**DRAFT**

Full paper published in the proceedings of

**10th International Multi-Conference on Advanced Computer Systems (ACS 2003), 22-24 October 2003, Miedzyzdroje, Poland**

# Tool support for detecting defects in object-oriented models

JANUSZ GÓRSKI, ALEKSANDER JARZĘBOWICZ, RAFAŁ LESZCZYNA, JAKUB MILER, MARCIN OLSZEWSKI

*Department of Software Engineering, Gdańsk University of Technology*

*Narutowicza 11/12, 80-952 Gdańsk*

*e-mail: {jango,olek,r.leszczyna,jakubm,olszes}@eti.pg.gda.pl*

Abstract: Object-oriented models are commonly used in software projects. They may be affected, however, by various defects introduced easily due to e.g. wrong understanding of modelled reality, making wrong assumptions or editorial mistakes. The defects should be identified and corrected as early as possible, preferably before the model is used as the basis for the subsequent representations of the system. To assure the effectiveness of the defect detection process we need both, better analysis methods and effective tool support. The paper introduces a new analytical method called UML-HAZOP and presents a tool supporting the application of this method.

Key words: UML, defect detection, inspection, object-oriented modelling

## 1. INTRODUCTION

During a software development project multiple products are created including, apart from code, the documentation of the system and of the development process. This documentation takes different forms depending on the chosen development cycle model, the project management decisions, the supporting tools used etc. There are, however, a number of products that are common for most projects. First of all they include a (logical) sequence of subsequent representations of the system, starting from the requirements specification and ending at code. Among these are system models of different level of abstraction (business, analytical, design). It is a present tendency that such models are expressed in object-oriented terms.

Software products are not free from defects. A defect can be introduced during any phase of the development process and then it remains in the product of this phase. If the defect is not detected and removed at the end of the phase, it is passed to the products of the subsequent phases. A typical example of such propagation is a defect in the requirements. If goes undetected, it will affect all other representations, including the code. Experience shows that the cost of finding and removing a defect increases significantly if the defect goes undetected through a number of development phases. This causes a strong motivation to detect and remove defects as early as possible, preferably at the same phase where they were born. To fulfil this expectation we need adequate analytical methods, capable to detect defects in various software representations, including those that are created early in the development process.

Object-oriented models increase their popularity in the software development community. Such models are built early in the development process and therefore they are of particular interest from the defect detection standpoint. Partially formalized syntax of such models provides for more precise definition of the criteria and rules of analysis than it was possible with respect to the "raw" or partially structured text which is found in more "traditional" software project documents. The advent of the Unified *Modelling Language*

(UML) which is becoming *de facto* standard for object-oriented modelling increases the motivation in researching towards new analytical methods of UML models because of their potential applicability to many software projects.

Defect detection in documentation usually takes a form of a review. Such reviews can take different forms depending on process definition, scope and level of formalisation. The reviews are supported by checklists that are used to store information about the most common defects. Such checklists can be developed in advance by systematic analysis of the language (UML) and the resulting modelling constructs.

In this paper we propose a new method of analysis called *UML-HAZOP*. Its brief description is presented in section 2 (a more thorough description can be found in [2]). The analysis of software models is an expensive process which requires the participation of humans and consumes significant resources. To decrease this effort, the application of the method should be automated, to as great extent as possible. The discussion of the scope of automation is presented in section 3. The conclusions from this discussion were used to design and implement a software tool supporting the application of UML-HAZOP. The paper presents the tool, its functionality and the support provided to particular activities of the method (section 4), as well as its architecture and technologies used in the implementation (section 5). We conclude by recalling some experiences with using the method and present plans for the future development.

# 2. UML-HAZOP METHOD

While building an object-oriented model, the most essential class of defects is related to incorrect representation of the modelled reality (resulting from e.g. wrong understanding of the modelled problem or making incorrect assumptions). Syntax errors are also possible, but their detection or prevention (e.g. by the mechanisms built into a CASE tool) is much easier, so we consider them as less interesting. We assume that the main focus of the analysis should be on the semantics of the model, not on its syntax. The essence of the modelling is to describe a part of reality using elements of a particular notation. By checking if a particular model element is used correctly (in the sense of proper representation of the reality being modelled) we can help in detecting defects that are present in the model. It is possible to systematically generate such checks and to store them in a form of checklists. The development of such checklists is not an easy task, however.

In the proposed approach we use HAZOP (*Hazard and Operability Study*) to support a systematic generation of checklists. HAZOP is an analytical method originally developed to detect hazards in safety critical systems. The method originated in the petrochemical industry and was adapted to software engineering in early nineties [3]. The method (in its original form) concentrates on systematic reviewing of the connections present in a system model. In the petrochemical domain, these connections are the pipes included in a chemical installation. In the software domain “connections” can be interpreted in different ways – it could be a physical communication channels but also a logical connection e.g. the relationship on a class diagram or the transition on a state diagram.

The idea of the method is very simple. For each reviewed connection, its attributes are first identified. An attribute is a well defined, atomic part or property of the connection. Then the method makes use of guidewords - a short phrases suggesting errors and deviations from the correct state of the connection e.g. NO, LESS, OTHER THAN, LATE. The application of the guidewords to the attributes results in the list of hypothetical deviations related to that connection.

UML-HAZOP method is an adaptation of HAZOP to the analysis of UML diagrams. At present, we have a complete set of checklists related to connections in class diagrams. The checklists for other UML diagrams are under elaboration. During the analysis of a class diagram the connections are e.g. the relationships between classes, in particular *Association* and *Generalization*. These relationships have a number of attributes e.g. *aggregation*, *name*, *multiplicity* etc. The guidewords form the set given in Table 1 are applied to the relationships and their attributes to suggest possible deviations. Not every suggestion gives a sensible result, but in many cases they refer to potential defects. The example interpretations related to the *aggregation* attribute are given in Table 2. All sensible interpretations (for the attributes included in the analysis) form so called *HAZOP table* for a given type of relationship e.g. *Association*. The analysis of an UML diagram is carried out by applying the HAZOP table (the checklist) to each relationship represented in the diagram.

| Guideword | General interpretation |
|---|---|
| NO | The complete negation of the design intention. No part of the intention is achieved and nothing else happens. |
| MORE | A quantitative increase. |
| LESS | A quantitative decrease. |
| AS WELL AS | All the design intention is achieved together with additions. |
| PART OF | Only some of the design intention is achieved. |
| REVERSE | The logical opposite of the intention is achieved. |
| OTHER THAN | Complete substitution, where no part of the original intention is achieved but something quite different happens. |
| EARLY | Something happens earlier than expected relative to clock time. |
| LATE | Something happens later than expected relative to clock time. |
| BEFORE | Something happens before it is expected, relating to order or sequence. |
| AFTER | Something happens after it is expected, relating to order or sequence. |

*Tab.1.* Generic HAZOP guidewords

| Guideword | Interpretation |
|---|---|
| NO | The aggregation that should be present is not marked in the diagram. |
| AS WELL AS | The aggregation is wrong and should be removed. |
| OTHER THAN | Wrong aggregation type is marked i.e. strong instead of weak or inversely. |
| MORE | Too many classes are aggregated; some of them should not be included in this aggregation. |
| LESS | The aggregation does not include some classes (present in the diagram or not) that should be aggregated. |

*Tab. 2.* Interpretations for *AssociationEnd.aggregation* attribute

The UML-HAZOP analysis is performed within a defined and managed process. The structure of this process is shown in Fig.1.

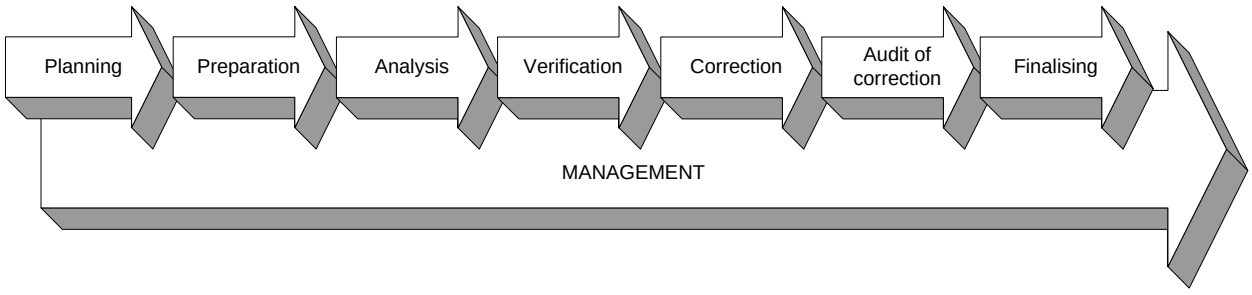


*Fig. 1.* Structure of UML-HAZOP analysis process

The process begins with the decision that a given UML model should be subjected to the UML-HAZOP analysis. Planning of the overall process is the first step and includes schedule preparation, assigning participants to tasks and selecting the checklists for all connections that are present in the model. The preparation phase consists of acquainting the analysts with the project documentation and with the method of analysis (if it is not already known). In the analysis phase, a systematic review of the model with use of checklists is performed and all detected defects are registered. Next, the list of defects is passed to the author of the model for verification (he/she accepts or denies their presence). The accepted defects are then corrected by the author during the correction phase. After that, an audit takes place to ensure that all defects have been removed. During the finalising step, the process is formally closed, the final product (the corrected model) is released and the process statistics (the number of detected defects, efficiency metrics etc.) are collected. To ensure its smooth realisation the process has to be managed by a person assigned to this task.

The detailed description of the method and its experimental results can be found in [2].

# 3. SCOPE OF AUTOMATION

As UML-HAZOP heavily involves human judgement the extent of its potential automation is limited in scope. The decision, if a potential defect suggested by a checklist does really occur in the analysed diagram, often requires the use of additional knowledge, which is not directly expressed in the model and has to be derived and interpreted by the analyst. If the analysed models represent a single domain then such interpretations can be (partially) delegated to an "intelligent" tool as it was already proposed for chemical plants [4]. However, UML-HAZOP does not make any assumptions about the type of analysed systems nor the elements they are composed of. The method focuses on the features of UML – the language that can be used to model any system. Therefore, the decision, whether a suggested defect is present in the model or not, requires a human expert intervention.

The part of the UML-HAZOP analysis process in which a human can be replaced by a tool is the preparation of HAZOP tables. It is a laborious task that can result in many errors (e.g. if some connections are mistakenly omitted). Having a tool taking as the input an UML diagram (e.g. created by a CASE tool) and capable of generating HAZOP tables could save much time and effort. The tool could take care for the completeness and correctness of the generated tables as well. The application of a simple algorithm to eliminate redundant items from the HAZOP tables (those that suggest impossible defects for a particular connection) could be another advantage. For example, if in a given diagram, aggregation is not used with the *Association* relationship, it is pointless to consider the defect addressing "wrong type of aggregation" (strong instead of weak or inversely).

The tool could also support the analysis by providing a user friendly interface to collect and store the analysts' decisions and comments and by generating various kinds of statistics.

Another important issue is a support for teamwork. A network based tool could provide for communication and information exchange between the participants of the analysis process without the necessity of gathering all of them in one place.

# 4. FUNCTIONALITY OF THE TOOL

## 4.1 Scope of functionality

The tool supports the following phases of the UML-HAZOP process: planning, analysis, verification and management.

In the planning phase the tool allows to create a new project, to read in the data about diagrams to be analysed and to generate, for each diagram, the corresponding HAZOP tables. It also supports the assignment of the users to the project and to their roles within the project.

Analysis and verification are entirely performed with the use of the tool. All users' decisions concerning defects are registered and stored by the system.

Concerning the management of the analysis process, the tool supports users management (assigning to project, exclusion from project, change of roles), projects management and diagrams management.

## 4.2 Users and roles

The system is accessible only for registered users. A user has to enter his/her password before beginning his work. Creating and maintaining user accounts is the task of the administrator. It is in fact his only task – the administrator does not participate in the analysis process and does not have access to any other data.

Distinct participants of the analysis process usually have different duties; this is why the tool supports a number of predefined roles:

- **Manager** - a person responsible for the whole analysis of particular project documentation. He/she opens and closes the project and decides about granting to the other users the access rights to the project documents. The access rights can be read/write (in case of Analyst and Designer) or read-only (in case of Observer).
- **Analyst** – a user who performs the analysis and inserts the results to the HAZOP tables.
- **Designer** – the author of the UML diagram subjected to the analysis. His task is to verify the results of the analysis and, if necessary, to correct the diagram.
- **Observer** – a user who reviews the results of the analysis for e.g. training or verifying purposes, but does not directly participate in the analysis process. Observer does not enter any data to the system.

Within a project it is possible to assign more than one role to a user (e.g. Manager and Analyst) and to assign a given role to many users (Manager is an exception – there can be only one Manager per project). The assignment of roles is first done when opening a new project, but later it can be changed anytime by the Manager.

## 4.3 Managing projects and diagrams

The information stored in the system has been structured into multiple layers. Each autonomous analytical task is called a *project*. Each project has a Manager and other participants assigned. A project encompasses a number (usually many) UML *diagrams*. Each diagram has its Designer (the author of the diagram). With each diagram there is a set of associated *HAZOP tables* concerning its connections or other elements that are the subject of analysis. HAZOP tables are generated automatically and the results of analysis and verification are registered in them.

A new project is created by its Manager. The diagrams are imported in an electronic form from a CASE tool (e.g. Rational Rose) and the corresponding HAZOP tables are generated automatically (see section 4.4). Projects as well as diagrams can be deleted together with all their contents. It is also possible to close and reopen them many times. A closed diagram or project is still stored in the system and is visible to the users (but no changes are allowed) – this way it is possible to "freeze" the current state of work.

## 4.4 Importing data from a CASE tool

The import of UML diagrams from a CASE tool (*Rational Rose* in our case) is achieved by the upload of the file containing the diagram. It is necessary to specify the path to the MDL file in which *Rose* stores the complete information about the UML model of interest. After the file is read, the system interprets it and extracts the list of all diagrams contained in the model. For each diagram, the information about its name, type and package it belongs to is given to the user together with the possibility to choose for which diagrams HAZOP tables should be generated. The user can give a new name to the diagram, in other case the original names (sometimes ambiguous) extracted from the Rose document would be preserved.

The following tasks are performed in an entirely automatic way. First the system reads from the *Rose* document the contents of each chosen diagram (in particular the data of all connections present in the diagram). Then using the built-in list of potential defects, a separate HAZOP table for each connection is created. The redundant rows of tables representing impossible defects of a given connection are identified and removed, so the user is provided with a set of HAZOP tables ready to be analysed.

## 4.5 Analysis and verification of HAZOP tables

When a user selects a diagram for further analysis, he/she is given a list of HAZOP tables generated for this diagram. Each table represents one connection (a relationship in case of the class diagram) and is described by its name and names of associated elements (e.g. classes). It provides for traceability between the table and the diagram elements. The task of the Analyst is, for each row of HAZOP table, to decide about the credibility of the defect suggested by this row. Those decisions are reflected by changing the colour of the row on the user display. This allows to notice, in an easy way, all the defects detected by the Analyst. The inspected rows of the HAZOP table can be also extended with comments giving the rationale for the undertaken decision.

The Designer is in charge of verifying the analysis results. The comments given by the Analyst can be helpful in performing this task. Designer is the author of the analysed model and his task is to accept or deny the decisions of the Analyst. For each defect suggested by Analyst, Designer confirms or rejects its credibility. The tool again uses colours to distinguish the rows with confirmed defects and those defects that have been denied.

Apart from taking decision, the Designer can write an additional comment e.g. to explain Analyst's doubts. Comments of the Designer are presented together with the Analyst's ones making a record of discussion about the inspected diagram.

The Observer has a possibility to review the current state of HAZOP tables. His/her user interface is similar to that used by the other participants, with the read-only access.

# 5. ARCHITECTURE OF THE TOOL

UML-HAZOP tool is a client – server Internet application. The server side is responsible for receiving, processing and storing the data as well as for assuring the concurrent access to its services. The client side is responsible for the interaction with a user. An important feature of the system is the integration with a CASE tool which is the source UML models to be inspected. The current prototype version of our system interfaces with *Rational Rose*.

All what a user needs to work remotely with the UML-HAZOP tool is a workstation equipped with an Internet browser and connected to the Internet. No dedicated software is used on the client side. Input data required for the analysis are MDL files (*Rational Rose* format). A Rose document is uploaded to the server from the user workstation (optionally after being compressed to ZIP format). It is then processed on the server to extract the necessary information. Although the *Rose* tool is not directly used in the analysis, it is useful for reviewing and correcting the original diagrams. Fig. 2 shows the configuration of user's workstation.

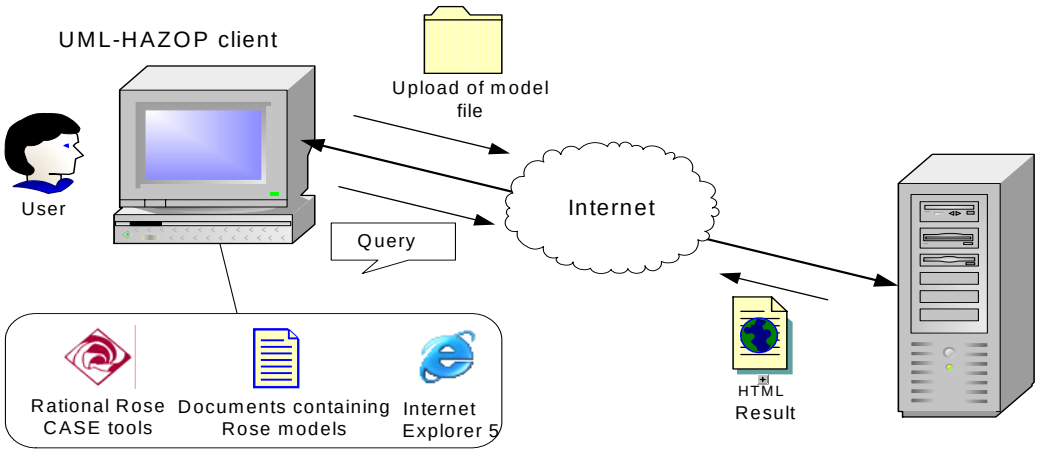


*Fig. 2.* Architecture of UML-HAZOP system from client perspective

The configuration of UML-HAZOP server is shown on Fig. 3. The main element of the system is a WWW server which processes users' queries. For security reasons, the communication is permitted only using HTTPS protocol. The WWW server works in a multi-threaded mode, so the concurrent access of many users is possible. The business logic of UML-HAZOP application is realised by *Active Server Pages* (ASP) in cooperation with *ActiveX* components and *Java* classes. The results of user's queries are returned by ASP scripts as HTML pages and sent by WWW server to client's workstation.

To ensure persistent and reliable data storage, a transactional DBMS system (SQL server) is used. The stored procedures encapsulate the internal data structure, making the use of defined interfaces the only way to get access to data.

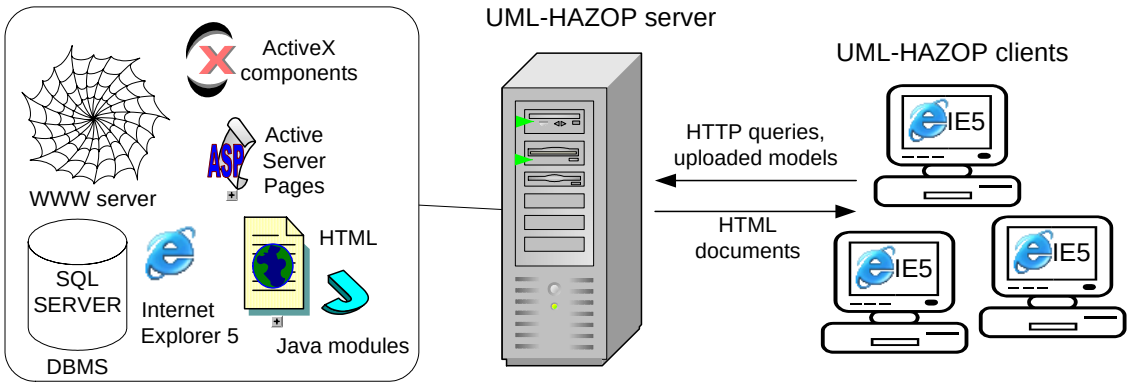


*Fig. 3.* Architecture of UML-HAZOP system from server perspective

Internal mechanisms of UML-HAZOP system are illustrated on Fig. 4. *Rational Rose* documents sent to the server by users contain data about UML models and diagrams expressed in a format used only by *Rational* products. To make integration of our system with other CASE tools possible, the introduction of an internal, intermediate data format was necessary. The purpose of this format is to form an interface between CASE tools using documents of different internal structure and UML-HAZOP tool data processing subsystems. We have decided to use XML language (and its standard interchange model – XMI), because it provides a complete representation of UML models and the processing of XML documents is supported by a number of Java packages available in the Internet. XML is also used in most of internal interfaces between UML-HAZOP tool subsystems.

The source data describing models to be analyzed is translated first to intermediate XMI format and then to the form of HAZOP tables, which are stored in the database. A user during the analysis process activities communicates with the WWW server and gets the visualization of current state of analysis as HTML pages. User's commands are received by the server as HTTP queries and translated by *Active Server Pages* scripts, which also update database contents if necessary.

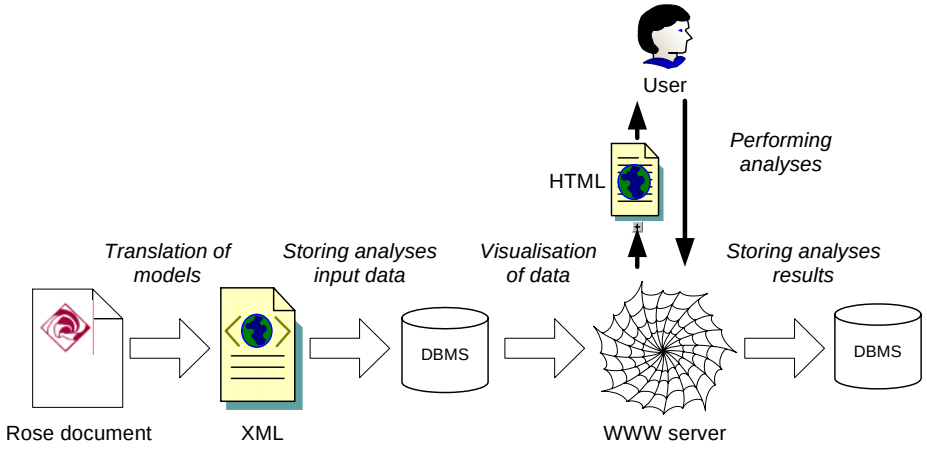


*Fig. 4.* Internal data processing of UML-HAZOP system

# 6. CONCLUSION

Development of UML-HAZOP and its supporting tool was a contribution to the EU 5th Framework Programme project DRIVE [1]. The project concerned the application of information technologies to support drugs distribution and application processes. UML-HAZOP method was applied to increase the assurance of software developed in DRIVE. It was used to analyze selected essential models e.g. the electronic patient record. The analysis results were included in the overall argumentation about the trustworthiness of the DRIVE solution (so called *trust case*) [6].

The tool served its purpose well. Its application resulted in the decrease of total effort related to the analysis process. The work spent on preparation of HAZOP tables was significantly reduced by the use of the tool. An improvement of the analysis and verification phases was also achieved due to the elimination of impossible defects from the HAZOP tables and the support provided by the user interface.

The client-server architecture and the use of Internet solutions provide for using the tool from remote locations. The possibility of cooperation and information exchange is especially important in distributed projects involving many partners. In our case, it resulted e.g. in performing analyses from Italy while some other analyses were performed at the same time from locations in Poland. The exchange of information between both sides was possible through the use of the tool.

The tool functionality is expected to expand with the progress of research on the UML-HAZOP method. The next step is to provide for analyzing the other types of UML diagrams. We will also experiment with other analytical criteria than those originally proposed by HAZOP (different selection of attributes and guidewords, different interpretations). Such criteria might originate from other methods derived from HAZOP (e.g. [5]). Various industrial case studies are being planned as well.

# 7. REFERENCES

[1] DRug In Virtual Enterprise, EU IST-DRIVE 1999-12040 research project, http://www.e-mathesis.it/Drive.

[2] Górski J., Jarzębowicz A., 'Detecting defects in object-oriented diagrams using UML-HAZOP', Foundations of Computing and Decision Sciences, vol. 27 (2002) no. 4.

[3] HAZOP Studies on Systems Containing Programmable Electronics, MoD Defence Standard 00-58, issue 2, 2000.

[4] Venkatasurbramanian V., Zhao J., Viswanathan S., 'Intelligent systems for HAZOP analysis of complex process plants', Computers and chemical engineering 24 (2000).

[5] Winther R., Johnsen O., Gran B., 'Security assessments of safety critical systems using HAZOPs', Proceedings of Computer Safety, Reliability and Security, 20th International Conference SAFECOMP 2001, Springer Lecture Notes in Computer Science 2187.

[6] Górski J., Jarzębowicz A., Leszczyna R., Miler J., Olszewski M. , 'An approach to trust case development', Proc. SAFECOMP 2003, Edinbourgh, UK, 2003