UML-BASED DESIGN AND FORMAL ANALYSIS OF A SAFETY-CRITICAL RAILWAY CONTROL SOFTWARE MODULE

András Pataricza[1], István Majzik[1], Gábor Huszerl[1] and György Várnai[2]
[1]*Department of Measurement and Information Systems*
Budapest University of Technology and Economics, Budapest, Hungary
Address: Magyar Tudósok krt. 2., Budapest, Hungary, H-1117
Phone: (+36-1) 463-3582, Fax: (+36-1) 463-2667, E-mail: {pataric,majzik,huszerl}@mit.bme.hu
[2] *Prolan Process Control Co.*
Address: Szentendrei u. 1-3, Budakalász, Lenfonópark, Hungary, H-2011
Phone: (+36-26) 543-100, Fax: (+36-26) 543-101, E-mail: varnai-gy@prolan.hu

Abstract: A new equipment of safety relevance has been developed to upgrade ageing relay-based railway interlocking systems in Hungary. In course of the design process formal methods have been used in the development of a module realising a well-separable function of the system. Namely, the UML-based design process was extended by model based analysis and validation. The first kind of analysis checked the completeness and consistency of the behavioural description of the module. In the subsequent phases, the functional design was enriched by modelling the potential faults and their effects. This kind of extension allowed the analysis of the error propagation and testability.
Keywords: UML-based modelling, safety analysis, testability, error propagation analysis.

1. INTRODUCTION

The introduction of the new European standards for railway safety will significantly change the method in the design procedures in all cases when a new product of safety relevance is being developed or an existing safety system is to be upgraded.

In our case a new equipment has been developed for upgrading existing relay-based railway interlocking systems. In order to gain the necessary approval of the authorities, the appropriate safety level of the modified interlocking system had to be proved. The acceptance of the hardware structure selected for the new equipment could be made without major difficulties, since the design has met the requirements of the UIC 738 recommendation. There are other well-known methodologies to prove the appropriate safety of the hardware.

Acceptance procedure for the software was much different because it should have been carried out (and is still being carried out) according to EN 50126, EN 50128 and EN 50129, and it was a novelty both for the authority and for the manufacturer. Validation and verification processes involved lots of time-consuming activities.

A significant but well-separated function of the system, the permission-managing module was developed by using UML, the Unified Modeling Language (OMG, 2001). The UML based design enabled us to apply formal verification and validation techniques.

While a relevant effort is being devoted in the software engineering industry to the development of standardised design languages and methods, such as UML, much less attention has been dedicated up to now to the integration of these design technologies with the verification and validation techniques. Our approach aimed at the extension of the UML-based design process by model based mathematical analysis and validation. The first kind of analysis checked the completeness and consistency of the behavioural description of the module, this way reducing the potential safety problems originating in an incomplete and ambiguous specification. In the subsequent phases, the functional design was enriched by modelling the potential faults and their effects. This kind of extension allowed the analysis of the error propagation and testability in the case of the anticipated faults.

In the following, first we present the permission-managing module (its hardware environment and software architecture) and illustrate its UML-based design. Then the formal analysis techniques are detailed. The paper is closed by a short Conclusion.

2. HARDWARE ENVIRONMENT OF THE PERMISSION-MANAGING MODULE

There are more than 300 railway stations in Hungary that are equipped with relay-based interlocking systems. Their ages vary between 15-30 years, but due to the proper maintenance done (and will be done) most of the systems has an additional life span of 10-20 more years.

For providing the operability of those systems as well as for expanding their functionality to match state-of-the-art requirements, a development project was started 3 years before.

The project has targeted to develop a new equipment of safety relevance that can be used

- to replace the existing MMI (a traditional push button operated, electromechanical control desk with mimic panel) of the relay interlocking,
- to make remote supervision and control of the interlocking system of a station possible,
- to integrate and solve other control and data acquisition tasks in the station, not closely related to the interlocking system (e.g. track-point heating subsystem, air-line powering system control etc.).

The project has come to a successful end and 18 pieces of the target product, called “electronic control desk”, have recently been put into operation at the railway stations of both Hungarian railway companies (Várnai, 2001).

Concerning hardware, the electronic control desk is a complex of modular industrial microcomputers and high end commercial PCs. The structure of the equipment is shown in *Figure 1*.

The architecture of the electronic control desk has been made to satisfy simultaneously and in conformity with the requirements of the functionality, the safety and the availability.

There are two levels of the electronic control desk according to the differences of their functions and due to the different locations where the functions are realised:

- the terminal (RTU) level, that is a modular, industrial microcomputer, connected to the I/O-s of the relay system, installed typically into the relay room of the station and assigned for the primary processing of the site information;
- the workstation (WS) level, that is a high

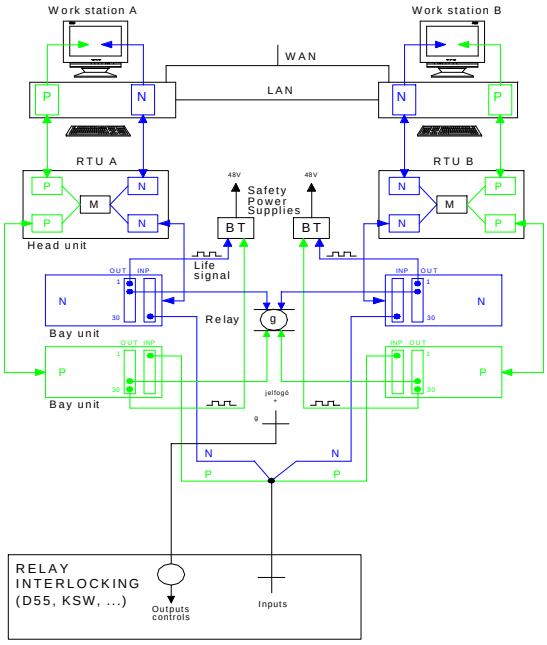

Fig. 1. Hardware architecture of the electronic control desk

end PC with LCD monitor, keyboard, track-ball, and printer, connected serially to the RTU level and through LAN/WAN to the regional traffic supervisory and control centres.

For meeting the requirements of the safety, the processing of the signals coming from the railway technology is redundant within an RTU, i.e. a single information bit of an object generates two inputs (true and false) for the RTU. In the same way, an output control can only be generated if two output channels operate simultaneously but with opposite direction. These two output signals are connected to the coil of a relay providing "END" function and outputting the real and safety output on its contact if the coil is energised.

The 2 out of 2 safety criteria is realised on the workstation level of the electronic desk as well, due to the doubled database and the simultaneously but independently processed data. The man-machine interface functions (display and control) can be executed only if the results of the parallel processing, the databases are equivalent.

The total hot-standby feature of the system extended to both levels and realised by a complete duplication fulfils the requirement of availability.

3. SOFTWARE ENVIRONMENT AND FUNCTIONS OF THE MODULE

The programs of the RTU-s run under a fast and simple real-time operating system (ROS 186). The software of the workstations is based on Linux that makes the safety version of XGRAM SCADA system running.

A simplified block diagram of the WS software is shown in *Figure 2*.

One can see from the diagram that the function of permission-managing is not among the real safety critical parts of the software, like e.g. controls, though inappropriate handling of train movements could cause delays in the traffic

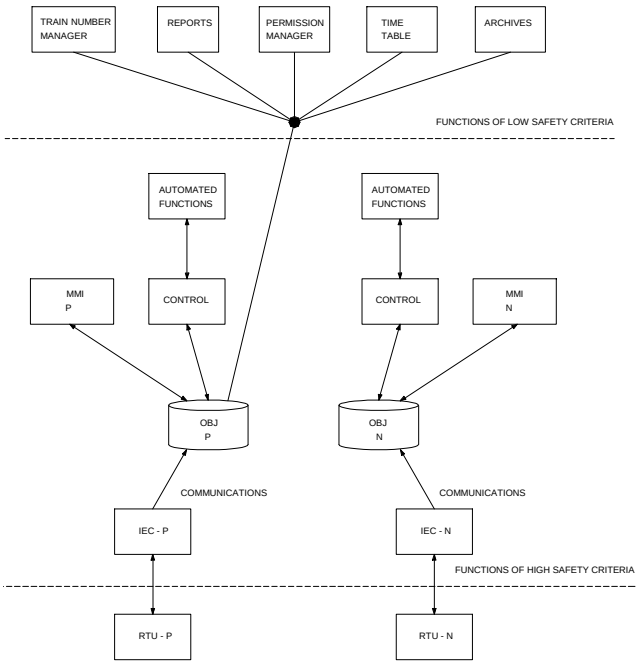

Fig.2. Block diagram of the WS software

of passengers and goods.

The permission-managing function can operate only between neighbouring stations, which are equipped with the electronic control desk and the stations are hooked up in a WAN.

In case of traditional equipment, permission request for starting a train is carried out by voice communication of the personnel through telephone lines. No train can be started from a station without the oral permission of the neighbour.

The whole process of requesting and granting the permission of train movements is strictly governed by the requirements of "F2" traffic regulation of MÁV, the Hungarian State Railways. When changing from the traditional verbal solution to the computerised version, the instructions of F2 had to be considered as functional specification of the permission-managing module.

A permission request telegram from the station of departure should include

– the train number,
– the mass, length and other characteristics of the train,
– the track, selected for the traffic, (right or left),
– the planned departure time.

The station of arrival can either grant or reject the permission. Permission can be granted with limitation or modification.

The personnel of the station can activate the permission-managing function on the screen of the workstation by clicking on the icon of "Permission". Blinking characters and sound effects warn the operator if a permission request arrives. Different colours of the train numbers signalises on the screen if a train

– has not yet requested to start,
– is under the permission requesting process,
– has got the permission to start,
– has been rejected by the neighbour.

The permission-managing module is closely connected to another module that handles the train numbers. A permission request can be issued only if the train to be started has a valid train number and it is staying at the station.

The permission request and grant procedure has to be continued either with the acknowledgement, or with the cancellation of the request reply of the requesting party. Finally the communicating computers will make an end of the process automatically, acknowledging the arrival of the train at the neighbouring station.

The module generates a complete report on the whole procedure of permission requesting and granting, including all information of the timing, the train and the personnel involved.

4. UML-BASED MODELLING

Several UML diagrams of the selected module were elaborated by using Rational Rose as a UML CASE tool. Use case diagrams were applied to describe the main functions. Use cases were refined by sequence diagrams depicting the messages among the entities of the system. The static structure of the module was specified by class and object diagrams, while the dynamic behaviour of the classes were given by statechart diagrams. As an example, we present a class diagram containing the classes belonging to the conceptual model of the communication subsystem (*Figure 3*). Main system components are the Course and the Station. The graphical

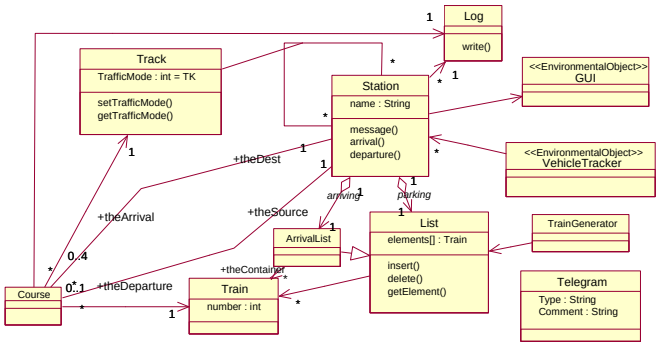

Fig. 3. Class diagram in the initial model

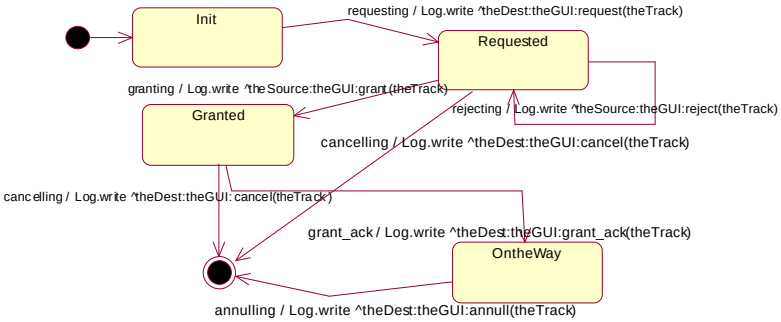

Fig. 4. Statechart diagram of the Course object

user interface (GUI) of the railway traffic control system (installed in each station) is a main environmental object.

In *Figure 4* the statechart diagram of the Course class is depicted which implements the logic of the handshake protocol between two stations. The source station requests a permission, which can be rejected or granted by the destination station. Permissions may be cancelled or annulled by the source station.

5. ANALYSIS OF COMPLETENESS AND CONSISTENCY

The implementation and even the detailed formal analysis of the UML model of the target system should be preceded by basic checks of the completeness and consistency of the model. These kinds of checks are motivated by experience showing that most of the safety-related malfunctions and accidents caused by computer programs occur due to flaws in the specification; mainly because of its *incompleteness and inconsistency* (Heimdahl and Leveson, 1996). The general well-formedness rules of UML (often implemented in CASE tools) are not enough to guarantee the correctness: they allow non-determinism, ambiguous "default behaviour" can be assumed etc. The specification flaws occur most probably in the statechart model, since it is the most complex view of the system (defining state-based behaviour) and its sophisticated constructs like hierarchy of states, concurrent sub-automata, priority relations among state transitions etc. are not easy to be handled by the designer.

In the literature, 47 safety criteria were collected as a general checklist for the specification of software systems (Leveson, 1995). These criteria can be grouped into several categories like state space completeness, input variable completeness, trigger event completeness, robustness, non-determinism, time and value limits. Although the criteria were specified for manual review, the most important group of criteria, the consistency and completeness rules can be checked mechanically. Indeed, in this category it is crucial to have mechanical tool support, because the manual checking of the behaviour of a complex system including hundreds of states and thousands of state transitions is error-prone and extremely time-consuming.

We have implemented tool support for the checking of the following criteria:

1. Exactly one initial state exists in the model and it is stereotyped as safe.
2. The system is deterministic: Two transitions starting from the same state and triggered by the same event are not enabled at the same time. Moreover, in concurrent sub-automata (pairs of) actions are not triggered by the same event (otherwise their execution order is non-deterministic).
3. The system is completely specified: For each state, possible trigger event and guard condition, there is a transition defined.
4. Each state is reachable (by state transitions).
5. Timeout transitions are defined in each state.
6. There is no transition in the model that is continuously disabled by another transition with higher priority. There is no transition with a guard that is always false.

These criteria can be verified by *static checking* directly on the model, i.e. without the construction of the global reachability graph of the system. This way state space explosion due to concurrent execution of sub-automata can be avoided thus the time and memory requirements of the checking can be drastically reduced.

The checking of these criteria is complicated by the hierarchic and concurrent nature of the statecharts, e.g. sub-states "inherit" the state transitions of their parent states (which has to be enumerated during the checking). Accordingly, before executing the checking, the statechart model has to be transformed to a *reduced form* which is a flattened model without state hierarchy and it contains only basic elements like states, events, transitions, and actions (Pap *et. al*, 2001). Moreover, the guard conditions are converted to a canonical form. The model reduction and the subsequent checking are performed automatically by approximately 60 rules in a general-purpose graph transformation system (Csertán *et al.*, 2002). The results of the checking identify the model elements where the rules are not satisfied.

Our checker is independent of the applied UML CASE tool since it is based on the standard XMI (XML Metadata Interchange) format of UML models. Accordingly, any tool supporting XMI can be used.

The checking of the statecharts of our example required 2-5 minutes and highlighted specification flaws like typing errors, malformed guard conditions and missing transitions. These flaws were corrected and the analysis was repeated until a complete and internally consistent model became available.

6. FAULT MODELLING

Usually, the UML models used in system design describe the behaviour of the system in the fault-free case. However, several analyses like testability and error propagation analysis require modelling the behaviour also in the presence of faults. Fault modelling includes the description of the behaviour of the components in faulty cases and the description of the error propagation among the (fault-free and faulty) components as well (Pataricza, 2002). We ask the designer to describe the effects of local faults (explicit fault modelling) and assume non-deterministic behaviour when an erroneous input is received (implicit fault modelling).

We use stereotypes (the standard extension mechanism of UML) and modelling conventions to identify the additional aspects as follows:

Description of the erroneous behaviour(s). In each component, error-free and erroneous behaviours are integrated into a single statechart. Modelling the effects of local permanent faults necessitates introducing additional states, events and messages. Erroneous states are identified by the stereotype <<FaultModel>>. Since we intend to support the analysis of testability and error propagation, our approach does not cover the modelling of internal transient faults and repair actions.

Completing the model by the handling of unexpected messages. When modelling erroneous communication one have to consider the reaction of receiver components to unexpected messages. If a message could arrive to the target both in erroneous and error-free case then the reaction is already described (the

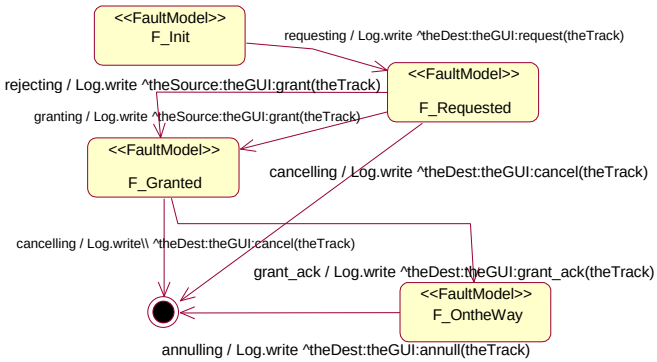

Fig. 5. Model of an erroneous behaviour of the Course object

completeness of the error-free description has already been analysed). Otherwise by default a non-deterministic state transitions is assumed. When the receiver is to interpret invalid messages, then the corresponding behaviour (e.g. exception handling) has to be modelled explicitly. If the target must not interpret a given message, then the name of the message has to be changed to InvalidMessage.

Labelling of components that cannot be confused by invalid messages. In case of some objects (e.g. hardware objects having a strictly restricted set of inputs) there is no need to model the effects of invalid messages. These objects are identified by the stereotype <<InvalidProof>>.

Labelling of error correcting transitions. When analysing error propagation, the analysis tool distinguishes between faulty (direct or indirect result of the erroneous operation of a component) and non-faulty messages. (This distinction, which is not apparent on the UML level of modelling, is not to be mixed up with the distinction between correct and invalid messages discussed above.) The automated processing requires the identification of the error correcting actions of the objects. The transitions that implement a correction are to be labelled with the stereotype <<ErrorCorrecting>>. This means, that the message sent by the given transition is non-faulty even if the triggering one was a faulty one.

Distinguishing environmental objects. From the point of view of error propagation the interface between the system and its environment is of central interest. UML models are usually open models containing no description of the environment. However, the subsequent analysis steps (Section 7) require a closed model where each input and output message has its source and target, respectively. Accordingly, the modeller has to include environmental objects labelled by stereotype <<EnvironmentalObject>>.

The fault modelling approach is illustrated in *Figure 5*. This statechart diagram extends the error-free behaviour presented previously in *Figure 4* by including the model of a potential erroneous behaviour. The diagram models the case when the handshake protocol is failed due to an erroneous control flow (in the event handler) that interprets rejection as granting. The UML stereotype <<FaultModel>> is used to distinguish the erroneous states.

Typically, statechart level extensions are used to model the effects of physical faults occurring in the used resources.

7. ANALYSIS OF TESTABILITY AND ERROR PROPAGATION

The enriched UML model (containing the fault model) allows, among others, the following two kinds of analysis:

– *Testability analysis* shows whether the faulty components can be identified by testing, i.e. by applying specific test data to the input of the system. This analysis characterises a fault as being testable (if its effects are observable on the primary or test outputs) or as being diagnosable (if its effects are specific, i.e. the fault can be identified on the basis of the outputs). In a properly designed system all faults are testable and diagnostic resolution correlates with the smallest replaceable (repairable) units. The analysis requires test generation and the simulation of the behaviour of the system under test in the case of the anticipated faults. In this way the quality of a test set in terms of coverage and redundancy can be estimated as well.

– *Analysis of error propagation* is used to asses the effects of local faults to system service (outputs) during operation. In this way propagation paths without proper error detection (or fault tolerance) and catastrophic consequences of a fault can be examined. This kind of analysis is performed by (parallel) simulation of the faulty behaviour.

Test generation and simulation are complex and time-consuming tasks. We were looking for a method that supports abstraction and successive refinement. Abstraction is needed to simplify the analysis tasks, moreover, it is required in the early phases of the design when a precise behavioural model is not available (but the architecture has to be established and the propagation paths and the error detection have to be estimated).

The basis of these analysis steps is the transformation of the UML model to a *data flow network*. Objects of the UML model are mapped to nodes while links among them are mapped to unidirectional channels connecting these nodes. In this representation, the fault effects and their propagation appear similarly to the flow of data in the functional model (Csertán, 1997). Tokens representing the data can be marked ("coloured") as correct or faulty. A set of error propagation paths can be estimated by tracing the token flow from the faulty component (data flow node) to the outputs of the network. Conditional error propagation can also be modelled.

This kind of tokenised approach of modelling data has several advantages.

- It supports abstraction. In the early phases of the design, when no precise behavioural description is available, modelling by using tokens like "good data", "corrupt data" and "missing data" already allows to estimate propagation paths and the required error detection. Later, as the behavioural diagrams will be available, more sophisticated classes of tokens can be defined. The refinement of the model can be driven by the analysis results available in the more abstract phase. This kind of abstraction delivers a superset of the propagated faults, thus in the model all potential consequences of a fault can be estimated.
- Data flow networks allow relatively simple simulation and test set generation. The algorithms well known in the logic gate design, e.g. PODEM (Goel, 1981) can be easily adapted and implemented (Csertán, 1997).

In our framework UML models of the system are transformed to data flow networks automatically, by using the graph transformation system mentioned in Section 5. The test set generation and simulation are performed on the resulting data flow network and the results of the analysis, like the test set, the set of faults that cannot be tested, and the effects on the outputs are reported to the designer in terms of the UML model elements. The data flow representation is available for simulation-based debugging purposes.

8. CONCLUSIONS

UML based design of a system enables formal analysis through mathematical formalisms like automata and data flow networks. We have implemented (1) a set of automatic model transformations in order to avoid the manual re-modelling of the system for analysis and (2) methods to perform the required analysis. In this way completeness and consistency of the specification, testability of anticipated faults and effects of error propagation could be established.

The formal analysis methods were applied successfully to a specific module of the railway control software. The early detection of the safety-related specification flaws and testability deficiencies contributed to the reduction of the costs related to the necessary corrections in later design phases or before the acceptance procedure.

Our future work will include the analysis of the *logic correctness of the control flow* by model transformations from UML statecharts to the input formalisms of model checker tools.

ACKNOWLEDGEMENT

This project has been supported by the Hungarian Ministry of Education under grant IKTA 065/2000 which is gratefully acknowledged.

REFERENCES

Csertán, Gy. (1997). A Framework for Early Testability Analysis. *Ph.D. thesis*, Budapest University of Technology.

Csertán, Gy., G. Huszerl, I. Majzik, Zs. Pap, A. Pataricza and D. Varró (2002). VIATRA - Visual Automated Transformations for Formal Verification and Validation of UML Models. *Proc. of the 17th Int. Conference on Automated Software Engineering (ASE 2002)*, pp 23-27.

Goel, P. (1981). An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Trans. on Computers*, 30(3), 215-222

Heimdahl, M. P. E. and N. G. Leveson (1996). Completeness and Consistency Checking of Software Requirements. *IEEE Trans, on Software Engineering*, **22**, (6).

Leveson, N. G. (1995). Safeware: System Safety and Computers. Addison-Wesley.

Object Management Group (OMG). Unified Modeling Language (UML) Specification v1.4. 2001. http://www.uml.org

Pap, Zs, I. Majzik, and A. Pataricza (2001): Checking General Safety Criteria on UML Statecharts. *Proc. SAFECOMP 2001, Budapest, Hungary*, 46-55, LNCS-2187, Springer Verlag.

Pataricza, A. (2002). From the General Resource Model to a General Fault Modelling Paradigm? *Proc. Workshop on Critical Systems Development with UML*. Dresden, Germany.

Várnai, Gy. (2002). Regional Remote Control of Railway Stations' Interlocking System within the Traffic Control System of GYSEV. Vezetékek Világa, 2002/1 9-12.