
Software Design

Friday 27 March 2015

2IO70

The purpose of this document is to present a Java program that realises the functions specified in the Software Specification document. This program is an intermediate step towards writing the PP2 code that controls the sorting machine.

Group 16

Rolf Verschuuren

Wigger Boelens

Stefan van den Berg

Dat Phung

Maarten Keet

Tudor Petrescu

Version 2

Table of Contents

Coding Standards3

Design decisions for the Java code4

Validation5

 Validation of java to transition table5

 Validation of timerManage.....5

 Control flow validation.....5

Appendix: Java Program.....6

Coding Standards

The java pseudo code follows the Google Java Style.

Source to Google Java Style: <https://google-styleguide.googlecode.com/svn/trunk/javaguide.html>.

PHP code used in this project follows the Zend Framework Coding Standard for PHP.

Source: <http://framework.zend.com/manual/1.12/en/coding-standard.html>.

Translating to pseudo java:

The java program starts by declaring the output variables. The names of the output variables will keep their original name, without spaces, in a camelCase form. The variable type will be determined from the Output table.

The inputs follow the same pattern.

Every states is represented as a function, keeping their name in the camelCase fashion, they will be all void functions due to the fact that they do not return anything.

Every state function will run preconditions if any, then check for specific input values using if statements, if an if statement is satisfied, there will be changes to the output values to match the next states output values, also the display is set to output the next states number, and then the next state function is called according to the state transition diagram, if no if statement is satisfied the current function is recalled.

The program is always looping, consequence of no deadlocks in the state machine as proven by the uppaal model test.

Example: Initial -> Calibrate_Sensor

So in this example the function initial is currently running, there are no preconditions to be checked, if the inputs have the desired value, in our case we check if the push button is pressed by the sorter, if so we will have the sorter moved down by activating the sorter motor via having the Hbridge0 variable set to 1. After this we set the display to showcase the number \$branchTO where to branch to2 then call calibrateSensor function and if the if statement wasn't satisfied we recall initial entering a loop.

Translating from Java to PHP

The java code was written such that the conversion process to php is as easy as possible.

All variable in java will have the "\$" sign added at the beginning of their name to comply with the php standards. The "\$" sign has no influence in the java program variable naming, while in php it is mandatory.

Design decisions for the Java code

In translating our transition table to a Java program we made a number of decisions shaping the code, these decisions are outlined in this section.

We started by looking at our transition table, in this table we had our transitions ordered by the “current state”, the state where the transition starts. Then there were some inputs that could trigger a transition from this state to a number of other states. Because of this we thought it would make sense to write a function for each state, since it would allow our code to essentially be a condensed version of the transition table. Where the code would be ordered by the “current state”, and each state would have a number of outgoing transitions to other states. This resulted in the following blueprint for each of our functions:

```
void function(){
    timerManage();          // The function that manages the outputs and PWM
    $temp = getButtons();    // Store the buttons currently being pressed in a temporary variable

    if( condition for transition ){      // Do this once for every outgoing transition from this state
        Changes required to change to the new statement;
    }

    function();              // Call on the same function again to recheck the buttons and
continue                   running the machine with timerManage()
}
```

Then we made an extra function which will be called from each function to do the PWM. This function is called timerManage. This function firstly gets the voltage which the output needs from the array.

This function has a variable called counter which increments each time the outputs have been set. That value is take modulo 12. So it will leave the outputs which need 12 volts on all the time. The reason why the values which need less than 12 volt will be turned off after they have been on for long enough. That goes as follows. First it checks if the engine needs to be on by checking if the voltage it needs is higher than counter. If the output needs to be on then it gets the location of the value in the array. And then does 2 to the power of the location. So now the correct output will be set on. Then the value of 2 to the power will be added to the variable engines. Then after all 7 outputs have been through that loop then it will set the output to the value of engines. So the lights which needed to be on will be on. Now the value of counter will increment each time and take modulo 12.

We also choose to save certain values, which may not be expected to be saved. In this section I will explain why we save the 2 variables. The first one is the variable of the location of the code. This has been saved because then we then we are capable of changing the return address after the timer interrupt. Because when an timer interrupt occurs we want to return to the initial state and the position where we were before. We also saved the original position of the stack pointer for when we come back from the timer interrupt to make sure that we empty the stack. Because there may be some values on the stack from before the timer interrupt. Thus to remove them we set the stack pointer to its original value.

Validation

Validation of java to transition table

Every state is represented by a function. The if statements in that function are the transitions which can occur from that state. The timer interrupt and the abort transitions are not represented as if statements, because interrupts go to a separate state(function). In those if statements the values that have to change are changed. The display will also be updated to the correct number of the state. The function timerManage is called in each state. Because with that function we make sure that the all outputs have the correct voltage.

We checked that all states are represented in the java code by a function. We also checked if they have all the transitions as if statements and that the correct values are changed.

Validation of timerManage

Loop invariant:

All elements before the current element of the array have been set on if they had to be on.

Initialize:

We start with the first element. Thus there are no elements before it and the loop invariant holds.

Step case:

If we're at element k, then according to the loop invariant all elements before k have been set on if they had to be on. Then if k has to be on (value of $k > \text{counter}$) it will be set on else it will stay off. So now the loop invariant holds for the element $k+1$

Termination:

The loop will terminate when k is greater than 7. Because we do not have any more outputs.

Control flow validation

Because the Java code has been validated to the state description and the transition table, which, in turn, have been validated with the UPPAAL model and shown to be correct and in tune with the initial description of the sorting machine. This means that the Java program, being a one-to-one translation of the finite state automaton, also has a correct control flow.

Appendix: Java Program

```
1 /**
2  * Sort of a simulation of the PP2 program
3  * controlling the Fischer
4  * Technik in order to sort black and white discs.
5  *
6  * @author Maarten Keet
7  * @author Stefan van den Berg
8  * @author Rolf Verschuuren
9  * @author Wigger Boelens
10 * @team Group 16
11 * @since 13/3/2015
12 */
13
14
15 class SoftwareDesign {
16     /**@CODE**
17     //inputs
18     int $push, $startStop, $abort, $position,
19         $colour;
20
21     //variables
22     int $state = 0;
23     int $sleep = 0;
24     int $temp = 0;
25     int $location;
26     int $counter = 0;
27     int $engines;
28
29
30     //constants
31     final int TIMEMOTORDOWN = 30;
32     final int BELTROUND = 2000;
33     final int BELT = 1200;
34     final int SORT = 850;
35     final int LENSAMPPOSITION = 5,
36         LENSAMPSORTER = 6,
37         HBRIDGE0 = 0,
38         HBRIDGE1 = 1,
39         CONVEYORBELT = 3,
40         FEEDERENGINE = 7,
41         DISPLAY = 8,
42         LEDSTATEINDICATOR = 9;
43
44     public static void main(String args[]) {
45         SoftwareDesign SoftwareDesign = new
46             SoftwareDesign();
47
48
49         //values for the data segment
50         SoftwareDesign.initVar("outputs", 12);
51         SoftwareDesign.initVar("stackpointer", 1);
52         SoftwareDesign.initVar("offset", 1);
53
54         //store the offset of the programm, this
55         // is used in the interrupt
56         SoftwareDesign.storeData(startofthecode,
57             "offset", 0);
58
59         //store the vluue of the stackpointer, so
60         // we can clear the stack
61         // easily
62         SoftwareDesign.storeData(SP,
63             "stackpointer",
64             0);
65
66         $counter = 0;
67
68
69         //reset outputs
70         SoftwareDesign.storeData(0, "outputs",
71             SoftwareDesign
72                 .HBRIDGE1);
73         SoftwareDesign.storeData(0, "outputs",
74             SoftwareDesign
75                 .LENSAMPPOSITION);
76         SoftwareDesign.storeData(0, "outputs",
77             SoftwareDesign
78                 .LENSAMPSORTER);
79         SoftwareDesign.storeData(0, "outputs",
80             SoftwareDesign
81                 .LEDSTATEINDICATOR);
82         SoftwareDesign.storeData(0, "outputs",
83             SoftwareDesign
84                 .DISPLAY);
85         SoftwareDesign.storeData(0, "outputs",
86             SoftwareDesign
87                 .CONVEYORBELT);
88         SoftwareDesign.storeData(0, "outputs",
89             SoftwareDesign
90                 .FEEDERENGINE);
91
92         //start moving the sorter up
93         SoftwareDesign.storeData(9, "outputs",
94             SoftwareDesign
95                 .HBRIDGE0);
96
97         //go to the first state and set the
98         // value for the display
99         SoftwareDesign.$state = 0;
100        SoftwareDesign.initial();
101    }
102
103    //state0
104    void initial() {
105        setStackPointer(
106            getData("stackpointer", 0));
107        timerManage();
108        //check if the sorter push button is
109        // pressed
110        $push = getButtonPressed(5);
111        if ($push == 1) {
112            //move the sorter down
113            storeData(0, "outputs", HBRIDGE0);
114            storeData(9, "outputs", HBRIDGE1);
115            //update the state
116            $state = 1;
117            //reset sleep for the next function
118            $sleep = 0;
119            calibrateSorter();
120        }
121    }
122    //loop
123    initial();
124 }
125
126 //state 1
127 void calibrateSorter() {
128     timerManage();
129     //the sorter is now moving down,
130     //and we're waiting for it to reach the
131     // bottom
132     if ($sleep == TIMEMOTORDOWN * 1000) {
133         //stop the sorter
134         storeData(0, "outputs", HBRIDGE1);
135         //update the state
136         $state = 2;
137         //reset sleep
138         $sleep = 0;
139         resting();
140     }
141     //loop
142     $sleep++;
143     calibrateSorter();
144 }
145
146 //state 2
147 void resting() {
148     timerManage();
149     //the program waits for the user to
150     // press the start/stop
151     $startStop = getButtonPressed(0);
152     if ($startStop == 1) {
153         //sleep so we don't go to the pause
154         // immediately
155         sleep(2000);
156         //power up the lights
157         storeData(12, "outputs",
158             LENSAMPPOSITION);
159         storeData(12, "outputs",
160             LENSAMPSORTER);
161         //start up the belt and the feeder
162         storeData(9, "outputs", CONVEYORBELT);
163         storeData(5, "outputs", FEEDERENGINE);
164         //set and start the countdown
165         setCountdown(BELTROUND + BELT);
166         startCountdown();
167         //update the state
168         $state = 3;
169         running();
170     }
171     //loop
172     resting();
173 }
174
175 //state 3
176 void running() {
177     timerManage();
178     //check if we need to pause
179     $startStop = getButtonPressed(0);
180     if ($startStop == 1) {
181         //stop the feeder engine
182         storeData(0, "outputs", FEEDERENGINE);
183         //set the timer
184         setCountdown(BELT);
185         //update the state
186         $state = 9;
187         runningTimer();
188     }
189     //check if a disk is at the position
```

```

190 // detector
191 $position = getButtonPressed(7);
192 if ($position == 1) {
193 //reset the countdown,because a
194 // disk was detected
195 setCountdown(BELTROUND + BELT);
196 //update the state
197 $state = 4;
198 runningWait();
199 }
200 //loop
201 running();
202 }
203
204 void runningWait() {
205 timerManage();
206 //check if we need to pause
207 $startStop = getButtonPressed(0);
208 if ($startStop == 1) {
209 //stop the feeder engine
210 storeData(0, "outputs", FEEDERENGINE);
211 //set the timer
212 setCountdown(BELT);
213 //update the state
214 $state = 9;
215 runningTimer();
216 }
217 //check if a disk is at the positiond
218 // detector
219 $position = getButtonPressed(7);
220 if ($position == 1) {
221 //reset the countdown,because a
222 // disk was detected
223 setCountdown(BELTROUND + BELT);
224 //update the state
225 $state = 5;
226 runningTimerReset();
227 }
228 //check if a white disk is at the color
229 // detector
230 $colour = getButtonPressed(6);
231 if ($colour == 1) {
232 //move the sorter up
233 storeData(9, "outputs", HBRIDGE0);
234 //update the state
235 $state = 6;
236 motorUp();
237 }
238 //loop
239 runningWait();
240 }
241
242 //state 5
243 void runningTimerReset() {
244 timerManage();
245 //update the state
246 $state = 5;
247 runningWait();
248 }
249
250 //state 6
251 void motorUp() {
252 timerManage();
253 //check if we need to pause
254 $startStop = getButtonPressed(0);
255 if ($startStop == 1) {
256 //stop the feeder engine
257 storeData(0, "outputs", FEEDERENGINE);
258 //set the timer
259 setCountdown(BELT);
260 motorUpTimer();
261 }
262 //check if the sorter push button is
263 // pressed
264 $push = getButtonPressed(5);
265 if ($push == 1) {
266 //stop the engine,because it is in
267 // the right position
268 storeData(0, "outputs", HBRIDGE0);
269 //update the state
270 $state = 7;
271 whiteWait();
272 }
273 //loop
274 motorUp();
275 }
276
277 //state 7
278 void whiteWait() {
279 timerManage();
280 //we are waiting for the white disk to
281 // be sorted
282 if ($sleep == SORT * 1000) {
283 //start moving the sorter down
284 storeData(9, "outputs", HBRIDGE1);
285 //update the state
286 $state = 8;
287 //reset sleep for the next function
288 $sleep = 0;
289 motorDown();
290
291 }
292 //check if we need to pause
293 $startStop = getButtonPressed(0);
294 if ($startStop == 1) {
295 //stop the feeder engine
296 storeData(0, "outputs", FEEDERENGINE);
297 //set the timer
298 setCountdown(BELT);
299 //update the state
300 $state = 11;
301 whiteWaitTimer();
302 }
303 //loop
304 $sleep++;
305 whiteWait();
306 }
307
308 //state 8
309 void motorDown() {
310 timerManage();
311 //the sorter is moving down
312 if ($sleep == TIMEMOTORDOWN * 1000) {
313 //stop the sorter
314 storeData(0, "outputs", HBRIDGE1);
315 //update the state
316 $state = 9;
317 //reset sleep for the next function
318 $sleep = 0;
319 runningWait();
320 }
321 //check if we need to pause
322 $startStop = getButtonPressed(0);
323 if ($startStop == 1) {
324 //stop the feeder engine
325 storeData(0, "outputs", FEEDERENGINE);
326 //set the timer
327 setCountdown(BELT);
328 motorDownTimer();
329 }
330 //loop
331 $sleep++;
332 motorDown();
333 }
334
335 //state 9
336 void runningTimer() {
337 timerManage();
338 //update state
339 $state = 13;
340 runningStop();
341 }
342
343 //state 10
344 void motorUpTimer() {
345 timerManage();
346 //update state
347 $state = 14;
348 motorUpStop();
349 }
350
351 //state 11
352 void whiteWaitTimer() {
353 timerManage();
354 //update state
355 $state = 15;
356 whiteWaitStop();
357 }
358
359 //state 12
360 void motorDownTimer() {
361 timerManage();
362 //update state
363 $state = 16;
364 motorDownStop();
365 }
366
367 //state 13
368 void runningStop() {
369 timerManage();
370 //check if a white disk is at the
371 // colour detector
372 $colour = getButtonPressed(6);
373 if ($colour == 1) {
374 //move the sorter engine up
375 storeData(9, "outputs", HBRIDGE0);
376 //update the state
377 $state = 10;
378 motorUpStop();
379 }
380 //loop
381 runningStop();
382 }
383
384 //state 14
385 void motorUpStop() {
386 timerManage();
387 //check if the sorter push button is
388 // pressed

```

```

390     $push = getButtonPressed(5);
391     if ($push == 1) {
392         //stop the engien for the sorter
393         storeData(0, "outputs", HBRIDGE0);
394         //update the state
395         $state = 11;
396         whiteWaitStop();
397     }
398     motorUpStop();
399 }
400
401 //state 15
402 void whiteWaitStop() {
403     timerManage();
404     //check if the white disk has been sorted
405     if ($sleep == SORT * 1000) {
406         //start moving the sorter down
407         storeData(9, "outputs", HBRIDGE1);
408         //update the state
409         $state = 12;
410         //reset the sleep for the next
411         // function
412         $sleep = 0;
413         motorDown();
414     }
415     //loop
416     $sleep++;
417     whiteWaitStop();
418 }
419
420 //state 16
421 void motorDownStop() {
422     timerManage();
423     //check if the sorter has moved down
424     if ($sleep == TIMEMOTORDOWN) {
425         //stop the engine of the sorter
426         storeData(0, "outputs", HBRIDGE1);
427         //update the state
428         $state = 9;
429         //reset sleep for the next function
430         $sleep = 0;
431         runningWait();
432     }
433     //loop
434     $sleep++;
435     motorDownStop();
436 }
437
438 //not a state
439 void timerInterrupt() {
440     //show that we have timer interrupt
441     $state = 18;
442     //make the sorter move up
443     storeData(9, "outputs", HBRIDGE0);
444     //stop all other outputs
445     storeData(0, "outputs", HBRIDGE1);
446     storeData(0, "outputs", LENSAMPPOSITION);
447     storeData(0, "outputs", LENSAMPSPORTER);
448     storeData(0, "outputs",
449         LEDSTATEINDICATOR);
450     storeData(0, "outputs", DISPLAY);
451     storeData(0, "outputs", CONVEYORBELT);
452     storeData(0, "outputs", FEEDERENGINE);
453     //make sure that the outputs get set
454     // immediatly
455     timerManage();
456     //set the display to the state of initial
457     $state = 0;
458
459     initial();
460
461 }
462
463 void abort() {
464     //stop all outputs
465     storeData(0, "outputs", HBRIDGE0);
466     storeData(0, "outputs", HBRIDGE1);
467     storeData(0, "outputs", LENSAMPPOSITION);
468     storeData(0, "outputs", LENSAMPSPORTER);
469     storeData(0, "outputs",
470         LEDSTATEINDICATOR);
471     storeData(0, "outputs", DISPLAY);
472     storeData(0, "outputs", CONVEYORBELT);
473     storeData(0, "outputs", FEEDERENGINE);
474     //make sure the outputs stop immediatly
475     timerManage();
476     //update the state to be correct in
477     // aborted
478     $state = 17;
479     aborted();
480
481 }
482
483 //state 17
484 void aborted() {
485     timerManage();
486     //check if we can start again
487     $startStop = getButtonPressed(0);
488     if ($startStop == 1) {
489         //start moving the sorter up for
490
491         // calibration
492         storeData(1, "outputs", HBRIDGE0);
493         //update the state
494         $state = 0;
495         initial();
496     }
497     //loop
498     aborted();
499 }
500
501 void timerManage() {
502
503     //make sure that when counter can not
504     // be higher than 12
505     mod(13, $counter);
506     //get the voltage of output $location
507     int $voltage = getData("outputs",
508         $location);
509     //power up the output when it needs to
510     if ($voltage > $counter) {
511         $engines += pow(2, $voltage);
512     }
513     //check if we are in a new itteration
514     if ($counter == 0) {
515         //set the first part of the display
516         $temp = getData("state", 0);
517         mod(10, $temp);
518         display($temp, "display", "1");
519     }
520
521     //check if we are at the end of the
522     // iteration
523     if ($counter == 12) {
524         //set the second part of the display;
525         $temp = getData("state", 0);
526         $temp = $temp / 10;
527         mod(10, $temp);
528         display($temp, "display", "01");
529     }
530
531     //check if we did all outputs
532     if ($location > 7) {
533         display($engines, "leds", "");
534         //set the variables for the next run
535         $engines = 0;
536         $location = 0;
537         $counter++;
538     }
539
540     //check if abort is pressed
541     $abort = getButtonPressed(1);
542     if ($abort == 1) {
543         abort();//stop the machine
544     }
545     return;
546 }
547
548 }
549
550 $location++;
551 timerManage();
552 }
553 }

```