From the General Resource Model to a General Fault Modeling Paradigm?

András Pataricza*

Budapest University of Technology and Economics
Department of Measurement and Information Systems
pataric@mit.bme.hu

Abstract. The paper presents a methodology to extend the OMG General Resource Modeling sub-profile to model component faults in a UML design. This extension introduces the notion of faults into the resources. The UML model of the target system is extended by the description of the error propagation attributes of the different objects. A qualitative abstraction supports a semi-decision style approximate analysis of the effects of local faults to the dependability attributes, like safety requirements of the system.

1 Objectives

Modern visual design methodologies, like UML, reduce the risk of design errors and simultaneously increase the productivity of the design process. Mainstream academic and some industrial tools use a transformation-based approach to formally analyze the UML model under design in order to prove its functional correctness [1]. For instance, model checking based approaches can detect general design errors, like deadlocks, and violations of application specific safety requirements.

However, the functional correctness of a design alone does not assure the correctness of the services delivered by it. The proof of the fulfillment of non-functional requirements, like dependability require a specific analysis, and potentially modifications in the target architecture. Typical analysis methodologies are based on the extension of the functional model of the target system by the QoS values of the components, and a subsequent transformation to the input of some QoS requirement specific analysis tool, which includes the composition of the system model from the UML diagram and component related QoS values. The analysis results are back-annotated to the original UML model for rendering.

This way a variety of specific questions of the type "Will the system function correctly?" can be answered, or at least a good approximate answer can be given. However, not only safety critical, but even demanding IT applications have to guarantee the seamless continuity of the services. Accordingly, another question

* This work was supported by the project OTKA T0308027 of the Hungarian National Scientific Fund.

of the form of "Will the system function correctly, even in the presence of a fault?" has to be answered, as well.

In a previous work [3] reliability and availability measures were estimated, however, this quantitative assessment did not intend to prove the correctness of the logic of the design. The main objective of our ongoing research is to elaborate a methodology compliant with the standardized UML for modeling operational faults and their system-wide effects. The intended UML compliance guarantees that standard simulation based and/or formal analysis methods can be further used without any modifications even for the extended model, as well.

Fault modeling and analysis should be performed as a follow-up phase of the initial system architecture design serving only for an extension of the model by the parameters needed for the analysis, probably done by a dependability expert. Note, that this approach avoids to build up a complete dependability model different from the functional one from scratch, thus the consistence between the designer's model and the dependability model is guaranteed by the process.

The main scope of the methodology is the modeling of permanent and transient faults in the resources, dominantly implemented by hardware. Error propagation via hardware and software components, and messages can be modeled by the method.

While the mathematical background would even allow for modeling faults in the very computer core (CPU-memory setup), the complexity of composing and analyzing a fine granular model would be practically infeasible. On the other hand, the computing core of any mission critical application is typically in itself fault-tolerant, (e.g. by modular replication and voting, by a master-checker setup or by a watchdog processor), this way faults resulting in catastrophic failures or rough distortions in the control flow of the computational process are only of secondary importance at the system design level.

The definition of a faithful fault model properly describing the most frequently occurring faults is crucial from the point of view of the quality of the design. A poor coverage of faults may result in an unacceptable QoS. Too pessimistic fault models result in an increased redundancy, thus superfluous costs.

The other aspect determining the location of fault modeling and analysis in the design process is the objective to introduce the aspect of dependability into the design flow as early, as possible in order to avoid costly redesign cycles. Initial phases of the design flow, like requirement capture and behavioral design producing only algorithmic specifications are premature due to the lack of information on resource allocation and correspondingly on the source of faults.

The first phase convenient to model potential faults is the architecture design phase where resource allocation is already carried out. OMG offers in a part of an adopted specification a profile for General Resource Modeling (GRM) [4] serving as a base for extention by the notion of faults and errors in our approach.

The General Resource Model

GRM is intended to describe resource types, their static or dynamic interaction with the system, and their management. Services required from and delivered by the resources are described by means of QoS parameters, which can be defined according to the actual analysis objective (Fig. 1).

Static resource usage models neglect the dynamics of the client-server interaction. Only a comparison of the QoS values required by the client and offered by the resource instance is performed. A typical system level static QoS analysis is a worst case assessment of the sufficiency of the capacity offered by the resources compared to the total maximum of the requests by the clients.

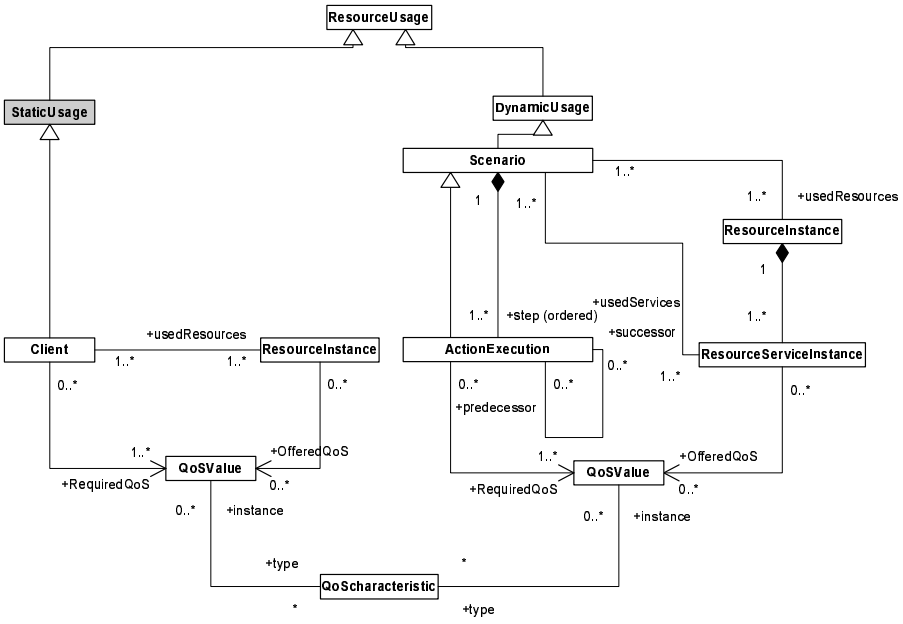

Fig. 1. The OMG General Resource Model

Dynamic usage modeling explicitly describes the order and timing of the client-resource interaction in the form of a scenario, a time ordered series of action executions, or steps. Each resource instance may offer single or multiple kind of services. Each action execution may require a resource and/or specific service(s) offered by a resource. QoS values are assigned to the service invocation (required QoS value), and to the resource service instance (offered QoS).

Resources can be classified either as protected or unprotected. Protected resources offer one or multiple exclusive service instances used simultaneously only by a single client selected by a resource broker implementing an access control policy. A resource manager supervises the creation and administration

of (typically software) resources corresponding to the resource control policy. The use of an exclusive service has to be started with a (blocking or non-blocking) acquire service action and terminated by a release service action.

Uninterpreted modeling

Current formal methods fail to completely validate and verify detailed models of medium and large scale applications due to run-time and state space complexity problems. Advanced analysis methods overcome these problems by using a combination of different techniques, like abstraction, exploitation of symmetries, counterexample dependent model refinement, etc [2].

One of the standard approaches is abstraction based non-interpreted modeling. While dynamic full (interpreted) models represent data items by their presence, type and value at a given node of the system at a temporal instant, non-interpreted models trace only the presence of data at the nodes of the system. Data value depending behavior is substituted by a nondeterministic choice between the abstract image of the alternatives in the interpreted model. This principle is most widely used for performance analysis by timed data-flow networks with probabilities assigned to the nondeterministic choices.

Obviously, every state sequence (trajectory) in the interpreted model has its counterpart in the non-interpreted model (a Galois connection is maintained). The information loss originating in the abstraction may introduce spurious trajectories (for instance, the termination of a counted loop cannot be proved by non-interpreted modeling, as even an infinite loop is allowed by the non-interpreted model). This way uninterpreted modeling leads to a semi-decision technique in validation and verification problems.

- If an exhaustive analysis performed on the uninterpreted model shows no violation of the validation objective, it guarantees, that even in the interpreted model no counterexample will occur. The use of a small abstract state space reduces the complexity of the check is by orders of magnitude compared to the interpreted model.
- However, it is indistinguishable, whether a violation detected on the uninterpreted model originates in a spurious solution due to the abstraction, or it is a true counterexample. The final decision needs a detailed analysis performed on the interpreted model, where the trace of the counterexample in the non-interpreted model can be used to confine the state space to be explored.

Fault and error modeling

Granularity requirements

The assessment of dependability attributes necessitates the modeling of two basic phenomena:

- The local effect of faults at the location of their appearance;
- The propagation of errors (fault induced wrong states of the system components) across the system in order to estimate, which faults may lead to a failure, i.e. an observable deviation from the specified system behavior.

The main challenge in fault modeling is the faithful mapping of the primary physical origins of faults to the model level (for instance, in the traditional gate level fault models a short in a transistor is described by a stuck-at constant logic value at a gate node). UML based fault analysis necessitates a proper modeling of the underlying platform, and the deployment of the logic objects to physical (engineering) ones as a fault in a shared resource can simultaneously affect functionally independent parts. This kind of correlated faults are undetectable in a pure functional system model. Additionally, a resource may expose different behaviors (so-called fault/failure mode) depending on the type of the fault affecting it.

Fault modeling has to be done in a fine granular way to reflect all the potential faults and error propagation paths. The UML model of the system architecture consists of abstract logic types, which are mapped during code generation to physical ones. A code generator can deploy a variable of an abstract integer subtype representing a room temperature in the range of $[-30...50]$ °C onto a 32 bit memory word, as physical carrier of the data. A single bit flip in this word can cause an illegal value out of the range of the logic type, absolutely not represented by the original UML model.

The modeling of error propagation needs the specification of the reactions of a component to the different kinds of erroneous inputs. The term "erroneous" means here that an input parameter may have a wrong value potentially violating the constraints imposed by its type definition, or in the case of real-time applications it can violate its temporal specifications (explicit error propagation). Errors affecting shared resources may cause "parasitic cross talks" between functionally independent units. For instance, a write operation to a location addressed by a wrong pointer value in a shared memory segment can corrupt a variable used by another method exclusively in the fault-free case.

On the other hand each dependability critical application is structured to have some well-defined damage confinement regions confining the potential propagation of errors to this region. For instance, in a pure software based multitasking system running on a well-protected computing platform the protection mechanisms typically confine the propagation of errors to a single task.

Obviously, all the above mentioned fault and error propagation specification problems can be sufficiently handled by a proper modeling style exposing potential parasitic couplings in an explicit way. A general rule for a proper modeling style is to model all the resources sharing within of a damage confinement region.

Modeling of fault occurrences

Dependability assessment necessitates the injection of faults into the system, i.e. to activate the selected fault or fault mode at the fault site in the model. An

individual analysis of all the temporal and permanent faults in the resources is infeasible due to the huge number of faulty models. All modern fault modeling methodologies exploit the fact that the faulty models differ from the fault free one only locally at the fault site, i.e., they are mutations of the original model. This way all the faults are merged into a single model by extending model of the fault-free behavior, which switches the behavior locally from the proper to the faulty one upon the activation of a particular fault by a separate fault injection module.

The fault injector is a separate process independent of the internal functioning of the system which has a direct access to the resources in order to activate/deactivate the selected fault(s), as shown in Fig. Permanent faults are modeled by a single activation event, in the case of temporal ones a deactivation event has to be triggered after the time interval corresponding to the expected duration of the fault, or after an error begun to propagate. Frequently, several other restrictions apply for the process generating faults like the assumption of the occurrence of a single fault only.

Qualitative fault modeling

Dependability analysis necessitates the simultaneous tracing of the information flow in the fault-free and in a faulty instance (or all candidate faulty instances) of the system in order to estimate where an observable difference in their behavior appears.

The main idea of our approach is to introduce qualitative fault models in the form of QoS parameters. This kind of modeling uses a small set of qualitative values from an enumerated type, like {`good, faulty, illegal`} to construct an abstract model simultaneously reflecting the state of the resources and the potential propagation of errors in the fault-free and faulty instances.

The designer can define the set of qualitative values arbitrarily, according to the objective of the analysis. For instance, in timeliness analysis, the set of values can be extended by `early` and `late`.

The most well developed formalism to analyze qualitative fault models is that of the data flow networks [5, 6], which is used in the industrial practice among others for fault tree analysis [7]. However, the principle can be used in any form of formal modeling supporting nondeterministic modeling.

The GRM based fault modeling methodology composes the model of three major parts:

- The **qualitative model of the system** specifies the functionality of the system in a skeletal form keeping all the objects and references to them, but substitutes the domain of every data type with a very reduced set of qualitative values. For instance, the model of an action in the fault-free case is simply, that upon the appearance of the good values on all its input pins, it will place good values onto its outputs.

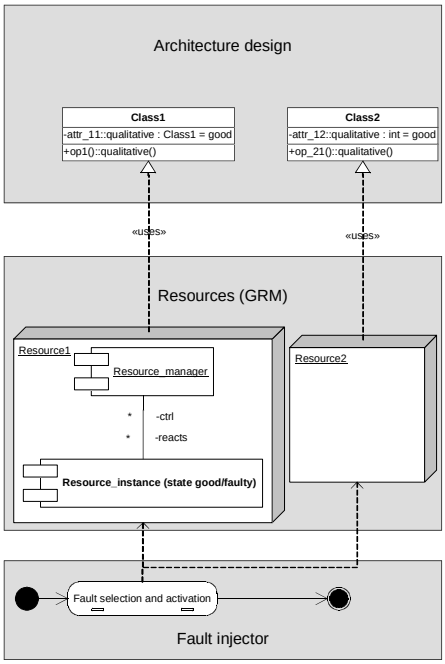

Fig. 2. The fault modeling architecture

An qualitative model of attributes can distinguish here between `good`, `faulty` and `illegal` values (for instance, in the example of temperature representation any value within of the subinterval, but differing from the `good` one is marked as faulty, and the values outside of the subinterval are `illegal`). The model of each method is extended by the description of its reactions to an invocation with a faulty input value combination. By default, an action having a faulty value at least on one of its input pins reacts by randomly placing values from the set of {`good, faulty,illegal`} at his outputs. The designer can override this default by stereotyping, for instance if the method invoked performs a syntax check on the input variables rising an interface exception. Typical fault tolerant structures, like voters, assertion based checks, etc. can be easily modeled this way.
Stateful methods execute their dynamics in an abstract form, where data dependent branches are transformed to random selections. The state space is extended by the state `illegal`, which models unhandled errors. Usually, this state should be unreachable in a proper design having a complete error exception handling system.

– The **model of resources** describes the underlying platform. The system and the resource models are interconnected by standard `<<deploys>>` mappings. Each resource state may have the value of `good` or `faulty`. (In a more detailed model even different failure modes can be distinguished).

In the case of static resource modeling (indicating only the dependence of some class or object instances on a resource) a mutual potential invalidation is used to model error propagation. The faulty state of a resource causes a potentially `faulty` or `illegal` value on the output of the method using it, or can block its termination. Similarly, an action using faulty or illegal input or state variables may put the resource into a faulty state. This way, error propagation between functionally independent object through shared resources can be modeled, as well. In order to support the modeling of recovery actions, they can be stereotyped as `<<recovery_action>>`, forcing the resource independently of its actual state to a good state.
In the case of dynamic resource modeling, the principles described above are applied to the resource manager and to the scenario defining the interaction between the service requesting object (`Client`) and the resource.

- Finally, a separate **fault injector** models the occurrence of faults. This part accesses the resources as a "virtual client". It uses high priority events to change the state of the resources from good to faulty to model permanent fault effects, and a two-phase event sequence to trigger a good-faulty-good state transition sequence characteristic to transient faults. The fault injector has to model additionally the constraints on the fault occurrence (e.g. single fault assumption). There is only a unidirectional information flow from the fault injector to the resources in order to assure the asynchronous occurrence of faults independently of the system operation.

Conclusion

The main message of the paper is, that the GRM profile and qualitative fault modeling are a promising combination to perform fault modeling to be used in simulation and formal analysis.

References

1. A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza and G. Savoia: *Dependability Analysis in the Early Phases of UML Based System Design*, Int. J. of Computer Systems - Science & Engineering, 16 (5), pp. 265-275.
2. E.M. Clarke, O. Grumberg and D.A. Peled: *Model Checking*. The MIT Press 2001.
3. A. Bondavalli, I. Majzik and I. Mura: *Automated Dependability Analysis of UML Designs* , Proc. ISORC'99, pp. 139-144, 1999.
4. *UML Profile for Schedulability, Performance, and Time Specification*, OMG Adopted Specification, March 2002
5. J.A. McDermid, D.J. Pumfrey: *A Development of Hazard Analysis to Aid Software Design*, Proc. COMPASS'94, pp. 17-26.
6. Gy. Csertán, A. Pataricza, and E. Selényi: *Dependability Analysis in HW-SW codesign*, Proc. IEEE IPDS'95, pp. 316-325.
7. Y. Papodopoulos, M. Maruhn: *Model-Based Synthesis of Fault Trees from Matlab-Simulink Models*, Proc. of DSN 2001, pp. 77-82.
8. P. Cousot, R. Cousot: *On Abstraction in Software Verification*. Proc. of the CAV 2002. pp. 37-56, Springer LNCS 2404.