# FORMAL SPECIFICATION IN “Z” LANGUAGE BY SOFTWARE Z/EVES

**J. Švec, J. Zahradník**

*Department of Control and Information Systems, Faculty of Electrical Engineering, University of Žilina, Univerzitná 8215/1, 010 26 Žilina, Slovak Republic*
*e-mail: jozef.svec@fel.utc.sk, jiri.zahradnik@fel.utc.sk*

**Summary** The paper shows a short overview of representation formalisms, which can be used for specification of technical system functional requirements. Some basic model schemas of function called User Identification of ITS are presented by formal specification “Z” language and software Z/EVES.

## 1. INTRODUCTION

Design of easy and usable models is process, which verifies human ability to understand problem and also it allows understanding to other. Operations in the system can be analyzed and improved more effectively by process modeling. Design of models is also used in transport applications [4], 5].
Functional requirements specification can be realized through following representation formalisms:

- informal: optional graphic representation, natural language, example description, animations, and so on;
- semi-formal: state machine diagrams, entity-relational diagrams, unified modeling language and so on;
- formal: specification languages or knowledge representation languages.

### 1.1. Informal specification

Informal specification is markedly oriented to user – it is akin to him, well-know and it has big expressive ability (include polyvalence expression, inconsistent and opposite propositions).

### 1.2. Semi-formal specification

Semi-formal specification is based on structured visualization of the system – it’s easy, presents good system overview and it’s usually used as a quasi-standard in industry. Unlike informal specification, semi-formal specification has partly formal defined semantic. The following diagrams (object oriented model, data flows diagrams, entity-relational diagrams, Petri nets, etc.) can be included to the semi-formal specification.

### 1.3. Formal specification

Formal specification is expression of traced attributes list, which is expressed by formal specification language and with specific abstraction level. Formal specification languages have better defined semantic. Based on these language attributes it is possible to conclude about most of represented knowledge including entire or partial code generation. Formal specification language creates mathematic basis for formal method. This method shows, what specification has to say. Language shows in detail how reality involved in specification can be expressed. Languages like VDM, Z, B, EVES, LOTOS should be included to this language group.

## 2. FORMAL SPECIFICATION “Z” LANGUAGE

“Z” language is a formal specification language, which is used for description and functions modelling of computer systems. This language enables to write formal specification of computer programs and to formulate evidences of system behaviour. Specification languages are situated between natural and program languages. These types of languages enable to eliminate internal ambiguity, which is characteristic for natural languages. “Z” language specification is organized into specific units, which are reciprocally re-bounded by structural relations. Each of these units has a declaration and prepositional part. State schema, operating schema and axiomatic definition are the most frequent used schemas. Software pack called Z/EVES, and other software packs, also serves for “Z” language schema design support. Software pack Z/EVES enables writing, developing and analysing of “Z” language specification. Z/EVES consists of two parts. First part is virtual server, which insures syntactic propriety revision and activities for execution of theorems and paragraphs logic validation. Second part is graphical interface, which enables work with specification. Transcription from informal specification to “Z” language is realized by schema design. There are six basic types of schemas (Free Type Definition, Axiomatic Definition, Vertical Definition of State Schema, State Initialization, Operating Schema and Horizontal Schema). Other schemas can be applied on this six basic schema types. Schemas in next part are shifted from application model example of Identifikácia používateľa (User Identification) function in ITS.

Free Type Definition

This type of definition implements set of all system states. Individual elements (constants) differ from each other, and their sequel in definition is arbitrary.

*State* ::= *basic* | *detection* | *identification* | *record*

*PresenceOfUser* ::= *present* | *absent*

*TypeOfUser* ::= *passenger* | *driver* | *vehicle*

*IdentificationRequest* ::= *sent_to_PSG*
| *sent_to_DRV*
| *sent_to_VHC*

*Identification* ::= *recieved_from_PSG*
| *recieved_from_DRV*
| *recieved_from_VHC*
| *not_recieved_from_PSG*
| *not_recieved_from_DRV*
| *not_recieved_from_VHC*

*Fig. 1. Free Type Definition*

In this case, this type of definition defines system states that befit to type *State* (*basic*, *detection*, *identification*, *record*). Next type is type *PresenceOfUser* (*present*, *absent*), *TypeOfUser* (*passenger*, *driver*, *vehicle*), *IdentificationRequest* (which represents to whom the request was sent – driver, passenger, vehicle) and *Identification* (represents, if identification is received or not and from whom– driver, passenger, vehicle).

Axiomatic Definition

This type of definition implements one or more global variables. It consists of two parts. First part is declaration and implements new symbols. Second is predicate part (optional) and specifies boundary conditions. This type of schema is not show in application example.

State Schema

State Schema describes state of system. It consists of the schema title, declaration (name:type) and predicates (boundary conditions). This schema type specifies all allowed states of the system. System state is allowed, if all conditions from axiomatic part are fulfilled.

```
System
state: State
```

*Fig. 2. Vertical Definition of State Schema*

The schema title is *System* in this concrete event. The system from its definition should gain elements from type *State* and these elements are: *basic*, *detection*, *identification* and *record*.

State Initialization

This type of schema has to be in each specification and it serves to system initialization by some way.

```
IniciateSystem
System
state = basic
```

*Fig. 3. State Initialization*

The start state, in which the system is started up, is defined by type *State*. This state is defined as *basic*.

Operating Schema

Operating Schema specifies situations, in which the system state is changed (Δ) or unchanged (Ξ). Schema also specifies operations including state variables, main state variables and input and output variables.

```
Detection
ΞSystem
user?: PresenceOfUser
state = basic
user? = absent
state' = basic
```

*Fig. 4. Operating Schema – changeless system state*

Operating Schema (fig. 4.) characterizes a situation, in which the state is not changed after user detection. Title of this operating schema is *Detection*. Input to schema is *user?*. This input could have two elements *present*, *absent*, which are defined by type *PresenceOfUser*. Before operation the initial state, in which the system occurs, is *basic*. If input *user?* will gain element *absent*, than the system will stay in previous state (state *basic*). This is represented by title with apostrophe *state'*, which gain element *basic*.

```
PSGDetection
ΔSystem
user?: PresenceOfUser
type_of_user?: TypeOfUser
request!: IdentificationRequest
────────────────
state = basic
user? = present
type_of_user? = passenger
state' = detection
request! = sent_to_PSG
```

*Fig. 5. Operating Schema – change of system state*

Operating schema in fig. 5., in contrast to previous schema (fig. 4.), defines the change of the system state, in which the system was before operation. The title of schema is *PSGDetection*. Schema includes two inputs to operation. First of them is *user?*. This input could gain two elements *present*, *absent*, which are defined by type *PresenceOfUser*. Second input to operation is *type_of_user?*. This input could gain three elements (*passenger*, *driver* and *vehicle*) from type *TypeOfUser*. Schema also includes one output from operation, which is represented by title *request!*. This output represents sending of identification requests to user. It could gain elements from type *IdentificationRequest*. These elements are *sent_to_PSG, sent_to_DRV, sent_to_VHC*. Before operation the system is in the basic state. If input to operation *user?* will gain element *present* and input *type_of_user?* will gain element *passenger*, then system will transfer to after operation state. This after operation state is represented by title with apostrophe *state´*. This state has title *detection*. Output from operation *request!*, which gains element *sent_to_PSG*, is sent at the same time, representing sending of identification request to user.

Horizontal Schema
This type of schema is used for development of new combinations from existing schemas, by the using of schematic count operations. This schema type is not shown in the application example.

## 3. CONTRIBUTION

The main purpose of this paper is to show how it is possible to apply formal specification “Z” language and Z/EVES software pack on modeling of service User identification that was created based on defined functional requirements specification.
This work has been supported by the Grant Agency of the Slovak Republic VEGA, grant No.1/1044/04 "Theoretical foundations for implementing e-Safety principles into intelligent transportation systems" and project Coordination and Stimulation from Innovative ITS Activities in Central and Eastern European Countries - CONNECT, TEN-T Programme EK.

## REFERENCES

[1] JANOTA, A.: Formálna špecifikácia bezpečnostne kritických systémov. Habilitačná práca. 2004.

[2] Inteligentní dopravní systémy v podmínkách dopravně telekomunikačního prostředí v České republice. Available at: http://www.lt.fd.cvut.cz/its/

[3] SAALTINK, M.: The Z/EVES 2.0 User’s guide. Technical Report TR-99-5493-06a, ORA Canada, October 1999.

[4] JANOTA, A.: Using Z Specification for Railway Interlocking Safety. In: Periodica Polytechnica, Ser. Transport Engineering, Vol. 28, No. 1 - 2, Hungary 2000. (s. 39-53) ISSN 1587-3811.

[5] 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.