PERIODICA POLYTECHNICA SER. TRANSP. ENG. VOL. 28, NO. 1–2, PP. 39–53 (2000)

USING *Z* SPECIFICATION FOR RAILWAY INTERLOCKING SAFETY[1]

Aleš JANOTA

Department of Information and Safety Systems
Faculty of Electrical Engineering
University of Žilina, Vel'ký diel, Žilina 010 26, Slovak Republic
Fax: +421-89-5252241
e-mail: janot@fel.utc.sk
Phone: +421-89-5133261

Received: September 29, 2000

Abstract

Formal methods involve a mature development technology that can be used to provide the highest confidence and that is used in a wide and expanding variety of environments, especially in key areas where the integrity of systems is critical or where there is a high intensity of use. Formal methods allow the logical properties of a computer system to be predicted from a mathematical model of the system by means of a logical calculation, which is a process analogous to numerical calculation. They make it possible to calculate whether certain properties are consequences of proposed requirements or whether requirements have been interpreted correctly in the derivation of a design. The objective of this paper is to discuss the possible use of formal methods in the field of requirement specification of the railway interlocking system behaviour, followed by the check of its correctness. A simple example is used to demonstrate the use of concrete formal description (Z notation). Then the principles of realisation of more complex solutions in the field of railway interlocking are indicated. Despite the specific contents of the paper concerning the railway applications the conclusions can be generalised and applied to other areas.

Keywords: formal method, *Z* notation, specification, interlocking.

1. Introduction

It is generally acknowledged that natural languages and similar informal notations have many disadvantages when used for writing technical descriptions – it is difficult to write specifications with the required precision, clarity and economy of expression and transform them systematically and reliably into a code or hardware. Furthermore, it is impossible to carry out formal mathematical reasoning about informally written descriptions.

Formal methods provide a framework within which people can specify, develop and verify systems in a systematic rather than ad hoc manner. They give us techniques to formally specify the functionality of a system, to verify its correctness

[1]This paper has been supported by the VEGA Grant Agency of the Slovak Republic, within the project 1/5230/98 'Theoretical Apparatus for Safety Analysis of the System with Defined Level of Safety', at the University of Žilina.

or to develop the system gradually from an abstract specification to its implementation. All these aspects are important when designing safety-critical systems.

2. Formal Methods

There are a lot of definitions for formal methods. For the purpose of this paper let us assume a definition given in (HEIMDAHL et al, 1998) and (NASA, 1998) – a method consisting of a set of techniques and tools based on mathematical modelling and formal logic that are used to specify and verify requirements and designs for computer systems and software. Such a method must provide:

- a notation with a well-defined syntax and semantics,
- some guideline and procedures for using the notation, and
- techniques for analysing specifications expressed in the notation.

Formal methods allow defects in requirements to be detected earlier than otherwise and can greatly reduce the incidence of mistakes in interpreting, formalising and implementing correct requirements. They can be used to mathematically prove that certain properties (functionality, safety etc.) hold and can be potentially applicable in all engineering disciplines, mainly for the reason that the use of formal descriptions permits machine processing of specifications. Formal methods may be classified according to different criteria: according to whether their primary purpose is descriptive or analytic (descriptive and analytic methods), according to the level of formality (with low, medium and high level of formality) and/or according to the type of the used specification language (algebraic and model-oriented).

2.1. Formal Specification and Verification

Traditionally, formal methods have been used for *formal specification and formal verification. A formal specification supports formal reasoning* which (because it can be checked by machine) can be made very reliable indeed (with reduction of human involvement) and thus it enables formal verification. In formal verification, a proof is constructed, often with mechanical support, that the specification satisfies properties of interest. Formal proofs, though, are usually very detailed. Even with machine support for their construction, formal proofs are liable to require considerable efforts to construct and check. Formal methods analysis techniques are based on deductive rather than inductive reasoning about system descriptions. These techniques include writing formal specifications, internal checking (e.g. parsing and type correctness), traceability checking, specification animation, and proof of assertions. A formal specification language typically gives a mathematical basis of a formal method. The distinction between a specification method and a specification language is fundamental. A method states what a specification must say. A language determines in detail how the concepts in a specification can be expressed. The objective of a

formal specification notation is to assist in the production of descriptions that are complete, consistent and unambiguous.

2.2. Use, Applications and Standards

Formal methods are used in computer science mainly for improving the quality of software or hardware or for improving confidence in the correctness of critical systems. At present they are mostly used in applications of safety critical systems, security systems, the definition of standards, hardware development, operating systems, transaction processing systems and others. For information on domains of formal method applications, see e.g. (NICHOLLS, 1992) or follow relevant discussion groups and computer science bibliographies on Internet. Due to the widening range of applications the use of formal methods is increasingly recommended by many standards (including safety ones) as a possible method of improving dependability. Highlighting the domain of railways only, the railway application standard (CENELEC, 1996) recommends formal methods (*CCS*, *CSP*, *HOL*, *LOTOS*, *OBJ*, *Temporal Logic*, *VDM* and *Z*) for development of software in a way that is based on mathematics (this includes formal design and formal coding techniques) and their use for writing specifications and for verifying the safety properties of them. Similarly, relevant standards could be identified in other application domains, too. Despite the given recommendations, there are several barriers that must be overcome of putting the formal techniques in the hands of formal developers: the notations must be more accessible and easy to use, robust tools must be produced, education must be improved and cost-effectiveness must be demonstrated. In addition, formal specification languages and their semantics are also to be standardised.

3. *Z* Notation

The *Z* notation is one of the best known formal methods which is often declared as a formal specification language and which is gaining widespread acceptance as a useful means of specifying software systems. The *Z* notation is a descriptive method with a medium level of formality. It means that the method can be used together with computer-aided tools. In the sense of a specification language it is model oriented, based upon the set theory and mathematical logic, i.e. it is based on model design which is given by description of the system state and operations over this state. The representation, structure and meaning of the formal part of specification, written in the *Z* notation, is defined in the draft standard (ISO, 1995). In the *Z* notation there are two languages: the mathematical language and the schema language. The former is used to describe different aspects of design: objects and the relationships between them. The latter is used to structure and compose descriptions: collating pieces of information, encapsulating them and naming them for re-use (WOODCOCK and DAVIES, 1996).

3.1. Aspects of Z Notation

The *Z* notation is based upon set theory and mathematical logic (first-order predicate calculus). Together, they make up a mathematical language that is easy to learn and apply. However, the language is only one aspect of *Z*. Another aspect is the way in which the mathematics can be structured. Mathematical objects and their properties can be collected together in schemas: patterns of declaration and constraint. The schema language can be used to describe system properties. A characteristic feature of *Z* is the use of types. Every object in the mathematical language has a unique type, represented as a maximal set in the current specification. This notion of types means that an algorithm can be written to check the type of every object in a specification; several type-checking tools exist to support the practical use of *Z* (e.g. $Z/EVES$, *Z Type Checker* etc.). The third aspect is the use of natural language that helps to relate mathematics to objects in the real world (partly by the judicious naming of variables, partly by additional commentary). A well-written specification should be perfectly obvious to the reader. The fourth aspect is refinement. We may develop a system by constructing a model of a design and then refine the description by constructing another model which respects the design decisions made, and yet is closer to implementation. Where appropriate, this process of refinement can be continued until an executable code is produced. Thanks to given aspects the *Z* notation is a mathematical language with a powerful structuring mechanism.

3.2. Trivial Application Example

For the sake of better understanding, a trivial example of *Z* specification is presented, based on (JANOTA, 2000), to illustrate a typical process of formal specification design:

- Giving an informal problem statement of a system: this step consists in informal specifying the system requirements, using informal means of expression (natural language, diagrams, graphs, tables etc.).
- Making *Z* style of presentation: *Z* specification consists of many small sections, each beginning with informal descriptions followed by formal texts. Natural order for easy understanding is from top down, from general to detail, from common to exception and avoids forward referencing. If necessary, larger entities can be built. *Z* adopts the state machine view of systems (*Fig. 1*).
- Writing *Z* specification using a *Z*-based tool: in this step the *Z* specification is transformed into the required computer-based format.
- Performing tool-supported activities: a variety of activities (pretty-printing, editing and browsing, type-checking, consistency checking, proof-obligation generation, theorem proving, code generation) can be available, based on the tool to be used.

3.2.1. Informal Problem Statement

The task consists in developing a system (hardware or software) whose behaviour makes substitution of electro-mechanical relay possible and whose specification is the task. There are different ways of how to describe a system behaviour informally – using verbal descriptions, specifying a voltage dependency that represents the hysteresis characteristic of relay operation, writing mathematical equations, making logical decision tables etc.

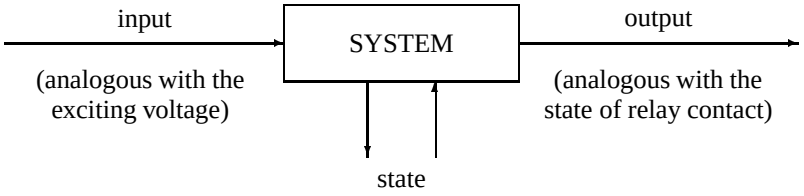

Fig. 1. State machine view to the system substituting an electro-mechanical relay

Behaviour of the system in *Fig. 1* can be described verbally:

a) If the (internal) state of the system is analogous with the relay drop-away, the imaginary relay pick-up will occur if the input voltage reaches the value equivalent to or higher than the nominal (just-operate) pick-up voltage (U_{pn});

b) If the internal state of the system is analogous with the relay pick-up, the imaginary drop-away will occur if the input voltage falls down to the value equivalent to or lower than the nominal drop-away voltage (U_{on}).

For the particular relay type, the value can be specified e.g. $U_{pn} = 15$ V and $U_{on} = 10$ V with the input voltage range 0 to 48 V.

3.2.2. Formal Z Specification

In this section a way of *Z* specification design is presented with brief explanation of particular parts. More details about *Z* syntax and semantics are available e.g. in (WOODCOCK and DAVIES, 1996) or (ISO, 1995).

$$States ::= pick\_up / drop\_away \tag{1}$$

Eq. (1) is called a free-type definition, which introduces a set of all possible states of the system containing two distinct elements. The order in which these elements are introduced is unimportant.

$$\begin{array}{|l} voltage, U_{pn}, U_{on} : \mathbb{Z} \\ \hline U_{pn} = 15 \\ U_{on} = 10 \end{array} \tag{2}$$

Axiomatic definition (2) introduces new symbols in its declarative (upper) part as elements of the set $\mathbb{Z}$. In the lower (predicate) part there are constraints – in our case the nominal voltage values. Such a definition is said to be axiomatic, as the constraint is assumed to hold whenever the symbol is used: it is an axiom for the object.

$$Response ::= close/open \tag{3}$$

Eq. (3) gives a free-type definition of the system response. Let's suppose a response analogous with the behaviour of the relay contact system (contacts *closed* or *open*).

$$\begin{array}{|l} \textit{System} \\ \hline state: States \\ voltage: \mathbb{Z} \\ \hline voltage \in 0..48 \end{array} \tag{4}$$

The system state is described by the so-called vertical definition of the state schema (4). The schema consists of the following parts: the name of schema (*System*), declaration (expressed as *name: type*) and constraints. The state schema defines all the allowable or valid system states (sometimes called cases). The system state is allowed if all the conditions in the axiomatic part are satisfied. As an example let's assume a particular situation where the system is in the state close and the *voltage* is 30 V.

$$\begin{array}{|l} \textit{InitSystem} \\ \hline System \\ \hline state = drop\_away \\ voltage = 0 \end{array} \tag{5}$$

The schema (5) says how the system should be initiated. The internal state is defined as drop away together with the zero input voltage. After initiation we specify the normal situations first. Every operation is specified using an *operational schema*.

PickUp

$\Delta System$
$voltage? : \mathbb{Z}$
$response! : Response$

$state = drop\_away$
$voltage? \geq U_{pn}$
$state' = pick\_up$
$response! = close$

(6)

Operation (6) is analogous with the relay pick-up. According to Z conventions, $\Delta System$ indicates that the operation *PickUp* may modify the state. The name *state* refers to the state before the operation (called the *pre-state*), the dashed name *state'* refers to the state after the operation (called the *post-state*). Suffixes '?' and '!' indicate an input name and an output name, respectively. The schema (7) describes the situation analogous with the relay holding ($\Xi System$ indicates that the system state is unchanged) where its output remains *close*.

Holding

$\Xi System$
$voltage? : \mathbb{Z}$
$response! : Response$

$state = pick\_up$
$voltage? > U_{on}$
$state' = state$
$response! = close$

(7)

DropAway

$\Delta System$
$voltage? : \mathbb{Z}$
$response! : Response$

$state = pick\_up$
$voltage? \leq U_{on}$
$state' = drop\_away$
$response! = open$

(8)

NoPickUp

$\Xi System$
$voltage? : \mathbb{Z}$
$response! : Response$

$state = drop\_away$
$voltage? < U_{pn}$
$state' = state$
$response! = open$

(9)

Schemas (8) and (9) describe other situations. Finally, there are so-called *horizontal definitions* (10) and (11). Their names correspond to states analogous with states of the relay contact system.

$$Close \stackrel{\wedge}{=} PickUp \vee Holding \tag{10}$$

$$Open \stackrel{\wedge}{=} NoPickUp \vee DropAway \tag{11}$$

Let's consider the specification definitive though it could be developed and refined.

3.2.3. Machine Processing of the Specification

The computer-based format of the specification is given by requirements of the tool used to check syntax correctness (alphabet of symbols and grammar rules

which define well-formed formulations) and semantics (mathematical proof of the contents correctness). A particular semantic construction is a model of a particular *Z* specification (or predicate) if the relationship between terms in the specification also holds in the model (if it makes the predicate true). A predicate is logically valid if it holds in all models. It is logically valid relative to a specification if it holds in all models of that specification. Logic for *Z* is a collection of axioms and rules of inference written using the *Z* notation. These rules and axioms may be used inductively to define which predicates are logical consequences of others. Those, which are the logical consequence of the empty set of predicates (or, equivalently, the logical consequence of only the axioms of the logic), are known as theorems. A logic for Z may be proven sound in some model by showing that each of its theorems is logically valid. It is traditional to ask whether a logic is also complete, i. e. whether every logically valid predicate is also a theorem. More details on distinguishing between the different philosophies of proof in *Z* are available in (MARTIN, 1999).

All the specifications presented in this paper were transformed to the format required by the software package *Z/EVES ver. 2.1* by *ORA Canada*, running under *MS Windows 98*. The tool uses its own file format (∗*.zev*, ∗*.zcs*) or can import the files in LaTeX2e format (∗*.tex*). *Z/EVES* has two parts: a server that checks paragraphs and executes proof commands, and a GUI that manages specifications, sends commands to the server, and displays results. An example display of the specification window is given in *Fig. 2* and relates to the contents of Section 4.

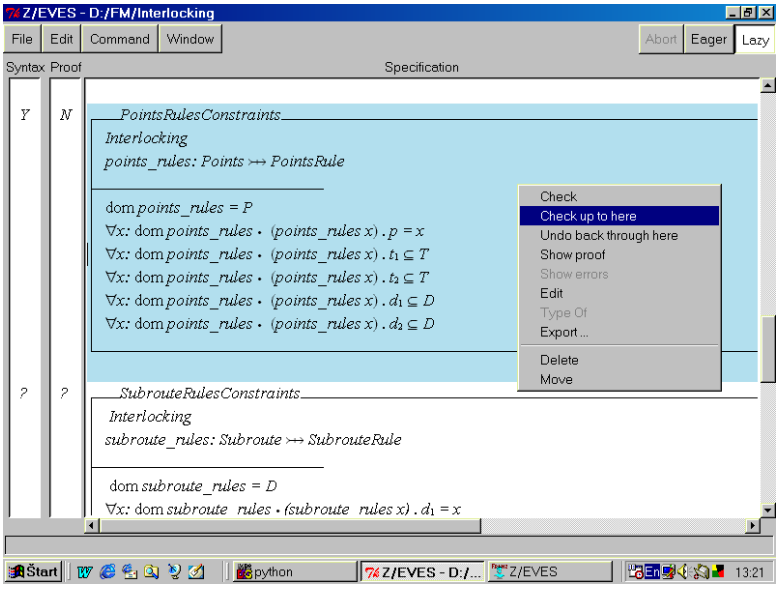

Fig. 2. Example of a *Z/EVES* specification window

The status of each paragraph is shown in two columns to the left of the paragraph. The leftmost column shows one of three symbols:

- '?' indicates that the paragraph has not been checked,
- 'Y' indicates that the paragraph has been checked and has no syntax or type errors,
- 'N' indicates that the paragraph has been checked and has errors.

The next column shows the proof status for the paragraph, using one of three symbols:

- '?' indicates that the paragraph has not been successfully checked (so proof status cannot be determined),
- 'N' indicates that the paragraph has unproved goals,
- 'Y' indicates that the paragraph has no unproved goals.

4. The Use of Formal Specification in Railway Interlocking

Modern computer interlocking systems typically consist of hardware and software where the software generally consists of:

- Generic programmes, common for different installations,
- Geographic programmes, considering local data.

Correctness of requirement specification must be assured in order to achieve functional safety of the interlocking system. Due to its high complexity it is apparent that verification of functional dependencies is an extremely complex task. Since the internal state has extremely many possible configurations there is an effort to perform the automatic analysis of the safety features of programmes with local (geographic) data. This task can be solved with the help of formal specification and verification methods.

4.1. Starting Points of Formal Analysis

Following the methodology introduced in the previous section the first step consists in informal problem definition. Any railway authority defines own regulations and standards concerning the rules for railway vehicle movement. Under conditions of ŽSR (Slovak National Railways) the relevant requirements are given in the form of government and technical standards (mainly of the series TNŽ 34 26xx) and other relevant regulations. Developing a concrete application these rules are applied to the given railway topology. This results in the table form containing all the possible operational situations. From our viewpoint this can be considered to be an informal requirements specification for the interlocking under specific local conditions. Not all railways use such table forms, some substitute them with alternative descriptions

of possible situations through rules or logical equations for route settings (with final document called User Case). However, the final specification is a product of manual activity of the staff and is subjected to strict validation and certification process. This process may be automated with the use of formal methods.

4.2. State of an Interlocking System

The state of an interlocking system is generally defined by combination of particular states of its components. These include mainly the following (physical) objects:

- Track circuits, dividing the track into sections and detecting their states (occupied or clear),
- Points, steering trains across junctions and finding themselves in one of defined positions (controlled plus, controlled minus or undefined),
- Signals, allowing or refusing the entry of the railway vehicle onto particular sections of the track and situated in advance of the section which they control.

Apart from the previous (physical) objects there are also so-called logical objects:

- Routes as sections of track between two signals, which proceed from an entry signal to an exit signal (the route set or unset),
- Sub-routes as subsections of routes that are associated with specific track circuits. The concept of the route as a set of sub-routes is typical for several computer interlockings. The sub-route can be locked or free.

The process of defining the local dependencies (based on geographic railway topology) most often concentrates on the following invariant conditions (SIMPSON, 1998):

- For every track circuit, no more than one of the sub-routes passing over it should be locked for a route at any time,
- If a sub-route over a track section containing points is locked for a route, then the points are correctly aligned with that sub-route,
- If a route is set, then all of its sub-routes are locked,
- If a track circuit containing points is occupied, than the points are locked,
- If a sub-route is locked for a route, then all sub-routes ahead of it on that route are also locked.

If any of these invariants are not satisfied, then there is an error in the geographic data. The task defined in the way above enables to automate the process of relevant data model design (geographic database) and verify its safety.

4.3. Concept of Z Representation

Within the first step the sets of particular components are defined. In the *Z* notation, there are several ways of defining an object – declaration, abbreviation, generic abbreviation, axiomatic definition and generic definition (WOODCOCK and DAVIES, 1996). We use the first way, declaration, to define new basic types, representing components of the interlocking:

$$[TCircuit, Points, Signal, Route, Subroute] \tag{12}$$

The schema *Components* describes the sets of track circuits, points, signals, routes and sub-routes with which a particular interlocking is concerned.

Components

$$\begin{aligned} T &: \mathbb{P}\ TCircuit \\ P &: \mathbb{P}\ Points \\ S &: \mathbb{P}\ Signal \\ R &: \mathbb{P}\ Route \\ D &: \mathbb{P}\ Subroute \end{aligned} \tag{13}$$

Relationship among different components of an interlocking is specified by the schema (14).

Structure

$$\begin{aligned} &locks: Route \rightarrow\!\!\!\!\!\!+\ \mathbb{P}\ Points \\ &sroute: TCircuit \rightarrowtail\!\!\!\!\!\!+\ \mathbb{P}\ Subroute \\ &berth: Points \rightarrow\!\!\!\!\!\!+\ \mathbb{P}\ TCircuit \\ &tcircuit: Subroute \rightarrowtail\!\!\!\!\!\!+\ TCircuit \\ &sroutes: Route \rightarrowtail\!\!\!\!\!\!+\ \mathrm{seq}\ Subroute \\ &plus: Points \rightarrow\!\!\!\!\!\!+\ \mathbb{P}\ Subroute \\ &minus: Points \rightarrow\!\!\!\!\!\!+\ \mathbb{P}\ Subroute \end{aligned} \tag{14}$$

Components and *Structure* together describe the layout of an interlocking. Points can be in one of two positions: controlled plus (*cp*) or controlled minus (*cn*). Therefore, we introduce the free type *points_position* (16). A point can be controlled free to go plus (cfp) or controlled free to go minus (cfm) or both at any time. Thus, the free type *points_state* is introduced by (17).

The free type *tcircuit_state*, *route_state* and *sroute_state* are introduced similarly. Track sections, with train detection devices, inform the interlocking if a

section is occupied (o) or clear (c), (18). A route may be in one of two states (19): set (s) or unset (us). Sub-routes are subsections of routes, which are associated with specific track circuits. They may be in one of two states (20): locked (l) or free (f).

Interlocking

Components

Structure

$$\begin{aligned}
&\text{dom } \mathit{locks} = R \land \cup (\text{ran } \mathit{locks}) \subseteq P \\
&\text{dom } \mathit{sroute} = T \land \cup (\text{ran } \mathit{sroute}) \subseteq D \\
&\text{dom } \mathit{berth} = P \land \cup (\text{ran } \mathit{berth}) \subseteq T \\
&\text{dom } \mathit{tcircuit} = D \land \text{ran } \mathit{tcircuit} = T \\
&\text{dom } \mathit{sroutes} = R \land \text{ran } (\cup (\text{ran } \mathit{sroutes})) \subseteq D \\
&\text{dom } \mathit{plus} = P \land \cup (\text{ran } \mathit{plus}) \subseteq D \\
&\text{dom } \mathit{minus} = P \land \cup (\text{ran } \mathit{minus}) \subseteq D
\end{aligned} \tag{15}$$

$$\mathit{points\_position} ::= cp \mid cm \tag{16}$$

$$\mathit{points\_state} ::= cfp \mid cfm \tag{17}$$

$$\mathit{tcircuit\_state} ::= c \mid o \tag{18}$$

$$\mathit{route\_state} ::= s \mid us \tag{19}$$

$$\mathit{sroute\_state} ::= f \mid l \tag{20}$$

To formalise the structure of a route setting rule, a number of entities must be concerned: the route which is to be set (r), a set of conditions on points which must be satisfied before the route can be set (p_1), a set of point movements form part of the setting process for the route (p_2), a set of sub-routes which must be free before the route can be set (d_1), and a set of sub-routes which must be locked when the route is set (d_2). Then the route setting rule can be specified (21).

RouteRule

$$\begin{aligned}
&r : \mathit{Route} \\
&p_1 : \mathbb{P}\ (\mathit{Points} \times \mathit{points_state}) \\
&p_2 : \mathbb{P}\ (\mathit{Points} \times \mathit{points_position}) \\
&d_1 : \mathbb{P}\ \mathit{Subroute} \\
&d_2 : \mathbb{P}\ \mathit{Subroute}
\end{aligned} \tag{21}$$

The schema (21) describes a sub-route release rule. This involves a set of track circuits which are required to be clear, a (possibly empty) set of sub-routes which are required to be free, and a (possibly empty) set of routes which are required to be unset.

Finally, we can also represent the rule for points (23) where p is the point with which we are concerned, t_1 is the set of track circuits associated with the free to go plus condition, t_2 is the set of track circuits associated with the free to minus position, d_1 is the set of sub-routes associated with the free to go plus condition, and d_2 is the set of sub-routes associated with the free to go minus condition.

SubrouteRule

$d_1 : Subroute$
$t : \mathbb{P}\ TCircuit$
$r : \mathbb{P}\ Route$
$d_2 : \mathbb{P}\ Subroute$

(22)

PointsRule

$p : Points$
$t_1 : \mathbb{P}\ TCircuit$
$t_2 : \mathbb{P}\ TCircuit$
$d_1 : \mathbb{P}\ Subroute$
$d_2 : \mathbb{P}\ Subroute$

(23)

With restriction to the three types of rules with which we are concerned, the overall structure of a geographic database may be represented by the schema (24). This contains three schemas, which specify detailed conditions (constraints) for each rule and are not specified here.

Database

RouteRulesConstraints
SubrouteRulesConstraints
PointsRulesConstraints

(24)

5. Conclusions

There are different reasons for applying formal techniques: an effort to improve quality of the whole development process, to improve integrity, reliability or other

characteristics of the system, to reduce specification errors, to improve requirements definition and documentation, to provide a firmer foundation for maintenance and enhancement and a more rational basis for choosing test data, to explore properties of a design architecture, to be as certain as possible that the design and implementation are error-free, to meet particular customer or standard requirements. The model-based approach results in a model of the system that acts as a specification and is constructed using well-understood mathematical entities. Setting out pre and post conditions for the function can specify functions. However, this approach does not scale up to large or medium sized systems. Thus, the *Z* notation is not intended for the description of non-functional properties, such as usability, performance, size and reliability. Neither is it intended for the description of timed or concurrent behaviour. This brings certain limitations when using it for formal specification of railway interlocking systems. However, there are other formal methods that may be used in combination with *Z* to relate state and state-change information to complementary aspects of design. For such purpose there are often used *Communicating Sequential Processes* (CSP). This formalism is able to describe concurrent behaviour of more objects or processes through interaction with their environment. More details about the way of modelling a geographic database using this formalism and checking it by $FDR2$ (*Failures-Divergences Refinement2*) can be found e.g. in (SIMPSON, 1998). Despite all discussed limitations, the functional view of using *Z* is widely accepted, and combined with the behavioural view supported by other formal techniques rather than used independently.

References

[1] CENELEC European Committee for Electro-technical Standardisation, Railway Applications: Software for Railway Control and Protection Systems. Draft prEN50128, August 1996.

[2] HEIMDAHL, Mats P. E. – HEITMEYER, Constance L., Formal Methods for Developing High Assurance Computer Systems: Working Group Report, *Proc. 2nd IEEE Workshop on Industrial-strength Formal Techniques (WIFT'98)*, Boca Raton, FL. October 19, 1998.

[3] ISO SC22 Working Group 19, *Z* Notation. Technical report, ISO/IEC JTCS1, SC22 N1970. ISO CD 13568, Committee Draft of the proposed *Z* standard, 1995.

[4] JANOTA, A., Formal Specification of a Railway Interlocking System. *Studies of the Faculty of Electrical Engineering*, University of Žilina, **26** (2000), pp. 13–25. (In Slovak).

[5] MARTIN, A., Relating Z and First-Order Logic. In: J. Wing, J. Woodcock, J. Davis (Eds.): *Proc. FM'99*, **II** (1999), LNCS 1709, pp. 1266–1280.

[6] NASA/TP-98-208193, NASA Formal Methods Specification and Verification Guidebook for Software and Computer Systems, Vol. I: Planning and Technology Insertion. 88 p, 1998.

[7] NICHOLLS, J. E., Domains of Applications for Formal Methods. *Proc. of Z User Workshop*, University of York, Workshops in Computing, Springer-Verlag, 1992.

[8] SIMPSON, A. C., Model Checking for Interlocking Safety. *Proc. FMERail Workshop*, Vol. 2 of 2nd FMERail Seminar, Parks Road, Oxford OX1 3QD, England, October 1998, ESSI Project 26538.

[9] WOODCOCK, J. C. P. – DAVIES, J., Using *Z*: Specification, Refinement and Proof. Prentice Hall International Series in Computer Science, 1996.