Formal Development of a Washing Machine Controller Model Based on Formal Design Patterns

XIN BEN LI[1], FENG XIA ZHAO[2]
[1]Department of Computer Science and Technology, Zhejiang Wanli University
No.8 South Qianhu Road, Ningbo, Zhejiang, 315100, P.R. CHINA,
[2]School of Nursing, Ningbo college of Health Science
No.51, Xuefu Road, Ningbo, Zhejiang, 315100, P.R.CHINA,

Abstract: - Formal methods approach to software construction can significantly increase the reliability and correctness of the resulting software. Formal methods users are given sophisticated languages and tools for constructing software models, but they often lack some systematic methodological measures to help. Formal design patterns can help formal methods users speed up the development process by re-using and incorporating some pre-defined proved and refined models, as design patterns do in object-oriented software. Some formal design patterns are presented and applied to the development of a washing machine controller model in Event-B that is a formal method for modeling and reasoning about complex discrete system.

Key-Words: - Formal methods, Event-B, Design Patterns, Washing Machine Controller

1 Introduction

Reactive systems are a class of software and/or hardware systems whose role is to maintain an ongoing interaction with their environment rather than produce some final value upon termination. Such systems are often critical systems, as errors occurring during their execution may have dramatic economical or human consequences. Typical examples of reactive systems are embedded systems, computer operating systems, communication protocols, or process control systems such as a nuclear reactor [1].

Formal methods are the application of mathematics to model and verify software or hardware systems [2]. The main advantages of the formal methods approach to software or systems construction is that, whenever applicable, it can lead to an increase of the reliability and correctness of the resulting software or systems by several orders of magnitude. Event-B is a formal method for modeling and reasoning about systems based on set theory and predicate logic. It uses refinement to represent systems at different abstraction levels and uses mathematical proof to verify consistency between refinement levels. The Event-B method has been devised for modeling reactive and distributed systems [3]. However, formal methods have been criticized for their lack of accessibility especially for novice users.

Design patterns in object-oriented software are a way of communicating expertise by capturing the solutions to similar design problems and re-using those solutions [4]. Introducing design patterns within formal methods can help users of formal methods speed up the development process by re-using some pre-defined proved and refined mini-models. We in this paper present some Event-B design patterns that can be used for specifying those reactive systems. The design patterns are constructed deliberately and proved correctly. They can be re-used to specify the behaviour as part of a specification of any reactive system. A case is given to show the application of the design patterns in formal Event-B method. This was done within the framework of the Rodin Platform that is an open tool set and provides effective support for refinement and mathematical proof [3].

2 Event-B Method

Event-B is a formal method for system-level modelling and analysis. It is designed for long-running reactive hardware/software systems that respond to stimuli from user and/or environment. The set-theoretic language in first-order logic takes as semantic model a transition system with guarded transitions between states. The correctness of a model is defined by an invariant property, i.e. a predicate, or constraint, which every state in the system must satisfy. More practically, every event in the system must be shown to preserve this invariant; this verification requirement is expressed in a number

of proof obligations (POs). This verification is performed either by model checking or theorem proving (or both).

An Event-B model consists of contexts and machines. Contexts contain the static parts of a model. These are sets, constants and axioms that describe the properties of these sets and constants. Machines contain the dynamic parts of a model. A machine is made of a state, which is defined by means of variables. Variables, like constants, correspond to simple mathematical objects: sets, binary relations, functions, numbers, etc. They are constrained by invariants I(v) where v are the variables of the machine. Invariants are supposed to hold whenever variable values change, but this must be proved first.

Besides its state, a machine also contains a number of events that show the way it may evolve. Each event is composed of a guard and an action. The guard is the necessary condition under which the event may occur. The action determines the way in which the state variables are going to evolve when the event occurs. An event may be executed only when its guard holds. Events are atomic and when the guards of several events hold simultaneously, then at most one of them may be executed at any one moment. The choice of event to be executed is non-deterministic [2].

In order to progress towards implementation, the process of refinement is used in Event B. A refinement is a (usually) more elaborate model than its predecessor. The refinement of a context is simply the addition of new sets, constants and axioms to it. The refinement of a machine consists of refining its state variables and its events. All state variables v are replaced by new ones w, some simply by renaming - i.e. of the same type and meaning - and others by variables of different type. Existing events are transformed to work on the new variables. New events can be defined; that is, the behaviour of an abstract event E can be refined by some sequence of E and new events. The new behaviour will usually reduce nondeterminism.

More details about Event B can be seen in [3].

3 Event-B Patterns

The purpose of a design pattern is to capture structures and decisions within a design that are common to similar modeling and analysis tasks. They can be re-applied when undertaking similar tasks to in order reduce the duplication of effort [4].

Reactive systems are characterized by the fact that their behavior is defined in terms of reactions to input events (or actions). Sequences of inputs are recognized, and outputs can be emitted in response. So, the most basic elements in reactive systems are action and reaction, action-reaction relations as well as the interactions between action-reaction pairs. The action can be viewed as an abstraction of an input command from environment, such as a start impulse signal, a login request, or an output instruction to the controlled agent. We recognized the following five design patterns when modeling this kind of systems according to the relationship between the action and corresponding reaction. The design pattern described in this paper is an Event-B model.

3.1 Pattern for Action and Weak Reaction

Name: Action&WeakReaction

Problem: When an action is emitted, a reaction should start in response to the action. If the action stops to stimulate sequentially, the reaction should also follow it to stop. However, the reaction sometimes has not enough time to react, because the action moves too quickly (the continuance of an action is too short, or the interval between actions is too short). This is so-called action and weak reaction.

Solution: The initial model for action and reaction is made of a state and some events. The state includes two variable, *actionW* and *reactionW* that denote the action and reaction respectively, and two invariant INV0_1 and INV0_2 that are used to constrain the variable's type to be set {0,1}. The essence of action and weak reaction is that the number of reactions is less than the actions'. To formalize the model for action and weak reaction, two new variable caW and crW are introduced within the initial model to count the exact times that action and reaction is set to 1. Of course, two new invariant *INV0_3* and *INV0_4* are needed to constrain the type of caW and crW to be natural number only. The real weak reaction relation in a action-reaction block is described by the third new invariant *INV0_5: crW* $\leqslant$ *caW*.

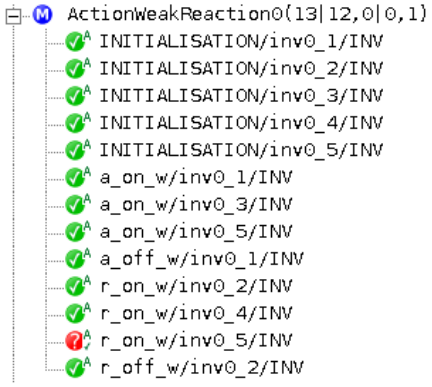

Fig.1 Proving perspective for action&WeakReaction

In addition to the initialization event, four events *a_on_w*, *a_off_w*, *r_on_w* and *r_off_w* are used to describe the behaviors of action and reaction. The event *a_on_w* simply sets variable *actionW* to 1 and increases the counter variable *caW* by 1 when its guard satisfies *actionW=0*. The event *a_off_w* only sets variable *actionW* to 0 when its guard satisfies *actionW=1*.The event *r_on_w* and *r_off_w* are similar.

The model, at this moment, with four variable, five invariants and four events seems to be built completely. However, the proving perspective of Event B shows that the event *r_on_w* fails to preserve the invariant *INV0_5*, as shown in Fig.1, which means something in our model is wrong or incomplete. To guarantee the model is built correctly, while proving invariant preservation by event r_on_w, another new invariant *INV0_6: reactionW=0 ∧ actionW=1 => crW < caW* has to be added. Thus, the model for weak reaction is specified correctly because all 18 Proving Obligation has been proved and discharged automatically.

```
MACHINE  Action&WeakReaction
VARIABLES  actionW, reactionW
INVARIANTS
    INV0_1 : actionW ∈ {0,1}
    INV0_2 : reactionW ∈ {0,1}
EVENTS
  INITIALISATION ≙ WHICH IS ordinary
    BEGIN
       act1 : actionW = 0
       act2 : reactionW = 0
    END
  a_on_w ≙ WHICH IS ordinary _
    WHEN
       grd1 : actionW = 0
    THEN
       act1 : actionW ≔ 1
    END
  a_off_w   ≙ WHICH IS ordinary
    WHEN
       grd1 : actionW = 1
    THEN
       act1 : actionW = 0
    END
  r_on_w ≙ WHICH IS ordinary
    WHEN
       grd1 : actionW = 1
       grd2 : reactionW = 0
    THEN
       act1 : reactionW = 1
    END
  r_off_w ≙ WHICH IS ordinary
    WHEN
       grd1 : actionW = 0
       grd2 : reactionW = 1
    THEN
       act1 : reactionW = 0
    END
END
```

Fig.2 Pattern for action&WeakReaction

After removing the counter variable caW, crW and pertinent invariants as well as action of events from the model, which were introduced for modeling only, we have the pattern for action and weak reaction as shown in Fig.2.

3.2 Pattern for Action and Strong Reaction

Name: Action&StrongReaction

Problem: In addition to the action and weak reaction described above, in some situations, we hope that the reaction could always follow the action's movement on and off, that is to say, the reaction and action can always keep synchronization. This is so-called action and strong reaction.

Solution: The model for action and strong reaction can be built by refine the model for action and weak reaction. The synchronization relation between action and strong reaction requires that the next action should not start before the reaction that tracked the previous action finished. This requirement can be formalized by a new invariant INV1_1: caW ≤ crW+1, based on the model for weak reaction.

To make this invariant is preserved by all events, the invariant *INV0_5*, *INV0_6* and *INV1_1* are modified into *INV2_1: reactionW=0 ∧ actionW=1 => caW = crW+1* and *INV2_2: actionW=0 ∨ reactionW=1 => caW = crW*, at the same time, the guards of event a_on_w and a_off_w are strengthened by predicate *reactionW=0* and *reactionW=1* respectively. At this stage, we have 20 Proving Obligations that have been proved automatically for this model as Fig.4 below.

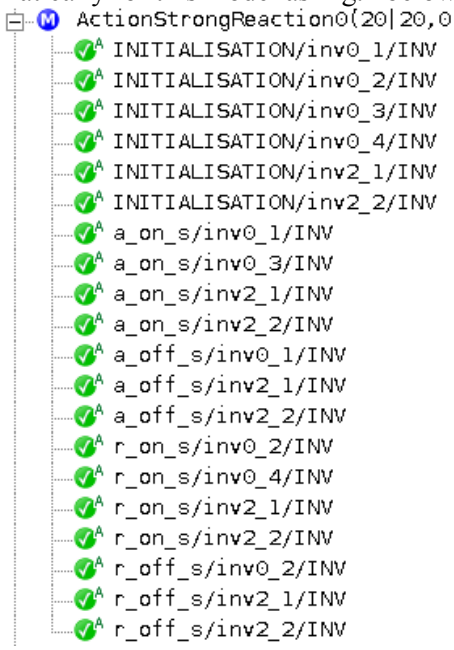

Fig.3 Proving Obligation for action&StrongReaction

Thus, we have the pattern for action and strong reaction as showed in Fig.4 where the counter variable caW, crW and pertinent invariants as well as action of events are removed from the model, and some names for variable and event are changed to symbolize the strong reaction.

```
MACHINE  Action&StrongReaction
VARIABLES  actionS,reactionS,
INVARIANTS
   INV1_1 : actionS ∈ {0,1}
   INV1_2 : reactionS ∈ {0,1}
EVENTS
  INITIALISATION   ≙ WHICH IS ordinary
    BEGIN
      act1 : actionS = 0
      act2 : reactionS = 0
    END
  a_on_s   ≙ WHICH IS ordinary
    WHEN
      grd1 : actionS = 0
      grd2 : reactionS = 0
    THEN
      act1 : actionS = 1
    END
  a_off_s   ≙ WHICH IS ordinary
    WHEN
      grd1 : actionS = 1
      grd2 : reactionS = 1
    THEN
      act1 : actionS = 0
    END
  r_on_s   ≙ WHICH IS ordinary
    WHEN
      grd1 : actionS = 1
      grd2 : reactionS = 0
    THEN
      act1 : reactionS = 1
    END
  r_off_s   ≙ WHICH IS ordinary
    WHEN
      grd1 : actionS = 0
      grd2 : reactionS = 1
    THEN
      act1 : reactionS = 0
    END
END
```

Fig.4 Pattern for action&StrongReactions

3.3 Pattern for Composite Weak and Strong reactions

Name: CompositeWeak&StrongReactions

Problem: Action and Reaction (weak or strong) are only the basic blocks for modeling discrete event system. In most cases, system to be modeled has some complex situations to handle, because functions of a large complex system depend on some sequences of events, in which some events may be of action-reaction relation (even a action-reaction chains) and some may occur simultaneously. The interaction between two action-reaction blocks can be modeled as composite or synchronization, which depend on that the two blocks are of weak-strong reactions or strong-strong reactions. When the weak reaction of a specific action-reaction block results eventually in the specific strong reaction of some action-reaction, it can be recognized as the composite for weak and strong reactions.

```
MACHINE  Composite_Weak&Strong_Reaction
VARIABLES
   actionW,reactionW,
   actionS,reactionS,...
INVARIANTS
   INV1_1 : actionW ∈ {0,1}
   INV1_2 : reactionW ∈ {0,1}
   INV1_3 : actionS ∈ {0,1}
   INV1_4 : reactionS ∈ {0,1}
   ......
EVENTS
  INITIALISATION  ≙ WHICH IS ordinary
    BEGIN
      act1 : actionW = 0
      act2 : reactioW = 0
      act3 : actionS = 0
      act4 : reactioS = 0
      ...
    END
  a_off_s  ≙ ...
  r_on_s   ≙ ...
  r_off_s  ≙ ...
  a_on_w   ≙ ...
  a_off_w  ≙ ...
  r_on_w ≙ WHICH IS ordinary
    REFINES a_on_s
    WHEN
       grd14 : actionW = 1
       grd13 : reactionW = 0
       grd11 : actionS = 0
       grd12 : reactionS = 0
    THEN
       act11 : reactionW = 1
       act12 : actionS = 1
    END
  r_on_w_false ≙ WHICH IS ordinary
    WHEN
       grd11 : actionW = 1
       grd12 : reactionW = 0
       grd13 : ¬(actionS=0∧reactionS=0)
    THEN
       act11 : reactionW = 1
    END
  r_off_s ≙ ...
END
```

Fig.5 Pattern for composite weak & strong reaction

Solution: The model for composite weak and strong reaction can be built by refinement of the strong reaction model. The variables, invariants and events of the refined model are the union set of strong reaction model and weak reaction model respectively, but the event *a_on_s* of strong reaction model need to be refined by event *r_on_w* of weak reaction model. What the composite really means is that the weak

reaction will eventually results in the strong reaction that is stimulated by the strong action, so the refinement of event *a_on_s* can be obtained by superpose the event *r_on_w* of weak reaction model onto it. That is to say, the event r_on_w is the refinement of the event a_on_s, and its guards (grd1: actionS=0 and grd2: reactionS=0) are strengthened by grd3: actionW=1 and grd4: reactionW=0, and its actions are refined by act3: reactionW:=1 and act4: crW:= crW +1. This composite event describes the fact that a weak action results in an event sequence: a weak reaction, a strong action and eventually a strong reaction. However, in case actionS=0∧reactionS=0 is false, that means the strong action had started but its strong reaction has not finished yet, a weak action results in a weak reaction only. Thus, a new event r_on_w_false is added to record the situation, and accordingly four another proving obligations for invariant preservation should be proved.

Finally, the model for composite weak and strong reaction we built has totally 42 proving obligations. After removing all the counter variables from the model, the pattern for composite weak & strong reaction can be obtained as Fig.5

3.4 Pattern for weak synchronization of two Strong reactions

Name: Wsynchronization2StrongReactions

Problem: As far as the synchronization of two strong action-reaction blocks is concerned, two kinds of synchronizations could be identified, which can be recognized as weak synchronization and strong synchronization. When a system requires that, the prerequisite the second strong reaction *s* can be set in on state is that the first strong reaction *r* have already been set in on state, but neither constrains how many times the *s* is set to on nor requests what state the r will be if the s is off. This is what we called weak synchronization of two strong reactions.

Solution: The initial model for weak synchronization of two strong reactions can be built by, firstly incorporating the two strong reaction models so that the variables, invariants and events of the model are the union of two strong reactions but some names are renamed to differentiate the two reactions, and then adding the synchronization invariant *dbl1_1: s=1 => r=1* into the model to describe the fact that the strong reaction *s* are synchronous weakly with the strong reaction *r*, as it is shown in Fig.6.

After the weak synchronization invariant *dbl1_1* was introduced into the model, the event *r_off_s* and *s_on_s* fail to preserve *dbl1_1*. To guarantee the model maintains *dbl1_1* correctly, a series of refinements have to be finished, in which some new invariants need to be added and some events need to be modified. In the end, three invariants:

dbl1_2: b=1 => r=1
dbl1_3: a=0 => r=1
dbl1_4: a=0 => b=0

are appended into the model. The event *a_off_s* is refined by strengthening its guards with *grd13: s=0* and *grd14: b=0*, and event *b_on_s* is refined by strengthening its guards with grd13: r=1 and grd14: a=1.

```
MACHINE
   WSynchronization2StrongReactions
VARIABLES
   a,r,b,s
INVARIANTS
   dbl0_1 : a ∈ {0,1}
   dbl0_2 : r ∈ {0,1}
   dbl0_7 : b ∈ {0,1}
   dbl0_8 : s ∈ {0,1}
   dbl1_1 : s=1 => r=1
   dbl1_2 : b=1 => r=1
   dbl1_3 : a=0 => s=0
   dbl1_4 : a=0 => b=0
EVENTS
   INITIALISATION   ≡ WHICH IS ordinary
     BEGIN
        act1 : a ≔ 0
        act2 : r ≔ 0
        act5 : b ≔ 0
        act6 : s ≔ 0
     END
   a_on_s  ≙ ......
   a_off_s ≙ WHICH IS ordinary
       WHEN
           grd11 : a = 1
           grd12 : r = 1
           grd13 : s = 0
           grd14 : b = 0
       THEN
           act11 : a ≔ 0
       END
   r_on_s  ≙ ......
   r_off_s ≙ ......
   b_on_s  ≙ WHICH IS ordinary
       WHEN
           grd11 : b = 0
           grd12 : s = 0
           grd13 : r = 1
           grd14 : a = 1
       THEN
           act11 : b ≔ 1
       END
   b_off_s ≙ ......
   s_on_s  ≙ ......
   s_off_s ≙ ......
END
```

Fig.6 Pattern for weak synchronization of two strong reactions

The last model for weak synchronization of two strong reactions has 20 new proving obligations that

have been discharged automatically, in addition to 2*20 old proving obligations for two strong reactions. So, the pattern for weak synchronization of two strong reactions are found if all the counter variables for modeling and related invariants are removed from the model, and it is shown in the Fig.6.

3.5 Pattern for Strong synchronization of two Strong reactions

Name: Ssynchronization2StrongReaction

Problem: Another kind of synchronization between two strong action-reaction blocks is so-called strong synchronization of two strong reactions. What the problem with the weak synchronization of two strong reactions is that, although it satisfies the prerequisite that the first strong reaction r will have been in on state whenever the second strong reaction s is going to be on, it neither constrains how many times s can be set in on nor requests what states r will go while s is in off state. The strong synchronization between two strong action-reaction blocks really means that the second reaction *s* will strictly run after the first reaction *r*, which reacts to the first action *a* and changes its value into on or off regularly.

Solution: The model for the strong synchronization of two strong reactions can be built by refining the weak synchronization model. As a matter of fact, the strong synchronization can be formalized by two predicate expressions $ca=cb \vee ca=cb+1$ and $cr=cs \vee cr=cs+1$, where *ca* and *cr* are counters variables for the first action and reaction respectively, and *cb* and *cs* are for the second. In order for the refined model to satisfy the two expressions, a new variable m is introduced into the model, and seven new invariants:

$dbl2\_1 : m \in \{0,1\}$
$dbl2\_2 : m=1 \Rightarrow ca=cb+1$
$dbl2\_3 : m=0 \Rightarrow ca=cb$
$dbl2\_4 : r=1 \wedge s=0 \wedge (m=1 \vee b=1) \Rightarrow cr=cs+1$
$dbl2\_5 : r=1 \vee s=0 \vee (m=1 \wedge b=1) \Rightarrow cr=cs$
$dbl2\_6 : r=1 \wedge a=0 \Rightarrow m=0$
$dbl2\_7 : m=1 \Rightarrow s=0$

are appended gradually into the model to constrain the synchronization conditions and simplify the model proving. Meanwhile, the event a_on_s is refined by a new action *act13: m:=1* and the guards of event a_off_s is strengthened by *grd15: m=0* , and the event b_on_s is refined by a new action *act13: m:=0 and a* strengthened guard *grd15: m=1*. The final model for the strong synchronization has 38 new proving obligations that have been discharged automatically or manually, except for the old 60 proving obligations that have been proved in the weak synchronization.

Similarly, if all the counter variables and related invariants are removed from the model, we have the pattern for the strong synchronization of two strong reactions as in Fig.7.

```
MACHINE
   Ssynchronization2StrongReactions
VARIABLES  a,r,b,s
INVARIANTS
   dbl0_1 : a ∈ {0,1}
   dbl0_2 : r ∈ {0,1}
   dbl0_7 : b ∈ {0,1}
   dbl0_8 : s ∈ {0,1}
   dbl1_1 : s=1 => r=1
   dbl1_2 : b=1 => r=1
   dbl1_3 : a=0 => s=0
   dbl1_4 : a=0 => b=0
   dbl2_1 : m ∈ {0,1}
   dbl2_6 : r=1∧a=0 => m=0
   dbl2_7 : m=1 => s=0
EVENTS
   INITIALISATION   ≡ WHICH IS ordinary
     BEGIN
        act1 : a ≔ 0
        act2 : r ≔ 0
        act5 : b ≔ 0
        act6 : s ≔ 0
        act9 : m ≔ 0
     END
   a_on_s  ≙ WHICH IS ordinary
       WHEN
           grd11 : a = 0
           grd12 : r = 0
       THEN
           act11 : a ≔ 0
           act13 : m ≔ 1
       END
   a_off_s ≙ WHICH IS ordinary
       WHEN
           grd11 : a = 1
           grd12 : r = 1
           grd13 : s = 0
           grd14 : b = 0
           grd15 : m = 0
       THEN
           act11 : a ≔ 0
       END
   r_on_s  ≙ ......
   r_off_s ≙ ......
   b_on_s  ≙ WHICH IS ordinary
       WHEN
           grd11 : b = 0
           grd12 : s = 0
           grd13 : r = 1
           grd14 : a = 1
           grd15 : m = 1
       THEN
           act11 : b ≔ 1
           act13 : m ≔ 0
       END
   b_off_s ≙ ......
   s_on_s  ≙ ......
   s_off_s ≙ ......
END
```

Fig.7 Pattern for strong synchronization of two strong reactions

4 Using Event-B Patterns To Develop the Controller Model for Washing Machine

In object-oriented technology, a design pattern is general reusable solution to a commonly occurring problem in software design, and using design patterns can result in adapting and incorporating some pre-defined pieces of codes in a software project. In formal methods of software development, formal design patterns could also be applied to adapt and incorporate pre-defined proved and refined mini-model into a large one. Although the formal Event-B design patterns we presented in the previous section are focused on reactive systems and may be only a part of them, they are very significant while developing a controller for an automatic washing machine.

4.1 Informal Description of the controller for Washing Machine

A washing machine must sense various inputs and perform a wash cycle by timing and controlling outputs. Fig.8 is a simplified diagram for the washing machine. Since the washing machine controller is a typical reactive discrete-state process control system, the task structure for the controller is shown in Fig.9. It should be noted that the safety of the controller is of specific significant. Hardware interlocks (or control task synchronizes), such as a switch that cuts off the motor and prevents the washing machine's tub from rotating when lid is open, should be used for wherever possible.

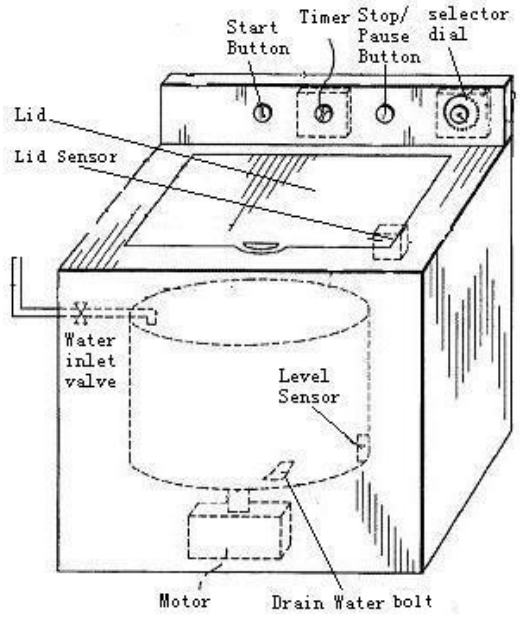

Fig.8 Washing Machine Diagram

Some of the sequence of events through which the controller accomplishes some washing cycle might be as follows:

(1) The user presses the 'Start' button to start the cycle.

(2) The lid is put down so that the lock is closed.

(3) The water valve is opened to allow water into the washing tub. When the 'Full water level' is sensed, the water valve is closed.

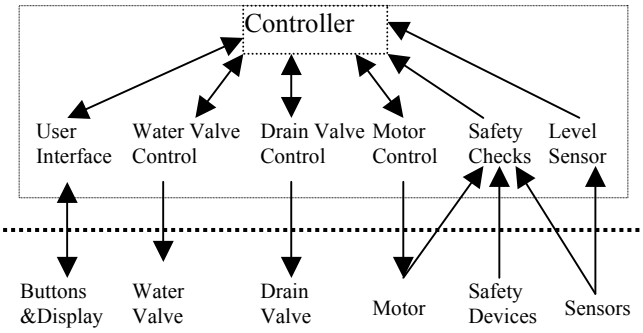

Fig.9 Washing Machine Diagram

(4) The washer motor is turned on to rotate the tub. The motor then goes through a series of movements, both forward and reverse (at various speeds) to wash the clothes. (The precise set of movements carried out depends on the wash program that user has selected.) At the end of the wash cycle, the motor is stopped.

(5) The drain water valve is opened to drain the rub. When the tub is empty, the valve is closed.

(6) The motor is turned on to spin the rub for a while, and then the motor is turned off.

4.2 Requirements of the Controller

The controller we are going to develop is a piece of software connected to some equipment. Some requirements related to environmental equipments include following three statements marked EQP-1, EQP-2 and EQP-3.

The controller has got the following pieces of Equipment: a Motor, a Lid, a water inlet Valve, and a Drain water valve.	EQP-1
A controller is supposed to manage this equipment.	EQP-2
Two Buttons are used to start /stop the Motor, and the Lid bar can be lifted up or put down.	EQP-3

Some functional requirements and/or connection constraints are described in following five statements.

Buttons and Controller are weak syncronized	FUN-1

Controller and Equipment are strongly syncronized	FUN-2

When water Valve is open, the Motor is stopped. Before the Motor is working, the Level Sensor must be full	FUN-3

Some safety requirements include following three statements marked SAF-1 and SAF-2.

When Motor is working, the drainValve must be bolted	SAF-1

When Motor is working, the Lid must put down	SAF-2

4.3 Development of the Controller Model by using Design patterns

The basic roadmap for building the controller model is that create the initial model by instantiating the Action&WeakReaction pattern so that the start and pause/stop buttons are connected to the controller and the FUN-1 is specified. Then the model is refined step by step by using Event B design patterns and refinements, and by connecting one piece of equipment and/or realizing a system function.

For the initial model (requirement FUN-1), its state can be defined as following:

VARIABLES:
start_wash_button
start_wash_impulse
stop_wash_button
stop_wash_impulse

INVARIANTS:
inv1: start_wash_button ∈ BOOL
inv2: start_wash_impulse ∈ BOOL
inv3: stop_wash_button ∈ BOOL
inv4: stop_wash_impulse ∈ BOOL

The behaviors of the initial model can be instantiated by re-using Action&WeakReaction pattern twice so that the Start and Stop/pause Button are connected to the controller.

actionW → start_ wash_button
reactionW → start_ wash_impulse
0 → FALSE
1 → TRUE

a_on_w → push_start_wash_button
a_off_w → release_start_wash_button
r_on_w → treat_push_start_wash_button
r_off_w → treat_release_start_wash_button

actionW → stop_ wash_button
reactionW → stop_ wash_impulse
0 → FALSE
1 → TRUE
a_on_w → push_stop_ wash_button
a_off_w → release_stop_ wash_button
r_on_w → treat_push__stop_ wash_button
r_off_w → treat_release_stop_ wash_button

So we have the initial Event-B model for the controller as Fig. 9

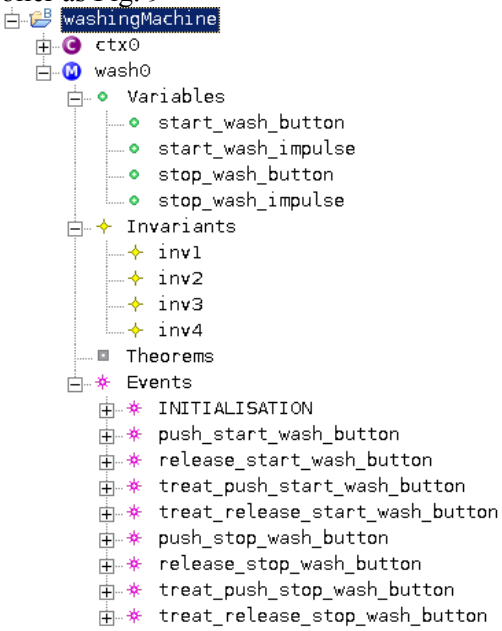

Fig.9 Initial Event-B model of the controller

The first refinement of the model is to connect the motor to controller. Here, a context ctx1 is needed to define the status of the motor as follow:

SETS: MOTOR_STATUS
CONSTANTS: working, stopped
AXIOMS:
axm1: MOTOR_STATUS ={ working, stopped}
axm2: working ≠ stopped

After extend the state of initial model by defining new variables motor_actuator and motor_sensor as well as the related invariants, we instantiate the action&StrongReaction pattern by:

actionS → motor_actuator
reactionS → motor_sensor
0 → stopped
1 → working

a_on_s → treat_start_wash
a_off_s → treat_stop_wash
r_on_s → wash_start
r_off_s → wash_stop

While refining event *treat_start_wash* and *treat_stop_wash*, the Pattern for Composite Weak and Strong reactions should be applied, since the controller reacts weakly to user's push of Start/Stop button (FUN-1) but washer should strongly synchronize the controller (FUN-2) instructions to start or Stop the washer. Fig.10 gives the first refinement machine wash1.

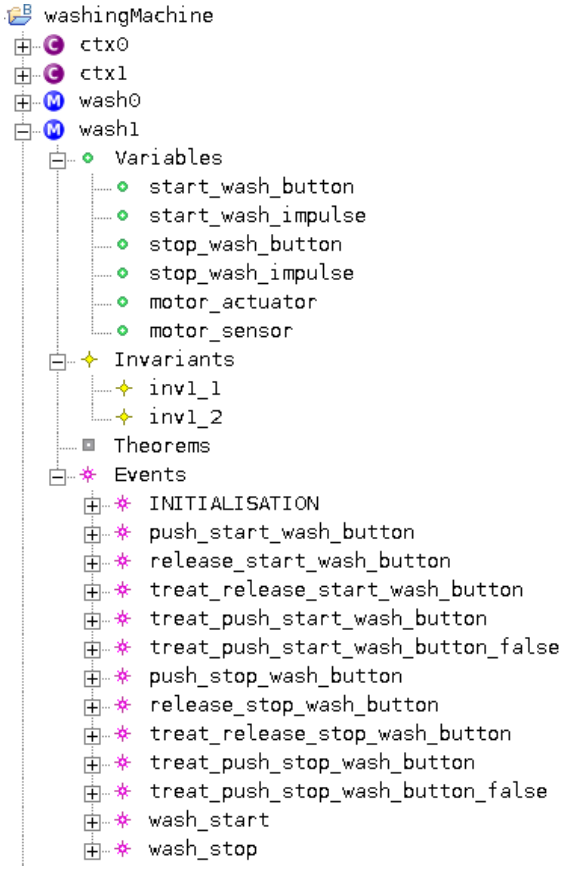

Fig.10 The first refinement of the controller

The second refinement of the model is to connect the water inlet valve, drain water bolt and lid onto the controller (it may also be dealt with a series of refinements to connect them onto the controller one after another). Since the FUN-2 requests that Controller and Equipment are strongly synchronization, the refinement of the controller model in this step can be handled by means of re-using the strong reaction pattern three times to deal with the valve, drain bolt or lid, which are similar to the motor we did during the first refinement. In the first place, a new context *ctx2* is extended from the *ctx1* to define the type sets and constants used by the water valve, drain bolt and lid. Axioms are defined also in the *ctx2* to constrain properties of the sets and constants. Fig.11 is a summary of the *ctx2*. As far as the machine is concern, the refined machine *wash2* includes 12 variables, 12 invariants and 24 events totally, half of which are new for the connecting equipments and the other of which are derived from the abstract machine *wash1*.

CONTEXT ctx2
EXTENDS ctx1
SETS VLAVE, DRAIN, LID
CONSTANTS close,open,bolt,unbolt,up,down
AXIOMS
axm1: VALVE = {close,open}
axm2: close ≠ open
axm3: DRAIN = {bolt,unbolt}
axm4: bolt ≠ unbolt
axm5: LID = {up,down}
axm6: up ≠ down
END

Fig.11 The context ctx2 for the 2nd refinement

The third refinement of the model is to cope with the functional requirement FUN-3, which requests that the water inlet valve should synchronize the washing motor, and the drain water bolt do the spinning motor. These all are weak synchronizations between two strong reactions, therefore the pattern for weak synchronization will be applied. As a result, the relatively abstract machine *wash2* is refined into a more concrete machine *wash3*, which 8 new invariants for synchronization are appended and 4 events are modified. With respect to the context, there is no new one during the refinement of wash3. It still sees the *ctx2*.

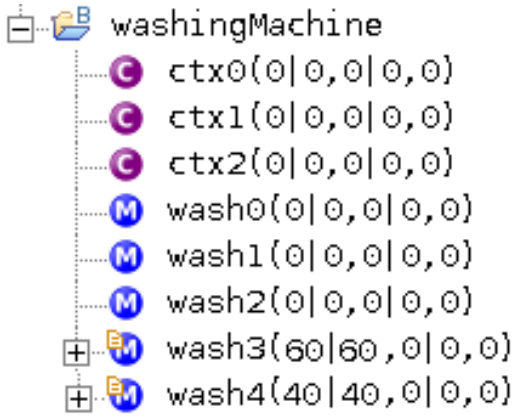

Fig.12 The context ctx2 for the 2nd refinement

The fourth (and last) refinement is about the safety requirement SAF-1 and SAF-2, which claim that the drain valve must be bolted when Motor is washing or the Lid must put down if the motor is spinning. These can be considered as strong synchronization relationship between two strong reactions. Applying the pattern for strong synchronization of two strong reactions to the machine *wash3*, we have the refined machine wash4 as shown in Fig.12, there the last two machines have some proving obligations that had been discharged automatically.

5 Conclusion

Formal methods approach to software construction can significantly increase the reliability and correctness of the resulting software, but formal methods users often lack some systematic methodological measures to help. In object-oriented technology, using design patterns can result in adapting and incorporating some pre-defined pieces of codes in a software project. Event-B design patterns play a similar role in the formal development of complex reactive systems, so that the development process can be accelerated by re-use some pre-defined proved and refined mini-models.

In this paper some Event-B design patterns are presented and applied to specifying the controller model of a washing machine. Although it may be possible not to use design patterns specifying the controller model even the reactive systems, the usage of the formal design patterns leads to a rigorous and systematic development. This will be quite helpful if the target software is critical systems. Other patterns for refinement and interaction among patterns are possible directions for future work.

References:

[1] Wen YJ, Wang J, Qi ZC. , *Compositional Model Checking and Compositional Refinement Checking of Concurrent Reactive Systems,* Journal of Software, Vol.18, No.6, June 2007, pp.1270−1281 (in Chinese with English abstract).

[2] J.R. Abrial, M. Butler, S. Hallestede, L. Voisin. *An open extensible tool environment for Event-B* . In Z.Liu and J.He(Eds): Formal Methods and Software Engineering-ICFEM2006, LNCS 4260 pp588-605,2006.

[3] E Ball, M Butler, *Event-B patterns for Specifying Fault-Tolerance in Multi-Agent Interaction,* in Proceedings of Workshop on Methods, Models and Tools for Fault Tolerance, 2007,

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA , 1995.

[5] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for real-Time Systems*, Pearson Education, Inc, 2003, Chinese Simplified language edition, Beijing University of Aeronautics and Astronautics Press, 2004.

[6] J.R. Abrial and Thai Son Hoang, *Using Design Patterns in Formal Methods: An Event-B Approach (Extended Abstract)*, in J.S.Fitzgerald, A.E.Haxthausen and H.Yenigun (Eds). Theoretical Aspects of Computing – ICTAC 2008, Volume 5160 of Lecture Notes in Computer Science. Springer, Berlin (2008), pp1–2.

[7] B. Arief, A. Iliasov, and A. Romanovsky, Rigorous Development of Ambient Campus Applications that Recover from Errors. Proceedings of the International Conference on Integrated Formal Methods 2007 (IFM 2007), pp. 103-110, 3 July 2007.

[8] Jean-Raymond Abrial and Stefan Hallerstede. *Refinement, decomposition and instantiation of discrete models: Application to Event-B.* Fundamentae Informatica, Volume 77, Number 1-2 / 2007.

[9] D.M.Auslander, J.R.Ridgely, J.D.Ringgenberg *Control Software for Mechanical Systems: Objected-Oriented Design in a Real-Time World*, Pearson Education Asia Limited and Tsinghua University Press, 2004