Towards Unified Dependability Modeling and Analysis

András Pataricza, Ferenc Győr

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Magyar tudósok körútja 1.
1117 Budapest
pataric@mit.bme.hu
gyf@iit.bme.hu

Abstract: Unified dependability modeling and analysis consists of both functional and non-functional modeling and analysis techniques. Nowadays one of the most popular modeling techniques is UML. Functional properties of an UML model can be validated and verified by existing modeling tools.

Checking of non-functional properties, like those related to dependability is of a growing importance while they cannot be easily derived from UML models. Despite the fact of the existence of a unified concept and terminology of dependability notions and mechanisms, little convergence is observable between the specific fields in dependability engineering. This paper presents a methodology for the uniform modeling of the different dependability related attributes.

1 Introduction

Dependability plays an increasingly important role in the assurance of the quality of services delivered by information technology systems. The objective of the paper is a mathematically sound modeling methodology extending UML to cover dependability properties as well.

Most of the functional properties in UML models can be checked by existing CASE tools with a growing support by formal methods guaranteeing the functional correctness of the target design. However, non-functional properties (including some very important aspects, such as security and safety) cannot be handled efficiently.

The paper is based on the simple observation, that different approaches addressing specific aspects of dependability use essentially the same algorithms for analysis of different parts and aspects of the model.

- For instance, faults in *software testability analysis* are associated with coding faults, and error propagation happens via invocation and inheritance [1].

- Faults are associated with the resources in assessment of the consequences of permanent or transient *hardware faults*, and error propagation originates in the interaction between the SW components and further extended by the interaction between different components previously affected by errors [2].

- Similarly, in *security analysis* interactions initiated at the interface points may propagate security and access right violations.

Additionally, the analysis methods are identical in many cases as well.

- The estimation of *damage confinement regions* necessitates the estimation of the transitive closure of the graph having the objects and resources as nodes, and their arbitrary connections as directed arcs starting from the node representing the fault site. This transitive closure as a cover for the objects potentially reachable from the fault site delivers a probable pessimistic estimate of the damage containment region.

- In *dynamic analysis of error propagation* (i) the model has to be extended by the transitions potentially occurring in a faulty system and subsequently (ii) the dynamic effects of faults are estimated by simulation or by an exhaustive traversal of the state space by model checking.

However, a contradiction exists between the uniform high level view of dependability and the actual practice in its UML based modeling and analysis.

- The IFIP WG 10.4 conceptually unified the different forms of appearance of the general notions of faults, errors, propagation etc. [4].

- Some papers already use this hierarchy to derive a uniform modeling concept [5], but no paper on UML-based dependability modeling defines and implements general algorithms covering all aspects of dependability in a uniform way.

2 Modeling concepts

Subsequently, the general approach will be referenced further to as meta-algorithms, i.e. general-purpose mathematical algorithms, which can be specialized in an automated way by meta-modeling based model refinement.

Traditional domain specific profiles focus only on elements *extending the target model* by the attributes needed for modeling of a specific aspect during model creation time.

Our proposal includes additional metamodel-level elements to be used in transformation design as well, like (i) the description of the effects of faults at a general level[1] and (ii) the mapping of UML system models to some mathematical analysis domain.

An advantage of our methodology is to introduce all these elements as a refinement hierarchy starting from the most general view. Specializations of the model at the highest level of abstraction can be used for the different analysis aspects, while if different analysis objectives share common concepts, a joint analysis method can be used.

The following modeling domains extend the basic UML metamodel in the case of dependability analysis complementing the architecture design phase (Fig.1.):

- The standard UML is enriched by dependability attributes to be used by the modeler of the target system.
- The standard General Resource Model is used to describe interactions by means of QoS parameters defined according to the actual analysis objective between the application and the underlying resources in either the form of static or dynamic usage, together with their management.
- The basic notions for analysis of dependability attributes (e.g. error propagation path or step) are added to the standard UML.

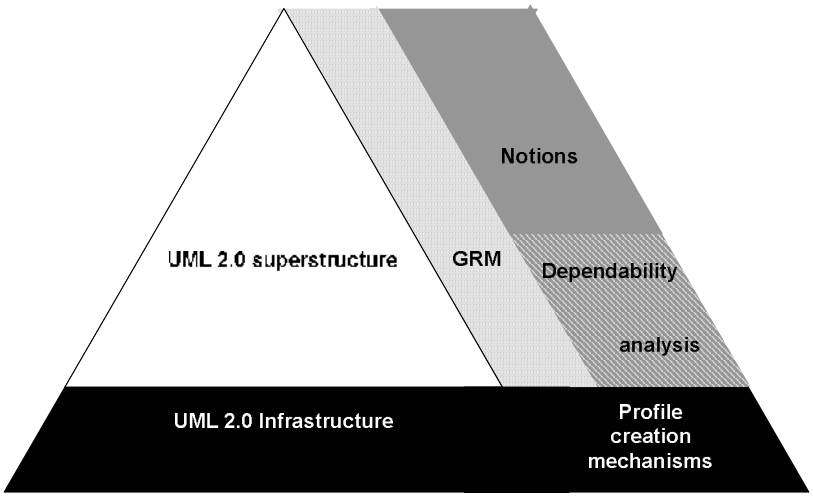

Figure 1: Modeling elements

[1] For instance, the basic notion of stuck-at faults in traditional gate level testing is introduced at the metalevel and the potentially faulty model is derived by applying this metalevel transformation to the faults free circuit. A similar approach to automatically derive faulty instances of a UML model was proposed in [8].

For instance, the flow of errors must be tracked along all explicit (via the ordinary data and control flow) and implicit (through shared resources) error propagation paths in order to check a system's dependability. This error propagation process is independent of the particular origin of the errors, thus it is identical if an error originates in a transient HW error, or it is caused by an intruder. In dependability analysis both kinds of errors may share the same propagation mechanism with slightly differing propagation paths.

3 Implementation technology

The implementation of the concepts described above uses two main technologies:

- Hierarchical modeling is used to relate dependability modeling and analysis concepts to the UML metamodel. However, the OMG standard Metaobject Modeling Facility (MOF) suffers of several drawbacks, like having only informal semantics, introducing a rigid four-level structure on metamodeling levels, and confining the refinement operators (for instance, the refinement of packages or associations is not supported). Our approach uses Visual Precise Metamodeling (VPM), an extended metamodeling and model refinement method [6], providing a precise refinement calculus, allowing an arbitrary number of metamodeling levels and the refinement of all modeling constructs.

- Transformations are described by the easy-to-understand but precise formalism of graph transformations, a multidimensional extension of the Chomsky-grammars [7]. A transformation is defined by an ordered set of simple visual graph manipulation rules executed in series. Each step specifies a graph pattern to be searched in the source graph (in the UML model of the application) and another one inserted into the target graph (into the mathematical analysis model). Complex transformations can be composed by cascading multiple simple ones.

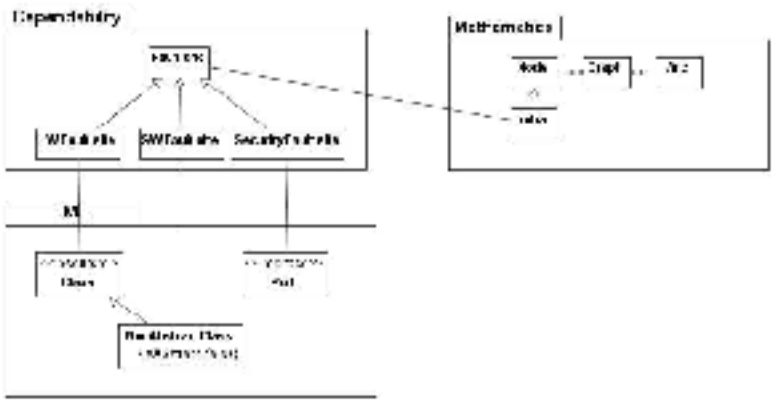

Figure 2: Linking modeling domains

The use of hierarchical modeling is illustrated by the small fragment in Fig.2.

Here three domains are related: one describes the main concepts of dependability, another one the target mathematical analysis tool, while the third one is the extended UML metamodel.

The dependability metamodel defines the different forms of appearance of the notion of a fault site. The association to the UML metamodel defines the correlation between the dependability concepts, and UML as a modeling language. For instance, in the case of the analysis of the consequences of hardware faults `Fault_site` is assumed to be a `Resource`. The association between the abstract class `Fault_site` and the `Initial_node` in a graph can be used during the analysis to define the starting point from which the transitive closure has to be calculated.

The typical flow of transformations consists of the following steps (Fig.3.): as a first step the dependability analysis related elements are extracted from the UML model of the system and they are labeled according to the rules of the individual components in the dependability metamodel. For instance, some of the resources is labeled as the fault site in hardware error propagation analysis.

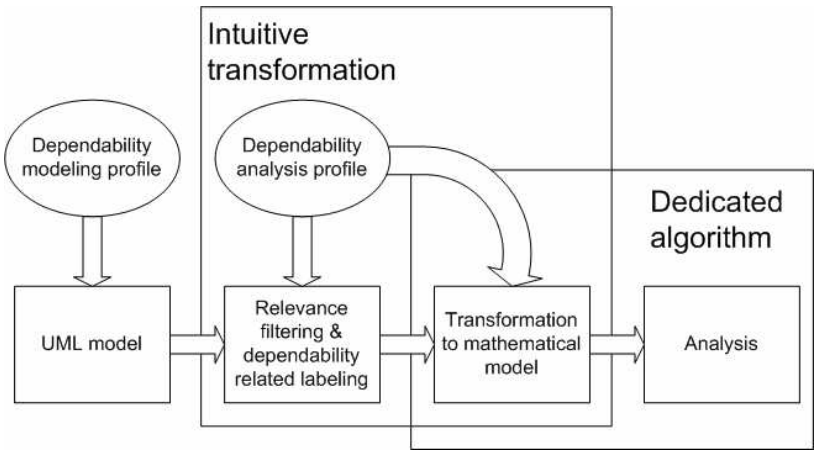

Figure 3: Transformation workflow

A subsequent transformation maps this model to the input of the mathematical analysis tool by using the associations between the dependability analysis model model and the metamodel of the target mathematical notion. For instance, the fault site becomes through this transformation to the initiative node in the graph of which the transitive closure has to be estimated.

It is worth to note, that by merging different steps in the transformation flow we may get to the counterparts of different known methodologies. The traditional intuitive transformations correspond to a direct and unstructured implementation of the filtering and labeling steps. Dedicated algorithms correspond to the merging of the dependability to mathematics transformation and the subsequent mathematical analysis.

The workflow indicates the benefits of using a hierarchical multistep approach. The rules used in the „filtering and dependability related labeling“ step are derived from a few of associations between the UML metamodel and dependability notions. The mapping from the „dependability labeled“ model to the mathematical one is a pure definition of the interface of the evaluation algorithm. Finally, the pure mathematical analysis algorithm can be shared between all analysis tasks necessitating the solution of a specific mathematical problem. This way, a small library of potentially highly optimized algorithms can serve for a variety of analysis objectives.

4 A pilot example

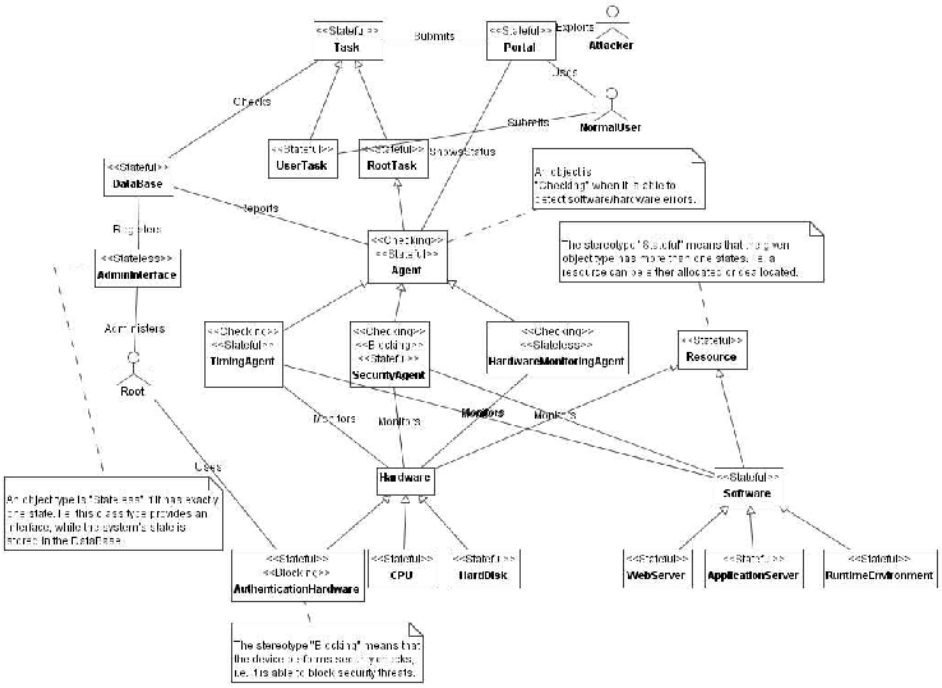

Figure 4: Class diagram of the system

The theoretical results are illustrated by an example describing a web-based task management system where a user can start a task only after getting an approval from the system administrator or “root”. The main components of this system (Fig. 4) are

- a finite set of resources (both hardware and software), that are used by tasks submitted by users;
- the set of “normal users” who want to submit tasks;
- the “root” responsible of the fair distribution of system resources;

- various “agents” keeping the system in an error and security flaw free state;
- a “database” that maintains all information about current system state;
- an “administration interface” converting root's commands into database queries;
- an “authentication hardware” preventing access to the administration interface for non-root users;
- a “portal” displaying actual information about running tasks for the users.

During normal operation (Fig 5).when a “`user`” intends to initiate a task, he logs in the portal, and submits it. After an approval is granted by the “`root`” a corresponding entry is created in the database, and the system will start it as soon as possible. Otherwise the task will be rejected. If a running task has been interrupted for some reason, an agent will restart the task. Upon termination of a task, its owner is notified via the portal. In the following, we will shortly summarize the hardware fault and security flaw scenarios.

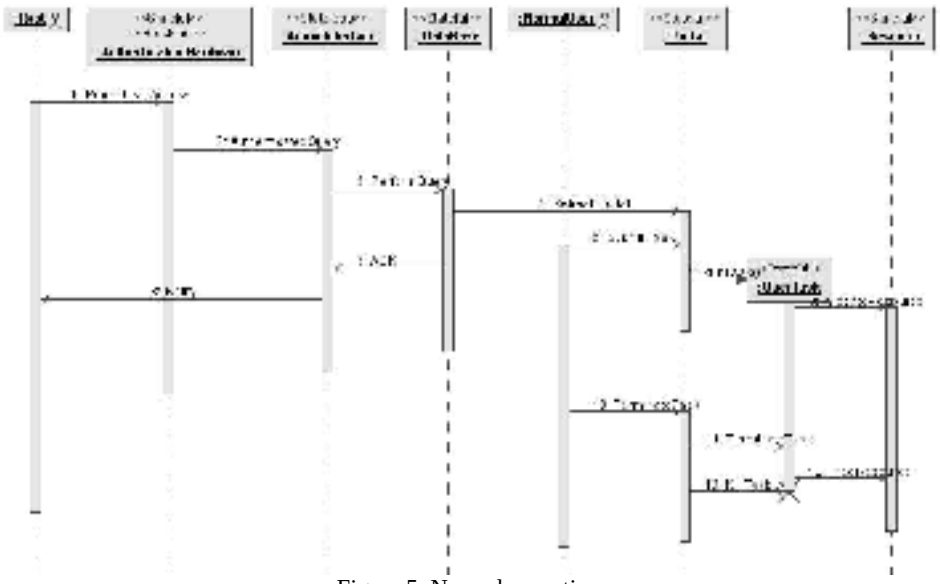

Figure 5: Normal operation

The classes described in UML diagram are stereotyped as: `Stateful` if it is capable of storing errors by preserving an erroneous state; `Stateless` if it has no memory; `Checking` if it is able to detect software and/or hardware errors thus preventing further error distribution; `Blocking` if it performs security checks, and prevents further distribution of the effects of one or more security flaws.

4.1 Software and hardware fault handling

The set of anticipated software and hardware faults consists of four main error categories: (i) sticking into a state; (ii) no response from a component; (iii) Random state transitions; (iv) a bad functionality manifested as a data error.

Software and hardware errors are detected by various agents continuously monitoring the system. One or more components forming an error-containment region will be halted depending on the type of the error to prevent further error propagation. After a subsequent diagnosis and repair, the corresponding components will be restarted. If the system can not recover from an error, it will go to a “Global Failure” state, and stop.

A hardware fault may even induce a software error with potential security related side-effects. A fault in the authentication hardware may generate a huge amount of random database queries leading to database inconsistency by overloading it. A database inconsistency can prohibit a successful recovery triggering a fail-stop of the system.

The damage confinement region is estimated by calculating the transitive closure of the error propagation graph as in the general case. The nodes of this graph are the software and the hardware resource classes; however the interruption of the error propagation by `checking` and `blocking` nodes is modeled by omitting them from the set of graph nodes in the association rules (Fig.6).

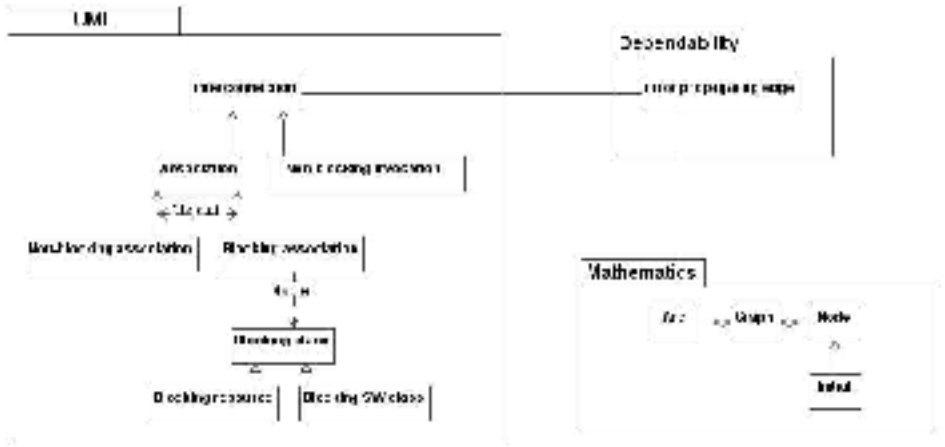

Figure 6: Error propagation rules

The result of the analysis shows that the errors in the database (a crucial component of the system) are detected and blocked by several agents (Fig. 7).

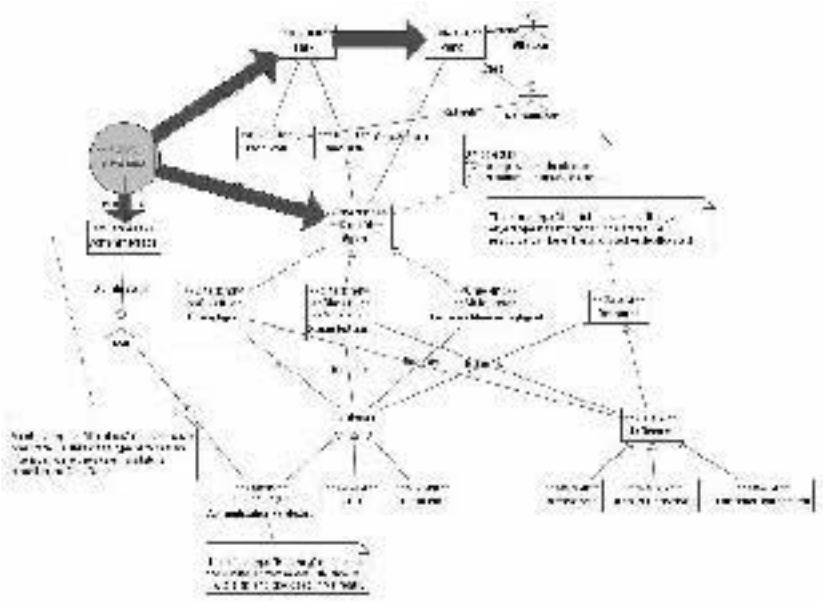

Figure 7: Damage confinement region for a database fault

4.2 Security faults

A “malicious user”, may try an internal security attack, in order to run malicious tasks over its limit by illegally modifying the database entry by deceiving the authentication hardware, so aliasing the root. Fortunately a security agent notices this kind of attack, and is able to ban the user and all of his tasks from the system.

An “external attacker” tries to violate the security of the system by illegally obtaining valuable information, aborting running tasks, or starting unapproved tasks. An imperfect security agent will be unable to detect this security threat. This way an attacker may modify the database in order to let kill a non-malicious task by the misleaded security agent, and let him spawn other, malicious tasks recognized by the agent erroneously as interrupted benign tasks (Fig. 8) thus breaking the Bell-LaPadula's “no-read-up” rule [9].

The description of the error propagation can be done by using the same association rules as in the case of hardware errors. The imperfectness of the security agent can be expressed by omitting this stereotyped class from the abstract class labeled as `blocking`.

5 Conclusions

A well-layered approach was presented in the paper to associate the notions of UML models, dependability, and mathematical analysis.

The main advantage of the methods is that it reduces the elaboration of transformation rules to the formalization of the general notions of dependability and analysis. Transformation rules can be derived in an automated way from this description.

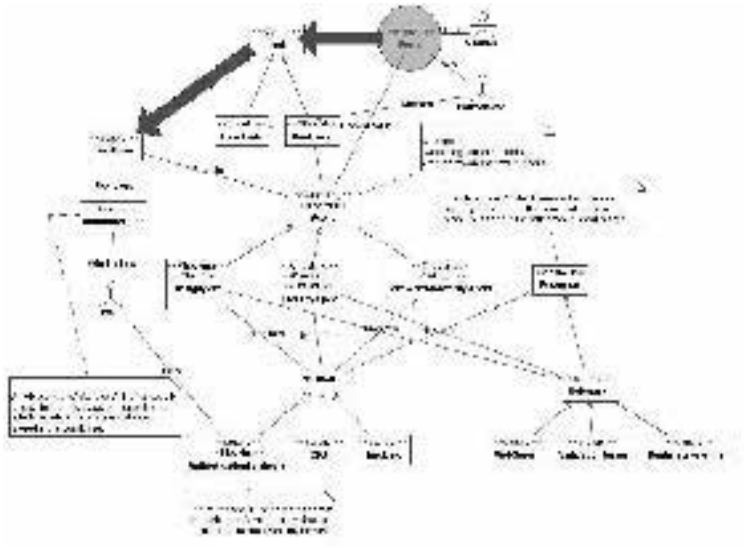

Figure 8: Security analysis

References

[1] B. Baudry. Y. Le Traon, G. Sunyé: Testability Analysis of a UML Class Diagram. Proc. IEEE Software METRICS'02, pp. 54 -63, 2002.

[2] A. Pataricza. From the general resource model to a general fault modeling paradigm? – Proc. of the UML'02 Workshop Critical Systems Development with UML, volume TUM-I0208, pp. 163-171. Technische Universität München, Oct 2002.

[3] J. Jürjens: Towards Development of Secure Systems Using UMLsec. Proc. FASE 2001, pp. 187-200, Springer LNCS-2029, 2001

[4] J.-C. Laprie: Dependability: Basic Concepts. Springer, 1992,

[5] S. Bernardi: Building Stochastic Petri Net models for the verification of complex software systems. PhD thesis, Universit`a degli Studi di Torino, 2003.

[6] D. Varró, A. Pataricza. VPM: Mathematics of metamodeling is metamodeling mathematics. Journal of Software and Systems Modelling, 2(3):187-210, October 2003.

[7] Gy. Csertán, et al. VIATRA - visual automated transformations for formal verification of UML models. In Proc. IEEE ASE 2002, pp 267-270. 2002.

[8] A. Pataricza. Metamodel based fault modeling in UML designs. In Suppl. Vol. of IEEE DSN-2003, pp. 72-73, 2003.

[9] A. Silberschatz, H. F. Korth, S. Sudershan: Database System Concepts, Third Edition, McGraw-Hill