Received January 20, 2019, accepted February 3, 2019, date of publication February 15, 2019, date of current version March 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2899761

Real-Time System Modeling and Verification Through Labeled Transition System Analyzer

YILONG YANG[1], QUAN ZU[1,2], WEI KE[3], MIAOMIAO ZHANG[2], AND XIAOSHAN LI[1]
[1]Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau 999087, China
[2]School of Software Engineering, Tongji University, Shanghai 201804, China
[3]Macau Polytechnic Institute, Macau 999087, China

Corresponding author: Xiaoshan Li (xsl@umac.mo)

This work was supported in part by the Macau Science and Technology Development Fund (FDCT) under Grant 103/2015/A3, in part by the University of Macau under Grant MYRG 2017-00141-FST, and in part by the National Natural Science Foundation of China (NSFC) under Grant 61562011 and Grant 61472279.

ABSTRACT Model checking as a computer-assisted verification method is widely used in many fields to verify whether a design model satisfies the requirements specifications of the target system. In practice, it is difficult to design a system without the sophisticated requirements analysis. Unlike other model checking tools, the labeled transition system analyzer (LTSA) not only can specify the property specifications of the target system but also provides a structure diagram to specify the system architecture of the requirements model, which can be further used to design the target system. In this paper, we demonstrate the abilities of LTSA shipped with the classic case study of the steam boiler system. In the requirements analysis, the LTSA can specify the cyber and physical components of the target system and interactions between the components and the safety properties of the target system. In system design, the LTSA can automatically generate a start-up design model as the finite state process from the requirements model, and then a design model can be further accomplished by system architects and developers. Finally, the LTSA can automatically verify whether the design model meets the requirements specifications. Our work demonstrates the potential power of model checking tools can be applied and useful in software engineering for requirements analysis, system design, and verification.

INDEX TERMS LTSA, model checking, steam boiler, UML.

I. INTRODUCTION

Real-time computing [1] describes hardware and software systems depending not only on the logical correctness of computation but also on the time constraints. The critical real-time systems must be dependable because any failure of the system could cause an economic disaster or even loss of human lives. The safety properties of such systems must be verified in the design stage as well as the testing stage of development before any deployment. Formal methods [2], [3] provide promising approaches to verify the safety properties of system. The papers [4]–[7] provide the case studies of general train control system, pacemaker, and cell phone, to illustrate how to verify the safety properties of the real-time systems [8], [9]. Model checking [10], [11] is a promising approach of formal methods, which can automatically verify the safety and liveness properties by exploring the state space of the target system. Compared with other formal methods, it has threefold strengths: a) Model checking is a systemic and algorithmic methodology which can be fully automated without the assistance of verification experts. b) Model checking can be applied at different stages of software engineering from requirements modeling system design, to system implementation. c) Model checking is particularly well suited for finding concurrency bugs and proving their absence, based on the algorithmic exploration of large state space.

In this paper, we use the model checking tool: Labeled Transition System Analyzer (LTSA), to model real-time systems and verify the properties of the system [12]. Because it has many good features: 1) LTSA provides both graphic and textual models to describe the target system. 2) LTSA can automatically generate state machines from textual process models. 3) LTSA provides structure diagrams to describe system architectures for requirements modeling and analysis [13]. 4) LTSA can automatically generate the skeleton of textual process model from system architectures for the next round fine-grained design [14].

Although LTSA has many good features, there are few examples of real-time systems modeling through LTSA.

The associate editor coordinating the review of this manuscript and approving it for publication was Roberto Pietrantuono.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

The steam boiler [15], [16] is a classic case study for real-time system modeling in many studies. The steam boiler is the minimal real-time system that contains all the essential parts of a real-time system: a controller and a controlled object with the sensors and the actuators. The controller can periodically sample the state of the controlled object by the sensors, then strategically change the state of the controlled object through the actuators to guarantee the safety of the whole system. All the components of the real-time system are synchronized through time. In the physical world, time is implicitly contained in the physical phenomenon; e.g., the current temperature of the water was determined by the heating power and the heating time. Therefore, when modeling the real-time system, the time should be considered as inside of physical law of the controller object. However, for abstraction of the physical world and easy understanding, many studies explicitly model the time as a component of the system named timer, and then use that timer component to synchronize with other components of the system.

A. RELATED WORK

As a classic real-time system, the steam boiler has been widely studied. The paper [17] presents a formalization of the steam boiler problem using Circus, which is a unified theory of Z and CSP. It utilizes the strength of Z notation to describe the specifications of the system and their refinement, the strength of CSP to describe and reason about concurrency. Using Timed Automata to solve the steam boiler problem is mentioned in [18], which describes time constraints in the model with clocks and certain safety requirements can be guaranteed under the described assumptions and failure model. The paper [19] adopts Mean Value Calculus to model the steam boiler, which could be used to specify and reason about time and logical constraint of the real-time system. The paper [20] uses SPIN to model and verify the steam boiler system and generate the PROMELA source code. Signal-coq [21] combines the programming language SIGNAL and Coq proof assistant to model and verify the steam boiler system. It demonstrates that Signal-coq is well-suited and practical approach for the validation of reactive systems. Action System [22] is an approach to the specification of control programs based on action systems and refinement, which is demonstrated through steam boiler system. PLUSS [23] combines algebraic specification with a notion of implicit state to model and verify steam boiler system. FOCUS [24] demonstrates the usefulness of FOCUS by its application of the requirements specification of a steam boiler. LOTOS [25] presents a technique for the automated verification of an object-oriented language OBject LOGic (OBLOG) through the steam boiler case study.

Although the steam boiler problem has been elaborately studied, all case studies above have the following defects: 1) They directly present a design model from the description of the target problem without the sophisticated requirements modeling and analysis, which is not the case of the practical projects. 2) They do not describe the system architecture, that will make it hard to elicit the components from the target problem. 3) They do not illustrate the differences between the implicit timer and explicit timer, and the relationship between the physical world timer and timer in the real-time modeling.

B. CONTRIBUTIONS

Although most model checking tools support system modeling and verification, to our best knowledge, there is no one that involves all stages from requirements modeling, system design, and verification. The contributions of our paper are:

1) Demonstrating LTSA for requirements modeling and analysis, and then generate the high level of design model from the structure diagram.
2) Discussing how to model the explicit and implicit timer in the real-time system, and the relationship of time between the system modeling and physical world.
3) Demonstrating LTSA requirements modeling, system design, and verification abilities for real-time system shipped by a classical steam boiler problem.

The remainder of this paper is organized as follows: Section 2 is preliminary of FSP specification and overviews the steam boiler problem. Section 3 shows the requirements modeling and analysis. Section 4 presents how to archive the design model of the steam boiler in LTSA, especially for the time modeling. And then Section 5 shows requirements verification. Finally, Section 6 concludes this paper and puts forward the future work.

II. PRELIMINARY

LTSA adopts FSP as textual model to describe the system. To make this paper self-contained, we present the specification of FSP and the brief introduction of steam boiler problem in this section. The more details of FSP could be found in the textbook [12].

A. THE SPECIFICATION OF FSP

Finite State Process (FSP) is CSP syntax-liked formal language for modeling concurrency system [12], it uses concept of *Primitive Process* to define the component of the system which contains the sequences of *actions*. Component composition could be defined as *Composited Processes* in which concurrent executions of actions could be synchronized or interleaved. The requirements of system could be captured as the *Properties* of FSP. Once both properties and processes of the system are defined, LTSA can check the satisfiability of properties for a particular system. The brief summary of FSP specification is provided as follows:

1) PRIMITIVE PROCESS

A primitive process is the execution of a sequential program. The state of primitive process is transformed by executing actions. We use primitive process to define the component of the system. Like any programming languages, primitive process may contain control flow such as choice and condition.

- Action Prefix −>: (a −> P) describes a process which engages in the action a and then behaves as described by P.

- Choice | : (a −> P | b −> Q) describes a process which initially engages in either of the actions a or b. After the first action has been performed, the subsequent behavior is described by P if the first event was a, or by Q if the first event was b.
- STOP: It is sometimes (if rarely) necessary to specify a primitive process which terminates. Consequently, a local process STOP is predefined which engages in no further actions.
- Alphabet Extension + { } : Each primitive process has an alphabet consisting of the actions it may take part in. A process may only engage in the actions contained in its alphabet although the converse does not hold. It is sometimes useful to extend the alphabet of a process with actions that it does not engage in and consequently actions that are not used in its definition. This may be done to prevent another process executing the action.
- Indexing: Both local process names and action names may be indexed. Both local processes and actions may have more than one index as illustrated by this example (for actions). A process which outputs the sum of two integers (in the range 0..N).
- Conditional: A conditional takes the form: if *expr* then *local_process* else *local_process*.
- Guards: A guarded transition takes the form (when B a −> P) which means that the action a is eligible when the guard B is true, otherwise a cannot be chosen for execution.

2) COMPOSITED PROCESSES

Parallel Composition ‖: (P ‖ Q) expresses the parallel composition of the processes P and Q. It allows all the possible interleaving of the actions of the two processes. The shared actions synchronize the execution of the two processes. If the processes contain no shared actions then the composite state machine will describe all interleaving.

3) SAFETY PROPERTIES

A safety property asserts that nothing bad happens. Informally, a property process specifies a set of acceptable behaviors for the system it is composed with. A system S will satisfy a property P if S can only generate sequences of actions (traces) which when restricted to the alphabet of P, are acceptable to P.

4) PROGRESS PROPERTIES (LIVENESS)

A progress property asserts that it is always the case that an action is eventually executed. We will define progress to check the steam boiler is still work or not. Liveness and progress are exchangeable concepts in the remains of this paper.

B. STEAM BOILER PROBLEM

The steam boiler problem [15], [16] is one of the typical real-time system, which divides the system into several components. It has a physical steam boiler, three water/steam/pump sensors and pumps, and a controller which could get the value from the sensors, make a decision, and send orders of switching on/off to the pumps. The communication among components are through messages.

TABLE 1. The summary of constants and variables.

Type	Unit	Comment
Interval	$\mathbb{N}$	Sample Period/Delay of PumpOn
Quantity of Water		
C	litre	Maximal Capacity
M_1	litre	Minimal Limit
M_2	litre	Maximal Limit
N_1	litre	Minimal Normal Limit
N_2	litre	Maximal Normal Limit
Outcome of Steam		
W	litre/sec	Maximal Rate
U_1	litre/sec/sec	Increase Rate
U_2	litre/sec/sec	Decrease Rate
Pump Parameters		
P	litre/sec	Capacity of each Pump
Current Variables		
q	litre	Quantity of Water
v	litre/sec	Steam Rate
p	litre/sec	Throughput of pumps

The summary of constants and variables are in Table 1. The constants are as follows: *Interval* is the sample cycle and delay of *PumpOn* action, C is the maximal capacity of the steam boiler. M_1 is minimal limit of water quantity, M_2 is maximal limitation, N_1 is minimal limitation and N_2 is maximal limitation in normal mode, W is maximal quantity of outcome steam, the increase rate of outcome steam are defined by U_1, the decrease rate of outcome steam is U_2, the number of pumps is 5, P is capacity of each pump. Those constants satisfy:

$$0 < M_1 < N_1 < N_2 < M_2 < C$$

Variable q represents the quantity of water, v is the current outcome rate of steam, p is the current throughput of pumps. Those variables must be satisfied the following invariants:

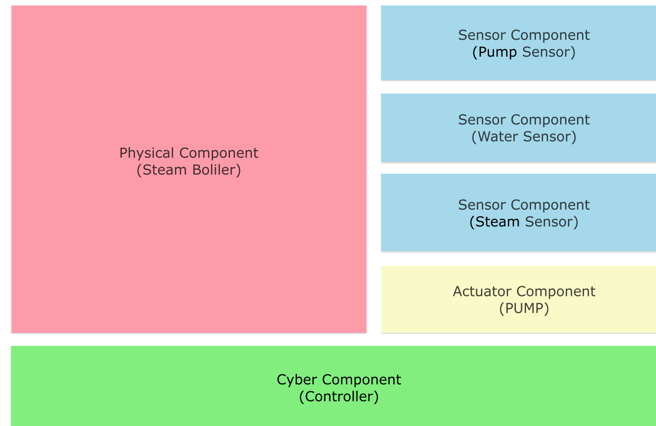
$$0 \leq q \leq C$$
$$0 \leq v \leq W$$
$$0 \leq p \leq 5P$$

III. REQUIREMENTS MODELING AND ANALYSIS

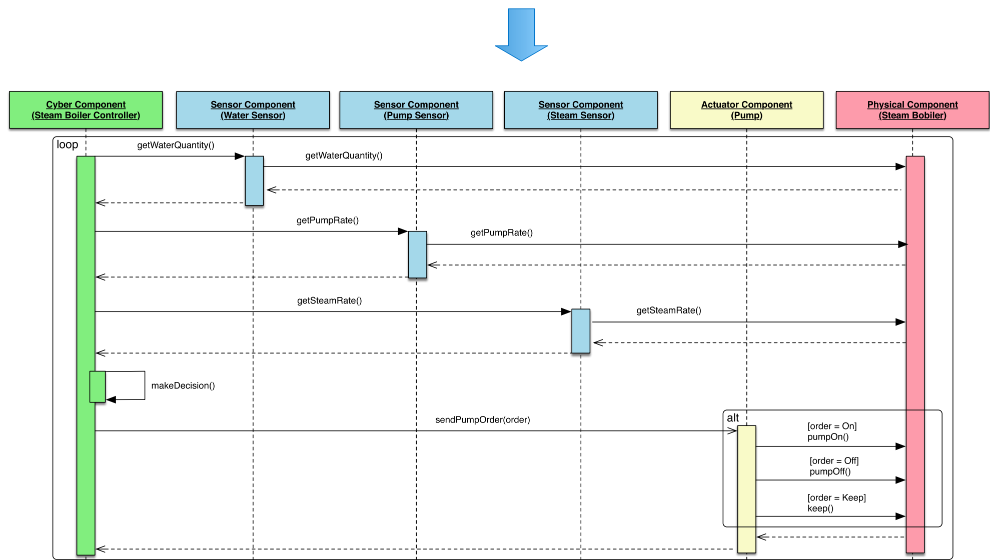
In this section, we use LTSA to do requirements modeling and analysis, which is shown in Figure 1.

1) We identity the components of system based on the description of the steam boiler problem in the last section. The steam boiler system contains a physical component - steam boiler, three sensor components - pump, water, and steam sensors, and a cyber-component - controller.

1. System Components



2. Component Sequence Diagram



3. Structure Diagram

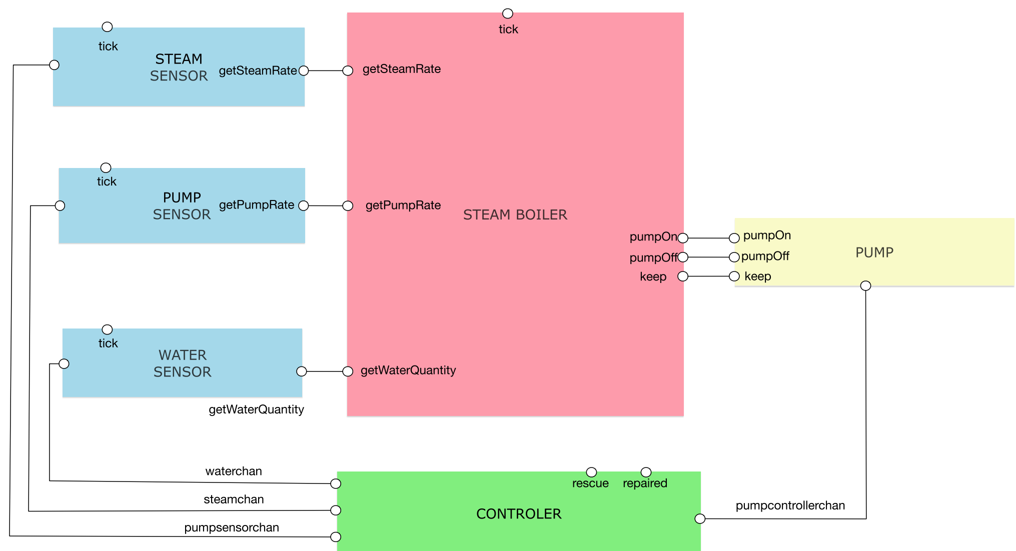


FIGURE 1. **Requirements modeling through LTSA.**

2) Based on the identified components, we further identify the external interactions among the components in a component sequence diagram. In which, the cyber component - steam boiler controller periodically gets the values of steam rate v, pump rate p, and water quantity q from the sensor components - water, pump, and steam sensors, and then sends the pump order to actuator component, the actuator will open or close the pump of steam boiler based on the order. Moreover, we specify the safety and progress properties in requirements model. In steam boiler system, the interactions of components must make the water quantity q kept between N_1 and N_2 in normal mode, and between M_1 and M_2 in the rescue mode when correct q cannot be obtained from the water sensor. That is called as safety properties in real-time system. Formally, the safety properties of the steam boiler system are described as follows:

$$REQ_{NormalMode} \mathrel{\widehat{=}} N_1 \leq q \leq N_2 \tag{1}$$

$$REQ_{RescueMode} \mathrel{\widehat{=}} M_1 \leq q \leq M_2 \tag{2}$$

Livelock-free and deadlock-free properties are describe as:

$$REQ_{AllMode} \mathrel{\widehat{=}} livelockfree(SteamBoiler) \tag{3}$$

$$REQ_{AllMode} \mathrel{\widehat{=}} deadlockfree(SteamBoiler) \tag{4}$$

Note that component sequence diagram is different from sequence diagram, it is used to identify the external interactions among the component as the parts of a design model.

3) From the identified interaction of component sequence diagram, we can smoothly derive the FSP structure diagram. This diagram connects components with their actions in the system. The component STEAM BOILER represents the physical steam boiler, which is assumed to keep boiling (boiling action) all the time. The state of STEAM BOILER includes the quantity of water, the quantity of steam and the throughput of the pumps, which are respectively denoted by the variables p, v, q. They are measured in a fixed sampling cycle via the sensors STEAM SENSOR, PUMP SENSOR and WATER SENSOR by the actions *getPumpRate*, *getSteamRate* and *getWaterQuantity* respectively. The state of STEAM BOILER is changed according to the state of PUMP, which is controlled by the actions *pumpOn*, *pumpOff* and *keep*. In each sampling cycle, the control system CONTROLLER receives the measures of p, v, q, according to which CONTROLLER will decide the message (*pumpOn* or *pumpOff* or *keep*) sent to PUMP through the channel *pumpcontrollerchannel*. Each message triggers the corresponding action of PUMP.

IV. SYSTEM DESIGN

In this section, we generate high level design of FSP processes and action sets from the FSP structure diagram by the ADL plugin of LTSA - Darwin [13], then present the details of design for each FSP component.

A. FSP GENERATION

LTSA with Darwin can automatically generate a FSP process for each component from structure diagram and then composite them as system design. Moreover, it can generate an action set for each component. For example, steam boiler component is included in structure diagram of steam boiler. Darwin can generate action sets, which contain *getSteamRate*, *getPumpRate*, *getWaterQuantity*, *pumpOn*, *pumpOff*, and *keep*. From the FSP structure diagram in Figure 1, SYSTEMDESIGN is generated as follows:

```
||SYSTEMDESIGN = (STEAMBOILER || CONTROLSYSTEM || PUMPCONTROLLER
        || WATERSENSOR || STEAMSENSOR || PUMPSENSOR || TIMER)
```

The sets of actions of each component are as follows:

```
set Timer = {tick}
set PumpSensor = {tick, getPumpRate, pumpsensorchan}
set SteamSensor = {tick, getSteamRate, steamchan}
set WaterSensor = {tick, getWaterQuantity, waterchan}
set Pump = {pumpOn, pumpOff, keep, pumpcontrollerchan.}
set SteamBoiler = {tick, getPumpRate, getSteamRate,
            getWaterQuantity, pumpOn, pumpOff, keep}
set Controller = {rescue, repaired, makedecision, waterchan,
        pumpsensorchan, steamchan, pumpcontrollerchan}
```

We present the detail design of each component in remaining subsections.

B. TIMER

Assume there is a start time denoted by t_0. We use t_i, $i \in \mathbb{N}$, to denote the time point which is elapsed i seconds since the start time. Therefore, the trace of time is represented as $\langle t_0, t_1, \ldots, t_i, \ldots, t_n \rangle$ in the system. For example, since the sample period is 5 seconds in this system, the next sampling point will be $t_{i+5} = t_i + 5$ if the previous sampling point is t_i. For q and v, they would be changed in every second. For p, if sampling starts from time point t_i, sampling period is 5 seconds, it would only be changed in the time points $\{t_{i+5j} \mid j \in \mathbb{N}\}$.

In order to model the time in the system, we use a component TIMER that synchronizes with STEAM BOILER and all the sensor components by the action *tick*. TIMER in FSP form is:

```
TIMER =  (tick -> TIMER)
```

where each *tick* represents the pass of one second. Note that if the sampling period is identical with the delay of pumping, which is the case in the steam boiler specification [15], we can model the system in a time-implicit way. That is, we don't need a Timer component to explicitly specify the time passing. We compare these two different modeling methods in the next subsection.

C. STEAM BOILER COMPONENT

The specification [15] specifies part of the behaviors in the steam boiler system. For instance, after switching on the pump, the water starts pouring into the boiler in 5 seconds. But some details are not given, including the variation law of the steam rate and the control strategy of the pump, which is crucial in the real industrial case. We design a pumping control strategy in next subsection. In this subsection, we hypothesize a physical variation law for the steam rate and show the model of the steam boiler in FSP.

1) QUANTITY OF WATER

We use q_i, v_i, p_i to respectively denote the quality of water, the value of steam rate, and the value of the pumping rate at the t_i time point. Then clearly we have the following equation:

$$q_{i+1} = q_i + (p_i - v_i) * \Delta t \tag{5}$$

where $\Delta t = 1$.

2) STEAM RATE

The hypothesis of the law of the steam rate is based on the fact that the steam rate is influenced by the quantity of the water. Between N_1 and N_2, we add another two quantities of water level B_1 and B_2, the best minimal limit and the best maximal limit, which are used to construct the hypothesis. We use VMINOUT to denote the minimal steam rate. It is required that $N_1 < B_1 < B_2 < N_2$ and $0 <$ VMINOUT $< min\{U_1, U_2\}$ are hold. We use v_i to denote the steam rate at the t_i time point. The law of steam rate is depicted in Figure 2 and Equation 6.

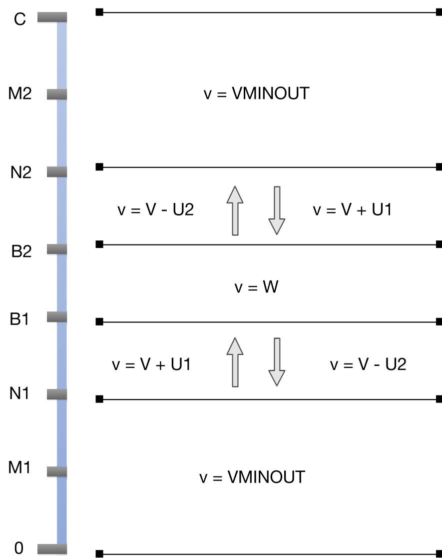


FIGURE 2. Steam rate.

If current water level is between N_2 and C or between N_1 and 0, the steam rate is minimal, denoted by VMINOUT. If current water level is between B_1 and B_2, the rate is maximal, denoted by W. If current water level is between N_1 and B_1 and it is increasing ($p_i - v_i > 0$), or if the current water level is between B_2 and N_2 and it is decreasing ($p_i - v_i < 0$), the steam rate will have an increment of U_1. If current water level is between N_1 and B_1 and it is decreasing ($p_i - v_i < 0$), or if the current water level is between B_2 and N_2 and it is increasing ($p_i - v_i > 0$), the steam rate will have a decrement of U_2.

$$v_{i+1} = \begin{cases} \text{VMINOUT} & \text{if } N_2 \leq q_i \leq C \text{ or } 0 \leq q_i \leq N_1 \\ v_i - U_2 & \text{if } B_2 \leq q_i < N_2 \text{ and } p_i - v_i > 0 \\ v_i - U_2 & \text{if } N_1 < q_i < B_1 \text{ and } p_i - v_i < 0 \\ v_i + U_1 & \text{if } B_2 \leq q_i < N_2 \text{ and } p_i - v_i < 0 \\ v_i + U_1 & \text{if } N_1 < q_i < B_1 \text{ and } p_i - v_i > 0 \\ W & \text{if } B_1 \leq q_i \leq B_2 \end{cases} \tag{6}$$

3) THROUGHPUT OF PUMPS

It is specified that, after switching on, the pump needs 5 seconds to pour the water into the boiler. That is, given the action pumpOn is triggered at t_i, the pumping rate p will have an increment of P at $t_i + 5$, assuming that there is at most one pump opening in one second. We use a sequence of 5 bits to represent the actions of pumping in the next 5 seconds. The position of a bit represents a time point, where the rightmost bit or the last one represents the current time point and the leftmost one or the first one represents the 5th time point. At the end of each second, the last bit will be checked, where 1 implies the pumping rate will increase and 0 implies no action. In the meantime, each bit in the sequence will move to right for one position, where the original last one is removed and the new first one is set to 0. We use integer t to represent the sequence of 5 bits, i.e., utilizing the binary representation of the integer to represent the sequence of bits. Thus, when the action pumpOn is triggered at a point, we increase t by 16, which in binary is 10000, implying that there is a pumping rate increment in 5 seconds. Note that due to the physical features of the steam boiler, only the action pumpOn has the 5 seconds delay for increasing the pumping rate but not the case for the action pumpOff. The action pumpOff will immediately decrease the pumping rate without any time delay.

4) STEAM BOILER IN FSP

STEAMBOILER has two subcomponents STEAMBOILERUN and PUMPDELAY. The sensors get the measures from the steam boiler by the actions *getPumpRate*, *getSteamRate* and *getWaterQuantity*. The steam boiler communicates with the pump by the actions *pumpOn*, *pumpOff* and *keep*. If *pumpOff*, the pump rate decreases immediately. If *pumpOn*, t is increased by 16, which is 10000 in binary. The steam boiler synchronizes with Timer by the action *tick*. In each second, the quantity of the water is changing according to Formula 5 and the steam rate changes according to the law. The pumping rate will be checked in PUMPDELAY. It firstly checks whether the last bit is 1, representing current increment. Then it moves the sequence one position to the right by the division by 2. The following is part of the model.

```
STEAMBOILER = (start->STEAMBOILERUN[INITQ][W][PUMPQ][0]),
    STEAMBOILERUN[q:Q][v:V][p:PUMPQ][t:PUMPMAXDELAY] = (
      getWaterQuantity[q] -> getSteamRate[v]
        -> getPumpRate[p] -> STEAMBOILERUN[q][v][p][t]
    | pumpOn -> STEAMBOILERUN[q][v][p][16+t]
    | pumpOff -> STEAMBOILERUN[q][v][p-PQ][t]
    | keep -> STEAMBOILERUN[q][v][p][t]
    | tick -> (
       when (q >= N2 )
        boiling -> PUMPDELAY[q+(p-v)][VMINOUT][p][t]
     | when (BEST2 < q && q < N2 && (p-v) < 0 && (v+UP) < W)
        boilingBEST2toN21[q][v][p] ->
                        PUMPDELAY[q+(p-v)][v+UP][p][t]
        ...... ) ),
    PUMPDELAY[q:Q][v:V][p:P][t:PUMPMAXDELAY] = (
      when (t % 2 == 0) pumping -> STEAMBOILERUN[q][v][p][t/2]
    | when (t % 2 != 0) pumping ->
                            STEAMBOILERUN[q][v][p+PQ][(t-1)/2]).
```

5) THROUGHPUT OF PUMPS IN IMPLICIT TIME

In the case of the sampling period is identical with the pumping delay time, we can use the implicit way to model the system. For the function of p, when the action *pumpOn* is triggered at t_i, the fixed delay is required before p is changed. Therefore, p is unchanged from the time point t_i to t_{i+4}. The value of p_{i+5} is determined by the following factors: 1) p_i, 2) whether the last pump order $lastpo \in \{True, False\}$ is *pumpOn* or not, 3) the previous order $actionp \in \{pumpOn, pumpOff, keep\}$, and 4) the current order $actionc \in \{pumpOn, pumpOff, keep\}$ of the pump.

$$p_{i+5} = \begin{cases} p_i+P & \text{if } lastpo = True \text{ and } actionc = pumpOn \\ p_i+P & \text{if } lastpo = True \text{ and } actionc = keep \\ p_i & \text{if } lastpo = True \text{ and } actionc = pumpOff \\ p_i & \text{if } lastpo = False \text{ and } actionc = pumpOn \\ p_i & \text{if } lastpo = False \text{ and } actionc = keep \end{cases} \tag{7}$$

$$p_{i+1} = \begin{cases} p_i-P & \text{if } lastpo = False \text{ and } actionc = pumpOff \\ p_i & \text{Otherwise} \end{cases} \tag{8}$$

$$lastpo = \begin{cases} True & \text{if } actionp = pumpOn \\ False & \text{if } actionp \in \{keep, pumpOff\} \end{cases} \tag{9}$$

If the last order is *pumpOn* and current order is *pumpOn*, p_{i+5} is p_i plus P. If the last order is *pumpOn* and current order is *keep*, p_{i+5} is p_i plus P. If the last order is *pumpOn*, and current order is *pumpOff*, p_{i+5} will not change. It is the same case that if the previous order is not *pumpOn* and current order is *pumpOn* or *keep*. p_{k+1} is p_k minus P, if the last order is not *pumpOn* and current order is *keep*. The reader may refer to the details of the implicit model in our Github repository.[1]

D. SENSOR COMPONENTS

Sensors observe the state of the steam-boiler and the values of the sensors are transmitted to the control system through networks or cables. Sensor components are specified by FSP as below:

[1]https://github.com/yylonly/LTSA

```
WATERSENSOR = ( getWaterQuantity[q:Q] -> waterchan.send[q] ->
        tick -> tick -> tick -> tick -> tick -> WATERSENSOR).
STEAMSENSOR = ( getSteamRate[v:V] -> steamchan.send[v] ->
        tick -> tick -> tick -> tick -> tick -> STEAMSENSOR).
PUMPSENSOR = ( getPumpRate[p:5*P] -> pumpsensorchan.send[p] ->
        tick -> tick -> tick -> tick -> tick -> PUMPSENSOR).
```

E. PUMP COMPONENT

The pump controller component is defined as:

```
PUMPCONTROLLER = (pumpcontrollerchan.receive[o:PUMPORDER] ->
   (when (o == ON)    pumpOn ->  PUMPTICK |
    when (o == OFF)   pumpOff -> PUMPTICK |
    when (o == KEEP) keep -> PUMPTICK)),
PUMPTICK = (tick -> tick -> tick -> tick -> tick ->
                                         PUMPCONTROLLER).
```

Pump controller would do the action corresponding to the order $o \in \{ON, KEEP, OFF\}$ received from pump controller channel. Furthermore, the state machine of pump component is shown in Figure 3.

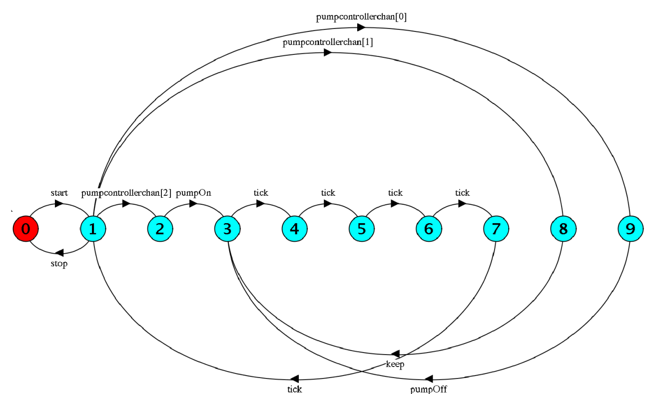


FIGURE 3. Pump component in LTSA.

F. CONTROLLER COMPONENT

The controller is the key component in the steam boiler system, and an appropriate strategy should make to keep the quality of water q between a specific range. After receiving the sensor measures q, v and p, the controller component will do the action *makedecision* to generate an order o, then do action *pumpcontrollerchan.send[o]* to send order o to pump controller channel, the function o is defined as:

$$o = \begin{cases} \text{ON} & \text{if } q < B_1 + \text{FTRD and } p \leq 3P \\ \text{ON} & \text{if } q < B_1 + \text{FTRD and } p = 4P \\ & \quad \text{and } lastpo = \text{False} \\ \text{KEEP} & \text{if } q < B_1 + \text{FTRD and } p = 5P \\ \text{KEEP} & \text{if } q < B_1 + \text{FTRD and } p \leq 4P \\ & \quad \text{and } lastpo = \text{True} \\ \text{KEEP} & \text{if } B_1 + \text{FTRD} \leq q \leq B_2 - \text{FTRU} \\ \text{KEEP} & \text{if } q > B_2 + \text{FTRU and } v \geq 0 \text{ and } p = 0 \\ \text{OFF} & \text{if } q > B_2 + \text{FTRU and } (p - v) \geq 0 \text{ and } p > 0 \end{cases} \tag{10}$$

Note that we introduce two integers FTRD and FTRU as thresholds ($0 \leq \text{FTRD} \leq B_2 - B_1$, $0 \leq \text{FTRU} \leq B_2 - B_1$)

with the integers B_1 and B_2 of Figure 2 to deal with the pump delay in the steam boiler. If the water level is above B_2 plus FTRU, and the water level is not decreasing and the pump rate is greater than zero, the decision OFF is made. If the water level is under B_1 + FTRD, and either throughout of pumps is less than $3P$ or throughout of pumps is $4P$ without the previous order *pumpON*, the decision is ON. The decision is KEEP, if water level is between that two boundaries, or water level is above B_2 plus FTRU besides steam rate is great than zero, throughput of pumps is zero, or water level is under B_1 plus FTRD besides throughput of pumps is maximal or $4P$ and previous order is *pumpOn*. The corresponding FSP design of the controller component is:

```
1  CONTROLSYSTEM = (init -> SYSCONTROLRUN[OFF]),
2  SYSCONTROLRUN[po:PUMPORDER] = (waterchan.receive[q:Q] ->
3       steamchan.receive[v:V] ->  pumpsensorchan.receive[p:5*P] ->
4       makedecision ->
5  --------------------- q < BEST1+FTRD ------------------------
6     ( when (q < BEST1+FTRD && p <= 3*P))
7          pumpcontrollerchan.send[ON] -> CONTROLTICK[True][q]
8     | when (q < BEST1+FTRD && p == 4*P && lastpo == False)
9          pumpcontrollerchan.send[ON] -> CONTROLTICK[True][q]
10    | when (q < BEST1+FTRD && p <= 4*P && lastpo == True)
11         pumpcontrollerchan.send[KEEP] -> CONTROLTICK[False][q]
12    | when (q < BEST1+FTRD && p == 5*P)
13         pumpcontrollerchan.send[KEEP] -> CONTROLTICK[False][q]
14 -------------------- q > BEST2-FTRU -------------------------
15    | when (q > BEST2-FTRU && (p-v) >= 0 && p > 0)
16         pumpcontrollerchan.send[OFF] -> CONTROLTICK[False][q]
17    | when (q > BEST2-FTRU && v >= 0 && p == 0)
18         pumpcontrollerchan.send[KEEP] -> CONTROLTICK[False][q]
19 ----------------BEST1+FTRD <= q && q <= BEST2-FTRU--------------
20    | when (BEST1+FTRD <= q && q <= BEST2-FTRU)
21         pumpcontrollerchan.send[KEEP] -> CONTROLTICK[False][q])),
22
23 CONTROLTICK[o:PUMPORDER][q:Q] = (tick -> tick -> tick -> tick
24                        -> tick -> SYSCONTROLRUN[o][q]).
```

V. MODEL VERIFICATION

This section verifies the designed FSP model against the safety properties and the progress properties in LTSA. Safety property checking guarantees that there is no deadlock in the system and the water quantity level keeps in the specified ranges. Progress property checking guarantees that there is no local loop in the system state.

A. REQUIREMENT SPECIFICATIONS IN FSP

1) PROGRESS PROPERTY (LIVENESS)

The designed steam boiler system must be livelock-free and deadlock-free. The system is livelock-free, only if all the processes of system are progressing without any divergence. For steam boiler system, it is livelock-free, only if all the components are progressing without divergences. LTSA use keyword *progress* to specify the progress property for each component. The components includes the steam boiler, steam sensor, pump sensor, water sensor, pump, and controller. Therefore, we specify six progress properties to verify livelock-free properties in the steam boiler system, each progress property contains the actions from the corresponding components, which are shown as follows:

```
progress WaterSensor = {getWaterQuantity[q:Q]}
progress SteamSensor = {getSteamRate[v:V]}
progress PumpSensor = {getPumpRate[v:V]}
progress PumpController = {pumpOn, pumpOff, keep}
progress Controler = {makedecision, rescue, repaired}
progress STEAMBOILER = {boiling[q:Q][v:V][p:P]}
```

Note that LTSA does not require specifying deadlock property explicitly, deadlock can be checked through the *check* menu of LTSA in Figure 4.

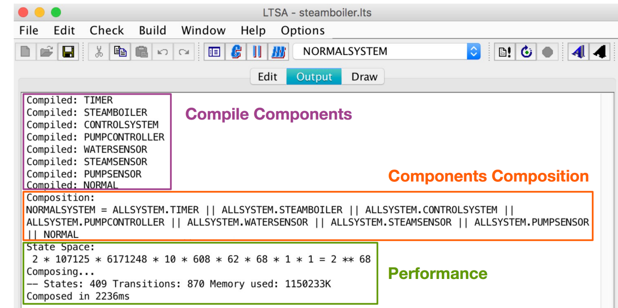


FIGURE 4. Safety property verification.

2) SAFETY PROPERTY

For safety property checking, we define two properties BASIC and NORMAL according to Requirement $REQ_{NormalMode}$ and $REQ_{RescueMode}$ as follows:

```
property BASIC = (getWaterQuantity[q:M1..M2] -> BASIC)
        + {getWaterQuantity[0..M1-1], getWaterQuantity[M2+1..C]}.
property NORMAL = (getWaterQuantity[q:N1..N2] -> NORMAL)
        + {getWaterQuantity[0..N1-1], getWaterQuantity[N2+1..C]}.
```

Note that LTSA does not provide a mechanism for describing invariant directly, it uses action set with parameters to describe those requirements. The Basic property is described as getWaterQuantity[q:M1..M2], which requires water quantity q between M_1 and M_2. The Normal property as getWaterQuantity[q:N1..N2] requires water quantity q between N_1 and N_2.

B. LTSA VERIFICATION

In this section, we use LTSA to verify the properties of the steam boiler model. The experiment settings are as follows: LTSA version 3.0, MacOS 10.14, 3.5 GHz Intel Core i5, 16GB 1600 MHz DDR3, and 512 SSD.[2]

1) LIVELOCK-FREE AND LOCKLOCK-FREE VERIFICATION

Livelock and deadlock can be directly verified in LTSA. The checking result of the progress (liveness) property is:

```
Progress Check...
-- States: 409 Transitions: 870 Memory used: 1633516K
No progress violations detected.
Progress Check in: 6ms
```

The checking result of deadlock is:

[2]http://www.doc.ic.ac.uk/ltsa/

```
Analysing...
-- States: 409 Transitions: 870 Memory used: 1544363K
No deadlocks/errors
Analysed using Supertrace in: 23ms
```

This result shows that the steam boiler model has no deadlock and livelock issues.

2) SAFETY PROPERTY VERIFICATION

We defined two safety properties in the requirement specifications of steam boiler system. In the normal mode, the steam boiler must not violate normal property. In the rescue mode, the steam boiler must not violate basic property within the constraint time, but not the normal property. LTSA use composition to verify the safety properties. If the property violates the designed model, the composition will be failure. To verify safety properties, we define the following compositions:

```
||BASICSYSTEM = (SYSTEMDESIGN || BASIC).
||NORMALSYSTEM = (SYSTEMDESIGN || NORMAL).
```

BASICSYSTEM represent composite the designed system with BASIC property and NORMALSYSTEM represent composite the designed system with NORMAL property. Both of compositions are passed without any failure. The composition result of NORMALSYSTEM is shown in Figure 4. All the components of steam boiler are successfully compiled, and they are composited with NORMAL property without any violation. The composition contains 409 states and 870 transitions. The composition spends 2236 ms and uses 1,150,233K memory. Furthermore, the source files for the model are made available at Github for interested readers.[3]

VI. CONCLUSIONS AND FUTURE WORK

In this paper, a nontrivial case study is presented to demonstrate how to use LTSA model and verify the real-time system. In requirements modeling, we show how to archive the structure diagram and generate the start-up design model. Furthermore, we design a variation law for the steam rate. We illustrate how to model the explicit and implicit timer in the components of the steam boiler system. For the most important effect of our paper, we show the potential power of integrating UML with the model checking tools for requirement modeling of both software and hardware components, system design and verification.

In the future, we consider to integrate LTSA with our requirements modeling and validation tool - RM2PT [26] to support formal verifying on the requirements model. Furthermore, we consider to generate prototype directly from the verified FSP model. Hopefully, this paper should be useful for in industry and academic worlds.

REFERENCES

[1] K. G. Shin and P. Ramanathan, "Real-time computing: A new discipline of computer science and engineering," *Proc. IEEE*, vol. 82, no. 1, pp. 6–24, Jan. 1994.
[2] T. Oda, K. Araki, and P. G. Larsen, "A formal modeling tool for exploratory modeling in software development," *IEICE Trans. Inf. Syst.*, vol. 100, no. 6, pp. 1210–1217, 2017.
[3] S. Saeeiab and M. Saeki, "Method integration with formal description techniques," *IEICE Trans. Inf. Syst.*, vol. 83, no. 4, pp. 616–626, 2000.
[4] E. Ahmad, Y. Dong, B. Larson, J. D. Lü, T. Tang, and N. Zhan, "Behavior modeling and verification of movement authority scenario of Chinese train control system using AADL," *Sci. China Inf. Sci.*, vol. 58, no. 11, pp. 1–20, Nov. 2015.
[5] A. Platzer and J.-D. Quesel, "European train control system: A case study in formal verification," in *Formal Methods and Software Engineering*, K. Breitman and A. Cavalcanti, Eds. Berlin, Germany: Springer, 2009, pp. 246–265.
[6] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam, "From verification to implementation: A model translation tool and a pacemaker case study," in *Proc. IEEE 18th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2012, pp. 173–184.
[7] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti, "Real-time urban monitoring using cell phones: A case study in Rome," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 1, pp. 141–151, Mar. 2011.
[8] Z. Hou, D. Sanán, A. Tiu, Y. Liu, and K. C. Hoa, "An executable formalisation of the SPARCv8 instruction set architecture: A case study for the LEON3 processor," in *Proc. 21st Int. Symp. Formal Methods (FM)*, in Lecture Notes in Computer Science, Limassol, Cyprus, vol. 9995, J. S. Fitzgerald, C. L. Heitmeyer, S. Gnesi, and A. Philippou, Eds. Cham, Switzerland: Springer, 2016, pp. 388–405.
[9] T. P. Khoo and J. Sun, "The miles before formal methods—A case study on modeling and analyzing a passenger lift system," in *Proc. Int. Conf. Formal Methods Softw. Eng. (ICFEM)*, J. Sun and M. Sun, Eds. Cham, Switzerland: Springer, 2018, pp. 54–69.
[10] C. Baier and J. P. Katoen, *Principles of Model Checking* (Representation and Mind Series). Cambridge, MA, USA: MIT Press, 2008.
[11] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., *Handbook of Model Checking*. Cham, Switzerland: Springer, 2018.
[12] J. Magee and J. Kramer, *State Models and Java Programs*. Hoboken, NJ, USA: Wiley, 1999.
[13] J. Magee, N. Dulay, and J. Kramer, "Regis: A constructive development environment for distributed programs," *Distrib. Syst. Eng.*, vol. 1, no. 5, p. 304, 1994.
[14] J. Magee, J. Kramer, and D. Giannakopoulou, "Analysing the behaviour of distributed software architectures: A case study," in *Proc. 6th IEEE Comput. Soc. Workshop Future Trends Distrib. Comput. Syst.*, Oct. 1997, pp. 240–245.
[15] J. R. Abrial, "Steam-boiler control specification problem," in *Formal Methods for Industrial Applications*. Springer, 1996, pp. 500–509.
[16] J. R. Abrial, E. Börger, and H. Langmaack, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Springer, 1996.
[17] J. Woodcock and A. Cavalcanti, "The steam boiler in a unified theory of Z and CSP," in *Proc. APSEC*, Dec. 2001, pp. 291–298.
[18] G. Leeb and N. Lynch, "Proving safety properties of the steam boiler controller," in *Formal Methods for Industrial Applications*. Springer, 1996, pp. 318–338.
[19] X. Li and J. Wang, "Specifying optimal design for a steam-boiler system," in *Formal Methods for Industrial Applications*. Springer, 1996, pp. 359–378.
[20] S. Löffler and A. Serhrouchni, "Creating a validated implementation of the steam boiler control," in *Proc. 3rd SPIN Workshop (SPIN)*, Apr. 1997, pp. 1–13.
[21] M. Kerboeuf, D. Nowak, and J. P. Talpin, "Specification and verification of a steam-boiler with signal-coq," in *Theorem Proving in Higher Order Logics*. Springer, 2000, pp. 356–371.
[22] M. Butler, E. Sekerinski, and K. Sere, "An action system approach to the steam boiler problem," in *Formal Methods for Industrial Applications*. Springer, 1996, pp. 129–148.
[23] M.-C. Gaudel, P. Dauchy, and C. Khoury, "A formal specification of the Steam-Boiler Control problem by algebraic specifications with implicit state," in *Formal Methods for Industrial Applications*. Springer, 1996, pp. 233–264.
[24] M. Broy, F. Regensburger, B. Schätz, and K. Spies, "Streams of steam—The steam boiler specification case study," Dept. Fac. Inform., Tech. Univ. Munich, Munich, Germany, Tech. Rep. 9202, 1998.

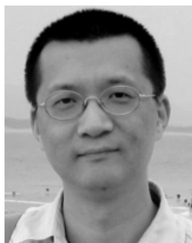
[3] https://github.com/yylonly/LTSA

[25] P. J. Carreira and M. E. Costa, "Automatically verifying an object-oriented specification of the steam-boiler system," in *Proc. 5th Int. ERCIM Workshop Formal Methods Ind. Crit. Syst. (FMICS)*, 2000, pp. 345–360.

[26] Y. Yang, X. Li, Z. Liu, and W. Ke, "RM2PT: A tool for automated prototype generation from requirements model," presented at the 41th Int. Conf. Softw. Eng. (ICSE), Montreal, QC, Canada, May 2019. Online. Available: https://2019.icse-conferences.org/event/icse-2019-demonstrations-rm2pt-a-tool-for-automated-prototype-generation-from-requirements-model

YILONG YANG received the B.S. degree in computer science from the China University of Mining and Technology, China, in 2010, and the M.S. degree from Guizhou University, China, in 2013. He is currently pursuing the Ph.D. degree in software engineering with the University of Macau. He has been a Fellow with United Nations University - International Institute for Software Technology, Macau. His research interests include automated software engineering and machine learning.

QUAN ZU received the Ph.D. degree from the College of Electronics and Information Engineering, Tongji University, China, in 2016. He is currently a Postdoctoral Research Fellow with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau. His research interests include formal methods, model checking, and algorithm design and analysis.



WEI KE received the Ph.D. degree in computer applied technology from Beihang University, in 2012. He is currently an Associate Professor with the School of Public Administration, Macau Polytechnic Institute. His research interests include programming languages, formal methods, software engineering tool support, and software implementation. He had successfully applied in a couple of research projects funded by Macau FDCT, including the areas of formal methods and software engineering.

MIAOMIAO ZHANG received the Ph.D. degree in automation from Shanghai Jiaotong University, Shanghai, China, in 2001. She held a Postdoctoral position with the Faculty of Computer Science, Radbound University, Nijmegen, The Netherland, from 2001 to 2004. Since then she joined the School of Software Engineering, Tongji University, as an Associate Professor, and was a Full Professor, in 2008. Her current research interests include model checking, real-time embedded systems, and model learning.

XIAOSHAN LI received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing, in 1994. He is currently an Associate Professor with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau. His research interests include formal methods, object-oriented software engineering with UML, real-time specification and verification, and the semantics of programming language.