig.2 shows its snapshot. There are 5 major areas on the interface, i.e., *Menu*, *Toolbar*, *Process*, *Plotting* and *Information area*.



Name	Initiator	Receiver	Content
int1	ifl	ABS	rs1
int2	ifl	ABS	R1
int3	ifr	ABS	rs2
int4	ifr	ABS	R2
int5	irl	ABS	rs3
int6	irl	ABS	R3
int7	irr	ABS	rs4
int8	irr	ABS	R4
int10	ABS	Ofl	bp1
int12	ABS	Ofr	bp2
int13	ABS	Orl	bp3
int14	ABS	Orr	bp4
int19	Ofl	ABS	speed1
int21	Ofr	ABS	speed2
int22	Orl	ABS	speed3
int23	Orr	ABS	speed4

```
a: ifl! {rs1, R1}
b: ifr! {rs2, R2}
c: irl! {rs3, R3}
d: irr! {rs4, R4}
e: ABS! {bp1}
f: ABS! {bp2}
g: ABS! {bp3}
h: ABS! {bp4}
```

Fig. 2. Snapshot of TimePF

- There are 5 items in the *Menu*, i.e., *File*, *Ontology*, *Edit*, *Check*, *Help*. The *File* menu is responsible for creating, opening and saving a project. The *Ontology* menu can load, show and check environment ontology. The *Edit* menu edits operations about clock and clock constraint. And the *Check* menu executes some checking operations such as *Check consistency*. The *Help* menu shows some help information about the tool.
- The *Toolbar* gives three kinds of qualitative relations among clocks. They are *precedence*, *coincidence* and *strict precedence*.
- The area of *Process* shows the steps to be followed in the tool.
- The *Plotting area* shows the diagrams drew by the tool, including problem diagram, scenario diagram, clock diagram and time point diagram.
- The *Information area* shows some information related to the diagram in the *Plotting area.* The information includes diagrams, phenomenon, interactions and citation information.

2 Process and Illustrating Example

Fig.3 shows the process of modeling and verifying timing requirements. It includes 2 major steps, i.e., modeling timing requirements, and verifying timing specification. In order to be clearly understood, we use the Anti-lock Braking System (ABS) as an example to show the steps of using TimePF. The description of ABS is as follows.
It is composed by four sensors and four actuators. The four sensors (ifl, ifr, irl, irr) are used to measure the rotational *speed of wheels. And the four actuators (ofl, ofr, orr, orl) represent the braking pressure on each wheel. The ABS is triggered by R. The signals of four sensors must arrive in a certain input delay (for example 0.5ms).*

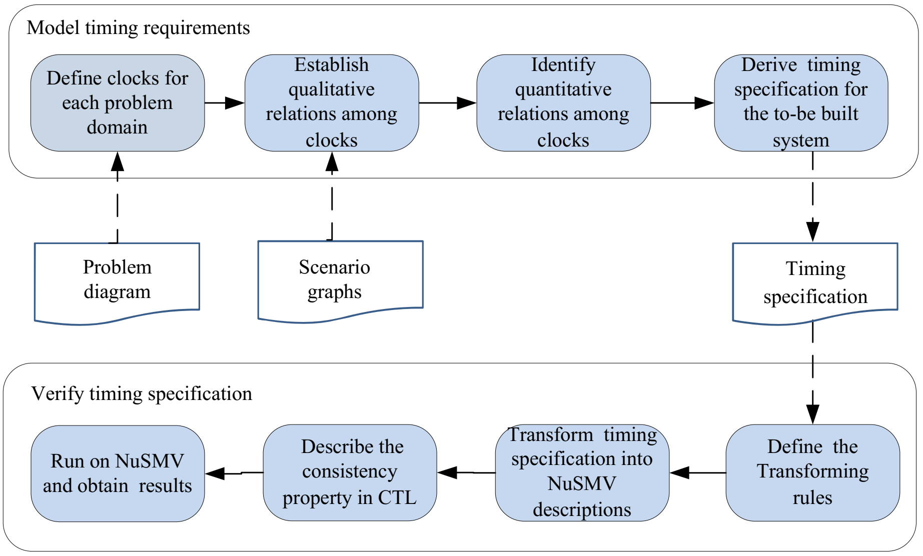
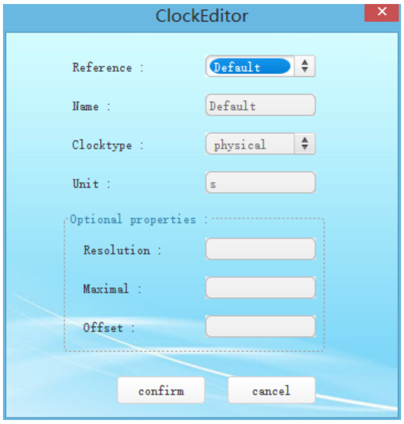

Model timing requirements
Define clocks for each problem domain
Establish qualitative relations among clocks
Identify quantitative relations among clocks
Derive timing specification for the to-be built system
Problem diagram
Scenario graphs
Timing specification
Verify timing specification
Run on NuSMV and obtain results
Describe the consistency property in CTL
Transform timing specification into NuSMV descriptions
Define the Transforming rules

Fig. 3. Process for modeling and verifying timing requirements

Step 1: Model Timing Requirements. The inputs of this step are a problem diagram and scenario graphs. The output is the timing specification of the to-be built system. It has 4 sub-steps:

1) Define clocks for each problem domain in the problem diagram. A clock is defined as *C*:=< *I*, <>, where, C is the clock, *I* is the set of time points, and < is the partial order relation defined on *I* which named *StricPre*. Clocks are classified into two kinds, domain clocks and interaction clocks. The domain clock is expressed as *d.C*, where *d* represents the problem domain and *C* represents the clock. And the interaction clock can be defined as *int.C,* where *int* represents the interaction and *C* represents the clock. Since each problem domain can initiate or receive many interactions, each domain clock is composed by the interaction clocks of this domain using the union operator of CCSL [6]. If a domain is composite, its clock will be established by the clocks of sub-domains using clock operators of CCSL including *sup*, *inf* and *union* [6].

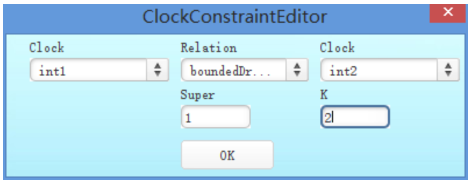

ClockEditor

Reference :	Default
Name :	Default
Clocktype :	physical
Unit :	s

Optional properties :

Resolution :	
Maximal :	
Offset :	

Fig. 4. Define domain clocks


ClockConstraintEditor

Clock	Relation	Clock
int1	boundedDr...	int2
	Super: 1	K: 2

Fig. 5. Define clock constraints

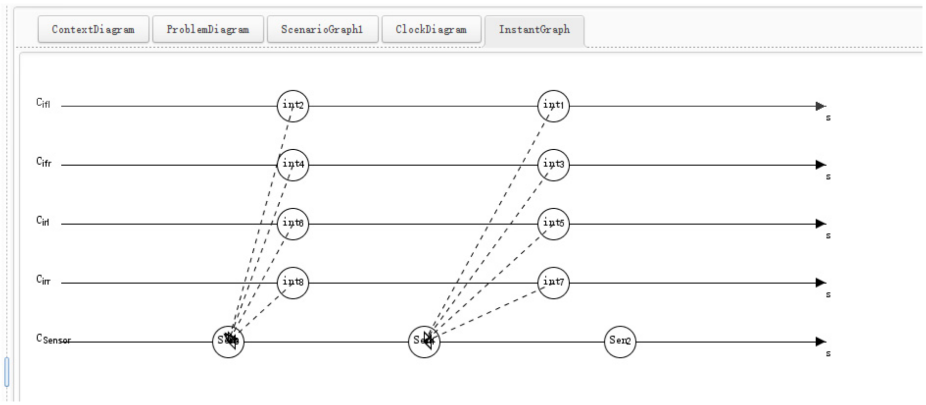


Fig. 6. Clock relations

Example: *In the ABS system, int_1 is an interaction that ABS software initiates and domain ifl receives R. It has a corresponding clock, $int_1.C_1$. Domain ifl has a clock too, $ifl.C_{ifl}$. As domain ifl has two related interactions, int_1 and int_2, then the Time-PF automatically generates $ifl.C_{ifl}= int_1.C_{1ifl}$ union $int_2.C_{2ifl}$. In the tool, Clock editor (Fig. 4) defines domain clock, and union constraints are shown in Fig.6.*

2) Establish qualitative relations among clocks. The qualitative relation operators provided by CCSL consist of *subclock*, *fasterThan* and *alternate*. These qualitative relations could be obtained directly from existing scenario graphs. We argue that interaction relations of the same problem domain could be directly transformed to the following clock relations:

- *int_1 behOrd int_2* implies *C_{int1} strictPre C_{int2}*.
- *int_1 behEna int_2* implies *C_{int1} strictPre C_{int2}*.
- *int_1 reqOrd int_2* implies *C_{int1} strictPre C_{int2}*.
- *int_1 syncBehReq int_2* implies *C_{int1} subClock C_{int2}*, and *C_{int2} subclock C_{int1}*.

Example: *In the scenario graph of ABS**,** int_1 behOrd int_2, then* we get *C_{int1} strictPre C_{int2} (Fig.6). This is generated automatically.*

3) Identify quantitative relations among clocks expressed as *C_1 bounderDrift(i, j) C_2*. It means that the time interval between the time points C_1 and C_2 is within the range $[i, j]$, where i is a negative integer and j is a positive integer.

Example: *In the ABS, int1 happens before int2 is within the range [1, 2], then we use bounderDrift relation as shown in Fig.5.*

4) Derive the timing specification of the to-be built system. Firstly, define clock C_{sys} for system by *union* each problem domain clock. The timing specification contains three parts: system clock C_{sys}, related clocks of C_{sys} and clock relations including the qualitative relations obtained from step 2) and quantitative relations obtained from step 3).

Example: *In the ABS system, we define C_{ABS} for the system. It is composed by Sensor and Actuator. So we get $C_{ABS} = C_{Sensor}$ union $C_{Actuator}$. The Sensor clock has relations with other clocks as shown in Fig.6. Finally, a specification in format of .txt is obtained.*

Table 1. Transformation rules from timing specification to the NuSMV descriptions

Timing specification in LTS	NuSMV descriptions
Clock C	C:boolean
Clock relation	constraint MODULE
State s	VAR s:boolean;

Transition
s {C_1, C_2,…,C_n} s’
	TRANS
	case
	s=TRUE:
	next(C_1)=TRUE & next(C_2)=TRUE &…& next(C_n)=TRUE &
	next(s)=FALSE & next(s’)=TRUE
	case;

Step 2: Verify Timing Specification. The input of this step is the timing specification obtained in step 1. And the output of this step is the verification results and the verified timing specification. It has 4 sub-steps:

1) Define the transformation algorithm from the timing specification to the descriptions of NuSMV. Firstly, we define the operational semantics of timing specification using Labelled Transition System (LTS). By direct mapping, the transformation rules form LTS to NuSMV descriptions are given in Table 1. Then the transformation algorithm could be defined (omitted due to limited space).

2) Transform the timing specification into the NuSMV descriptions according to the transformation rules in 1).

***Example**: According to the transformation algorithm, the constraint* $ifl.C_{ifl}= int_1.C_{1ifl}$ *union* $int_2.C_{2ifl}$ *could be transformed into ctrl: union* (C_1, C_2, C_{ifl})*. According to the transformation rules in Table 1, union could be transformed to:*

```
        MODULE union(left, right, new)
        TRANS
           (next(left)=TRUE & next(new)=TRUE)|(next(right)=TRUE &
  next(new)=TRUE)|(next(left)=FALSE & next(right)=FALSE & next(new)=FALSE)
```

3) Describe the consistency property of timing specification in CTL. For the timing specification with clock set T= $\{C_1, C_2,\ldots,C_n\}$, we say it is consistent if it satisfies the following two CTL formula:

- *EF(AGp)*, where $p=!(C_1|C_2|\ldots|C_n)$;
- $\forall C_i \in$ T, *EF (AGq)*, where $q=!C_i$.

***Example**: In the ABS system,* $T=\{C_{ABS}, C_{Sensor}, C_{Actuator}, C_{ifl}, C_{ifr}, C_{irl}, C_{irr}, C_{ofl}, C_{ofr}, C_{orl}, C_{orr}, C_{1ifl}, C_{2ifl}, C_{1irr}, C_{2irr}, C_{1irl}, C_{2ifr}, C_{2irl}, C_{1ifr}, C_{3ofl}, C_{4ofl}, C_{3ofr}, C_{4ofr}, C_{3orr}, C_{4orr}, C_{3orl}, C_{4orl}, C_{int1}, C_{int2}, C_{int22}, C_{int3}, C_{1Sensor}, C_{1sensor2}, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{sen1}, C_{sen2}, C_{sen3}, C_{act1}, C_{act2}, C_{act3}\}$*. Its consistency formula is generated automatically by our tool.*

4) Check the consistency of timing specification using NuSMV, and obtain the verification results and the verified timing specification.

***Example**: After clicking Check consistency menu, the consistency of the ABS system could be checked automatically. There is no inconsistency condition. Then the timing specification could be output in the format of .txt at the same folder with the project.*

Acknowledgement. This research was supported by the National Natural Science Foundation of China under grants 61202104 and 91418203, and the Doctoral Fund of Ministry of Education of China under Grant 20120076120016.

References

1. Chen, X., Liu, J., Mallet, F., Jin, Z.: Modeling timing requirements in problem frames using CCSL. In: Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC 2011), pp. 381–388 (2011)
2. Yin, L., Chen, X., Liu, J.: Consistency analysis of timing requirements for cyber-physical system. Journal of Software, 25(2), 400−418 (2014, in Chinese)
3. Chen, X., Liu, J.: Modeling Software Timing Requirements: An Environment Based Approach, 36(1), 88–103 (2013, in Chinese)
4. Jackson, M.: Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley (2001)
5. Mallet, F.: Clock constraint specification language: specifying clock constraints with UML/MARTE. Innovations in Systems and Software Engineering. **4**(3), 309–314 (2008)
6. Chen, X., Yin, B., Jin, Z.: DPTool: a tool for guiding the problem description and the problem projection. In: Proceeding of the 18th IEEE International Requirements Engineering Conference, pp. 401–402 (2010)