THE ARCHIVES OF TRANSPORT
VOL. XXII NO 2 2010
10.2478/v10174-010-0014-1

Object-oriented programming as a method for developing software in rail-traffic-control computer systems

Marek Sumiła*
Andrzej Lewiński**

Received January 2010

Abstract

The paper focuses on a new method for specifying safe software for rail traffic control systems. The presented method is particularly convenient to define typical devices and subsystems used in traffic control, defined as software blocks in which control algorithms are directly applicable. The method uses the object-oriented methodology and the UML language. The program in UML language allows for modelling, verification, functional testing, and simulation in an environment that is typical for rail traffic control. The method takes the software implementation requirements into account, having in mind the safety and control in real time in accordance with the UIC and CENELEC standards and recommendations.

Keywords: rail control systems, object-oriented programming, safe software

1. Introduction

Rail traffic control devices are structured on the basis of computer technology in fulfilling their tasks under the two control platforms: the hardware platform and the software platform. The first one is responsible for the proper co-operation with devices of the railway tracks infrastructure and is a working environment for the computer control program whose task is to control rail traffic control processes in the operational area of the device.

* Faculty of Transport, Warsaw University of Technology, 75 Koszykowa St., 00-665 Warsaw, Poland, e-mail: sumila@it.pw.edu.pl

** Faculty of Transport, Technical University of Radom, 29 Malczewskiego St., 26-600 Radom, Poland, e-mail: alewinski@pr.radom.pl

Taking the safety aspect into account with respect to this type of devices leads to an analysis of both the hardware platform and the software. The safety of hardware is often expressed by its operational reliability rate as well as by a structural design of devices, so that one type of damages would be much more likely than the other ones [16] [108]. This approach led to having developed a concept of safe damages and it is considered as a specific philosophy for designing safe rail traffic control systems. Assessment of the rail traffic control system safety is expressed in this case by the reliability rates, developed based on the obtained empirical data, e.g.: a number of the device defects per year or by means of analysis of the so-called Fault Tree Analysis (*FTA*).

The software security is understood differently. It is conditional on a correct specification, coding, and compiling the program in the target environment (rail traffic control device). It is closely linked to the ability of software to detect disturbances occurring during operation of the system and upon their occurrence, the ability to such a response that guarantees safety of the set-up activities.

Previous methods for developing the rail traffic control systems software in the low-level languages (e.g. assembler) proved to be hardly universal. At the moment, they are prohibited. The higher level of algorithmic pragmatics languages (e.g. PASCAL, C), known as structural languages, has become more convenient. However, it seems that the use of those languages also causes a high level of abstraction of the program functions that are attributed to it in the hardware specification. This results in situations where part of the specification and coding errors remain undetected, and which consequently may lead to dangerous situations.

2. General review of the existing software development methods

This method is the procedure for implementation of a part of the *life cycle programming* system. A concept of the method directly refers to the concept of methodics and methodology. The methodics defines a set of methods and procedures of technical and organisational character, allowing for the life cycle of the system to be implemented by an executive team. Methodology is understood as a science on methods.

The *life cycle programming* is a process consisting of a sequence of mutually consistent steps, allowing for a full and effective development of IT systems and then use thereof. The general life cycle of the system programming consists of the following stages: strategic planning, requirements analysis, designing, implementation and operation. By introducing a more detailed diagnosis, we can talk about stages aiming at: specification, designing, coding, integration, verification, validation, installation, and maintaining operability.

The system analysis has occurred to be extremely important in the software development process as its objective was to understand the problem areas, namely

to control rail traffic in this case, provide a description of the current state of the information system, and consequently also to provide the future software model. By the software model we mean a certain abstraction of the system being designed and viewed from a particular perspective and with a certain level of detailedness.

There are many different concepts of software development currently in the world, which partly result from many years of the programmers' experience and partly is the result of studies conducted by various research and industrial centres. The first group includes a method that is based on the cascade model (*Waterfall Model*), model V, incremental model, prototype model, spiral model, or evolutionary model (*Evolutionary Delivery*). The second group of methods includes the method based on *eXtreme Programming* (XP), *Scrum, Dynamic Systems Development Method, Adaptive Software Development, Crystal Clear, Feature Driven Development,* and *Pragmatic Programming.* The vast majority of the secure systems designers is in favour of applying classic methods as good and therefore more predictable.

2.1. Classic and modern structural approach

The structural approach is considered as a classic model-building approach, which was presented out in the seventies by T. DeMarco. The author of this concept has divided the work into four primary stages:

- to develop a physical model of the existing system,
- to develop a logical model of the existing system,
- to develop a logical model of the required system,
- to develop a physical model of the required system.

Modern structural analysis was developed based on the classic approach criticism, carried out by E. Yourdon in the late eighties [27]. In his criticism Yourdon proved that the more accurate proceeding is that which leads to the development of so-called *Essentials Model*, describing both the environment model and the behavioural model and based on it the *Implementation Model* is developed. A description of models is usually presented by DFD diagrams (*Data Flow Diagram*). The method allows for prioritisation and making a description more detailed as well as for use of additional descriptive forms.

There are three concepts to develop hierarchical DFD diagrams: *top-down*, *bottom-up,* and *hybrid*. The top-down method involves a description of:

- context diagram,
- defining very general processes,
- to reach the final solution by further detailed clarifications (decomposition of processes).

The method requires experience, especially in large systems, and a separation of processes at a high level may appear to be difficult.

The bottom-up method involves:

- creating a list of independent processes that shall not be decomposed any further,

- grouping of processes to the form of higher-order processes up to the highest level – the context diagram.

This method also requires experience, because there is a large number of processes that must be examined already at the beginning.

The hybrid method is to define:

- context diagram,
- general diagram,
- a list of processes that will not be decomposed,
- to assign the lowest-order processes to the appropriate branch of the general diagram,
- to group processes within a single branch until a single process (with the general diagram) has been created for all the branches.

Component criteria are used for assessing the quality of DFD diagrams:

- presence of at least one process,
- proper names of all the elements introduced into the diagram,
- correct use of the revision elements,
- presence of at least one input flow that is assigned to each process,
- presence of at least one output flow from each process,
- a known source of each data flow,
- no flows that have no destination of their own,
- no flows introduced between the external unit and a storage,
- correctness of the flows structure to and from the storages.

The following can be mentioned as quality criteria of semantic meaning for DFD diagrams:

- level of detailedness of the processes specification,
- usefulness of input streams to each process separately,
- reality of processing (a set of required data),
- usefulness of each process output flow,
- purposefulness of the data flows decomposition.

The above criteria are an element that verifies correctness of the method applied in the software development process.

2.2. Methods for software development in Real-Time Systems

The presented overview of the computer systems designing methods would not be comprehensive without specifying the methods developed for purposes of real-time systems, which include traffic control systems in transport [2]. Currently, there are several methodics taking into account the issue of limitations effecting from a time flow. Representatives that can be mentioned are as follows:

- Ward-Mellor Methodics,
- Hatley's-Pirbhai Methodics,
- Lavi-Harel Methodics.

All of the methodics listed herein operate on three basic aspects. They are models, methods and tools[1]. Due to the fact that real-time systems are inextricably linked with the environment, resources understood here as hardware, inside which a given software shall operate, should also be included in the description of each system in addition to models describing the structure and behaviour on the basis of the specification.

The methodics, proposed by Paul T. Ward and Stephen J. Mellor called *Structured Design for Real-Time Systems* (SDRTS) or in other words *Real-Time Structured Analysis* (RTSA) shall be presented as an example. It is an extension of classic structural methods SA/SD (DeMarco, Yourdon, Constantine) for real-time application. Therefore, the basic model specifies the system through a model of environment in which the system operates and via the behaviour model as a description of response to events occurring in its environment. The environment model structural design requires establishment of the context diagram understood as the system environment and the boundaries between the system and the environment, and a list of events occurring in the environment. The behaviour model design includes a description of the transformation pattern, the system functions, i.e. the responses of the system to external events and data formula described in the form of data structures, information flow and its storage.

The implementation model includes a three-level description:

- processes level – a division of functions and data in a system distributed between processors or computers,
- level of tasks – tasks are assigned to individual processors,
- level of modules – a division of the task into modules that form the task structure.

There are two basic tools for describing the system in the Ward-Mellor methodics:

- extended DFD data flow diagrams – they define continuous and discrete flows,
- *Control Flow Diagrams* (CFD) – they define activities and control signals.

These include:

  - Standby E (*Enable*) – they allow for operation if the input conditions are met,
  - Disabling the stand-by state D (*Disable*),
  - Conditioning the stand-by state on the T (*Trigger*) signal.

Events are defined, depending on the existing conditions. They can be divided into three main groups:

- events occurring exclusively outside the system, i.e. in its environment,
- each event must face a response of the system,
- events occur at certain moments.

The designing phase includes three models: processors, tasks, and modules to which the following criteria are applicable:

- gradual introduction of the time element in operation, the inclusion of erroneous operation and the limited resources, such as e.g. buffers memories,

[1] A character of the hetero mentioned aspects may be considered general.

- gradual elaboration in the decomposition phase by choosing the type of processors, determining time parameters, probability in algorithms in a layer of modules,
- looking for similar types of data for groups of modules.

The paper proposes using the object-oriented methodology as a direction of research studies on new and more efficient software development methodics and a method has been identified that may facilitate the work of the rail traffic control systems programmers.

3. Object-Oriented Methodology

The gist of OOM (*Object-Oriented Methodology*) is to define computer programs as a set of objects each of which stores static and dynamic data (*attributes*) and a behaviour expressed by procedures (*methods*) [3][17][20][26]. The object is an abstract entity representing or describing a certain thing or a concept existing in the real or software world. Is an independent, asynchronous, concurrent unit which stores data, performs services and cooperates with other objects by exchanging messages.

The idea for application of the object-oriented methodology (OOM) appeared in the sixties of the last century, as a method supporting the simulation of behaviours of objects in computer programs. The creators of the first language (Simuli 67) satisfying this assumption was Ole-Johan Dahl and Kristen Nygaard from Norsk Regnesentral in Oslo [20]. The success of the language led to development of the object orientation concept in the form of the Smalltalk language (Xerox) PARC, and in the eighties the Ada language (military applications), and C++[2] (the extension of C language). Modern languages, using the object-oriented methodology are an important group of software languages. The most popular include IncrTCL[3], Java[4], Object Pascal[5], Perl[6], PHP[7], Python[8], Ruby[9], UML[10] and many others.

The object-oriented programming differs from traditional structured programming in the way so that data and procedures are interlinked in order to encourage development of software and multiuse of programs or their portions, described by the objects. For that purpose both the objects and relations are parameterised. The process in the event of objects is about grouping them into classes. Therefore, the

[2] http://www.open-std.org/jtc1/sc22/wg21
[3] http://incrtcl.sourceforge.net
[4] http://java.sun.com
[5] http://www.borland.com
[6] http://www.perl.org
[7] http://www.php.net
[8] http://www.python.org
[9] http://www.ruby-lang.org
[10] http://www.uml.org

class is a collection of objects that have the same attributes, methods, relationships, and meaning.

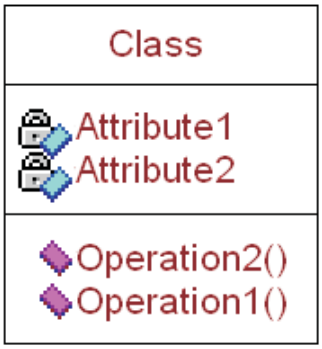

Fig. 1. Class diagram. Graphical presentation of the objects class

Parameterisation of relations involves determining the level of dependencies of the affected objects. These include: *associations*, *aggregation*s, *compositions*.

The greatest benefit effecting from application of the object-oriented methodology in the software development process is a natural transition from the concept of the program operation concept to its code in the modelling process. This means that the result and the purpose of the object-oriented programming is a model of future software. The most important features of the object-oriented methodology are considered to be [3] [6] [8] [17] [26]:

– *Abstraction* – is used to generalise the meaning of the object with respect to its operation, state, rules of communication with other objects,
– *Encapsulation* – means hiding the implementation of each object. The scope of external access to internal attributes of the object is strictly defined,
– *Polymorphism* – causes alterations in the type of the object, depending on a reference or a collection of the initiating objects,
– *Inheritance* – allows for creating specialised objects on the basis of the more general ones,
– *Reuse* – means creating, sharing, and using project components, software, and documentation, which are suitable for multiple applications at the same project – system.

An important stage in the development of object-oriented methodology was the introduction of the graphic environment as a means of expression of modelled structures. This has somehow become a natural stage of OOM development in the continuously growing complexity and a size of the software being developed. The shift away from a traditional recording as a line of code caused a more human-friendly perception and understanding of the saved program.

Various ways of presenting software structures have occurred along with the introduction of the graphic environment. Documentation of each of those structures

is envisaged for another (a correct) type of the diagram. Currently, there are several types of diagrams. Each of them serves to present (program) another aspect of the software being developed. Depending on the type of the diagram, a notation of elements has been created, allowing in a universal way for creating adequate model descriptions. Both the scope of illustration and a range of available components strongly depends on the type of diagram and its purpose. Currently known graphical object modelling languages often have different names, such as graphic design identifying the same elements. According to the authors, this introduces a big disorder in the programmer's work and leads to many misunderstandings.

Generally, we can distinguish the following types of diagrams [8]:

- Structure diagrams (represented include: classes diagram, components diagram, objects diagram, packages diagram, deployment diagram).
- Behaviours diagrams (represented include: activities diagram, state machine diagram, use cases diagram).
- Interaction diagrams (represented include: interaction diagram, sequence diagram, states diagram, communication diagram, and state process flow diagram).

It is worth noting here that while the idea of the structure diagrams partly corresponds to the traditional approach where parts of the software constitute the source code blocks, then the idea of behaviours and interactions diagrams is consistent with the theory of *Finite State Machine* (*FSM*). The FSMs are an important theoretical tool in creating and testing models of wider processes as applied to mathematics and logic, linguistics, philosophy, or biology. In the case of IT techniques they have become the way to describe behaviours of programs and software-controlled machines. In all languages where the basis of notation was a text, the finite state machines were created independently as an element that supports programmer's work. Physically created on paper they had no direct representation in the programming language. For languages with graphic notation, the state machine that had been built from states and transitions became synonymous with the program content (it was its accurate transition to the programming language).

The object-oriented methodology also allows for a possibility of a hierarchical building of a state machine, the so-called *states nesting*. This feature allows for hiding complex state structures in the form of abstract states. This is consistent with the generally accepted assumption of object-oriented programming, which assumes a principle of transition "from general to specific."

All of the above-mentioned features of object-oriented modelling led to the situation that OOM started to be treated as a serious tool for creating a sophisticated and complex software for IT systems.

3.1. UML language

Using the object-oriented methodology for software modelling of the rail traffic control system is associated with the use of one of the existing object-oriented languages. For purposes of this paper, the UML has been selected.

The UML language (Unified Modelling Language) was created in 1996 (version 0.9) as initiative of three object-oriented language developers: Grady Booch, James Rumbaugh and Ivar Jacobson [3]. Their purpose was to harmonise notations and vocabulary used for the systems modelling, since in early nineties there were more than fifty languages of object-oriented character [29]. Starting from version 1.0, the Object Management Group (OMG) organisation[11], which to date facilitates its development, took control over the language. The UML language in 1.4.2 version was adopted as ISO/IEC 19501 standard. In February 2009, version 2.2 was released significantly improving (compared to 1.4.2 version) the modelling of *embedded systems*.

The UML was designed as a language to define, visualise, structure, and document the systems software, however, it is also used for modelling business processes, systems engineering, and representing organisational structures. The latest version specifies fourteen major diagram types and three abstract types (of structures, behaviours, and interactions). Currently, SysML (*Systems Modelling Language*)[12], which is a language for modelling specific issues in systems engineering, was developed based on the UML language.

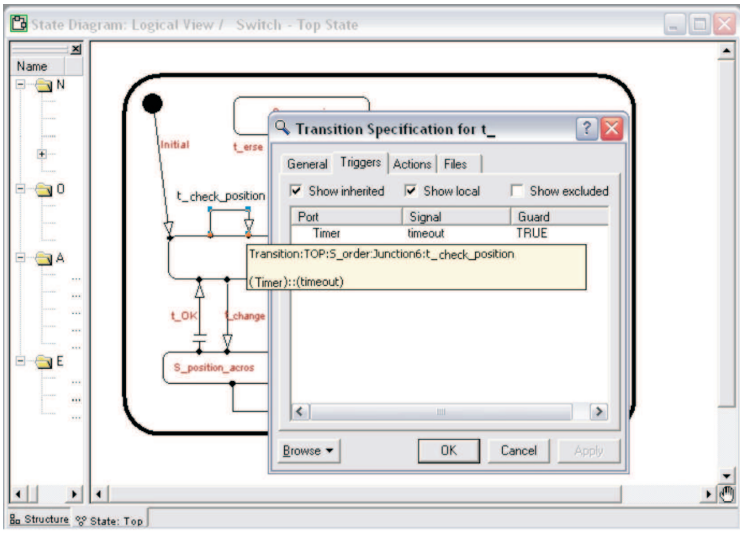

Fig. 2. Rose RealTime Tool. The window for defining the timeout signal

[11] http://www.uml.org/
[12] http://www.omgsysml.org/

3.2. Working tools with UML language

Software development by object-oriented modelling in UML language is inherently associated with CASE (*Computer-Aided Software Engineering*) tools. The first one of them was developed by same language authors, calling it the *Rational Rose*.

Currently, there is a significant number of tools on the market, allowing to work with the language. Generally, they can be divided into two groups: commercial and free. The scope of their possibilities is very diverse. Some of them enable only to create diagrams and their analysis, others allow far-reaching simulation and compilation in terms of devices in which they will operate as driver programs.

Using the publicly available sources[13] both commercial and non-commercial modelling tools in UML language can be mentioned and briefly characterised.

Despite a large number of tools available, relatively few of them are suitable for modelling software for control systems in transport. This is because of the vast responsibility that rests with the tool for correctness of the generated code, for the system responsible for transportation of people and goods.

4. The use of object-oriented methodology in software development for Rail Traffic Control Systems

4.1. Modelling and specification of Rail Traffic Control Devices

The traffic control process in the control circuit is usually presented with verbal descriptions in an informal way. The description includes four primary phases: the first of which aims at preparing the process route, the second one is vehicle movement supervision, the third one is release of the run, and finally, the fourth one is driving record. An example of such a description can be found e.g. in paper [7].

Formal descriptions of the rail traffic control structures and processes occur much more rarely. They are a tool to assess correctness of the control functions implementation and a pattern for designers of future control systems. Classic description of the rail traffic control elements and processes is provided by the paper [1], however, a full description of the formal processes associated with implementation of the runs is included in the paper [30][31].

4.2. Elaborated method assumptions

The elaborated method for the rail traffic control devices software development systematises the programmers' work by introducing a series of assumptions about the rules for developing a software model in UML language, conditions for selection of CASE environment and a compiler. The general assumptions include as follows:

[13] http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

1. Software for the rail traffic control computer system is developed on the basis of the existing formal and semi-formal descriptions contained in the device specification.
2. The method use is largely independent from the target machine's operating system.
3. Independent teams of programmers are assigned to develop software modules.
4. The designing process is implemented via the CASE tool whose manufacturer guarantees the correctness of the post-compilation program code.
5. UML is the software designing language deriving from a family of object-oriented languages, based on object-oriented methodology (OOM).
6. A limited number of UML diagrams is used in the software development process in order to reduce redundancy.
7. Diagrams of the resulting model can be part of the rail traffic control system documentation and the proof-of-safety element.

Assumptions on the software development rules:

1. Software is developed in the modelling process.
2. The method use allows for implementing the rail traffic control concept based both on the process method and the geographic method.
3. An object in the model is seen as an elementary machine with precisely defined boundaries of activities and input – output signals.
4. Representative of the Finite State Machine (FSM) is an active object.
5. Access to facilities is limited for safety reasons.
6. Communication with objects is possible using special interfaces, i.e. ports.
7. Each port has a defined set of rules governing the exchange of messages, as described in the protocol.
8. Each pair of ports (two communicating objects) is linked to one protocol.
9. The model has a hierarchical structure.
10. The model in UML language may become a framework of a proper software of the rail traffic control computer system, or its proper software, following its supplementation by the missing components in the low-level languages and after using the certified and secure compilers.
11. Diagrams of the resulting model can be part of the rail traffic control system documentation and the proof-of-safety element.

Assumptions on the software development process:

1. Production of the model is carried out in two phases:
   a. PHASE I – a model that takes into account proper operation of the system,
   b. PHASE II – a model extended by the emergency situations handling components;
2. The model is divided into two blocks:
   a. Dependence block,
   b. Block of interfaces to control external devices;

3. Production of the model is carried out on two planes: the software static structure and the software dynamic structure.
   a. The static structure plane is formed as the first plane and it is a collection of objects and their linking relationships,
   b. The dynamic structure plane supplements each object with a suitable control machine;
4. A description of each of the model planes is presented using diagrams:
   a. The static structure plane: class diagrams and structures diagrams,
   b. The dynamic structure plane: states diagrams and sequences diagrams;
5. Transition to Phase II is implemented through extension of the model developed in Phase I by applying object-oriented methodology (OOM) properties – inheritance.
6. The model, resulting from application of the method, is supplemented with an additional code in low level language (e.g. C, C++) appropriate for a specific operating system, functions, time operation, etc.
7. Control of external devices, especially controlled devices, such as switches, semaphores, is done by using intermediate software objects hereinafter referred to as virtual elements.
8. The virtual elements constitute a software interface for standard signals of a logical plane to control technological process.
9. Each of the software model development phases is verified in the validation and simulation process.

4.3. Examples of the method applications

The software functionality model, created as a result of the method application, is implemented in two blocks. The first is responsible for interlocking functions and the other one for cooperation of the track construction facilities.

4.3.1. Interlocking functions model

The interlocking functions model describes a proper control of track installation elements so that traffic of rail traction units would be carried out smoothly and collision free. The development of such a model requires the programmers to know a specific character of the route adjustment process and the process influencing factors. The use of object-oriented methodology allows for dividing work into two subsequent stages: a model for ideal conditions and a model for actual conditions. A transition from the first model to the other one takes place in the process of extending the first one by adding additional objects, states, and transitions. This process occurs through the application of the objects properties inheritance.

For the ideally ongoing process of setting a perfect route, the specifications model can be presented in the form of a machine having the following states/stages [30]:

– state of idle route – the rail traffic control system expects to receive a request for performance of a task, and also it performs a self-testing of the system.
– the choice stage – is a process that aims to select those elements of the track system the use of which shall allow for creating a route path and a side protection. The stage ends when elements, which are required for implementation of the route, have been marked. This marking excludes a possibility to select items booked for another route.
– the set-up stage – is a process that involves checking and possible switching of the driving path elements into a desired position as well as closing the elements serving as lateral protection for the ongoing route.
– the confirming stage – at this task implementation stage, activities are carried out to prevent from making changes to the state of the elements, involved in the process, and elements of the lateral protection.
– the process release stage - (the process implementation) – the rail traffic control system that at this monitors and supervises train movement, correct taking and releasing of individual isolated sections, thus changing their state into free. The process of gradual release of the process route elements is called the sectional releasing.

The hereto presented machine also allows for giving up the ongoing task in the railway traffic system. For the adopted process method, the control process in accordance with the machine described above is illustrated in Figure 3.

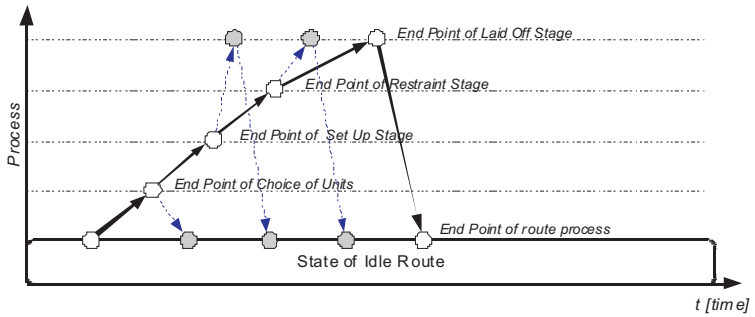

Fig. 3. Model of the machine for a route setting process. White circle – point of the completed control stage, grey circle – a point of early termination of the control process, black arrow – a progress of the control process, dashed arrow – a progress of the control process early termination

A transition from the model, presented in Figure 3, to the software in the UML is largely the mapping of the first one. The subsequent stages of the setting-up process have been mapped in the form of active objects. A combining relation of the objects indicates an interdependence of the next object from the previous one, which is also the mapping of the existing reality. Data on the allowed processes is stored in the object called the interlocking table *Tablica_zaleznosci*, and the current values of states of each of the railway infrastructure elements are stored in an additional object:

the table of states of track systems elements *Tablica_stanow_elementów_ukl_tor*. Each of the active objects is assigned with a number of ports, such as: an internal port *Timer* being the source of the system clock beat signal and a traffic controller port *Port_dyz_ruchu*, allowing for bidirectional transmission of signals between the system and the managing traffic controller on duty.

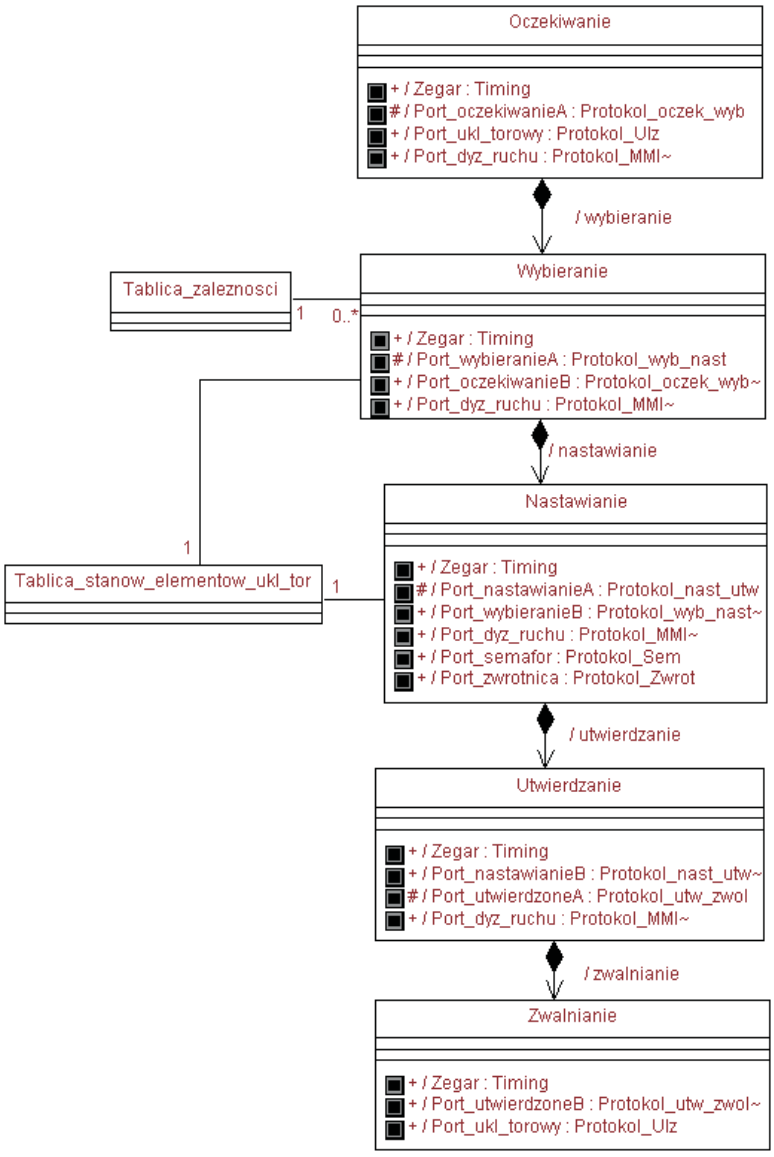

Fig. 4. Class diagram. Structural model of the Route Process Set-up Machines

Extension of the concept of the machine that performs the route setting-up process by an additional state of failure is illustrated in Figure 5. There is an assumption in this machine that a situation may occur distorting a scheduled sequence of actions.

The emergence of such an event may occur at any time of the ongoing process and should result in a transition to a safe state, *the State of Failure*. This has been illustrated in the above figure by the vertical red arrows towards the top.

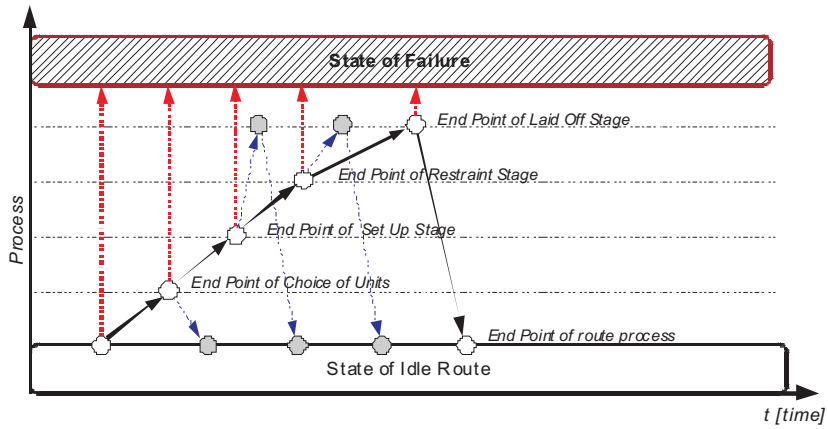

Fig. 5. Model of the route setting real process. White circle – point of the completed control stage, grey circle – a point of early termination of the control process, black arrow – a progress of the control process, dashed arrow – a progress of the control process early termination, vertical dashed arrow – the transition to the state of failure

Introduction of an additional state in the system model leads to a supplementation of the software, as illustrated in Figure 5, with an additional object failure handling *Obsluga_Awarii*. Its task is to gain control over the system and bring their ongoing operations to the state that guarantees safety. Extension of the object model also involves adding additional states, initiating safety sequences, to each of the existing active objects. Additional states were added as extensions of the existing objects in the form of child objects using the inheritance relationship. Such an approach does not affect the current structure, but only adds a new functionality.

The post-modification object model is presented in Figure 6.

4.3.2. Example of the Set-Up Process Executing Object Machine Modelling

Each of the above stages of the interlocking functions implementation is modelled by an active object. The machine of each of those objects is responsible for a part of the set-up process. The way of implementing such a machine may be

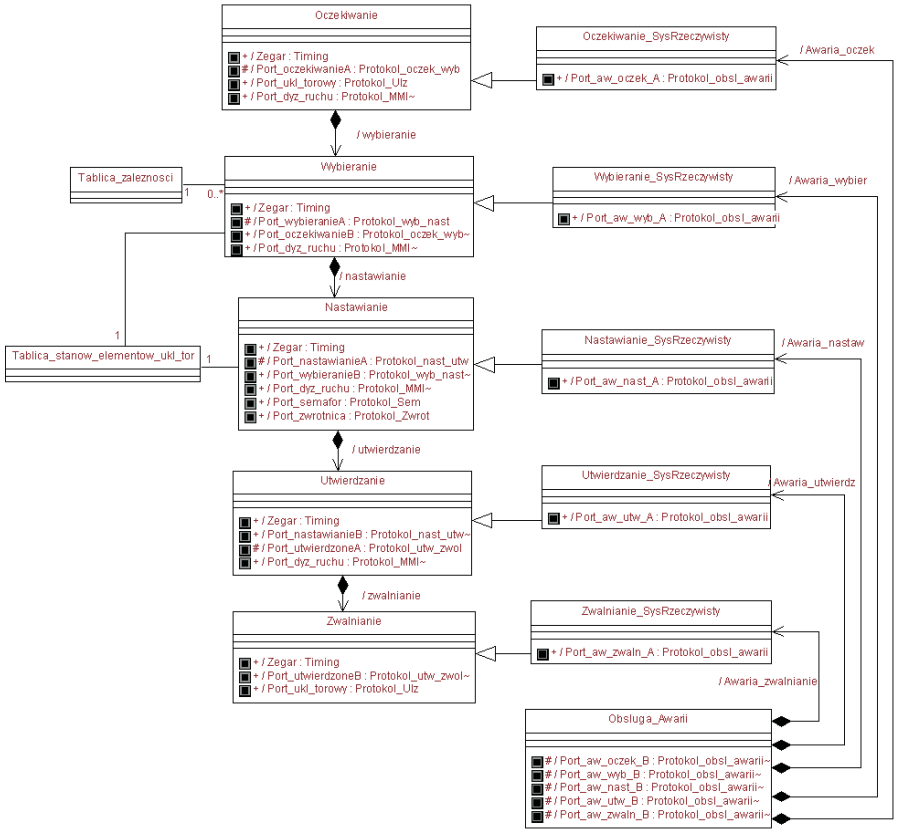

Fig. 6. Class diagram. The extended model

illustrated with an example of the stage where elements are selected for a new process route.

Activation of the *Initial* transition launches the machine. The process begins when an object makes a transition of the object from state *St_Choice* to the state of *St_Actual_element* with transition of *t_take_el*. As a result of this transition information is collected about the current state of the section. If the attribute value, corresponding to the occupancy, indicates the state of ”free,” there is a change of the section object’s attribute into the selected one, and a check of the condition for the last element of the process route in the state of *CP_Last_ways_el* decision. Depending on the results of the operation, subsequent checks are carried out or the operation is terminated and transferred into the next stage in the set-up process. Whenever the section dedicated to the process is found busy, it is followed by release of elements that have already booked by changing the attribute of each object into not selected one in the state of *St_Release_elements*.

Other stages of the route set-up process can be presented in a similar way.

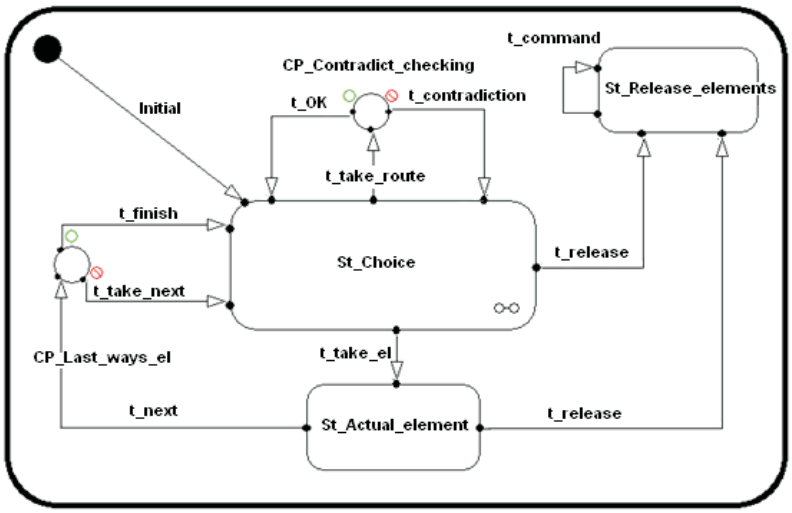

Fig. 7. Diagram of states. Active object *Choice*

5. Summary

The main purpose for developing the method presented above was to show how to design software for the rail traffic control system that shall allow for a simpler implementation than ever before while maintaining high reliability and safety requirements. Taking into account the existing CENELEC standards [18] [19], the use of object-oriented methodology and object-oriented modelling language UML operating in CASE (Rose RealTime) environment has been proposed.

The presented method involves describing static and dynamic structures of the system software in the form of graphic diagrams. The source of this description may be an informal (verbal) record as well as a formal one (mathematically verified.) The result of the description is a certain model that reflects the software structure and functionality. By applying the object-oriented language, having a graphic interpretation of the program structures, it has become easier than ever before to create software modules with clearly defined boundaries, transparent rules of communication, and evidently defined internal states.

Development of a complete and coherent model allows for performing a compilation using the standard compiler of the chosen target language (C, C++), or another compiler selected by the designer. This property allows the application of the compilers whose producers provide a guarantee for the accuracy of the code for the target machine. This is an important element in the broadly understood safety, in the context of tasks that are performed by the hardware equipped in software that was developed in this way.

In the course of the conducted research studies, it has been found that the choice of the CASE tool is of a great importance in the process of software development

in the UML language. A list of available commercial and free tools on the one hand encourages to experimentation, but on the other hand, it forces to identify advantages and limitations by future software designers. The right choice is of the special importance in the last stages of the software development work involving testing, simulation, and documentation of the system model.

The method, presented in this publication, may facilitate the improvement of the new rail traffic control systems implementation process. Its application indicates the direction of how to solve important issues related to security verification of computer control systems.

References

1. Apuniewicz S.: Principles of mathematical modelling of objects and rail traffic control processes. Scientific Papers of Kielce University of Technology, Kielce, 1977.
2. Ben–Ari M.: Fundamentals of concurrent and distributed programming. WNT, Warsaw, 1996.
3. Booch G., Rumbaugh J., Jacobson I.: UML User Guide. WNT, Warsaw, 2001.
4. Christov Ch.: Problems of safety electronic systems of railway signalling. Monograph, The University of Technology in Sophia, Transport Department, Sophia, 1988.
5. CNTK 4T12C00529: Impact of new information technologies to improve the functionality and safety of the railway traffic. Warsaw, 2006.
6. Coad P., Yourdan E.: Object Analysis. WNT, Warsaw, 1993.
7. Dąbrowa-Bajon M.: Fundamentals of rail traffic control. Publishing House of the Warsaw University of Technology, Warsaw, 2002.
8. Douglass B. P.: Doing Hard Time – developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. ADDISON-WESLEY, Massachusetts, 1999.
9. Evans A.,France R.,Lano K.,Rumpe B.: Developing the UML as a Formal Modelling Notation. In Proceedings of UML'98 – The United Modelling Language. Beyond the Notation. Volume 1618 in Lecture Notes in Computer Science. Springer–Verlag, 1998.
10. Hooman J., Mulyar N., Posta L.: Validating UML models of Embedded Systems by Coupling Tools. Embedded Systems Institute, Eindhoven 2004.
11. Huzar Z.: Real-Time Semantics of State Maps in UML. Conference Materials – Ninth Conference on Real Time Systems, Ustroń, 2002.
12. ISO/IEC 19501: Unified Modelling Language Specification. Version 1.4.2. Formal/05-04-01.
13. Kopetz H.: Real-Time Systems. Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, Boston/Dordrecht/London, 1998.
14. Lewiński A., Konopiński L., Siergiejczyk M.: Reliability and security of selected computer rail traffic control systems. Conference Materials – National Conference on Security and Reliability '99 KONBiN. Zakopane–Kościelisko, 1999.
15. Lewiński A., Sumiła M.: The Semi-Functional and Reliability Modelling of Railway Control Systems. Scientific Papers, TRANSPORT, Paper 51, Gliwice 2003.
16. Lewiński A.: Problems of Secure Computer Systems Programming in Railway Transport Applications. Monograph. Publishing House of Radom University of Technology, Radom, 2001.
17. Meyer B.: Object-Oriented Programming. Helion, Warsaw, 2005.
18. PN-EN 50126: Railway Applications: Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS). European Standard CENELEC, September 1999.
19. PN-EN 50128: Railway Applications: Communication, signalling and processing systems – Software for railway control and protection systems. European Standard CENELEC, March 2001.

20. Subieta K.: Object Orientation in Designing and Databases. Academic Publishing House PLJ, Warsaw, 1998.
21. Sumiła M., "Method for Development of Control Software in the Railway Traffic Control Systems", Publications of Warsaw University of Technology, Warsaw, 2007.
22. Sumiła M.: Designing of Control Systems Using Rose RealTime. Conference Materials – Ninth Conference on Real Time Systems, Ustroń, 2002.
23. Sumiła M.: Attempt to Implement Secure Software for the Safe Rail Traffic Control System Using Object-Oriented Techniques. Scientific Works of Radom University of Technology, TRANSPORT 1(17), Radom, 2003.
24. Wawrzyński W., Kochan A.: The Object–Oriented Modelling of Control Systems in Transport. Scientific Works of Warsaw University of Technology, TRANSPORT, Warsaw, 2001.
25. Wawrzyński W.: Safety of Control Systems in Transport. Maintenance Problems Library, Warsaw-Radom, 2004.
26. Yourdon E., Argila C.: Object-Oriented Analysis and Designing – Examples of Applications. WNT, Warsaw, 2000.
27. Yourdon E.: Modern Structured Analysis. Prentice Hall, 1989.
28. Zabłocki W.: Station-Based Model of Traffic Control Devices. Scientific Works of Radom University of Technology, TRANSPORT 1(13), Radom, 2000.
29. Zabłocki W.: Modelling of Station Rail Traffic Control Systems. Scientific Works of Warsaw University of Technology, TRANSPORT, Paper 65, Warsaw, 2008.
30. Zabłocki W.: Modelling of rail traffic control systems information structure and formal description elements. Scientific Works of Warsaw University of Technology, TRANSPORT, Paper 57, Warsaw, 2006.
31. Zabłocki W.: Modelling of rail traffic control systems. Scientific Works of Radom University of Technology, TRANSPORT 1(17), Radom 2003.