Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2

Erik Poll[1], Patrice Chalin[2], David Cok[3], Joe Kiniry[4], and Gary T. ,9.9/F89wd.4693-(ns)1(y)]TJ/F76.9738T

2.1 Method contracts

We begin with the specification of a `TickTockClock`

```
1  //@ model import org.jmlspecs.lang.JMLDataGroup;
2  public class TickTockClock {
3    //@ public model JMLDataGroup _time_state;
4
5    //@ protected invariant 0 <= hour && hour <= 23;
6    protected int hour; //@ in _time_state;
7    //@ protected invariant 0 <= minute && minute <= 59;
8    protected int minute; //@ in _time_state;
9    //@ protected invariant 0 <= second && second <= 59;
10   protected int second; //@ in _time_state;
11
12   //@ ensures getHour() == 12  && getMinute() == 0  &&  getSecond() == 0;
13   public /*@ pure @*/ TickTockClock() {
14       hour = 12; minute = 0; second = 0;
15   }
16
17   //@ requires true;
18   //@ ensures 0 <= \result && \result <= 23;
19   public /*@ pure @*/ int getHour() { return hour; }
20
21   //@ ensures 0 <= \result && \result <= 59;
22   public /*@ pure @*/ int getMinute() { return minute; }
23
24   //@ ensures 0 <= \result;
25   //@ ensures \result <= 59;
26   public /*@ pure @*/ int getSecond() { return second; }
27
28   /*@  requires   getSecond() < 59;
29     @  assigeable _time_state;
30     @  ensures    getSecond() == \old(getSecond() + 1) &&
31     @             getMinute() == \old(getMinute()) &&
32     @             getHour() == \old(getHour());
33     @ also
34     @  requires   getSecond() == 59;
35     @  assigeable _time_state;
36     @  ensures    getSecond() == 0;
37     @  ensures    (* hours and minutes are updated appropriately *);
38     @*/
39   public void tick() {
40       second++;
41       if (second == 60) { second = 0; minute++; }
42       if (minute == 60) { minute = 0; hour++; }
43       if (hour == 24)   { hour = 0; }
44   }
45 }
```

Fig. 1. JML specification for `TickTockClock`. The datagroup `_time_state`, the associated `assigeable` in

2.2 Purity

In the DBC approach, o8ly query methods can be used in assertion expressions because they are required to be side-e□ect free [Mey97]. The correspondi8g co8-cept in JML is k8own as method *purity*; i.e. o8ly methods declared as `pure` can be used in assertion expressions. E.g., since the method `getSecond()` is declared `pure`, it is legal to make use of it in the postcondition of `tick()`.

```
class SettableClock extends TickTockClock {

    // ...

    /*@ public normal_behavior
      @   requires 0 <= hour && hour <= 23 &&
      @            0 <= minute && minute <= 59;
      @   assignable _time_state;
      @   ensures  getHour() == hour &&
      @            getMinute() == minute && getSecond() == 0;
      @ also
      @  public exceptional_behavior
      @   requires !(0 <= hour && hour <= 23 &&
      @              0 <= minute && minute <= 59);
      @   assignable \nothing;
      @   signals (IllegalArgumentException e) true;
      @   signals_only IllegalArgumentException;
      @*/
    public void setTime(int hour, int minute) {
        if(!(0 <= hour && hour <= 23 && 0 <= minute && minute <= 59)){
             throw new IllegalArgumentException();
        }
        this.hour = hour;
        this.minute = minute;
        this.second = 0;
    }
}
```

of any runtime exceptions, making the specification a lot stronger than it might appear at first sight.

2.5 Instance and static invariants (and the callback problem)

A JML `invariant` clause declared with a `static` modifier is called a *static invariant*. Static invariants express properties which must hold of the static attributes of a class. An assertion that appears in a non-static `invariant` clause is called a *instance invariant* or an *object invariant*. Note that while this terminology is contrary to the literature, it is more accurate with respect to the nomenclature of Java. In this paper, an unqualified use of the term "invariant"

starts or ends – are called the *visible states*. The visible state semantics for invariants says that all invariants of all objects have to hold at these visible

```
public class Clock {
  //@ public model long _time;
  //@ private represents _time = second + minute*60 + hour*60*60;

  //@ public invariant _time == getSecond() + getMinute()*60 + getHour()*60*60;
  //@ public invariant 0 <= _time && _time < 24*60*60;

  //@ private invariant 0 <= hour && hour <= 23;
  private int hour; //@ in _time;
  //@ private invariant 0 <= minute && minute <= 59;
  private int minute; //@ in _time;
  //@ private invariant 0 <= second && second <= 59;
  private int second; //@ in _time;

  //@ ensures _time == 12*60*60;
  public /*@ pure @*/ Clock() { hour = 12; minute = 0; second = 0; }

  //@ ensures 0 <= \result && \result <= 23;
  public /*@ pure @*/ int getHour() { return hour; }

  //@ ensures 0 <= \result && \result <= 59;
  public /*@ pure @*/ int getMinute() { return minute; }

  //@ ensures 0 <= \result && \result <= 59;
  public /*@ pure @*/ int getSecond() { return second; }

  /*@ requires 0 <= hour && hour <= 23;
    @ requires 0 <= minute && minute <= 59;
    @ assignable _time;
    @ ensures _time == hour*60*60 + minute*60;
    @*/
  public void setTime(int hour, int minute) {
    this.hour = hour; this.minute = minute; this.second = 0;
  }

  //@ assignable _time;
  //@ ensures _time == \old(_time + 1) % 24*60*60;
  public void tick() {
```

```
class AlarmClock extends Clock {
  //@ public model int _alarmTime;
  //@ private represents _alarmTime = alarmMinute*60 + alarmHour*60*60;

  //@ public ghost boolean _alarmEnabled = false; //@ in _time;

  //@ private invariant 0 <= alarmHour && alarmHour <= 23;
  private int alarmHour; //@ in _alarmTime;

  //@ private invariant 0 <= alarmMinute && alarmMinute <= 59;
  private int alarmMinute; //@ in _alarmTime;

  private /*@ non_null @*/ AlarmInterface alarm;

  public /*@ pure @*/ AlarmClock(/*@ non_null @*/ AlarmInterface alarm) {
    this.alarm = alarm;
  }

  /*@ requires 0 <= hour && hour <= 23;
    @ requires 0 <= minute && minute <= 59;
    @ assignable _alarmTime;
    @*/
  public void setAlarmTime(int hour, int minute) {
    alarmHour = hour;
    alarmMinute = minute;
  }

  // specla
```

```
1  public interface AlarmInterface {
2    public void on();
3    public void off();
4  }
```

Fig. 5. Interface of the alarm used in `AlarmClock`

```
invariant _alarmTime+60 <= 24*60*60 ==>
    ( _alarmEnabled
      <==> _alarmTime <= time && time <= _alarmTime+60);
```

```
        //@ set _alarmEnabled = true;
    }
}
```

```
public class DigitalDisplayClock {
 //@ public model long _time;
 //@ private represents _time = getSecond()+getMinute()*60+getHour()*60*60;

 //@ private invariant time.length == 6;
 //@ private invariant 0 <= time[0] && time[0] <= 9; // sec
 //@ private invariant 0 <= time[1] && time[1] <= 5; // sec
 //@ private invariant 0 <= time[2] && time[2] <= 9; // min
 //@ private invariant 0 <= time[3] && time[3] <= 5; // min
 //@ private invariant 0 <= time[4] && time[4] <= 9; // hr
 //@ private invariant 0 <= time[5] && time[5] <= 2; // hr
 //@ private invariant time[5] == 2 ==> time[4] <= 3; // hr
 private /*@ non_null rep @*/ int[] time; // NB rep modifier
 /*@ pure @*/ public DigitalDisplayClock() {
        time = new rep int [6]; }   // NB rep modifier

 //@ ensures 0 <= \result && \result <= 23;
 public /*@ pure @*/ int getHour() { return  time[5]*10 + time[4]; }

 //@ ensures 0 <= \result && \result <= 59;
 public /*@ pure @*/ int getMinute() { return time[3]*10 + time[2]; }

 //@ ensures 0 <= \result && \result <= 59;
 public /*@ pure @*/ int getSecond() { return time[1]*10 + time[0]; }


```

```
class MyDigitalDisplayClock extends DigitalDisplayClock{

  //@ requires time.length == 6;
  /*@ pure @*/ public BrokenDigitalDisplayClock( /*@ non_null @*/ int[] time) {
     this.time = time;
  }


```

$[BDF^+$

port 98-06-rev29, Iowa State University, Department of Computer Science, January 2006. To appear in

[Win90] Jeannett2 M. Wing. A specifier's introduction to formal methods. *Computer*