 TOOL-SUPPORTED DEPENDABILITY EVALUATION OF REDUNDANT ARCHITECTURES IN COMPUTER-BASED CONTROL SYSTEMS

I. Majzik, P. Domokos, M. Magyar
Budapest University of Technology and Economics, Department of Measurement and Information Systems
Address: Magyar Tudósok krt. 2., Budapest, Hungary, H-1117
Phone: (+36-1) 463-3598, Fax: (+36-1) 463-2667, e-mail: majzik|pdomokos|mmelinda@mit.bme.hu

Abstract: Architectural choices have a profound influence on the dependability of a computer system. Dependability modelling and analysis is a method proposed for the early evaluation of the system architecture and the related design decisions. We describe the design, implementation and application of a tool that is able to construct automatically a dependability model (in the form of Generalized Stochastic Petri Nets) on the basis of a system architecture model. The dependability model can be solved by an external solver, computing in this way the system-level reliability or availability measures. The tool is modular, extensible and supports the aspect-oriented design of redundancy structures.

Keywords: Dependability evaluation, dependability modelling, automated modelling, UML, stochastic Petri nets

1 INTRODUCTION

Computer-based applications are expected to play a prominent role in railway control systems. Architectural choices have a profound influence on the dependability (reliability and availability) of a computer system. Since the prevention of faults cannot be guaranteed neither by high quality hardware components nor by thorough validation of software components (since in the latter case the complexity typically prevents a complete proof of correctness), fault tolerance techniques based on redundancy have to be applied. The degree of redundancy (i.e., the number of redundant components, the redundancy management techniques applied in the system) is determined in the architecture design phase.

The CENELEC standard EN 50128 prescribes the thorough evaluation of hardware and software interactions, the possible failures and the protection mechanisms. In the architecture design phase, some basic questions to be answered by the evaluation are the following: What is the reliability/availability of a system if it is built from components of given reliability/availability? Which redundancy scheme is the best choice? How many components are required in a given redundancy scheme? Which component can be considered as a dependability bottleneck? What is the proper maintenance policy?

Model-based dependability analysis is a method proposed especially for the early dependability evaluation of the system architecture and the related design decisions (Malhotra and Trivedi, 1994). The basis of the analysis is a so-called dependability model, which represents the failure and repair behaviour of system components, taking into account their interactions according to the given architecture. The goal of the analysis is to compute probabilistic measures of system failure. Models can be used to perform “what-if” kind of analysis to evaluate different architectural choices and answer the above presented questions.

Basically two types of mathematical models are used to construct dependability models. Combinatorial models capture static conditions (typically in terms of Boolean combinations of components failures) that lead to system failure. Fault trees, reliability block diagrams and reliability graphs are the most popular formalisms. Note, however, that combinatorial models can not handle the failure or maintenance dependencies between components and the imperfect coverage of fault tolerance schemes. These stochastic dependencies can be captured by stochastic state-space models. The prevalent type of state space models is the Continuous Time Markov Chains.
that is able to transform UML architecture models into GSPN based dependability models. The GSPN model can be solved by an external solver, computing in this way system-level reliability or availability measures. The challenges of the tool design can be summarized as follows:

- Find the proper level of abstraction that is effective in the architecture design phase. We followed the approach described in (Bondavalli et al., 1999 and Majzik et al., 2003).
- Support the dependability evaluation of redundancy structures and maintenance policies in a re-usable and extensible way: the designer shall be able to define her/his own architectural pattern and re-use it in different designs without the need of re-defining the underlying fault tolerance mechanisms (i.e., the logic of error handling).
- Provide extensions for a modularized, aspect-oriented design of fault tolerance: the redundancy-related design decisions (as a crosscutting concern) should be separated from the design of the functional aspects of the architecture.

The paper is structured as follows. Section 2 introduces the dependability modelling approach. Section 3 describes the architecture of the dependability modeller tool. The role of the tool in aspect-oriented modelling is summarized in Section 4. Section 5 demonstrates the use of the tool in a case study.

### 2 Dependability Modeling in the Architecture Design Phase

In the early phases of architecture design abstract dependability models are used that do not take into account the functional properties of components, only represent (1) fault occurrences, (2) error propagations and (3) repair mechanisms. The fault occurrence process describes how component faults occur and how these faults result in errors and failures. The main characteristics are the fault occurrence rate and the error delay. Error propagation processes describe how errors of a component propagate to other components through interconnections characterized by a propagation probability (that aggregates communication and workload parameters). In case of redundant components, the error propagation is nontrivial, since redundancy techniques (e.g., N-modular redundancy) are used to prevent the propagation. In these cases the “logic” of error propagation (i.e., how component failures result

...
in the failure of a redundant subsystem) can be described typically by a fault tree. Finally, repair mechanisms describe the removal of faults and errors by fault treatment, error recovery, etc.

The dependability model to be constructed by our tool, i.e., the mathematically precise description of these processes and mechanisms, is a Generalized Stochastic Petri Net (GSPN). The transformation approach described in (Bondavalli et al., 1999 and Majzik et al., 2003) can be summarized as follows:

- The architecture design model of the system is translated to a dependability model in a modular way: Each component’s fault occurrence process is represented by a module (fault occurrence subnet in GSPN terms), similarly, interconnections are represented by propagation subnets etc. In case of redundancy structures, the nontrivial error propagation is another subnet. These subnets are connected by interface places. The system level failure occurs when a component failure propagates to the system level, i.e., to the components that directly provide system services. Accordingly, the system failure is represented by a specific marking of the GSPN dependability model, and its probability can be computed by a GSPN solver tool.

- Since the dependability model abstracts from the functional properties of components, several types of components can be identified from dependability point of view (independently of their actual functionality): e.g., fault occurrence processes of stateless hardware components can be modelled by subnets of the same structure (but different parameters), since the fault → error → failure effect chain in these components is similar at this abstraction level.

- To support the transformation, the system architecture design (i.e., the set of UML class diagrams) shall be extended. In the case of components, these extensions identify the component type (if it is not obvious from the UML model) and accordingly describe the local dependability related characteristics of the components. Similarly, extensions identify types of interactions between components and describe their characteristics. As the type of a component or an interaction can be identified using an UML stereotype, and the local characteristics can be assigned as tagged values, these extensions form a UML profile.

Part of the UML profile supported by our tool is illustrated in Fig. 1. Here class symbols with attributes represent stereotypes with the corresponding tagged values. For example, a stateful hardware component is characterised by its fault occurrence rate, error latency time, a ratio of permanent and transient faults, and its repair delay. A stateless software component can be characterised solely by the fault occurrence rate, as there is no error latency, faults are design faults and the repair is assumed to be implicit (i.e., the activation of faults depend on the workload).

Redundancy structures, as composite components, are identified by a distinguished subcomponent that coordinates the execution of the other subcomponents and provides access to the service of the redundancy structure. Referring to its role, it is stereotyped as Redundancy Manager. It is characterised by a tagged value that identifies the nontrivial error propagation from the subcomponents to the composite component.

Fig. 1. Component types and the corresponding dependability attributes
The roles of different subcomponents (e.g., variants and adjudicators) are identified by stereotypes as well. The whole system can also be considered as a composite component, it can be characterised by the measures to be computed (the default measure is availability).

For example, a redundancy structure with three variants and a voter is modelled in Fig. 2. The stereotypes identify the roles (here we applied abbreviations, e.g. SFE_HW means a stateful hardware component), while the tagged value attached to the redundancy manager identifies the fault tree (i.e., the nontrivial error propagation process in the structure). This fault tree can be stored separately (as described later).

Fig. 2. A redundancy structure (example)

Note that the analysis subnets corresponding to the component and interaction types are usually constructed by a dependability expert who takes into account catalogue data (in case of hardware components), test results (in case of reused software components), and estimations based on previous experience in similar systems. Accordingly, the main task of a dependability expert is to identify component types and construct the subnets that can be parameterized using the actual characteristics of the individual components constituting the system. Moreover, error propagation subnets belonging to common redundancy techniques (e.g. N-modular redundancy, N-version programming) and repair mechanisms can also be constructed.

3 ARCHITECTURE OF THE DEPENDABILITY MODELLER TOOL

The architecture of the dependability modelling tool is summarized in Fig. 3. In the following subsections we will discuss the components of the tool in detail.

3.1 The input model

The input model is a UML architecture model (class diagram) that is constructed using the profile illustrated partially in Fig. 1. Note that the modular construction of the tool will allow the support of other input languages as well, for example AADL (SAE, 2004). The UML model shall be provided in standard XMI format, thus several UML modelling environments can be used by the designer to construct the architecture model.

3.2 The internal representation of the dependability model

The core part of the tool is the representation of the dependability model in the form of a hypergraph called Intermediate Model (IM) for historical reasons. It is based on a simple metamodel (see Fig. 4): it consists of typed nodes and typed edges between them. The nodes represent simple system components (hardware and software resources) and composite ones (redundancy structures, subsystems and the whole system). Edges represent error propagation dependencies between simple or composite components. Types of
nodes and edges are based on the same identification (abstraction) that was introduced in the input model. Similarly, nodes and edges have a set of attributes that originates from the dependability related extension of the input model.

In general, error propagation is modelled by an U type of edge (that refer to a “uses the service of” relation). Note that the direction of error propagation is the opposite of the direction of the edge. A node that represents a composite structure (e.g., a redundancy structure) is connected to its subcomponents using a C type of edge (“is composed of” relation) that is a hyperarc (single source and several targets).

The transformation from the UML input model to the IM is the following. Stereotyped classes are mapped to nodes of the corresponding type. Redundancy structures (that are identified by the redundancy managers and the subcomponents coordinated by them) are represented by (1) the individual nodes corresponding to the subcomponents and (2) an extra node that represents the redundancy structure as a composite structure itself. The type of the extra node is FTS (fault tolerance structure). Here C types of edges are used to connect the FTS node to the nodes representing the subcomponents. The FTS node is assigned an attribute that identifies the specific error propagation which is derived from the Fault-Tree attribute of the redundancy manager. All the other components that use the service of the redundancy structure are connected to the FTS node using U edges (or by C edges, if the fault tolerant structure is embedded in a higher level fault tolerant structure). By default, the system itself is represented by another extra node, its type is SYS. The SYS node is connected to all top level components (i.e., that are not parts of a composite one) by using C edges.

Note that the IM is a flexible basis of the potential tool extensions. When a different abstraction level is used, or the dependability expert identifies a new component type (e.g., a component with specific failure process) or error propagation then a new type of node or edge can be easily included in the IM.

### 3.3 The GSPN subnet library

According to the modular dependability modeling approach, analysis subnets (GSPN subnets in our case) are assigned to the node and edge types of the IM. These subnets are stored in a subnet library and they are re-used when the system level dependability model is constructed. The interface places of the subnets are clearly identified in order to support their composition.

As the elements of this library directly correspond to the node and edge types used in the IM, the extension of the IM (as described in the previous subsection) needs the proper completion of the GSPN subnet library; if the IM is extended by new types of nodes or edges then the corresponding GSPN subnets shall be stored in the library.

The subnets can be constructed in one of two ways:

- A Petri net editor (e.g., NetLab) can be used. The GSPN subnets assembled in the editor have to be exported in the standard PNML format (Billington et al., 2003), since it is the internal format of the library. The generic failure, repair and propagation subnets were constructed in this way.

- The GSPN subnet corresponding to a non-trivial error propagation in a redundancy structure can be constructed by an automatic transformation from a fault tree (Malhotra and Trivedi, 1995). Here the fault tree describes the combination of subcomponent failures that leads to the failure of the redundancy structure. The fault trees corresponding to common redundancy structures are available in the literature. We developed an editor (integrated into the Eclipse modelling framework) to construct the fault tree and transform it into the corresponding GSPN subnet.

![Fig. 5. Failure subnet of a stateful component](image)

Each node type is assigned a failure subnet and a repair subnet. Fig. 5 presents the failure subnet that models the failure process of a stateful component (H, E and F places model healthy,
erroneous and failure states, FO is the fault occurrence rate, EL is the error delay. In case of a stateless component, there is no E (error) place.

The failure subnets of FTS and SYS nodes consists of only a pair of H and F places, since the failure of a composite component is a consequence of the failures of the subcomponents.

U edges are assigned a generic error propagation subnet that is shown in Fig. 6. The propagation probability is abbreviated as PP. Each C edge is assigned a specific subnet identified by the attribute of the source FTS node; this subnet is typically transformed from a fault tree. If the source of a C edge is a SYS node then a default “OR” fault tree is assumed. The dual of the fault trees (obtained by exchanging each AND gate with an OR gate and vice versa) is used to represent the way the composite element gets repaired when the subcomponents are repaired.

3.4 The GSPN model

In the system level dependability model construction phase, subnets corresponding to the node and edge types of the IM are taken out of the library and are connected to each other according to the relations in the IM. The tool implements generic rules that describe the composition of subnets:

- First, the interface places (H and F) belonging to the composite nodes are generated.
- Then the failure subnets of the other nodes are loaded from the library. These subnets contain the interface places (H, F and optionally E). In case of modelling repair, the repair subnet is also loaded from the library, and connected to the failure subnet. In principle, the tool merges the interface places that have the same name in the subnets of the same node.
- Similarly, the fault trees identified by the composite nodes are loaded and connected. The role names are used to identify the corresponding interface places.
- Finally, the error propagation subnets are included. In this case (see Fig. 6) the [from] and [to] qualifiers identify the corresponding places in the source and target subnets.

When the system level model is constructed, it is passed to the GSPN solver to compute the dependability measures that were specified as attributes of interest.

3.5 The UML model library

The redundancy structures typically follow well-known and widely used schemes, like e.g., N-modular redundancy in the case of hardware components, and N-version programming in the case of software components. These schemes can be provided as design patterns in UML. Note that several UML based modelling environments support the use of design patterns. In the case of our tool, the following peculiarities shall be taken into account:

- In our approach the redundancy structures are identified by the distinguished redundancy manager. Accordingly, the design patterns shall include this component.
- The design pattern shall be decorated using the stereotypes (that identify the roles of components in the scheme) and empty tagged values (that can be filled when the design pattern is applied).
- As these design patterns are instantiated as redundancy structures, they realize a nontrivial error propagation which is identified by a specific attribute of the redundancy manager. The subnet corresponding to this error propagation shall be placed into the GSPN subnet library.

Note that the addition of a new design pattern requires a coordinated extension of the set of UML design patterns, optionally the IM (if a new role is identified) and the GSPN subnet library.

4 SUPPORT FOR ASPECT-ORIENTED MODELING OF REDUNDANCY

The approach of using design patterns can be further extended to aspect-oriented modelling and analysis. The basic idea is the following. The application of fault tolerance mechanisms can be considered as a crosscutting concern that influences several components in the system architecture and has system-wide effects determining the system level dependability attributes. The designer is in charge of integrating into the core system architecture (that initially implements only the functional requirements) those additional components that are responsible for the fulfillment of non-functional requirements like repair (recovery) and redundancy management.

Aspect-oriented modelling can be used to effectively separate functional and non-functional concerns. Here the additional components that correspond to the fault tolerance aspect are included in a separate model called *aspect model*. It can be integrated with the core architecture model by a specific tool called *model weaver*. The aspect model is divided into two parts: the set of design patterns (that are called here *advice models* since they represent the mechanisms that will be integrated with the core model) and a *weaving layer* that is responsible for identifying the points in the original model where the advice models have to be applied (Domokos, 2005).

Accordingly, design decisions related to fault tolerance are modularized in this weaving layer. By using the notation that resembles aspect-oriented languages (Kiczales et al., 1997), the designer working on the weaving layer is able to designate those locations of the core architecture model that need redundancy support and select the design pattern to be applied. This modularization facilitates not only the re-use of patterns, but the traceability and assessment of redundancy-related design decisions and their modification.

From the point of view of dependability evaluation of the design decisions described in the weaving layer, our dependability modeller tool is an integral part of the model weaver. Based on the information available in the weaving layer, the model weaver constructs automatically both the integrated architecture model of the application (as a basis of further refinement) and the corresponding dependability model. The approach is sketched in Fig. 7.

When a design pattern (advice model) is integrated into the design, the corresponding analysis subnet from the GSPN subnet library is integrated into the system-level dependability model. Thus, instead of identifying redundancy structures in the original architecture model (by looking for predefined UML stereotypes), the model

---

Fig. 7. The concept of model weaving

Fig. 8. Model weaving and dependability modelling
weaver identifies these structures on the basis of the weaving layer and "instructs" the dependability modeller tool to integrate the corresponding subnet. The integration of the dependability modeller into the aspect-oriented model weaver is illustrated in Fig. 8.

5  A CASE STUDY

In the following we demonstrate the usability of the tool in a case study. The modelled system is inspired by a railway supervisory and control system (electronic control desk) that is connected to the interlocking system of a railway station either locally or through remote interfaces (Várnai, 2001, Pataricza et al., 2003). The electronic control desk is built upon industrial microcomputers and high-end PCs.

Two parts of the system are distinguished. The workstation level provides the man-machine interface and it is responsible for the control-related and data processing tasks. The main software components of a workstation are the internal database, the visualization and the data processing module. At the lower level, the interface units (IUs) are responsible for the primary processing of the input/output signals from/to the controlled system. They are connected to the workstation by so-called data concentrators.

The processing of the signals is performed in two redundant channels following the "2 out of 2" principle. Accordingly, input signals are duplicated as early as possible. With regard to the output signals, the actuators connected to the system by the IUs work if they receive the same control from the two channels. Similarly, the operator can detect the discrepancy in the observed information: the two channels share a display in which the data from the two channels are visualized alternately (thus, a discrepancy results in blinking on the display).

To simplify the presentation of diagrams, in this paper we concentrate on the availability measures of the service (system functionality) that visualizes the input data from the controlled system. We evaluated two alternative architectures:

- Two workstations are in the two channels. Two data concentrators with two separate IUs are used, but the data concentrators can be shared between the channels. The block diagram of the architecture (that is referred to as the "original system" in the following) is presented in Fig. 9.

- The workstations are replicated, implementing in this way a highly available architecture. There is an active pair of workstations and the other one takes over the active role automatically if there is a discrepancy in the output of the active pair. Thus, the service can continue without waiting for the repair of the failed components.

5.1 UML Modelling

The UML model presented in Fig. 10 contains the hardware and software components of the original system and the associations among them (here software deployment is modelled by an association). The operator is represented explicitly.

Redundancy structures are identified as follows: The operator has a redundancy manager role since she/he compares the two channels. Moreover, the data processing software components act as redundancy managers since they can switch between the data concentrators. The variants managed by these redundancy managers are assigned Variant1 and Variant2 stereotype.

The other stereotypes identify the stateful/stateless software or hardware components (SLE_SW, SFE_HW etc). Stateless components are the visualization and data processing components, as well as the data concentrators and the IUs. Associations with potential error propagation are stereotyped (note that error propagation occurs in reverse direction w.r.t. the association).
The parameters used in the experiments were the following. Fault occurrence rates are 0.01 1/day for software components and workstation hardware, 0.005 1/day for data concentrators and IUs, the operator is considered always fault-free. Error latencies are uniformly 5 minutes in each stateful component. The repair delay of stateful software components is 30 minutes (restart), while the repair (replacement) in case of the permanent faults of hardware components needs 4 hours. It was assumed that 30% of the hardware faults are permanent.

5.2 The IM and the GSPN
The structure of the IM model is presented in Fig. 11. (The IM is constructed as an XML file, its graphical layout was visualized in our IM editor.) The redundancy structures are represented by FTS nodes, the subcomponents are connected by C edges (hyperarcs). The U edges represent error propagation. The service is represented by the top-level FTS node, the measure to be computed is the availability of this service.

On the basis of the IM and the GSPN subnet library, the GSPN dependability model was constructed by the tool automatically. It consists of more than 200 places and edges. The output of the transformation is a description of the network in standard PNML and in CSPL format. The latter is the input language of the SPNP tool (Ciardo et al., 1989) used for solving the model. This CSPL file contains not only the net structure but also includes the measures to be computed and the parameters of the solution method. Service availability is computed here as the expected number of tokens in the H place belonging to the top-level FTS node.

5.3 Results of the evaluation
The simulation-based solving of the GSPN required approx. 15 minutes (considering 95% confidence level and ±0.01% confidence interval).
The availability in the case of the original architecture was 99.772%, while the introduction of the extra components increased the availability to 99.997%.

We examined how the availability of the original architecture depends on the fault occurrence rate of the workstation hardware. The results are presented in Fig. 12. The same level of availability that was computed in the case of the highly available architecture cannot be reached by using a better workstation hardware.

![Fig. 11. IM model of the original architecture](image)

Fig. 11. IM model of the original architecture

6 CONCLUSION

The most important properties of the tool can be summarized as follows:

- **Automated construction of dependability models:** The GSPN dependability model is constructed from an (enhanced) design model automatically. Currently UML class diagrams are supported, in the next step we will support standard architecture description languages as well.

- **Adaptability of the tool to different input models:** The modular approach of dependability modelling and the concept of using the IM as a core mathematical model provides an easy way to adapt the tool to different input languages that describe various component types (e.g., actuators with specific fault occurrence or error propagation properties). Adaptation is possible by (1) adding new node and edge types to the IM metamodel and (2) inserting the corresponding analysis subnets into the library.

- **Extensibility of the subnet library:** Analysis subnets can be translated from fault trees, or can be constructed directly in GSPN editors. We assembled an initial library that includes subnets of common component types and redundancy structures. Refined subnets can be included in the library if more information about the dependability properties of a component become available in the design process.

- **Aspect-oriented modelling of redundancy structures:** Architectural design patterns defined in the design language can be selected in a separate model layer that modularizes dependability related design decisions. Our tool is part of a model weaver that integrates
the design patterns into the core architecture model and at the same time constructs the system-level dependability model using the subnets belonging to the patterns. Although the parameters of components are often estimations, this imprecise parameterization still allows to compare architectural choices (to check which solution is better), or to identify those components to which the system level availability is highly sensitive. Another type of usage is the “what-if” kind of analysis to estimate the effects of new components or solutions.

The dependability modeller tool was developed and successfully used in the framework of the Economic Competitiveness Operative Programme supported by the Hungarian National Office of Research and Technology, in the project entitled “EU-conform, constructive safety assessment of railway control systems”. Our industrial partner was the Prolan Control Co. The use of the tool is anticipated in two ongoing EC-funded research projects.

7 LITERATURE


