SYSTEMATIC GENERATION OF DEPENDABILITY CASES FROM FUNCTIONAL MODELS

András Pataricza
Technical University of Budapest, Department of Measurement and Information Systems
Address: Magyar tudósok krt. 2., Budapest, Hungary, H-1117
Phone: (+36-1) 463-3595, Fax: (+36-1) 463-2667, E-mail: pataric@mit.bme.hu

Abstract: A variety of applications relies on rough granular redundancy assuring safety by the architectural principle, but integrated solutions of a lower level of redundancy necessitate a detailed check in order to guarantee a high-level of safety. While formal methods are widely used in proofs of correctness of safety critical applications, little has been done to use them for checking the compliance to safety requirements in the case if some component fails. The paper presents a novel approach based on qualitative modeling at the level of fault/error/failure modes facilitating the reuse of existing formal methods for validating and verifying dependability including safety properties.
Keywords: fault impact modeling, formal methods, model-based engineering.

1. INTRODUCTION

The use of formal methods belongs more and more to the core technologies in the proof of correctness of IT systems, including validation and verification (V&V) but their use in dependability assessment is quite limited. A question of the type "*Does the system have a potential deadlock due to a design fault?*" belongs to the standard problems addressed by formal methods embedded in the design flow. However, very little has been done in the context of system design for dependability to answer questions, like "*May a faulty component result in a potential deadlock blocking some vital system functionality?*".

The paper presents an approach facilitating the use of formal methods in dependability analysis despite of the modeling and analysis complexity originating in the large number of faulty cases.

1.1 Research challenges

The main advantage of formal methods is, their exhaustiveness, as they examine the system thoroughly by checking all the potential cases. A major obstacle in their practical use for V&V of engineering designs is the strong limitation on model complexity for practical feasibility of the underlying computations, thus either the faithfulness of the model has to be sacrificed by a high level of abstraction, or a proper workaround has to be found reducing the complexity by decomposing the analysis problem into small fragments each one of them feasible to solve.

Knowing all the difficulties in proof of correctness problems dealing only with a single, fault-free instance of the system under evaluation, an exhaustive formal analysis of all the potential faulty instances of the system seems to be hopeless.

1.2 Abstractions in modelling

A recurring motive in the related research is the search for good abstractions which are faithful enough to deliver practically meaningful results but fit into the computational complexity limitations of formal methods.

A motivating example is provided by one of the fundamental theorems in automata theory on the equivalent states. As it is well known, the checking of an entire equivalence class (containing a finite or even infinite number of states) can be reduced to the examination of the behaviour of the class leader, a single element representing the entire class. The use of a similarly powerful

abstraction technique, qualitative modelling (Saidi, 2000) is the core idea in our paper.

Practical engineering and everyday's thinking uses frequently qualitative abstractions in reasoning about operations of a system.

For instance, the formulation of the simple statement "*If a car drives faster, than the speed limit, a radar control can take an expensive picture*" already uses this technique in an intuitive way. The main characteristic of the abstraction technique in the background is that all values belonging to a domain showing up a similar behaviour (the entire speed interval above the speed limit) are aggregated into a single representative state.

Naturally, this symmetry is violated if taking account the impact of breaking (for instance, after detecting a radar trap). In such a case, either the domain has to be splitted into two subdomains (one representing the case, if the speed can be reduced enough to avoid the fine and the other one, if not anymore) or a non-deterministic model has to be created stating that after braking we can potentially avoid the fine or not.

Note that this abstraction corresponds to a complete uniformization of the behaviour of all the concrete states in the particular domain by allowing in the abstract model a behavior for *all* the elements in a domain, what *at least one* element can do.

This permissive over-approximation[1] covers all the potential behaviours of the system guaranteeing that no important case will be neglected. On the other hand, over-approximation may introduce spurious solutions.

This way, the abstract system leads to a semi-decision. If a check over the abstract model delivers a negative result indicating that no execution having some target property exists, then this is a proof of non-existence of such a run in the concrete model. If an example is found, further checks are needed over a refined abstract or the concrete model to decide, where it is a spurious solution or a true example.

In general, abstract models generated by qualitative abstraction out of the full (concrete) model of the system represent the modes of operations inducing qualitatively different behaviours by a few values marking these operation domains and provide an upper approximation of the potential behaviours of the system.

Semi-decision techniques in safety analysis means, that if no dangerous situation is identified over the abstract model, then this can be taken as a proof of safety. Counterexamples need further evaluation.

The drastic aggregation of domains into single states avoids the traditional computational complexity problems originating in the over-detailed representations in full models at a price of introducing potentially spurious solutions.

1.3 Abstractions in dependability analysis

The principle of qualitative abstraction can be used in fault impact modelling. The basic model examines here the fault-free and a faulty model simultaneously in a similar form, as traditional gate level automated test pattern generation does it since more than four decades.

Qualitative abstractions in fault impact analysis describe the information flow in a potentially faulty system only at the level of resolution of fault/error/failure modes.

This way only so much is kept as data representation in modelling fault induced safety cases, whether the individual elements and signals are good or erroneous (or what kind of error mode is present) instead of using their concrete values necessitating the handling of huge state spaces.

The underlying core engineering heuristics is that errors of a similar order of magnitude (belonging to the domain of a particular error mode) in a physical system typically cause similar impacts, accordingly operation domains corresponding to the individual error modes can be aggregated into a single state representing the euchre domain.

Note, that the selection of fault/error/failure modes is a free design parameter in modelling, thus the dependability analyst can choose them for instance according to the designated diagnostic resolution.

Obviously, such a reduction of model complexity (typically of many orders of magnitude) has the price of introducing false alarms due to abstraction threatening its usability.

---

[1] allowing for instance in the abstract model, that braking at an extremely high speed may slow down the car below the speed limit, despite the fact, that the detailed model indicates clearly, that this is not the case

2. FORMAL DEPENDABILITY ANALYSIS

The research was focused on three, strongly correlated fields (Pataricza, 2006):

1. Static, syndrome-level dependability assessment models describe the signal flow in a system after a temporal compaction mapping them to a single attribute of failure mode. The set of failure modes contains in the simplest case of pass/fail categorization only the values of good and faulty, in the case of a more detailed modeling the faulty case is refined into multiple values according to failure modes.

The core idea of syndrome level analysis is the characterization of the sensitivity of the individual components in the system by means of relations between the error manifestations at their inputs and outputs. It serves as an early check of the appropriateness of the dependability concept, thus avoiding extremely costly redesign cycles due to violations of dependability requirements.

2. Another option, dynamic error propagation analysis deals with the dynamics of propagation of discrepancies appearing as fault impacts in a system both by an intuitive method creating the basic models heuristically and by an algorithm automatically deriving them from the functional description of the components.

3. Finally, the foundations of model transformation generating formal analysis models from engineering ones are addressed.

We assume that the model of the system under evaluation is available as a network of interconnected (functional or physical) components according the usual practice in modern model-driven engineering design practice.

The main objective is to create the system model by synthetizing it from composable component dependability models, where the component models can be individually and separately estimated and potentially stored in a library.

The approach working at the level of smaller components allows for using relatively expensive methods in the generation process of their respective dependability models.

2.1 Fundamental model

Dependability assessment necessitates the modeling of two basic phenomena: the local effects of *faults* at the fault location, and the propagation of *errors* (discrepancies) across the system in order to estimate, which faults may cause a *failure*, i.e. an observable deviation from the specified system behaviour.

The system model M is composed as a structure of interconnected *components* $\{C_1 \dots C_n\}$. A *set of anticipated faults* F_i is associated to the individual components C_i. Each component has a fault free *reference instance* C^{good}_i and a separate *faulty mutation* C_i is created for each anticipated *fault mode* f^j_i describing the behaviour in the presence of this particular fault. The actual instance is characterized by a *fault state variable* f_i.

The notion of *error* refers to impacts of faults observable as a discrepancy in the state of the actual system from that in the reference system. Accordingly, the analysis of error propagation necessitates the tracing of information flow in two concrete models: in the *reference* $M^0 = (S^r, S^r_I, next^r)$ and the *actual* $M^a = (S^a, S^a_I, next^a)$ characterized by their state spaces, initial states and state transition relations. *Composite states* are defined as the value pairs at the corresponding nodes in the reference and actual models.

2.2 Spatial compaction

Mutually exclusive e*rror predicates* $\varepsilon_0 \dots \varepsilon_e$ arbitrarily chosen by the designer completely partition the set of composite states into disjoint subsets and map them to the set $D'' = \{\delta_0 \dots \delta_e\}$ of *error modes*. Error modes are a spatial abstraction of the concrete value pairs appearing at the corresponding nodes.

The simplest classification reflects only the match and mismatch of the states in a composite state by taking ε_{good} as the "=" relation and ε_{faulty} as "≠". It partitions errors into the error modes of D={*good*, *faulty*} and neglects the particular type of the deviance (left in Fig. 1). More fine granular models differentiate between error modes by separating the class of *faulty* into disjoint subclasses, like differentiating *minor* and *major deviance*s and *out of range* values (right in *Figure 1*).

Naturally, more complex error mode representation may introduce values depending on the reference and actual states as well in addition to the deviance itself.

The formulation of error predicates is an important instrument to reflect the actual dependability objective. For instance, the integrity of data is the main aspect to be reflected by these predicates in reliability analysis; however, a correct data content published in a wrong way is an error in security analysis.

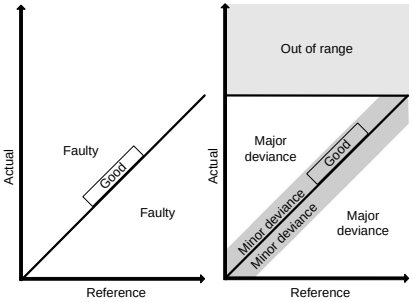

Fig.1. Error mode refinement

2.3 Dynamic model

Abstract dynamic error propagation analysis relies on a model of the dynamics of the target system (internal operation sequences in the components, their mutual interaction and invocation) in order to incorporate the activation sequences into the analysis.

The introduction of error modes as main representation facilitates the creation of a simple automaton describing the impacts of errors appearing at inputs of the individual components[2].

An input error may result in an error on the output of the corresponding component (error propagation), cause a discrepancy in the states in the reference and actual models (latent error potentially manifested later as a control flow distortion), or both. Another option is error absorption (e.g. a single input error of a TMR component).

This dynamic qualitative error propagation modelling automaton (*Figure 2.*) uses the error modes as input-output values to characterize the behaviour of the composite model. An *error sequence* $[\delta_0 \rightarrow \ldots \rightarrow \delta_m>$ is a temporal sequence of discrepancies represented by their respective error modes.

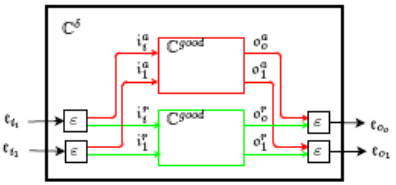

Fig. 2. Dynamic error propagation model

The principle of construction of the abstract automaton is a special form of predicate abstraction. A state transition will be included into the abstract state automaton from the actual composite state to a given successor state upon an input error vector and generating an output error vector, if there exists at least one arbitrary combination of input and output vectors, and actual and next states in the concrete model corresponding to it. This simplest form of model construction necessitates a satisfiability check for each combination of I/O vector and actualities states.

This model represents the control flow in the composite model in its full extent, but the representation of data dependencies is reduced to that on syndrome values thus aggregating entire operational data domains into a single syndrome value.

The abstract automaton can answer for instance, the typical question asking the impact of the first occurrence of a "minor" error. Here, the initial states in both the actual and reference concrete models are identical and the question is *"Over an arbitrary pair of input values differing only slightly at the corresponding inputs of the actual and reference models what are their successor states and what kind of deviance may appear on their outputs?"*.

Such a question may help to conclude whether minor deviances can cause divergences in the control flow executions (different runs) of the actual and reference systems (e.g. noise sensitivity of the control algorithm). Similarly, it can be checked whether such an input of a minor deviance may be amplified to a major one on the outputs. Both answers are approximate in the sense that they indicate only the potential occurrence of such a safety problem.

As the creation of an automaton working over the few values from the error modes as input and output alphabets needs an exhaustive checking of all value pairs constrained by the error mode for all the reachable state pairs. This way, this is one of the core elements in the algo-

[2] The modeling of the impacts of local faults in the components and the propagation of an error through a faulty component in a network containing groups follow exactly the same procedure with a slightly more complex model containing a good and a faulty instance in the composite model.

rithms consuming the most computational powers; however, for simple components this can be done even by hand.

One important characteristics of the dynamic modeling approach sketched above is that it uses only very basic constraints on the modeling language as the concepts used are confined to discrete state space and time, non-deterministic finite automata. This way, such popular modeling approaches, like dataflow, sequence or state chart diagrams can be extracted such a way that the abstract model uses the same modeling paradigm (Majzik *et. al.*, 2002; Majzik *et. al.*, 2003; Pataricza, 2003).

The system model is created by means of composing the individual component models to a network. Thanks to the drastic reduction in data representation the computational complexity of checking this model lies in between that related to the non-interpreted model (neglecting data values by decolouring data items) and proof of correctness models which are rich in data but treat only a single, good system. All the algorithms used for checking the particular language used in describing the original model can be reused without any alteration to check dependability attributes.

The advantage of the spatial compaction is that the abstract automaton has to operate over the very few discrete values corresponding to the error modes even in the case if the system is analog or hybrid. The complexity of the state spaces corresponds to the Cartesian product of the state spaces in the reference and actual models.

2.4 Temporal compaction

The spatial compaction method reaches huge complexity reduction by aggregating large data domains into a few of qualitative values. Another option for further reduction of computational complexity is along the temporal dimension by compacting arbitrarily long error sequences into a single value (syndrome).

The engineering practice frequently uses such simplified views of long sequences. For instance, software errors are frequently characterized at the level of resolution of severity class only according to the most severe error occurring in them, independently of the value and the temporal position of the error occurrence.

Similarly, a typical question related to fault-tolerant systems is whether a system is self-stabilizing, thus whether it always takes a proper compensation action after an observable error attenuating its impact.

The most convenient way to express temporal attributes is by using temporal logic, a formal logic widely used to specify, and reasoning about, logic propositions qualified in terms of time.

A complete and mutually exclusive set of predicates $\{\vartheta_0 \ldots \vartheta_t\}$ formulated as temporal logic expressions performs a temporal abstraction by partitioning the set of error sequences into disjoint subsets and compacting each of them into a single element from the set of *syndrome* values $Y = \{y_0 \ldots y_t\}$. Syndromes observable by the system user will be referred to as *failure modes*.

Once again, the selection of the syndrome value set is a free design choice offered to the dependability analyst according to the purpose of the analysis, similarly to error modes.

For instance, a set of syndrome values in fault impact analysis may contain the values of {*good, minor failure, major failure*} ordered according to the severity of the fault impacts.

Similarly, when examining the appropriateness of the fault compensation mechanisms, the values of {*good, attenuating, amplifying, oscillating* etc.} can be introduced.

Syndrome level modeling constitutes the topmost level of abstraction in error propagation modeling. The principle of creating syndrome level component models is similar to the estimation of dynamic ones.

A component syndrome model describes the sensitivity of its outputs to failure modes appearing on its inputs in the form of a set of input-output syndrome relations.

The estimation of a particular syndrome relation needs the checking for each input-output value vector combination whether there exist such error sequences which fulfil the candidate relation. This necessitates for each value combination a model checking run. The resulting computational complexity confines the maximum complexity of the component models to a moderate level.

2.5 Static model

Syndrome level static models (*Figure 3.*) composed of interconnected failure relation models of components use a high level of abstraction both in the spatial and temporal dimensions representing their mutual interdependency.

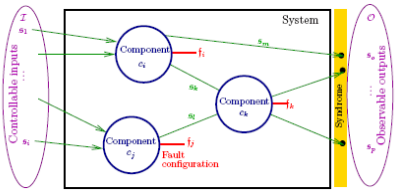

Fig. 3. Syndrome-level static network

Such models are composed as a network of relations and the problems described by such networks are referred to as Constraint Satisfaction Problems (CSP). CSPs are networks of variables interconnected by a set of relations called constraints (Tsang, 1993).

A CSP is solved, if a value assignment has been found for all variables, such that all the relations are satisfied. Constraints have several interesting properties (Barták, 2001); like a constraint does not need a unique specification of all the values of its variables thus they may specify partial information which can be exploited on-the-fly diagnosis procedures.

Constraints are non-directional, declarative (they specify what relationship must hold without specifying a computational procedure to enforce that relationship), additive (the conjunction of constraints is effective independently of the order of imposition of constraints), and compositional, as hierarchical refinement is supported.

The use of relations in CSPs provides a proper support for handling nondeterministic abstractions of complex systems.

Such models share a uniform structure, thus joint algorithms can be used to solve them. In all of our technology experiment we simply use Prolog as modeling language for constraint definition and solution.

The main advantage of the static analysis is that it delivers valuable results in a very fast way thanks to the simplicity of the solution. In particular, for safety analysis one of the main advantages is that the typical architectural redundancy pattern, like n-out-of-m redundancy structure can be easily modelled in a general form. Moreover, as CSP relies on the algebra of relation, it supports several refinement techniques shortly described later in the paper.

A hierarchical analysis along structural refinement can prove the appropriateness of built-in fault-tolerance of safety critical functionalities in the case of structural redundancy at the topmost level of abstraction without necessitating a detailed model of the individual functionalities.

Static, syndrome level dependability model based analysis delivers frequently overly pessimistic results by taking all the topologically feasible error propagation paths as potentially active independently, whether an actual scenario uses them at all.

3. MODEL REFINEMENT AND AGGREGATION

As mentioned before, the model analysis process is a trade-off between modeling detail (and avoidance of false alarms) and computational complexity needed for the analysis. *Figure 4.* summarizes the main possibilities for model abstraction and refinement.

Two strategies can be used for abstraction (creating more compact and easier to solve models at a price of an increased number of false alarms originating in spurious solutions) and refinement (introducing more details and faithfulness): the first one is model refinement within a single type of model, while the second one connects different types of models.

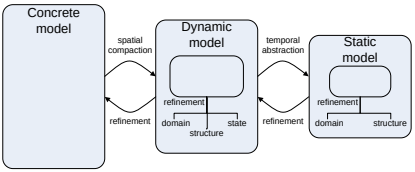

Fig. 4. Abstraction/refinement hierarchy

3.1 Refinements within of a model class

Model abstraction and refinement are well known concepts in the engineering design of computing systems additionally to their mathematical use in the formal analysis for V&V purposes (Clarke *et. al.*, 1994).

The basic principle of design processes guaranteeing correctness is the restriction of the set of allowed model enrichment operations to such ones, which constitute a correct and complete refinement in the terms of mathematics.

Both dynamic and static error propagation models can be created in two basic ways depending whether we start from a coarse or a fine granular model:

Heuristic top-down modeling serves for an early assessment of dependability attributes in the initial phases of system design. It may start for instance from a non-interpreted model of the control flow. The starting point is a skeletal form of the characterization of the dynamics substituting data dependencies with nondeterministic choices. This model will be enriched with dependencies on the syndrome values interpreting this way the dependence of the dynamic behaviour of the system on errors and failure modes of components. This method is intuitive, as it relies on an expert-made forecasting of the sensitivity of the individual components to input errors.

Abstraction based modeling derives the dynamic error propagation models from the functional description of the components. Obviously, this model necessitates a more detailed specification of the system than the sketch-like heuristic methodology, but it delivers an accurate estimation of error propagation effects and simultaneously assures the consistency between the functional and error propagation models.

3.2 Refinements in the design workflow

The primary use of these two models is in different phases of the design workflow, but their scope may cover either only the behaviour of the system or it can be extended to cover both the architecture of application and its deployment to resources (Pataricza, 2002).

Incremental top-down design strategies gradually enrich an initial abstract glass-box model of the system by elaborating either more and more detailed specification of the components (domain refinement) or the model structure (structure refinement) or the resolution of the representation of its temporal behaviour (state refinement). The incremental design flow is composed as the consecutive application of these elementary operations.

Domain refinement uses a finer resolution in the error/syndrome modeling or introduces a more refined separation of cases into the control flow. Input-output refinement introduces subtypes to error modes/syndromes (like in Fig.1.).

Control flow refinement decomposes the state space of a component by splitting an internal state into multiple ones and simultaneously modifying. The set of state transition relations is in order to assure the correspondence between the states in the original and refined models.

Structure refinement splits a single component into a sub-network composed of multiple components, while the interfaces and the state space remain unaltered. Changes of the network structure effect only the ”inner structure” of the refined node, but the other nodes as well as the original interconnections remain unaltered.

State refinement increases the temporal resolution of the model by splitting state transitions into a sequence of such transitions. For instance, an initial coarse model of an operation describes only the arrival of the request to the corresponding unit and the response given by it while a more fine granular one includes the internal control flow of the operation, as well.

There is a difference how a model will be constructed depending whether dependability analysis is executed concurrently with the design workflow or it is only an a posteriori activity.

In the first case the creation of the dependability model follows a step by step refinement synchronized with the design flow of the system.

It starts from a non-interpreted model of the target system representing data by their presence/absence at the different nodes of the system. The intuitive method introduces error modes by a gradual refinement into the non-interpreted model in order to get a description of the error propagation properties of the actual component instance. The subsequent refinements of the dependability model will be executed in synchrony with the functional design workflow.

The main drawback of the intuitive method is, that it is error prone, especially, if an input error may cause latent errors inside of a component (in its storage elements) postponing the manifestation of the error by several steps of operation.

An alternate way is given if at each individual level the functional specification is given in which case the dynamic and static models can be derived from the functional description by using the algorithms sketched above.

Frequently dependability (for instance, safety evaluation) is a separate activity following the functional design. According to our practice, the easiest way to generate good quality models starts from the most detailed models (simple components) and creates abstract models by automated aggregation.

For instance, the syndrome level static error sensitivity model of a sub-network performing a particular functionality will be generated by means of estimating of all the solutions of the corresponding constraint network and omitting

from these solution relations all the variables invisible as interface signals.

The basic definition of the temporal compaction described in Sect. 2.4 serves as a compliance checking criteria between dynamic and static models.

3.3 Run-time refinements

Counter example is given model refinement is one of the most fundamental techniques in model analysis.

The core idea here is that a specific refinement is performed after finding an example violating the required properties. This refinement can be performed locally around the location or temporal interval in which the problematic case was detected. Here the abstract counter example is used as a boundary constraint in the finer granular model.

All the refinement possibilities illustrated in Fig.4. can be included into the model checking process in order to support the elimination of spurious solutions originating in the over-abstraction characteristic to the more abstract models (Chang *et. al.*, 1994).

4. CONCLUDING REMARKS

A novel approach was presented in the paper introducing spatial and temporal compaction for deriving formal models for error propagation analysis. Its main advantage is the reusability of existing proof of correctness methods for analysing dependability attributes in the case of presence of errors.

Static analysis can be performed by means of constraint satisfaction programming supporting even on-the-fly diagnostics processing partial information available at error notifications.

The diagnostic image can be further refined by applying the costly but more accurate dynamic analysis.

Both methods are characterized by the use of over-abstraction of the system thus they guarantee that no critical safety situation will be overlooked.

This way, they fulfil the expectations against a worst case analysis. The price of the simplified analysis is that depending on the level of simplification they may generate false alarms (spurious solutions) to be eliminated by a sub-sequent detailed analysis.

The complete toolchain using VIATRA (Varró, 2003) as transformation engine between a UML dialect for the engineering models of safety-critical embedded systems and dependability analysis models is currently under elaboration in the framework of the GENESYS EU-FP7 research project.

REFERENCES

Barták, R. (2001) Theory and practice of constraint propagation. In Proceedings of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC2001), Wydavnictvo Pracovni Komputerowej, pp 7–14.

Clarke E. M., Grumberg O. and Long D. E. (1994) Model Checking and Abstraction. ACM Tansactions on Programming Languages and Systems, 16(5):1512–1542.

Chang E., Manna Z. and Pnueli A. (1994) Compositional Verification of Real-Time Systems. In Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science, pp 458–465. IEEE Computer Society Press.

Majzik I., Huszerl G., Pataricza A., Kosmidis K. and Dal Cin M. (2002) Quantitative analysis of UML Statechart models of dependable systems. The Computer Journal, 45(3):260–277.

Majzik I., Pataricza A. and Bondavalli A. (2003) Architecting Dependable Systems, volume 2677 of LNCS, chapter Stochastic Dependability Analysis of System Architecte Based on UML Models, pp 219–244. Springer-Verlag.

Pataricza A. (2002) From the general resource model to a general fault modeling paradigm? (2002) In Workshop on Critical Systems Development with UML, pp 114–115.

Pataricza A. (2003) Meta-model based fault modeling in UML designs. In Suppl. Proc. DSN 2003: The International IEEE Conference on Dependable Systems and Networks, IEEE Press, pp 72–73.

Pataricza A. (2006) Model-based dependability analysis. DSc Thesis, Hungarian Academy of Sciences

Saidi H. (2000) Model checking guided abstraction and analysis. In Proceedings of the Seventh International Static Analysis Symposium-SAS2000.

Tsang E. (1993) Foundations of constraint satisfaction. Academic Press.

Varró D. (2003) Automated model transformations for the analysis of IT systems. PhD thesis, Budapest University of Technology and Economics.