FORMAL SPECIFICATION AND VERIFICATION OF RAILWAY SYSTEMS USING UML

K. Lano, K. Androutsopolous, D. Clark
Dept. of Computer Science, King's College London, Strand, London WC2R 2LS, UK
Phone: +44 (0)207 848 2832. Fax: +44 (0)207 848 2851. email: { kcl | kelly | david }@dcs.kcl.ac.uk

Abstract: In this paper we investigate how UML can be used to formally specify and verify critical railway systems. A benefit of using UML is its status as an international standard and its widespread use in the software industry. One disadvantage is its unclear and incomplete semantics, we deal with this by defining a precise subset, RSDS, of UML which has a well defined semantics and verification tools and techniques.
Keywords: Critical systems, UML, RSDS, Formal Specification.

1. INTRODUCTION

A widening gap has opened up between the techniques (object-oriented design) used in the mainstream of software development, and those used and advised for critical system development. The motivation for the RSDS work at King's College is to reduce this gap and enable the use of some object-oriented techniques for critical systems, by combining existing formal and object-oriented approaches.

UML is a widely used notation for object-oriented specification and design, and is also an international standard. It may therefore be considered more appropriate than more experimental notations such as Object-Z (Smith, 2000) as a language for critical system specification. By providing a rigorous semantics for a subset of UML, and a systematic translation to code, we aim to support the use of UML as a formal specification language (Evans *et al*, 1998) and as an *executable* formal specification language (Booch, 2002). In addition, multiple verification tools can be used with RSDS/UML, to provide independent verification as recommended in IEC 61508 (IEC, 2000) and related railway industry standards such as CENELEC EN 50128 (CENELEC).

The formal specification of railway systems has been carried out in a number of formal methods, such as Z (Huber, 2002) and B (Dehbonei, 1995). The benefit of such specification is that properties, especially safety properties, of the systems can be analysed in a precise manner. A specification can also assist engineers in comprehending the functioning of a system.

In this paper we will investigate the use of RSDS/UML on the railway signalling system described in (CSRR, 1994), a semi-automated route management system.

2. RSDS/UML SPECIFICATION

RSDS uses a subset of UML class diagram and statechart notation to specify the data and behaviour of reactive systems, and to verify and analyse these specifications via translations to tools such as B (Lano, 1996) and SMV (Burch, 1990). Java code can also be synthesised from the specifications.

RSDS uses abbreviated forms of OCL constraints (OMG, 2002) attached to classes or sets of associations to specify safety and operational invariants of reactive systems in a concise manner. For example, in Figure 1, the constraint is interpreted as

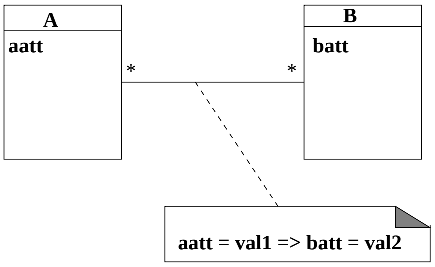

Fig. 1: Example RSDS Specification

$$\forall\, x, y \cdot (x, y) \in A\_B \;\wedge$$
$$x.aatt = val1 \;\Rightarrow$$
$$y.batt = val2$$

where A_B is the set of pairs of objects in the association between A and B.

3. COLD SPRING RAILROAD SPECIFICATION

This system supports dispatchers in organising the set-up and cancellation of routes and the blocking of parts of the network (eg, for maintenance) whilst enforcing safety rules such that at most one train can occupy a given route.

The railway network consists of a set of *locations*, each of which has an occupancy detector, and may also have a signal and/or a switch (Figure 2). Each switch may be in

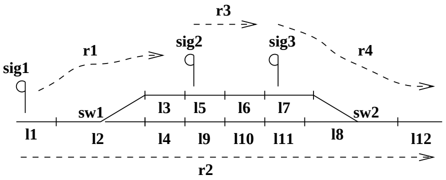

Fig. 2: Example Track Layout

either the normal (straight through) position or reverse (directing the train to the side) position, or neither. A train can only travel over the point if it is in the normal or reverse position. Each switch has sensors *swn*, *swr* to detect its position, and motor *swset* to change its position. Each signal may be set to clear (allowing full speed) or half, or stop. The network is divided into *routes*, which consist of a sequence of adjacent track locations extending from a location containing a signal, the *lead signal* of the route, to a location immediately before the next signal in the same direction. In Figure 2 the four routes have been indicated.

Safety invariants can be derived from hazard analysis, as described in (Lano *et al*, 2002). For example the hazard of train collision due to entry of a train into an already occupied track location leads to the invariants $C1$ and $C7$ below. The hazard of a train entering into a location under repair is prevented by invariants $C2$ and $C8$. The hazard of derailment due to a train passing over an unset switch is handled by invariant $C5$, and the hazard of derailment due to passing too rapidly over a reverse set switch is covered by invariant $C9$.

The dispatchers can issue commands of the form *cmd locn*, where *locn* is the number of the location the command is intended for. *cmd* may be one of the following:

- *sigclr* – set the signal clear;
- *sigstp* – set the signal to stop;
- *sigblk* – set the signal blocked;
- *sigubk* – unblock the signal;
- *swrev* – set the switch to reverse;
- *swnor* – set the switch to normal;
- *swblk* – block switch;
- *swubk* – unblock switch;
- *tkblk* – block track;
- *tkubk* – unblock track.

There may be checks required before a command can be accepted, for example a *sigclr* command can only be obeyed if the route associated with the signal is unoccupied, not blocked, none of its locations are used in another occupied or reserved route, none of the switches in the route are blocked, and they are all in a reverse or normal state, and the signal is not blocked. Alarms are raised if an invalid command is issued.

Figure 3 shows the abstract analysis model of the system as a UML class diagram. The annotation ? on an attribute indicates that it corresponds to a sensor input, the annotation ! indicates an actuator output. Other attributes are internal data defined by the software control system to record properties of the network.

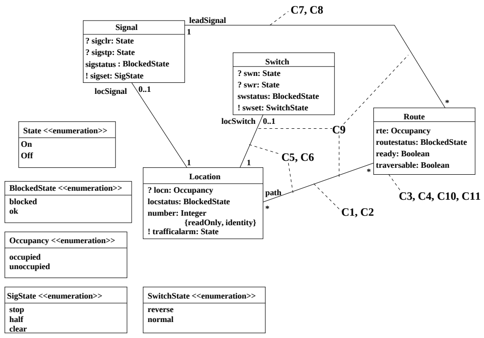

Fig. 3: Abstract Analysis Class Diagram

The safety invariants of the system can be expressed as constraints on the class diagram, as follows. As in (King, 1995), we separate general rules that apply to all networks, from specific rules defining the topology of a particular network.

$C1$ "If any location in a route is occupied, the route is occupied":

$$\begin{aligned}&\mathit{location} = \mathit{occupied} \;\Rightarrow\\ &\qquad \mathit{route} = \mathit{occupied}\end{aligned}$$

as a constraint on the association between *Route* and *Location*. This means

$$\begin{aligned}&\forall\, l : \mathit{Location};\; r : \mathit{Route} \cdot\\ &\quad (r, l) \in \mathit{Route\_Location} \;\wedge\\ &\quad l.\mathit{location} = \mathit{occupied} \;\Rightarrow\\ &\qquad r.\mathit{route} = \mathit{occupied}\end{aligned}$$

where *Route_Location* is the relation between *Location* and *Route*. We indicate on Figure 3 the associations that each of the following constraints applies to.

The inverse rule is a normal OCL invariant of *Route*:

$$\begin{aligned}&\mathit{path}.\mathit{location} = \mathit{unoccupied} \;\Rightarrow\\ &\qquad \mathit{route} = \mathit{unoccupied}\end{aligned}$$

$C2$ "If any location in a route is blocked, so is the route":

$$\begin{aligned}&\mathit{locstatus} = \mathit{blocked} \;\Rightarrow\\ &\qquad \mathit{routestatus} = \mathit{blocked}\end{aligned}$$

Similarly if any signal in the route is blocked, or if any switch in the route is blocked. The inverse rule is:

$$\begin{aligned}&\mathit{path}.\mathit{locstatus} = \mathit{ok}\ \&\\ &\mathit{path}.\mathit{locSwitch}.\mathit{swstatus} = \mathit{ok}\ \&\\ &\mathit{path}.\mathit{locSignal}.\mathit{sigstatus} = \mathit{ok} \;\Rightarrow\\ &\qquad \mathit{routestatus} = \mathit{ok}\end{aligned}$$

$C3$ "If a route is not ready, then it is not traversable":

$$\mathit{ready} = \mathit{false} \;\Rightarrow\; \mathit{traversable} = \mathit{false}$$

$C4$ "The lead signal of a route is actually in the route":

$$\mathit{leadSignal} : \mathit{path}.\mathit{locSignal}$$

$C5$ "A route is unready if some switch is not set to either normal or reverse":

$$\begin{aligned}&\mathit{swr} = \mathit{Off}\ \&\ \mathit{swn} = \mathit{Off} \;\Rightarrow\\ &\qquad \mathit{ready} = \mathit{false}\end{aligned}$$

Likewise if a switch is changing from its current position:

$$\begin{aligned}&\mathit{swr} = \mathit{On}\ \&\\ &\mathit{swset} = \mathit{normal} \;\Rightarrow\\ &\qquad \mathit{ready} = \mathit{false}\\ &\mathit{swn} = \mathit{On}\ \&\\ &\mathit{swset} = \mathit{reverse} \;\Rightarrow\\ &\qquad \mathit{ready} = \mathit{false}\end{aligned}$$

In these cases the lead signal is set to *stop*:

$$\begin{aligned}&\mathit{ready} = \mathit{false} \;\Rightarrow\\ &\qquad \mathit{leadSignal}.\mathit{sigset} = \mathit{stop}\end{aligned}$$

$C6$ The network-specific rules for when a route is ready depend on the particular track and switch configuration, for example in Figure 2, route $R1$ is ready if switch $SW1$ is set to reverse:

$$\begin{aligned}&SW1.\mathit{swn} = \mathit{Off}\ \&\\ &SW1.\mathit{swr} = \mathit{On}\ \&\\ &SW1.\mathit{swset} = \mathit{reverse} \;\Rightarrow\\ &\qquad R1.\mathit{ready} = \mathit{true}\end{aligned}$$

Similarly there are specific rules for *traversable*.

Constants (written in capitals) represent particular constant objects.

$C7$ "If a route is occupied, its signal must be set to stop":

$$\begin{aligned}&\mathit{route} = \mathit{occupied} \;\Rightarrow\\ &\qquad \mathit{sigset} = \mathit{stop}\end{aligned}$$

$C8$ "If a route is blocked, its signal must be set to stop":

$$\begin{aligned}&\mathit{routestatus} = \mathit{blocked} \;\Rightarrow\\ &\qquad \mathit{sigset} = \mathit{stop}\end{aligned}$$

$C9$ "If a route contains some switch set to reverse, then its lead signal should be set to stop or half speed":

$$\mathit{swr} = \mathit{On} \;\Rightarrow\; \mathit{sigset} \neq \mathit{clear}$$

$C10$: "If two routes share the same lead signal, and one is traversable, the other must be untraversable":

$$\begin{aligned}&r_1 \neq r_2\ \&\ r_1.\mathit{traversable} = \mathit{true}\ \&\\ &r_1.\mathit{leadSignal} = r_2.\mathit{leadSignal} \;\Rightarrow\\ &\qquad r_2.\mathit{traversable} = \mathit{false}\end{aligned}$$

This is a constraint on the forms of network allowed – it should never be the case that two alternative routes are both navigable from a given signal, since the settings of switches will define a unique traversable

route. The network of Figure 2 has this property.
$C11$: "If a route is occupied, the lead signal of any route that intersects it must be stop":

$$l : r1.path \ \& \ l : r2.path \ \& \\ r1.route = occupied \ \Rightarrow \\ r2.leadSignal.sigset = stop$$

An example of a temporal constraint could be that if a switch is commanded to move to the normal position then it will eventually reach that position:

$$swset = normal \ \Rightarrow \ AF(swn = On)$$

$AF(P)$ means that P is true in the current state or in some successor state.

The effect of operator commands are as follows:

- *sigstp*:

  $$sigstp \ \& \ sigstatus = ok \ \Rightarrow \\ AX(sigset = stop)$$

  $AX(P)$ means that P holds in all immediate successor states.

- *swnor*:

  $$swnor \ \& \ route = unoccupied \ \& \\ swstatus = ok \ \& \\ routestatus = ok \ \Rightarrow \\ AX(swset = normal)$$

- *swrev*:

  $$swrev \ \& \ route = unoccupied \ \& \\ swstatus = ok \ \& \\ routestatus = ok \ \Rightarrow \\ AX(swset = reverse)$$

- The blocking and unblocking commands have no validation checks, and simply modify the values of the *locstatus*, *swstatus* and *sigstatus* variables as appropriate (and therefore also possibly the values of *routestatus* and the state of signals).

An error message is given if a command fails its checks.

The following are environmental constraints whose failure will trigger the creation of a traffic alarm for the location concerned, using the RSDS fault detection architecture (Lano *et al*, 2002).

- A switch is failed if its *swn* and *swr* sensors are both set:

  $$swn = On \ \Rightarrow \ swr = Off$$

- Similarly for signals:

  $$sigclr = On \ \Rightarrow \ sigstp = Off$$

- A location should not become occupied if it is blocked:

  $$locationoccupied \ \Rightarrow \\ locstatus = ok$$

- A train should only enter a route at the lead signal location:

  $$locationoccupied \ \& \\ route = unoccupied \ \Rightarrow \\ locSignal = leadSignal$$

- A train must only enter a route when the lead signal is clear or half:

  $$locationoccupied \ \& \\ route = unoccupied \ \Rightarrow \\ leadSignal.sigclr = On \ \& \\ leadSignal.sigset \neq stop \\ locationoccupied \ \& \\ route = unoccupied \ \& \\ swr = On \ \Rightarrow \\ leadSignal.sigclr = On \ \& \\ leadSignal.sigset = half$$

4. VALIDATION AND VERIFICATION

The RSDS tools carry out extensive completeness and consistency checks on invariants: to determine if each sensor and actuator, and each sensor and actuator state, appears in some invariant, and that all logical combinations of cases are considered. Consistency checks identify for each pair of invariants $A \Rightarrow B$ and $C \Rightarrow D$ if B and D are contradictory whilst A and C are not.

Derived constraints are deduced from those supplied by the developer by means of contraposition and transitivity of implication laws. For example

$$traversable = true \ \Rightarrow \ ready = true$$

can be deduced from $C3$.

The B Toolkit (B Core, 2002) can also be used to check the consistency of the invariants and the specified component behaviours, ie, that the intended behaviour of the system does satisfy the invariants and therefore that these invariants are non-contradictory. In addition, B supports animation.

The class diagram of Figure 3 can be directly translated into a B specification that has a corresponding modular structure. This translation is systematic and automated by the RSDS tools.

The expanded versions of the UML constraints become invariants and control rules in the B code.

The B for locations is:

```
MACHINE Location
SEES SystemTypes
USES Signal, Switch
VARIABLES locations, location,
  locstatus, locSwitch, locSignal
INVARIANT
  locations <: LOCATION_OBJ &
  location: locations --> OCCUPANCY &
  locstatus: locations --> STATUS &
  locSwitch: locations --> FIN(switchs) &
  locSignal: locations --> FIN(signals)
INITIALISATION
  locations := {} || location := {} ||
  locstatus := {} || locSwitch := {}
  || locSignal := {}
OPERATIONS
  setlocation(oo,val) =
    PRE oo: locations &
        val: OCCUPANCY
    THEN location(oo) := val
    END;
  ...
END
```

Each class in the system is translated into a corresponding B machine, each of these is then included in a *Controller* machine, which coordinates the reaction of objects to events. For example, the *locationoccupied* event has code derived from the invariants which are affected by this event. In particular the invariant that if a location is occupied, all the routes it belongs to must also be occupied, is expressed in B by:

```
!ll.(ll: locations =>
  !rr.(rr: routes &
       ll: path(rr) =>
    (location(ll) = occupied  =>
       route(rr) = occupied)))
```

Similarly the invariant that all occupied routes must have a *stop* lead signal can be expressed in B. The resulting controller code generated to maintain these invariants is:

```
locationoccupied(oo) =
  PRE oo: locations
  THEN
    setlocation(oo,occupied) ||
    setAllroute(
      { rr | rr: routes &
           oo: path(rr) },
      occupied) ||
    setAllsigset(
      leadSignal[{ rr | rr: routes &
                        oo: path(rr) }],
      stop)
  END;
  ...
END
```

The second update is derived from $C1$, the third from $C7$.

Verification of temporal properties of a system can be achieved by translation to SMV (Burch *et al*, 1990). This translation provides an alternative proof capability, and the ability to explicitly exhibit counter-example scenarios to expected properties, and therefore assists in providing more precision to hazard analysis of the software. Details are in (Lano *et al*, 2003).

The RSDS, B and SMV semantics are very closely related, representing a model or execution of a specification as a sequence of time steps, each step consisting of no events or of a single sensor event and a set of other events (in reaction to the sensor event). Thus the results of analysis in B or SMV can be immediately related to the UML model.

Finally, a translation to Java provides an executable version of the system, for testing. The translation of RSDS/UML constraints into code is direct. For example, constraints $C1$ and $C7$ yield the (automatically generated) code:

```
public static void
  locationoccupied(Location oo)
{ oo.setlocation(occupied);
  for (int i0 = 0; i0 < routes.size(); i0++)
  { Route routex =
       (Route) routes.get(i0);
    if (routex.getpath() != null &&
        routex.getpath().contains(oo))
    { routex.setroute(occupied);
      Signal leadSignalx =
        routex.getleadSignal();
      leadSignalx.setsigset(stop);
    }
  }
}
```

The Java translation is described in (Lano *et al*, 2003).

5. COMPARISON

Related work on UML is the U2B tool of (Snook *et al*, 2003), and translations (Kim and Carrington, 2000) from UML to Object-Z (Smith, 2000). These translations emphasise the preservation of the UML class diagram structure in the formal notation, as the starting point of a formal development. In both cases the formal notation is central and the UML used only as a diagrammatic front end. In contrast, we envisage that most developers would prefer to use UML as their main notation, and only use B as one analysis tool for property checking and animation of the class diagram. The exact structure of formal modules is therefore less important than the traceability of formal notation elements to the original diagram elements. We have ensured such a traceability is possible by representing each class, attribute and role by a separate variable and each method by a B operation.

RSDS is similar in intent to pragmatic formal approaches such as SCR (Bharadwaj and Heitmeyer, 1999). However, instead of inventing a new diagrammatic formal method, we make precise an existing widely-used semi-formal method.

6. CONCLUSION

We have shown that it is possible to use a relatively simple subset of UML to specify complex railway systems in a precise manner, and to provide verification and analysis capabilities for such specifications. Future work is intended to extend the forms of constraints that are supported.

REFERENCES

B Core UK Ltd. (2002) *The B Tookit.*

Bharadwaj, R. and C. Heitmeyer (1999) *Model checking complete requirements specifications using abstraction.* In *Proceedings of Automated Software Engineering*, **6**, 37–68.

Booch, G. (2002) *Growing the UML*, Software and Systems Modelling, 1(2), 157–160.

CENELEC *EN 50128:Railway Applications – Software for Railway Control and Protection Systems.*

CSRR (1994) *CS-RR Software User Requirements Document.*

Dehbonei, B. and F. Mejia (1995) *Formal Development of Safety-Critical Software Systems in Railway Signalling*, Applications of Formal Methods, Series in Computer Science, pp. 227–252, Prentice Hall International.

J R Burch, E M Clarke, et al (1990) *Symbolic Model Checking:* 10^{20} *States and Beyond*, Proceedings of the Fifth Annual Symposium on Logic in Computer Science.

A. Evans, R. France, K. Lano, B. Rumpe (1998) *The UML as a Formal Modelling Notation*, in UML '98, LNCS Vol. 1618, pp. 336–348.

M. Huber and T. King (2002) *Towards an Integrated Model Checker for Railway Signalling Data*, FME 2002, Springer LNCS.

IEC (2000) *IEC 61508 – Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems.*

S. Kim and D. Carrington (2000) *A Formal Mapping Between UML Models and Object-Z Specifications*, in ZB2000, LNCS Vol. 1878, Springer-Verlag.

T. King (1995) *Formalising British Rail's Signalling Rules*, Praxis Plc, Bath.

K. Lano (1996) *The B Language and Method*, Springer-Verlag, FACIT series, ISBN 3-540-76033-4.

K. Lano, J. Fiadeiro, L. Andrade (2002) *Software Design using Java 2*, MacMillen.

K. Lano, D. Clark, K. Androutsopoulos (2002) *Safety and Security Analysis of Object-oriented Models*, Safecomp 2002.

K. Lano, D. Clark, K. Androutsopoulos (2003) *Synthesis of Code from UML Specifications*, DCS, King's College.

OMG (2002) *UML OCL Specification*, http://www.omg.org/uml/.

G. Smith (2000) *The Object-Z Specification Language*, Kluwer, 2000.

C. Snook, P. Wheeler, M. Butler (2003) *Preliminary Tool Extensions for Integration of UML and B*, IST-2000-30103 deliverable D4.1.2.

H. Treharne (2002) *Supplementing a UML Development Process with B*, FME '02.