Building Hybrid Real-Time Systems in XYZ/E——Implementation of the Steam-Boiler Control Specification Problem*

YAN An[1] TANG Zhi-song[2]

[1](Department of Computer Science Brown University Providence RI 02912)
[2](Laboratory of Computer Science Institute of Software Chinese Academy of Sciences Beijing 100080)
E-mail: ya@cs.brown.edu

Abstract XYZ/E is a temporal logic system as well as a programming language. The temporal logic language XYZ/E can represent both high level and low level specifications in the same framework, so that the specification and implementation of software systems are very convenient. A specification and an implementation of the Steam Boiler Control Specification Problem in XYZ/E is presented in this paper. A visual user interface is also implemented.

Key words Temporal logic language, real-time hybrid system, specification, XYZ/E.

XYZ is a software engineering system consisting of a temporal logic language XYZ/E and a set of CASE tools. The temporal logic language XYZ/E is based on Manna-Pnuli's Linear Time Temporal Logic. It can represent both high level and low level specifications in the same frame work, so that the specification and implementation of software systems are very convenient. XYZ/E is simple yet expressive enough to be accepted by engineers. Besides, the formal nature of this language makes it capable of programming verifications. The basic command of XYZ/E (called conditional element or *ce*) is

$$LB=y \wedge R \Diamond @(Q \wedge LB=z) \qquad (1)$$

Here y and z are called definitional label and forward label of the *ce* respectively. "@" represents $\$O$ (next time) or $\Diamond$ (eventually). R and Q are logic formula called condition part and action part respectively. More details about XYZ/E please refer to Ref. [1].

Steam Boiler Control Specification Problem is raised in Ref. [2], which encourages a formal specification and an implementation program to control the water level in a boiler. Safety property is the most important because the quantity of water has to be kept in a given range. Being too low or too high might lead to unexpected destructions.

A steam-boiler system consists of a boiler, four pumps, four pump controllers, a valve and two sensors. One sensor measures the quantity of water and the other measures the speed of the steam coming out of the boiler. There are also a message transmission system and an operator disk. Our goal is to specify the behaviors of the steam-boiler and verify some properties of the system in XYZ/E.

---

* This research is supported by the National Natural Science Foundation of China(国家自然科学基金,No. 69673019).
YAN An was born in 1975. He received a B. S. degree in Computer Science from University of Science & Technology of China, and a M. S. degree in computer science from Institute of Software at the Chinese Academy of Sciences and Brown University. His research interests are programming language, real-time systems, software engineering and object-oriented database. **TANG Zhi-song** was born in 1925. He is a professor, a doctoral supervisor and a member of the Chinese Academy of Sciences. His current research areas include temporal logic language, software engineering, CASE tools and software architecture.

Manuscript received 1998-09-04, accepted 1999-01-27.

The steam-boiler system is safety-critical. The program must keep the water level in a given range. The boiler needs to be stopped once the water goes beyond emergency lines. The system should also be fault-tolerant. It is expected to maintain working even if some malfunctions occur in the physical units. Please refer to Ref. [2] for a complete description of the steam-boiler problem.

In this paper, we present a specification and an implementation of the Steam Boiler Control Specification Problem in XYZ/E. A visual user interface is also implemented for this purpose. There are also some closely related papers. Reference [3] discusses the stepwise specification and verification of the steam-boiler system. Reference [4] discusses the mathematic model, formal specification and verification of general hybrid systems in the XYZ framework.

1 Architecture of the XYZ/Steam-Boiler System

To describe the steam boiler system, we present a process for each physical unit to simulate its behaviors, for example, the level process for LEVEL, the pump process for PUMP1-PUMP4. Another process is called "control center", which receives messages from the physical units, analyzes the received information, and transmits messages to the physical units. Besides, a process called "clock" generates a CLOCK message every 5 seconds to synchronize all the actions. The process "msg-center" serves to distribute messages within the processes. To simulate the events occurring on the physical units, for instance, a failure on LEVEL, we give another process called "event" to communicate with the user interface. When you click on the visual item which represents one of the physical units, such as the LEVEL, in the user interface, the event process will receive a message from the interface. Then it will set the state of the LEVEL in the level process to be BROKEN. In the next time cycle, the control center process will detect the failure by the odd behavior of the level process. All the processes are running concurrently, and their relationship is shown in Fig. 1.

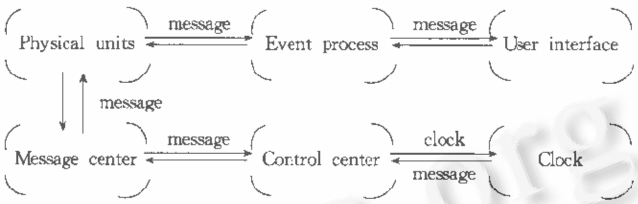


Fig. 1 Architecture of the XYZ/Steam-Boiler system

In specification form:

$[LB=STARTmain \Rightarrow \Diamond[ProgramRunning \vee ProgramStopped]$

$WHERE[$

$(ProgramRunning \leftrightarrow$

$(ControlCenterRunning \wedge ClockRunning \wedge EventRunning \wedge$

$SteamRunning \wedge LevelRunning \wedge Pump1Running \wedge Pump2Running \wedge$

$Pump3Running \wedge Pump4Running \wedge PumpCtrl1Running \wedge$

$PumpCtrl2Running \wedge PumpCtrl3Running \wedge PumpCtrl4Running \wedge ValveRunning))$

$(ProgramStopped \leftrightarrow$

$(LB=STOPmain \wedge ControlCenterStop \wedge ClockStop \wedge$

$EventStop \wedge SteamStop \wedge LevelStop \wedge Pump1Stop \wedge$

$Pump2Stop \wedge Pump3Stop \wedge Pump4Stop \wedge PumpCtrl1Stop \wedge$

$PumpCtrl2Stop \wedge PumpCtrl3Stop \wedge PumpCtrl4Stop \wedge ValveStop))$

$(ControlCenterRunning \leftrightarrow ControlCenter(state) \in RunningStateSet) \wedge$

$(ControlCenterRunning \leftrightarrow ControlCenter(state) \in RunningStateSet) \wedge$

$(SteamRunning \leftrightarrow Steam(state) \in RunningStateSet) \wedge$

$(LevelRunning \leftrightarrow LevelCenter(state) \in RunningStateSet) \wedge$

$(Pump1Running \leftrightarrow Pump1(state) \in RunningStateSet) \wedge$

$(Pump2Running \leftrightarrow Pump2(state) \in RunningStateSet) \wedge$

$(Pump3Running \leftrightarrow Pump3(state) \in RunningStateSet) \wedge$

$(Pump4Running \leftrightarrow Pump4(state) \in RunningStateSet) \wedge$

$(PumpControl1Running \leftrightarrow PumpControl1(state) \in RunningStateSet) \wedge$

$(PumpControl2Running \leftrightarrow PumpControl2(state) \in RunningStateSet) \wedge$

$(PumpControl3Running \leftrightarrow PumpControl3(state) \in RunningStateSet) \wedge$

$(PumpControl4Running \leftrightarrow PumpControl4(state) \in RunningStateSet) \wedge$

$(ValveRunning \leftrightarrow Valve(state) \in RunningStateSet) \wedge$

$(ClockRunning \leftrightarrow Clock(state) = WORKING) \wedge$

$(EventRunning \leftrightarrow Event(state) = WORKING) \wedge$

$(ControlCenterStop \leftrightarrow ControlCenter(state) = EMERGENCY\_STOP \wedge LBcc = STOP) \wedge$

$(SteamStop \leftrightarrow Steam(state) = EMERGENCY\_STOP \wedge LBsteam = STOP) \wedge$

$(LevelStop \leftrightarrow LevelCenter(state) = EMERGENCY\_STOP \wedge LBlevel = STOP) \wedge$

$(Pump1Stop \leftrightarrow Pump1(state) = EMERGENCY\_STOP \wedge LBpump1 = STOP) \wedge$

$(Pump2Stop \leftrightarrow Pump2(state) = EMERGENCY\_STOP \wedge LBpump2 = STOP) \wedge$

$(Pump3Stop \leftrightarrow Pump3(state) = EMERGENCY\_STOP \wedge LBpump3 = STOP) \wedge$

$(Pump4Stop \leftrightarrow Pump4(state) = EMERGENCY\_STOP \wedge LBpump4 = STOP) \wedge$

$(PumpControl1Stop \leftrightarrow PumpControl1(state) = EMERGENCY\_STOP \wedge LBpc1 = STOP) \wedge$

$(PumpControl2Stop \leftrightarrow PumpControl2(state) = EMERGENCY\_STOP \wedge LBpc2 = STOP) \wedge$

$(PumpControl3Stop \leftrightarrow PumpControl3(state) = EMERGENCY\_STOP \wedge LBpc3 = STOP) \wedge$

$(PumpControl4Stop \leftrightarrow PumpControl4(state) = EMERGENCY\_STOP \wedge LBpc4 = STOP) \wedge$

$(ValveStop \leftrightarrow Valve(state) = EMERGENCY\_STOP \wedge LBvalve = STOP) \wedge$

$(ClockStop \leftrightarrow Clock(state) = EMERGENCY\_STOP \wedge LBclock = STOP) \wedge$

$(EventStop \leftrightarrow Event(state) = EMERGENCY\_STOP \wedge LBevent = STOP) \wedge$

$RunningStateSet = \{INITIALIZATION, NORMAL, DEGRADED, RESCUE\}$

]

Here we assume that:

$LB = main\_START \wedge Pre\text{-}condition1 \wedge Pre\text{-}condition2 \wedge \ldots \wedge Pre\text{-}conditionk$

$\quad \Rightarrow \Diamond[Post\text{-}condition1; Post\text{-}condition2; \ldots; Post\text{-}conditionk]$

$WHERE((LB = pros1\_START \wedge Pre\text{-}condition1 \Rightarrow \Diamond Post\text{-}condition1)$

$\quad (LB = pros2\_START \wedge Pre\text{-}condition2 \Rightarrow \Diamond Post\text{-}condition2) \wedge$

$\quad (LB - prosk\_START \wedge Pre\text{-}conditionk \Rightarrow \Diamond Post\text{-}conditionk))$

Pros1, pros2, ..., prosk are processes which run concurrently.

This property is called decomposability. It will greatly reduce the complexity of specification and verification. Further discussion of the decomposability of a parallel program can be found in Ref. [1].

In executable code:

$[LB = START \Rightarrow \$OLB = InitProcessName;$

$LB = InitProcessName \Rightarrow$

$\quad \$Opump1 = pumppros \wedge \$Opump2 = pumppros \wedge$

```
    $Opump3=pumppros ∧ $Opump4=pumppros ∧
    $Opctrl1=pctrlpros ∧ $Opctrl2=pctrlpros ∧
    $Opctrl3=pctrlpros ∧ $Opctrl4=pctrlpros ∧
    $Osteam1=steampros ∧ $Olevel1=levelpros ∧
    $Ovalve1=valvepros ∧ $Ocontrol_center1=control_center ∧
    $Oclock1=clockpros ∧ $Oevent1=eventpros ∧
    $Omsg_center1=msg_center ∧ $OLB=run;
  LB =run ◊ ||[
     pump1 {1} (%INP/1|j; %CHN/ch_msgb_pump11(msg_centerb1, * )|ch_receive (con1, * );
       %CHN/ch_pump1_msgb1( * ,msg_centerb1) ch_send( * ,con2));
     pump2 {1} (%INP/2|j; %CHN/ch_msgb_pump21(msg_centerb1, * )|ch_receive(con1, * );
       %CHN/ch_pump2_msgb1( * ,msg_centerb1)|ch_send( * ,con2));
     ...
     pctrl1{1} (%INP/1|j; %CHN/ch_msgc_pctrl11(msg_centerc1, * )|ch_receive(con1, * );
       %CHN/ch_pctrl1_msgc1( * ,msg_centerc1)|ch_send( * ,con2));
     ...
     steam1 {1}(%CHN/ch_msgb_steam1(msg_centerb1, * )|ch_receive(con1, * );
       %CHN/ch_steam1_msgb( * ,msg_centerb1)|ch_send( * ,con2));
     control_center1{1}(%CHN/ch_msg_control_center1(msg_center1, * )|ch_receive(con1, * );
       %CHN/ch_control_center1_msg( * ,msg_center1)|ch_send( * ,con2));
     msg_center1{1} (%CHN/ch_valve1_msg(valve1, * )|ch_valve_msg(con11, * );
       %CHN/ch_clock1_msg(clock1, * )|ch_clock_msg(con12, * ))];
     LB=l2 ◊ $OLB=STOP ].
```

2 The Overall Operation of the Control Center

2.1

"The program follows a cycle and a priori does not terminate. This cycle takes place each five seconds and consists of the following actions: reception of messages coming from the physical units, analysis of the information received, transmission of messages to the physical units."

```
  [LB =START ◊ $OLB=mainloop;
     LB=mainloop ◊ $Och_receive?msg $OLB=checkmsg;
     LB=checkmsg ∧ (msg=CLOCK) ◊ $OLB=deal_clockmsg;
     LB=checkmsg ∧ (msg=LEVEL) ◊ $OLB=deal_levelvmsg;
     ...
     LB=deal_clockmsg ◊ $OLB=mainloop;
     LB=deal_levelvmsg ∧ (msg.value>M2 $ Vmsg.value<M1)
        ◊ $OLB=emergency_stop;
     LB=deal_levelvmsg ∧ (msg.value>M1 ∧ msg.value<N1)
        ◊ $Osendmsg[1]=OPENPUMP ∧ $Och_send!sendmsg ∧ OLB=mainloop;
     ...]
```

Note that we omit the specification of these behaviors because it is straightforwardly represented in the executable code. The informal description in the section title is quoted from Ref. [1].

2.2

"The control center operates in different modes, namely: initialization normal, degraded, rescue, emergency stop."

```
[LB = START ◇ $Omode=INITIALIZATION ∧ $OLB=linit;
    LB=linit ◇ ...
    ...
    ... ◇ $Omode=NORMAL ∧ $OLB=lnormal;
    ... ◇ $Omode=DEGRADED ∧ $OLB=ldegraded;
    ... ◇ $Omode=EMERGENCY_STOP ∧ $OLB=lemergeney_stop;
    ...
    LB=lnormal ◇ ...
    LB=ldegraded ◇ ...
    LB=lrescue ◇ ...
    LB=lemergency_stop ◇ ...
    ...
]
```

2.3

"The initialization mode is the mode to start with. The program enters a state in which it waits for the message STEAM_BOILER_WAITING to come from the physical units. As soon as this message is received the program checks whether the quantity of steam coming out of the steam boiler is really zero. If the unit for detection of the level of steam is defective, that is, when v is not equal to zero, the program enters the emergency stop mode."

```
[LB=linit ◇ $Och_rec?msg ∧ $OLB=linit_checkmsg;
        LB=linit_checkmsg ∧ (msg[0]=STEAM ∧ msg[1]=STEAM_BOILER_WAITING)
          ◇ $OLB=linit_steamwaiting;
        LB=linit_check ∧ msg(msg[1]=CLOCK) ◇ $OLB=check_timeout;
        LB=linit_checkmsg ∧ ... ◇ ...
        ...
        LB=linit_steamwaiting ∧ (msg[2]=0) ◇
          $OSteamWaiting= $T ∧ $OSteamOk= $T ∧ $OLB=linit;
        LB=linit_steamwaiting ∧ (msg[2]/=0) ◇
          $Omode=EMERGENCY_STOP ∧ $OLB=lemergeney_stop;
        ...
]
```

2.4

"If the quantity of water in the steam-boiler is above N2 then the program activates the valve of steam-boiler in order to empty it. If the quantity of water in the steam-boiler is below N1 then the program activates a pump to fill the steam-boiler. If the program realizes a failure of the water level detection unit it enters the emergency stop mode. As soon as a level of water between N1 and N2 is reached the program can send continuously the signal PROGRAM_READY to the physical units until it receives the signal PHYSICAL_UNITS_READY which must be emitted by the physical units. As soon as this signal is received, the program enters either the mode normal if all the physical units operate correctly or the mode degraded if any physical unit is defective. A transmission failure puts the program into the mode emergency stop."

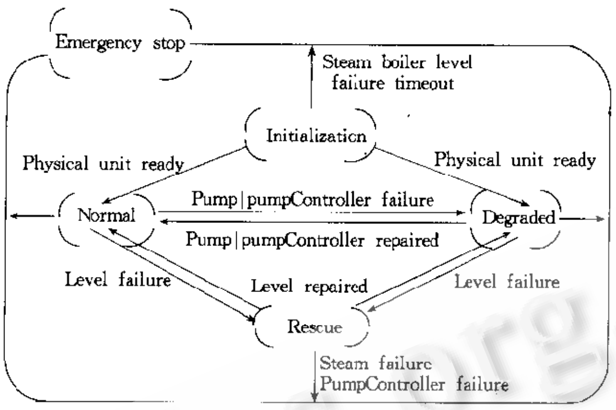


Fig. 2 Working modes of the control center

```
$[LB = linit \Rightarrow \$Och\_rec?msg \wedge \$OLB = linit\_check\_msg;$
    $LB = linit\_check\_msg \wedge (msg[1] = LEVEL) \Rightarrow \$OLB = linit\_check\_levelv;$
    $LB = linit\_check\_msg \wedge (msg[1] = CLOCK) \Rightarrow \$OLB = linit\_change\_mode;$
    $LB = linit\_check\_msg \wedge (msg[0] = STEAM \wedge msg[1] = PHYSICAL\_UNIT\_READY)$
      $\Rightarrow \$OSteamReady = \$T \wedge \$OLB = linit;$
    ...
    $LB = linit\_check\_msg \wedge (msg[0] = VALVE \wedge msg[1] = PHYSICAL\_UNIT\_READY)$
      $\Rightarrow \$OValveReady = \$T \wedge \$OLB = linit;$
    $LB = linit\_check\_levelv \wedge (msg.value > N2 \wedge msg.value < M2) \Rightarrow$
      $\$Osendmsg[0] = VALVE \wedge \$Osendmsg[1] = OPENVALVE \wedge$
        $\$Och\_send!sendmsg \wedge \$OLB = linit\_init;$
    $LB = linit\_check\_levelv \wedge (msg.value < N1 \wedge msg.value > M1) \Rightarrow$
      $\$Osendmsg[0] = PUMP1 \wedge \$Osendmsg[1] = OPENPUMP \wedge$
      $\$Och\_send!sendmsg \wedge$
      $\$Osendmsg[0] = PUMP2 \wedge \$Och\_send!sendmsg \wedge$
      $\$Osendmsg[0] = PUMP3 \wedge \$Och\_send!sendmsg \wedge$
      $\$Osendmsg[0] = PUMP4 \wedge \$Och\_send!sendmsg \wedge \$OLB = linit\_init;$
    $LB = linit\_check\_levelv \wedge (msg.value \geqslant N2 \wedge msg.value \leqslant N1) \Rightarrow$
      $\$Osendmsg[0] = STEAM \wedge \$Osendmsg[1] = PROGRAM\_READY \wedge \$Och\_send!sendmsg \wedge$
      $\$Osendmsg[0] = LEVEL \wedge \$Och\_send!sendmsg \wedge$
      $\$Osendmsg[0] = PUMP1 \wedge \$Och\_send!sendmsg \wedge$
    ...
      $\$Osendmsg[0] = VALVE \wedge \$Och\_send!sendmsg \wedge \$OLB = linit\_init;$
    ...
    $LB = linit\_change\_mode \wedge (SteamOk \wedge LevelOk \wedge \ldots \wedge ValveOk) \Rightarrow$
      $\$Omode = NORMAL \wedge \$OLB = lnormal;$
    $LB = linit\_change\_mode \wedge \sim(SteamOk \wedge LevelOk \wedge \ldots) \Rightarrow$
      $\$Omode = DEGRADED \wedge \$OLB = ldegraded;$
    ...
]
```

3 The Overall Operation of the msg_Center Process

The message passing schema is shown in Fig. 3. X represents steam, level, ..., valve. We introduce the msg_center process to distribute the messages. There is no difficulty in extending this mode to a point to point one, because it is clear that

msg.to = steam, level, ..., valve

is identical to

msg to steam,
msg to level,
...
msg to valve

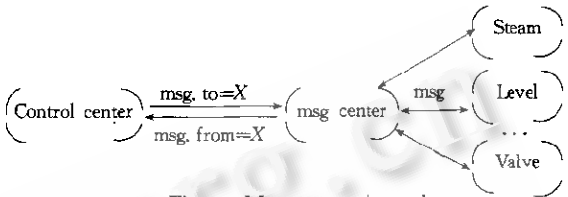


Fig. 3　Message passing schema

In the former mode, we can deal with all the messages in the same way, i.e. *ch_receive?msg*, *ch_send?msg*. If information about the source and destination is included in the message itself, that will greatly reduce the complexity of program coding. What is more, for communicating with a newly added physical unit, for example, pump5, we need not add ch_from_pump5 and ch_to_pump5 in the code of the control_center process. Thus the control center can operate in the same way as before.

4 The Operation of the Level Process

The procedure of specification and implementation of the level process is very much like that of the control center, except that it uses MODE message from the control center rather than the CLOCK message from the clock process for synchronization. The control center will receive a CLOCK message from the clock process for every 5 seconds, then it will send a MODE message to all the other processes. This message serves as the indication of the next time cycle in those processes. Following is the description of the time cycle of the level process:

```
[LB=STARTlevel ⇨ $LB=lreadmsg;
   LB=lreadmsg ⇨ $Och_receive?msg ∧ $OLB=check_msg;
   LB=check_msg ∧ (msg[1]=MODE) ⇨ $OLB=lsendlevelv;
   LB=check_msg ∧ ... ⇨ ...
   ...
   LB=lsendlevelv ⇨ CalculateWaterlevel(%IOP/reallevelv|q; %INP/time-time0|t);
   LB=lsendlevelv1 ⇨ $Osendmsg[0]=LEVEL ∧ $Osendmsg[1]=LEVELv ∧
      $Osendmsg[2]=reallevelv ∧ $Och_send!sendmsg ∧ $OLB=lreadmsg;
   ...
]
```

Here we use "*CalculateWaterlevel(%IOP/reallevelv|q; %INP/time-time0|t)*" to get a value for simulating the water level in a real steam boiler. Following is the specification of the CalculateWaterlevel procedure:

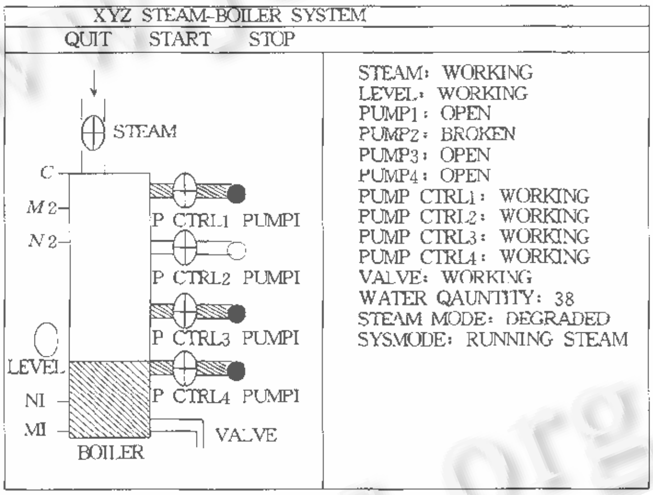
$$LB=START\_CalculateWaterlevel \wedge q \geqslant 0 \wedge q \leqslant C \wedge t > 0 \wedge u \geqslant -U1$$
$$\wedge u \leqslant U2 \wedge v \geqslant 0 \wedge v \leqslant W \wedge p \geqslant 0 \wedge p \leqslant 4P$$
$$⇨ \Diamond[q' = q + \int_0^t \dot{q}dt \wedge LB = RETURN]$$
$$WHERE[\dot{q} = p - \frac{\rho_w}{\rho_s} v \wedge v' = v + \int_0^t \dot{v}dt \wedge \dot{v} = u]$$

Here u is the increase of quantity of steam, v is the quantity of the steam exiting the steam-boiler, and p is the throughput of the pumps. ρ_w and ρ_s are densities of water and steam, respectively. We will discuss the formal specification and verification of hybrid systems in Ref. [3] and other related papers[4,5]. The safety and liveness properties of the steam-boiler system are also specified and verified in those papers. A new operator "$\$ O\epsilon$" will be introduced for that purpose.

5 The User's Interface

The visual user interface is coded in C language with Xlib. It runs in XWindow or OpenWindow. This interface works concurrently with the program coded in XYZ/E that we have described before. They will exchange data and control via the IPC message passing mechanism of UNIX.

The interface exhibits the current status of the steam boiler in a pictorial way. Water is drawn blue and will rise and fall as the data in the "steam" changes. Figure 4 is a running snapshot when PUMP2 is broken (drawn red), and all the other PUMPs have been opened automatically because the water level has fallen beyond the normal range.



STEAM: WORKING
LEVEL: WORKING
PUMP1: OPEN
PUMP2: BROKEN
PUMP3: OPEN
PUMP4: OPEN
PUMP CTRL1: WORKING
PUMP CTRL2: WORKING
PUMP CTRL3: WORKING
PUMP CTRL4: WORKING
VALVE: WORKING
WATER QAUNTITY: 38
STEAM MODE: DEGRADED
SYSMODE: RUNNING STEAM

Fig. 4 User interface

Also, the interface offers us the opportunity to simulate an exceptional event occurring on the physical units, for instance, PUMP1 is broken at an arbitrary time. The usage is described below.

Click at a physical unit drawn black: which indicates that there is a failure on the physical unit which we are simulating by computer. The corresponding process will change its state at instance. After a few seconds, the control center process will detect this failure and send FAILURE_DETECTION message to the corresponding process. The color of this element will change to red at the same time, showing that this physical unit is broken and the control center has acknowledged this situation.

·Click at a physical unit drawn red: which indicates that the broken physical unit is repaired. The corresponding process will change its state at instance and send PHYSICAL_UNIT_REPAIRED message to the control center. On receiving this message, the control center process will send PHYSICAL_UNIT_REPAIRED_ACKNOWLEDGEMENT message to the corresponding process. The color of this element will change to black at the same time, showing that this physical unit is repaired and the control center has acknowledged this situation.

6 Conclusions

This paper has discussed the specification and implementation of the Steam Boiler Control Specification Problem in the temporal logic language XYZ/E. It turns out that XYZ/E is suitable for hybrid, real-time and communication problems. In our future work, we plan to discuss the Steam Boiler's behavior in differential equations. We also plan to verify this program under our framework.

Acknowledgments

The authors would like to thank Dr. Zhao Chen and the other colleagues in the XYZ Group for their many helps during the implementation of this system.

References

1 Tang C S. An outline of the XYZ system. Logic of software engineering. In: Pnueli A, Lin H eds. Processdings of a Workshop 1995. Beijing, Singapore: World Scientific, 1996
2 Jean-Raymond Abrial. Steam-Boiler control specification. Method for Semantics and Specification. International Conference and Research Center, Schloss, Dagstuhl, Germany, 1994
3 Yan An, Tang C S. A unified linear-time temporal logic solution to the steam-boiler control specification problem. Science in China (Series E), 1999,42(2):244～251
4 Yan An, Tang C S. Hybrid systems in XYZ. Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Technical Report No. ISCAS-LCS-9704, 1997
5 Yan An. Hybrid systems in XYZ [MS Thesis]. Institute of Software, The Chinese Academy of Sciences, 1997
6 Tang C S. A temporal logic oriented toward software engineering—an introduction to XYZ system (I). Chinese Journal of Advanced Software Research, 1994,1(1):1～29
7 Zohar Manna, Amir Pnueli. Clocked transition systems. Logic & Software Engineering. In: Pnueli A, Lin H eds. Proceedings of a Workshop 1995. Beijing, Singapore: World Scientific, 1996
8 Zohar Manna, Amir Pnueli. Verifying hybrid systems——hybrid systems. In: Grossman R L, Nerode A, Ravn A eds. Lecture Notes in Computer Science. Springer-Verlag, 1993
9 Maler O, Manna Z, Pnueli A. From timed to hybrid systems. In: Proceedings of REX Workshop 1991. Springer-Verlag, 1992
10 Zhou Chao-chen, Wang Ji, Anders P Ravn. A formal description of hybrid systems. Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Technical Report, No. ISCAS-LCS-95-14, 1995

在 XYZ/E 中实现混成实时系统——蒸气锅炉控制问题的解决

闫安　唐稚松

(中国科学院软件所计算机科学开放研究实验室　北京　100080)

摘要　XYZ/E 是一个时序逻辑系统,同时也是一种时序逻辑程序设计语言.XYZ/E 能够在统一的框架下表示高层和低层的描述,所以便于软件系统的描述与实现.该文对基于 XYZ/E 的蒸气锅炉问题进行了描述与实现,并介绍了为该问题实现的图形用户界面.

关键词　时序逻辑语言,混成实时系统,描述,XYZ/E.

中图法分类号　TP311