Checking Safety Criteria under UML

Ph. D. thesis

Zsigmond Pap

Technical University of Technology and Economics
Department of Measurement and Information Systems

2005

Consultant: Dr. András Szegi
1. Premises and goals

Nowadays more and more software-controlled programmable units are used in safety critical applications. Additionally, the complexity and size of these control software exponentially increases, so the classic safety checking methods as the informatics’ version of HAZOP, or FMEA ([7]), are less effective. On the other hand, according to Leveson’s study, 90% of accidents is caused by programmed units because of software malfunction.

According to the literature, these faults are caused typically by three reasons:

1. Software specification is incomplete or inconsistent
2. Coding error then the program
3. (Human) Operator error (typically due to a user interface error).

Leveson’s study shows that, mainly the first and the third reasons are the most dangerous. Using automatic program code generators, automated validation and verification techniques, the resulted program certainly meets the specification. But if this specification is incomplete or inconsistent, the result will inherit this too. Incompleteness and faultiness includes all user interface properties too, which finally leads to human operator error. Since this, Paterno reckon user interface problems among specification errors [6].

During analyzing accidents, Leveson created a set of criteria of which can help to decrease probability of faults. These criteria are formed in natural text, like a rule of thumb.

My first goal is to update and convert these criteria to modern software, and if necessary, to extend or modify these.

Secondly, I should check how these criteria are usable to make automated specification checking. Important viewpoint, that this specification checking cannot use unacceptable amount of time and processing resources (like for example SPIN [3] does), yet in large specifications too.

There is another method to decrease specification error. This method is, when the field expert cannot make erroneous specification, since the tool or environment does not allow this. In this case, the program designers are forced to keep Leveson’s criteria. Naturally, this requires additional support from the modeling tool.

My third goal is to find a way to force designers to make complete and consistent specification. This can be done using a new and adequate formalism, or by extending an existing one. Very important to be compatible with the existing and widely used formalism to avoid learning a completely new technique.
Expedient to examine how these criteria support error checking and avoiding, and which already existing modeler tool can be used.

The effect of some specification error is indirect, for example the effect is an operator error. Typically these are user interface errors. Paterno, Leveson and Schneiderman demonstrated the most general fault reasons:
- Misunderstanding of some kind of indication or operation.
- Exceeding limits of human perception and reaction limits.
- Unexpected side effects or operation modes.
- Inconsistent operation.
- Missing of intervention right.

These are typical specification errors. Unfortunately there is no really well usable method in the literature to avoid these problems.

My fourth goal is to find method to avoid user interface specification errors, and its consequences. This method should be well usable in very different application areas.
2. Checking methods

To make automated verification, at first the specification must be formalized. To formalize specification the UML is chosen, because nowadays it is a widespread de-facto industrial standard.

The most complex part of a UML specification is the State chart (which is originally based on David Darel’s formalism [2]), and the functional part described by action semantics. Typically, most of the errors are found here, so these two formalisms are the most important. Unfortunately during the standardization of UML there was no focus on safety, so at first the UML formalism itself should be checked from the point of view of safety.

A UML specification essentially is a special graph, specified by UML metamodels. Following the graph-based approach, checking a specification model or converting to another form is a graph-transformation. The graph-transformation is a visual, easy-to-understand, and very flexible method. Additionally, since most of the criteria can be formalized as graphs, this can be a common base for the whole process.

The graph-transformation essentially is a substitution-modification, which can convert from one formalism to another one using rules and rule-sequences. As another benefit, the rules are graphically self-documented, while the formalism allows mathematical proofing of properties of transformation.

The UML metamodels [11] can also be handled as graphs, which makes possible to build a UML safety sub-language on the basis of the original metamodels.

Checking, and sub-language building is possible not only in the case of State charts, but in the case of Action Semantics too. Unfortunately Action Semantics is not yet supported by most UML modeler tools.

The method of defining sub-languages can be extended to Class diagrams too. The result is design pattern [5], which is essentially a well-documented template. Using design patterns most of the safety requirements are effectively formalizable, and usable in practice, but after applying a design pattern to an application, its safety properties, its “warranty” is unknown.

However using UML sub-language patterns, these properties remain checkable, for example using OCL clauses.

Design pattern together with OCL clauses makes possible design a “safe” user interface template, which can keep Leveson’s criteria. This user interface design template is based on the Model-Controller-View architecture.
3. New scientific results

1. A method is defined to check the behavior of software specification or model from the point of view of safety in a formal, automatic way using UML statecharts.
   a. 3 additional criteria are defined to extend the original 47, which describes:
      i. Inter-component timing rules
      ii. Parallel processing criteria
      iii. Event-handling and sequencing rules.
   b. Criteria are formalized for the specification core:
      i. Using OCL language
      ii. Using graph-grammar
      iii. Using linear temporal logic
   c. Temporary description system is defined, called Reduced Form, which:
      i. Is orthogonal (neither modeler item can be formed using another items)
      ii. Keeps the safety properties of the original model, but does not generate new ones (exceptions defined).
      iii. In which there are no chained constructions in the model, so every connections of two elements can be matched in one step.
      iv. Unambiguously convertible from the original model. Transformation rules are defined to convert State chart model to Reduced Form.
   d. I made the described system usable in practical systems, and this is proofed by writing an experimental checking tool. This tool:
      i. Can handle more, than 100000 graph points
      ii. Runtime does not increase exponentially with the size.
   e. Rules are defined to handle inheritance in an object-oriented safety-critical system, to increase checking efficiency.

2. An alternative state chart model is defined based on UML state chart metamodel, which can structurally avoid safety- and checkability problems.
   a. Most of safety criteria are generalized to the metamodell-level of statechart, so the problematic constructions can be eliminated.
   b. Safety State chart Metamodel is Defined, which:
      i. Does not contain iterate-only constructions (such as chained list).
      ii. Uses OCL clauses to define validity.
      iii. Is unambiguously generable from every proper models.
iv. Is unable to describe unsafe model.

v. Using managed definition guarantees the consistency of the model.

3. A new method is defined to keep application-specific criteria, and to check it. Generally these criteria define structural rules (not behavioral), where strict verification is impossible.

a. I defined an item set, which structurally meets the application-specific criteria. Actually this is a possible version of object-system definition of safety criteria, which:
   i. Contains watermark to help safety conformance test.
   ii. Consist of elements and it’s associations defined by structural safety criteria.
   iii. Generally usable.

b. Structurally not defined criteria are formed to OCL rules. This set of rules allows additional safety checking.

c. To help using the model, I defined some general program parts missing from the UML as design patterns:
   i. Data-handler unit, which can handle data validity time
   ii. Event sequencer and handler unit, which meets the event handling safety criteria.
   iii. Driver program, using safe inter-component timing method.

d. A safe user interface design patterns defined, which:
   i. Decreases operator error possibility
   ii. Based on generally used MVC (Model-View-Controller) architecture.
   iii. Meets general- and structural safety criteria and the most important ergonomic rules.

4. The safety problems of UML Action Semantics is demonstrated, as:
   i. Mutual access control is not specified
   ii. Parallel operation is not defined
   iii. Guarantee cycle-free operation is not defined.

b. I extended the formalism with safety rules

c. I gave a method to check specification completeness and consistency, using graph-transformation technique.
4. Using results

The specification checking system is implemented in the VIATRA visual graph-transformation system. It is designed to suit to the philosophy of project HIDE [1]. The goal of this project is to give an easy-to-use tool to the software developers, while hide deep mathematics. This philosophy enables using complex methodology in a simple way.

Since this, all transformation- and checker program modules are extended with a graphical user interface. The program reads the model from the UML development tool using standardized format, checks it, and generates an error list.

This system was used in two independent real embedded systems during an IKTA project: a railway permission requesting system (PROLAN RT), and a dialysis machine (B-Braun Medical Hungary kft.). In addition, this checker tool was used in a fire-alarm system, and in a roof-cooler sprinkler system. Both are developed by Meldetechnik kft.

Experiences show that, every real specification made with care contains 4-10 (depends on size and complexity) sly faults. Identification of these if very difficult, still reading the checker tool’s error list. Without checking, these faults would remain hidden.

Although this checker tool made to find safety-critical errors, well usable to find other hidden errors too; this makes the checker universally usable.

Safety UML metamodel is not yet implemented in real systems, because general UML modeling tools are not able to handle UML dialects (expect stereotype-based). Since dialects are part of UML2, sooner or later these will be implemented, and make my system really usable.

There are more embedded applications, which implement the safety design patterns and safety user interface. The two largest are the above fire alarm center and a building-inspection system’s data transmitting driver program. The second – mainly data handler program – uses the data handler pattern and the event queue, while in the fire alarm center main software embed the safe user interface. Because the fire alarm equipments have special requirements, the user interface is very complex.

The practice show that, these design patterns are readily used not only by safety experts, but by normal developers, since they save a lot of work using pre-generated structures. As a side effect, these developers “inherit” safety rules too.
5. My Publications according to safety

Vet journal articles:


Vet conference articles:


Non-vet conference articles:

    CSCS'2000, 2000, Szeged

    ISBN 963 42 682 4

12. Zs. Pap: Specification Completeness and Consistency Checking under UML. 

    Mini-symposium- 2000, Budapest, pp. 6-8

14. I. Majzik, Zs. Pap, A. Pataricza: Completeness and Consistency Analysis of UML 
    Statechart Specification, Dagstuhl szeminárium, 2000, Uni. Erlangen

Technical reports:

15. Pap Zs.: Biztonsági kritériumok ellenőrzésének technológiája, I2RT, Budapest.

16. Pap Zs.: Biztonságkritikus Felhasználói Felület tervezői mintája, I2RT, Budapest

17. Pap Zs. Biztonságkritikus UML, IKTA 065/2000, Keretrendszer nagymegbizhatóságú, 
    biztonságkritikus rendszerek fejlesztéséhez és teszteléséhez, Budapest, 2001 december
6. Bibliography


E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, Addison-Wesley, Reading Mass. 1994.


