The formal modelling and verification of safety critical ATP software design

F. Yan & T. Tang
School of Electronics and Information Engineering, Beijing Jiaotong University, Beijing, People’s Republic of China

Abstract

The safety of software is becoming increasingly important as computers pervade control systems on which human life depends. This has become more complex and in rail transportation fields and the methods to ensure its correctness have been slow in development. One feasible approach is to mathematically verify software design in such systems with Formal Methods. ATP (Automatic Train Protection) is a vital part of Train Control Systems. It assures safe train movement by a combination of train detection, separation of trains running on the same track or over interlocked routes, over speed prevention, and route interlocking. Obviously ATP is a safety-critical system and we regard it as a case study for our formal development methods. Firstly, the multi-tasks ATP onboard software model and state transitions will be modelled with UML; secondly, the timing model will be verified to meet the requirement of timing by SMV model checker; finally, the multi-tasking timing model will be realized with VxWorks (a real-time operating system by WindRiver). A major conclusion of the survey is that formal methods, while still immature in some respects, can be used successfully to assist in developing safety-critical systems.

Keywords: safety critical system; Formal Methods; ATP; UML; SMV.

1 Introduction

Nowadays, rail transportation is improving rapidly in China, and we need to assure that Train Control Systems meet safety requirements because they play an important role in the safety of the public. With the development of modern computer technology, it has become possible to protect against drivers’ errors; the generic label for systems to do this is Automatic Train Protection (ATP).
Operational experience indicates that ATP enhances the safety of a transit system because it safeguards against collisions and derailments more effectively than manual and procedural methods. Failing to meet the specified timing constraints will lead to a disaster. Safety concerns in computer systems like ATP are even more confusing. Such systems consist of many subcomponents, which are tightly coupled and have highly complex interactions. When such a system is further embedded in a larger system, the probability of failure greatly increases.

Formal methods are perceived by the community as a way of increasing confidence for safety critical systems. Formal methods are mathematically based techniques, often supported by reasoning tools, that can offer a rigorous and effective way for the specification, development and verification of software systems [1]. We try to apply formal methods into the ATP software development lifecycle. The ATP software specification will be modelled formally and verified to meet the requirement of safety.

2 ATP system

The Automatic Train Protection system (ATP) is defined as assuring safe train movement by a combination of train detection, separation of trains running on the same track or over interlocked routes, over speed prevention, and route interlocking. The ATP system incorporates all of the equipment that is required to ensure safe operation of trains. ATP systems continually calculate the maximum safe speed of a train in the light of current track and signal conditions, compare the actual speed with the maximum, and apply the brakes automatically if the train is going too fast. The design of ATP is based on fail-safe principles for all portions, both wayside and on-board. The sole purpose of the train protection system is to assure the safety of vehicle movement by preventing collisions and derailments.

From Figure 1, we can see a safe brake model of train. When the train is near the stop point, ATP will trigger Service Brake, and then the train will stop safely according to Service Brake Curve. If its speed is exceed limit, the Emergency Button will be triggered automatically, and the train will run according to Emergency Brake Curve, and must stop before the protection point.

Train protection functions and requirements override all other control system functions either through equipment design or, in a completely manual mode, by rules and procedures. The functions that make up train protection are [2]:

- Train detection—monitoring of the track to determine the presence and location of trains;
- Train separation—assuring that trains on the same track maintain a safe following distance to prevent collisions;
- Route interlocking—preventing trains on crossing, merging, or branching routes from making conflicting (unsafe) moves that would cause a collision or derailment;
- Overspeed protection—assuring that train speed remains at or below the commanded speed limit as to prevent collisions resulting from going too
fast to stop within the available distance and to prevent derailments due to excessive speed on curves or through switches;
- Train and track surveillance-observing conditions on and in the vicinity of the track ahead of the train and monitoring safety-related conditions on board the train.

Figure 1: Safe brake model of train.

3 Formal development method for ATP software

The safety of software is becoming increasingly important as computers pervade control systems on which human life depends. Whilst hardware has become significantly more reliable over the years, the same cannot be said of software. From a software perspective, developing safety critical systems is going to require significant advances in areas such as specification, architecture, verification, and process.

A significant problem of developing software for safety critical systems is how to guarantee that the functional behaviour of a developed software system will satisfy the corresponding functional requirements and will not violate the safety requirements for the associated overall system. In order to solve this problem, it is important to analyze thoroughly the safety properties of the overall system, to achieve accurate software functional requirements and to verify properly the implementation of the software.

The formal development method of safety critical software is studied based on international standards. As described in figure 2, every development phase of
system lifecycle forms a “V” diagram. From system hazard analysis, the SRS (System Requirement Specification) and system safety plan are constituted. Then according to formalized software specification, the software design is accomplished. After formal verification and validation, we can assure the implemented software meets the safety requirements. The formal specification is a useful communication tool between customer and designer, between designer and implementer, and between implementer and tester. We use UML as specification tool. In formal verification phase, we use SMV, a kind of model checker, as verification tool.

![Diagram of Formal Development Lifecycle](image)

Figure 2: Formal development lifecycle for onboard ATP systems.

### 3.1 UML Modelling

The most effective means to avoid accidents during a system’s operation is to eliminate or reduce dangers during the design and development of a system. Also, accurate requirements capture is very important in the design of any system. A mistake at this stage will be carried through the entire development process and will be very expensive to correct later. Studies have shown that a modification in service can cost up to 1,000 times more than a modification at the requirements stage. Even worse, two thirds of all errors are made at the requirements state [3]. It clearly makes sense to ensure that the requirements are correct before proceeding with development.

Unified Modelling Language (UML) is the OMG standard notation for object-oriented modelling. It is a general-purpose visual modelling language that is used to specify, visualize, construct and document the artefacts of a software system. UML provides a wide range of notations to model a software system from different perspectives, organizing them in five interrelated views: design view, process view, implementation view, deployment view, and use case view.
The role of each of these views is complementary with the others, and the last one must guide a software process. Each view presents both static and dynamic aspects. Dynamic issues are commonly described by means of interaction, state and activity diagrams [4].

Figure 3: ATP on-board system use case diagram.

According to ATP functions which are realized by on-board and track-side systems, we construct On-board ATP Use Case Diagram. In Figure 3, there are 4 actors, Driver, locomotive signal, shaft and ATP agent. The ATP agent receives the locomotive signals from track-side systems and shaft counter signals for speed measure. Then it ensures the safety operation of the train according to ATP rules. The driver is charge of driving train.

Figure 4: ATP on-board system package diagram.

Next, we divide ATP system into several subsystems by constructing Package Diagram, as described in Figure 4. In ATP process subsystem, TMR (Triple Modular Redundancy) computer systems are used to ensure the safety of the process of ATP system.
Figure 5: ATP on-board system collaboration diagram.

Figure 5 - ATP collaboration diagram shows interaction patterns among objects to perform operations. ATP processor compares the speed between current speed measured by shaft counter (marked as 1: SpeedInput) and the speed limit from ground communication (marked as 2: Signal). When current speed is close to the speed limit, the ATP system will give a warning. When the train is over speed, it will trigger ServiceBrake or Emergency Button automatically.

Table 1: On-board ATP tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Task ID</th>
<th>CPU Time (ms)</th>
<th>Period (ms)</th>
<th>PRI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shaft Input 1</td>
<td>req11</td>
<td>2</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Shaft Input 2</td>
<td>req12</td>
<td>2</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Speed Process</td>
<td>req13</td>
<td>5</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>Signal Interface</td>
<td>req21</td>
<td>2</td>
<td>200</td>
<td>3</td>
</tr>
<tr>
<td>ATP Process</td>
<td>req22</td>
<td>10</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>ATP Output</td>
<td>req23</td>
<td>5</td>
<td>200</td>
<td>5</td>
</tr>
<tr>
<td>ATP Record</td>
<td>req24</td>
<td>5</td>
<td>200</td>
<td>6</td>
</tr>
<tr>
<td>MMI Interface</td>
<td>req25</td>
<td>5</td>
<td>200</td>
<td>7</td>
</tr>
<tr>
<td>Dual Computers supervising</td>
<td>req31</td>
<td>5</td>
<td>500</td>
<td>8</td>
</tr>
</tbody>
</table>

By the UML model, we can construct nine tasks and allocate their priorities, as described in Table 1. To ensure safety, we must testify the multi-task model has liveness property, that is to say, there is no deadlock and every task can
finish in time. Besides, dual ATP systems A and B are used to deal with possible failures. When system A is found fail to function, system B, which is supervising the operation of system A will run at once.

3.2 Formal verification

3.2.1 SMV model checker
The term model checking designates a collection of techniques for the automatic analysis of reactive systems. Subtle errors in the design of safety-critical systems that often elude conventional simulation and testing techniques can be (and have been) found in this way. Because it has been proven cost-effective and integrates well with conventional design methods, model checking is being adopted as a standard procedure for the quality assurance of reactive systems.

Symbolic model checking based on binary decision diagrams (BDDs) is an efficient automatic verification technique that is simultaneously capable of scaling and of verifying a wide range of properties. It has been applied successfully to many industry-scale hardware circuits, but not aggressively to the analysis of software specifications. SMV is a CTL symbolic model checker using BDDs to represent state sets and transition relations [5]. SMV was designed by CMU, and we can get it from the website www-2.cs.cmu.edu/~modelcheck/code.htm.

In temporal-logic model checking, we are given a state transition system, which models a software or hardware system, and a property specified as a formula in a certain temporal logic, and determine whether the system satisfies the formula. A common logic for model checking is the branching-time Computation Tree Logic (CTL), which extends propositional logic with certain temporal operators. Typical formulas include the following (meanings of the temporal operators such as AG will be given later):

- AG safe: All reachable states are safe.
- AG AF stable: The system is stable infinitely often.
- AG (request → AF response): A request is always followed by a response sometime in the future.
- AG EF restart: It is possible to restart the system in any reachable state.

Each formula is evaluated at some state $q$. A proposition holds at $q$ if $q$ satisfies the proposition. The operator A means “for all paths starting at $q$,” E means “for some path starting at $q$,” G means “for every state along the path,” and F means “for some state along the path.” So AG safe holds at $q$ if every state (G) along every path (A) starting at $q$ satisfies the proposition safe.

The system satisfies a formula if the formula holds at all initial states. If not, a model checker typically attempts to find a counterexample. For example, if the formula AG safe is false, a counterexample is a finite path starting at some initial state and ending at a state that is not safe. [6]

3.2.2 Verification programme
The safety and liveness (non deadlock) of the ATP multi-task model is checked by SMV.
/*Set the time and period of every task, e.g. shaft counter 1*/

MODULE pipe11(timeout, processor_granted, request, finish)
VAR  state: 0..2;  /* program counter*/
DEFINE
  start  := state = 0 & timeout;
  finish := state = 2;
  request := case
    /*Tells the scheduler that the task wants to execute*/
    state = 0: 0;
    1:         1;
  esac;
ASSIGN
  init(state) := 0;
  next(state) := case
    start: 1;
    finish: 0;
    !processor_granted = p11: state;
    state = 0: 0;
    1: state + 1 mod 3;
  esac;

/*Constructing the pipeline*/
P11: pipe11(timeout100, processor_granted, req11, P11finish);
P12: pipe12(P11finish, processor_granted, req12, P12finish); .... etc.

/*Defining the priority of every task*/
DEFINE
  processor_granted := case
    req11: p11;
    req12: p12;
    req13: p13;
    req21: p21;
    req22: p22;
    req23: p23;
    req24: p24;
    req25: p25;
    req31: p31;
    1: idle;
  esac;

/*error would occur if a timeout occurs and the processor hasn't finished the previous execution yet*/
error :=  timeout100 & !(P11.state = 0) |
  timeout100 & !(P12.state = 0) |
  timeout100 & !(P13.state = 0) |
  timeout200 & !(P21.state = 0) |
  timeout200 & !(P22.state = 0) |
  timeout200 & !(P23.state = 0) |
  timeout200 & !(P24.state = 0) |
timeout200  & !(P25.state = 0) | 
timeout500  & !(P31.state = 0) ;
/* CTL formula which means non error for ever */
SPEC AG !error

The result of model checking:
-- specification AG (!error) is true

resources used:
processor time: 2.25 s,
BDD nodes allocated: 45730
Bytes allocated: 1957528

BDD nodes representing transition relation: 812 + 1

We can see that there is no error after 2.25s checking, that is to say, our design is safety.

4 Conclusions

Formal methods, while still immature in certain important respects, are beginning to be used seriously and successfully by railways to assist in developing safety-related systems. By applying Formal methods in ATP, we design and verify the safety and liveness (non deadlock) of the ATP multi-task model by SMV. This makes us gain more confidence for assuring the system design. Then we can use VxWorks (a famous real time operating system) to realize our safety design.

References