Beyond Assertions: Advanced Specification and

1.1 Approaches to Verification

2.1 Method contracts

We begin with the specification of a `TickTockClock`

```java
//@ model import org.jmlspecs.lang.JMLDataGroup;
public class TickTockClock {
  //@ public model JMLDataGroup _time_state;

  //@ protected invariant 0 <= hour && hour <= 23;
  protected int hour; //@ in _time_state;
  //@ protected invariant 0 <= minute && minute <= 59;
  protected int minute; //@ in _time_state;
  //@ protected invariant 0 <= second && second <= 59;
  protected int second; //@ in _time_state;

  //@ ensures getHour() == 12  && getMinute() == 0  &&  getSecond() == 0;
  public /*@ pure @*/ TickTockClock() {
      hour = 12; minute = 0; second = 0;
  }

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

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

inerficat((on)1.)]TJ/F8269.963Tf0e-8.6770Td[2.2ePur(te3)1y

```
class SettableClock extends TickTockClock {


```

details in frame properties and provides flexibility in specifications. This section explains these notions, and the need for them.

An `assignable`

4 Model fields

Model fields [CLSE05] are closely related to the notion of data abstraction

```
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 invariant 0 <= minute && minute <= 59Td[(9)]TJ/10
  private int hour; //@ in _time;
```

9

in `AlarmClock` has to meet the specification for that method given in `Clock`. For example, the method `tick()`, which is overridden in `AlarmClock`, has to meet the specification given for it in `Clock`. Note that any methods which are not overridden have to be re-verified, to ensure that they maintain any additional invariants of the subclass. (Ruby and Leavens have done some work [?] that

```
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;
  }

  // spec inherited from superclass Clock
  public void tick() {
    super.tick();
    if (getHour() == alarmHour & getMinute() == alarmMinute & getSecond() == 0) {
         alarm.on();
         //@ set _alarmEnabled = true;
    }
    if ((getHour() == alarmHour & getMinute() == alarmMinuteb5 & getSecond() == 0)
        ||
        (getHour() == alarmHourb5 & alarmMinute == 59 & getSecond() == 0) ) {
         alarm.off();
         //@ set _alarmEnabled = false;
    }
  }

}
```

Fig. 4. Example JML specification illustrating the concepts of specification inheritance and ghost fields.

```
        (0 <= _time && _time <= (_alarmTime+60) % 24*60*60)
     ) );
```

Verification by ESC/Java2 will immediately point out that these invariants can be violated, namely by invocations of `setTime` and `setAlarmTime`. This high-

e current time and the alarm time in the decision to turn the alarm o might

Of course, one could also choose to turn the ghost field into a real field, so that it can be used in the implementation. This would make the implementation simpler to understand.

Ghost vs model fields To recap, the crucial difference between a ghost and a model field is that a ghost field extends the state of an object, whereas a model field is an abstraction of the existing state of an object. A ghost field can be assigned to in annotations using the special `set` keywords. A model field cannot be assigned to, but it changes value automatically whenever some of the state

[LBR06] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report 98-06-rev29, Iowa State University, Department of Computer Science, January 2006. To appear in