

Simple Models for High-Availability Systems with Dependent Components

M. Walter & C. Trinitis

Institut für Informatik 10 der Technischen Universität München, Germany

ABSTRACT: When modeling fault-tolerant systems, state-based methods yield much more realistic results in comparison to traditional combinatorial methods. To avoid the difficult manual design of large state-based models, we advocate an approach, by which a high-level input model is used from which a semantically equivalent low-level model is automatically generated. This approach was implemented in the tool OpenSESAME (Simple but Extensive Structured Availability Modeling Environment). Its input uses reliability block diagrams as a wide-spread modeling technique favored by many reliability engineers. In addition, users can specify inter-component dependencies of the system without having to create a state-based model. The main contributions of this paper are, first, a detailed description of the input model showing the application areas and limitations of OpenSESAME; second, a detailed explanation of the transformation process into the state-space domain; and third, it contains a realistic industrial example modeling a water supply system of a city.

1 INTRODUCTION

The wealth, well-being, and safety of all humans more and more relies on the reliability, availability, safety and security of machines. Unfortunately, our machines become more and more complex, making it very difficult to ensure their correct and safe functionality. Fault tolerant systems will therefore play an important role in future systems.

For several reasons, it is of importance to quantitatively evaluate the dependability of these systems. First, for safety-critical systems, it is important to know the risk of using/operating the system. For instance, it is frequently quoted that components used in an airplane should have an unreliability of less than 10^{-9} per hour of flight. Second, it is often not trivial, how a fault tolerant systems must be built to achieve the highest level of dependability, given a fixed set of components (see e.g. Section 4 of this paper). A similar question is: which part of the system should be made more dependable to get the most increase in dependability? Third, increasing the dependability of a system by adding redundancy usually implies some performance losses and/or increased costs. Thus, the decreased performance/cost ratio must be justified by a credible increase in reliability or availability. Fourth, from a somewhat different perspective, a quantitative evaluation of the dependability of a system may help to make the right decisions when purchasing such a system.

For two obvious reasons, a *measurement* of the reliability/availability of a fault tolerant system is not practicable. First, if the system is in its design phase,

measurements cannot be done at all. Second, by definition, fault-tolerant system fail rarely which means that there is not enough data for statistically meaningful analyzes.

Therefore, *stochastic models* are used which are used to predict the dependability of systems from the dependability of their components. *Fault trees* (FT) and *reliability block diagrams* (RBD) are the most common modeling techniques for fault tolerant systems. Unfortunately, the traditional solution methods for these models rely on the assumptions that there are no inter-component dependencies between the components of the system. Therefore, these techniques yield dangerous, over-optimistic results as they do not take into account failures with a common cause, failure propagation, imperfect failure detectors, fail-over times (spent for fault detection, localization, and isolation as well as reconfiguration), physical disturbances (e.g. heat, electromagnetic noise, vibration) and limited repair personnel.

State based models like Markov chains can be used to model systems including these dependencies. As the state space grows exponentially with the number of components, it is usually implicitly defined using e.g. stochastic process algebras or stochastic Petri nets. However, these formal and rigorous modeling methods are difficult to learn and not very user friendly. Without going into detail, practice shows that e.g. Petri nets have the following undesired properties which hampers their applicability:

- They are not very intuitive. A Petri net cannot be understood without knowing the firing

rules. Additionally, they resemble finite automata which often leads to misinterpretation and confusion.

- Traditional nets do not support modularization and building hierarchic models. Modern extensions like shared places or super-transitions are likewise not very intuitive.
- Petri net models are difficult to modify. For instance, adding an inter-component dependency can mean that a completely new model has to be created.
- Not all aspects of the model can be parametrized. Often, the Petri net structure depends on some variables like the number of redundant components in the system. Thus, a new net has to be designed each time the parameter changes.
- As a conclusion, Petri nets do not support a modeling process based on the principle of stepwise refinement. That means, that starting with a simple model, and iterating over an evaluation and refinement phase is *not* possible.

On this account, we propose to not use Petri nets as an *input* model but rather use a tool which creates the nets from a model which has a higher degree of abstraction. In this paper, we advocate the tool OpenSESAME (Simple but Extensive Structured Availability Modeling Environment) which creates state based models from reliability block diagrams, which can be enriched with inter-component dependencies.

In Section 2, we describe the input model of OpenSESAME and evaluate it regarding its user-friendliness. In Section 3, we explain how the input model is transformed into a state-based model, which can then be evaluated by existing tools. Section 4 presents a real-world example and shows how it can be modeled using OpenSESAME. The related work is described in Section 5. Finally, Section 6 summarizes the paper and gives an outlook on open problems and future work.

2 OPENSESAME INPUT DIAGRAMS

An OpenSESAME model as seen by the user comprises component tables, reliability block diagrams, failure dependency diagrams, repair group tables, and variable tables. Not all model parts are necessary, usually one starts with one component table and a block diagram, only. Then, the model can be refined by adding additional tables and diagrams. In the following, the individual parts of the model are described.

The *component tables* list all components the system consists of. Each component type has a unique name, a mean time to failure (MTTF), and a mean time to repair (MTTR). If the system contains several components of the same type, the table also lists the number of components of this type. Furthermore,

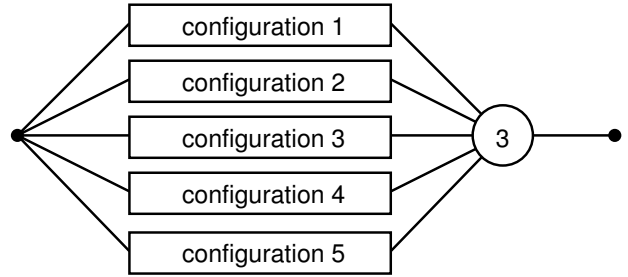


Figure 1: 3-of-5:G system. The system is available, if at least 3 out of 5 configurations are available.

each component may have either a dedicated repair person or is allotted to a repair group (see below). For small and medium sized models, a single component table will be sufficient. For large models, several tables can be used to group related components.

An extended version of *reliability block diagrams* (RBD) is used to model the redundancy structure of the system. RBDs are undirected graphs where each edge is labeled with a component. A component may appear several times in the same RBD. Two special nodes s and t define a Boolean system which is available, if there is a connection between these nodes and unavailable otherwise. As components can be unavailable so can the edges: calculating the probability whether s and t are connected yields the availability of the modeled system. In OpenSESAME, several modern extensions to traditional RBDs were implemented. First, the user may specify more than two border nodes. This allows for calculating the availabilities of subsystems in addition to the overall availability. Second, edges may be labeled with a sub-RBD instead of a component. This allows for building a hierarchy of RBDs. Thus, even large systems can be modeled in a concise way. Third, so-called k-of-N:G edges are supported.

As an example, Figure 1 shows a 3-of-5:G system which is available if at least 3 of its 5 so-called configurations are available. A configuration may be a simple component, or an arbitrary large sub-diagram. Alternatively, the system could be modeled using regular RBDs without k-of-N:G edges, however, such an RBD would comprise 30 edges.

Finally, as a unique feature of OpenSESAME, the model can be enriched with inter-component dependencies. Because some dependencies are related to the redundancy structure of the system, it makes sense to add these dependencies to the RBD. For example, in many systems fault tolerance is achieved using fault recovery techniques. In these systems, the redundant components are in passive or standby mode as long as the system is fault free. In contrast to so-called active-active systems which are based on fault masking, the redundant components can be used for non-critical tasks. Furthermore, in systems with fault recovery, a redundant component can possibly replace several components, which allows the construction of N+1 redundant systems. However, such systems also have a drawback compared to systems based on fault mask-

ing: the failure of an active component must be detected, localized and isolated, and the redundant component must be activated after the failure of the primary component. During this so-called fail-over time, the system is unavailable.

To avoid over-optimistic results and unfair comparisons, availability models should therefore take into account possible fail-over times of fault recovery mechanisms. In OpenSESAME, k -of- N : G edges can therefore be attributed with a fail-over time.

Not all inter-dependent dependencies result from the fault tolerance mechanisms. They can appear between any two or more components of the system. Adding these dependencies to the RBD might therefore be difficult. For instance, it might happen that there is a failure propagation between two components which were modeled in different sub-RBDs.

To explicitly make a distinction between the redundancy structure on the one hand, and failure dependencies on the other hand, OpenSESAME supports a second kind of diagram, called *failure propagation diagram* (FDD). These diagrams can be used to model failures with a common cause, different kinds of failure propagation as well as imperfect failure detection mechanisms (imperfect coverage).

FDDs are directed graphs: the nodes are labeled with components, whereas the edges are labeled with a type and a probability. Currently, two types of failure dependencies are supported: destructive failure propagation and blocking failure propagation. Drawing an edge of destructive failure propagation with parameter p between two components A and B means, that whenever A fails, the failure is propagated to B with probability p . Thus, B fails as well and must be repaired before it becomes available again. Destructive failure propagations are drawn using solid lines. Blocking failure propagations are drawn by dashed lines. In this case, component B is blocked, i.e. becomes unavailable. However, it cannot/must not be repaired by itself. However, as soon as component A is repaired, component B becomes available, too.

As an FDD can contain several edges, and one OpenSESAME project can comprise several FDDs, sophisticated failure dependencies can be modeled in a concise and clear way. For example, Figure 2b) and 2c) show several possible dependencies originating at source component S and affecting two target components T_i and T_j . In Figure b) the targets are independently affected: if there is a propagation from S to T_i , T_j is affected with the same probability as if there is no propagation from S to T_i . In contrast, Figure c) shows common failure propagation: here, either both components T_i and T_j are affected by the propagation, or none. This behavior is modeled by using a so-called pseudo component P. This component does not appear in the redundancy structure and cannot fail (its MTTF is ∞). However, it might be the target of a failure propagation from component S. If it fails, it propagates the failure to both T_i and T_j with probability 1.



Figure 3: Generic OpenSESAME model for a k -of- N : G system with blocking failure propagation.

ity 1. To ensure a correct model, T has to be attributed with a MTTR of zero (destructive propagation) or ∞ (blocking propagation), respectively.

As a final example how FDDs can be used, please refer to Figure 2d). Here, the pseudo component P is the source of the failure propagation. This FDD can be used to model common cause failures. P is attributed with the mean time between such events. If P fails, i.e. the common cause occurs, a failure is induced in both T_i and T_j .

Stochastic dependencies may also occur due to a limited amount of repair personnel. Unless every component has a dedicated repair person, delays in the repair may occur if multiple components are failed at the same time and the repair personnel is overburdened. In OpenSESAME, this can be modeled by means of a *repair group table*. Each entry of this table describes a repair group in terms of its size, i.e. number of repairs which can be done concurrently. In its current implementation, OpenSESAME assumes a random repair policy for each repair group. Later revisions of OpenSESAME might also support different repair strategies like first come (i.e. failed) first served, shortest repair first or closest repair next etc.

In many scenarios, a model must be analyzed with a varying set of parameters. For instance, one might want to evaluate the availability of a system with varying failure and repair rates or a varying number of redundant components. It is a big advantage of OpenSESAME, that *all* parameters of the model might be stated in the form of a variable.

For this purpose, OpenSESAME supports three types of variables:

- *Mean times* represent positive real numbers and are used for MTTF and MTTR values, as well as fail-over times in RBDs.
- *Probabilities* are numbers from the interval $[0:1]$ and are used to attribute edges in FDDs.
- *Naturals* can be used to specify the number of components of one type, the parameter K in k -of- N edges and the capacity of a repair group.

For example, Figure 3 shows a generic model of a system comprising N components. The system is available, if at least K components are available. If one of the components fails, which probability p , all other components become blocked until this component is repaired. With OpenSESAME, the model can be solved with different values for N , K , p and the MTTF and MTTR of the component without user interaction.

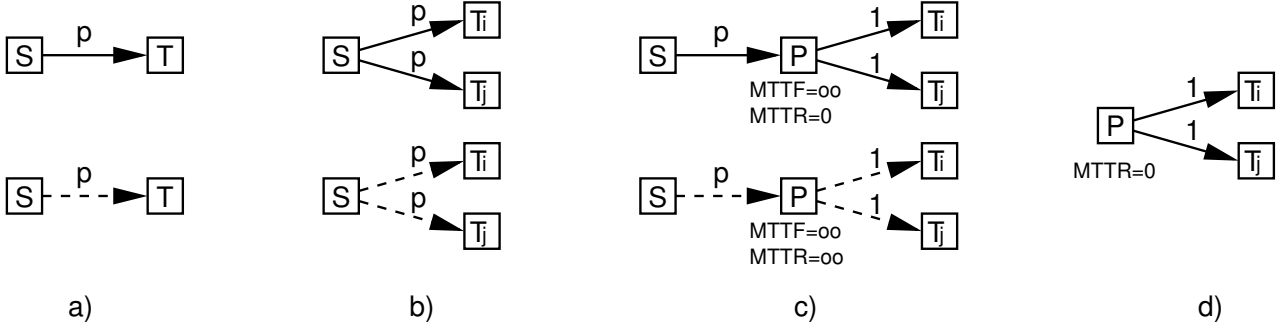


Figure 2: Failure dependency diagrams (FDD). Destructive propagation is depicted by solid arrows, whereas dashed arrows are used for blocking propagation.

3 TRANSFORMATION PROCESS

As the models of the proposed method contain inter-component dependencies between the failure and repair events, they cannot be evaluated with classical evaluation techniques for Boolean models. Thus, we propose another solution which is based on a transformation of the input diagrams into one or more state-based models, which in turn can be analyzed using an evaluation tool for this model class. Currently, Generalized Stochastic Petri Nets (GSPN, (Ajmone Marsan, Balbo, Conte, Donatelli, and Franceschinis 1995; Sahner, Trivedi, and Puliafito 1996)) are created from the input diagrams which are analyzed with the tool DSPNexpress (Lindemann 1998). However, similar to changing the back-end of a compiler to migrate it to a new architecture, the method can be adapted to support new modeling methods like stochastic process algebras (Hermanns, Herzog, and Katoen 2002) or tools like GreatSPN (Ajmone Marsan, Balbo, Conte, Donatelli, and Franceschinis 1995), Möbius (Clark, Courtney, Daly, Deavours, Derisavi, Doyle, Sanders, and Webster 2001), TimeNet (German, Kelling, Zimmermann, and Hommel 1995), CASPA (Kuntz, Siegle, and Werner 2004), or SHARPE (Sahner, Trivedi, and Puliafito 1996).

At a glance, the transformation process is made up of nine phases as shown in Figure 4. In the first phase, the model is simplified to facilitate the actual transformation process which follows this phase. The result of the first step is a simplified input model. This model does no longer contain any variables, hierarchies, and modularization. Next, the resulting RBD is converted into a Boolean structure formula. For small systems, this can be done by finding all minimal paths of the RBD. Each path then represents one conjunction-term of the redundancy structure given in disjunctive normal form. Thus, the redundancy structure is of the form:

$$\bigvee_{i=1,2,\dots,n} (c_{i1} \wedge c_{i2} \wedge \dots \wedge c_{im})$$

In the worst case, the number of paths grows exponentially with the size of the RBD. Thus, we are currently implementing a methods to directly convert the RBD into a binary decision diagram based on methods which are known to work well for a diverse set of

graphs (Kuo, Lu, and Yeh 1999).

In a third step, the interdependence-oriented input diagrams are converted into a component-oriented data structure. This data structure comprises one object for each component of the system, which is attributed by the inter-dependencies this component is involved in. Next, the model is divided into independent parts. To accomplish this, the components of the models are grouped by the inter-component dependencies in such a way that there are no dependent components which are in two different groups. In the worst case, there will be dependencies between all components of the system and therefore only one group after this evaluation step. However, in many real-life systems, there are independent subsystems that can be solved separately, which greatly reduces the computation costs in the next phases.

However, this also requires to divide the redundancy structure according to its variables. A simple way to do this is by using a method similar to the Shannon decomposition which is best shown using a small example. Consider a 3-of-4:G system with four components A, B, C, and D. Its redundancy structure looks like this:

$$\begin{aligned} \phi = & (A \wedge B \wedge C) \vee (A \wedge B \wedge D) \vee \\ & (A \wedge C \wedge D) \vee (B \wedge C \wedge D) \end{aligned}$$

Let us further assume that there are some dependencies between the components A and B as well as some dependencies between components C and D. However, the subsystems AB and CD are stochastically independent. Therefore, one Petri net for AB and one Petri net for CD is created. The redundancy structure ϕ can be divided as follows:

$$\begin{aligned} \phi = & [(A \wedge B) \wedge \phi_{A=\text{true},B=\text{true}}] \vee \\ & \vee [(A \wedge \bar{B}) \wedge \phi_{A=\text{true},B=\text{false}}] \vee \\ & \vee [(\bar{A} \wedge B) \wedge \phi_{A=\text{false},B=\text{true}}] \vee \\ & \vee [(\bar{A} \wedge \bar{B}) \wedge \phi_{A=\text{false},B=\text{false}}] \end{aligned}$$

| |
|--|
| 1. preprocess model |
| 2. gain redundancy structure |
| for each variable substitution |
| 3. find independent submodels |
| 4. create component oriented data structures |
| 5. decompose redundancy structure |
| for each submodel |
| 6. create Petri net |
| 7. evaluate Petri net |
| 8. compose result |
| 9. postprocess results |

Figure 4: Overview on the transformation process.

As all sub-terms are disjoint, it holds:

$$\begin{aligned}
Pr\{\phi\} &= Pr\{(A \wedge B) \wedge (C \vee D)\} + \\
&+ Pr\{(A \wedge \bar{B}) \wedge (C \wedge D)\} + \\
&+ Pr\{(\bar{A} \wedge B) \wedge (C \wedge D)\} + \\
&+ Pr\{(\bar{A} \wedge \bar{B}) \wedge (\text{false})\}
\end{aligned}$$

and finally, because the subsystems AB and CD are stochastically independent:

$$\begin{aligned}
Pr\{\phi\} &= Pr\{(A \wedge B)\} \cdot Pr\{(C \vee D)\} + \\
&+ Pr\{(A \wedge \bar{B})\} \cdot Pr\{(C \wedge D)\} + \\
&+ Pr\{(\bar{A} \wedge B)\} \cdot Pr\{(C \wedge D)\}
\end{aligned}$$

These probabilities can be computed by constructing and analyzing a stochastic Petri net (SPN) for each set of interdependent components. This is done by first choosing an appropriate SPN for each component, depending on the dependencies of this component and then connecting these sub-SPN in a way reflecting the dependencies between the components.

The SPNs will then be saved to disk in a format depending on the Petri Net tool being used to evaluate the state-based model. Inversely, after the evaluation of the Petri Net, the results are read from the disk by the software for post-processing. The most important step in the post-processing phase is to combine the results according to the structure formula.

Additionally, other post-processing, like summarizing the results of several solution runs with different parameter assignments in a table or diagram (see e.g. Figure 8), can follow.

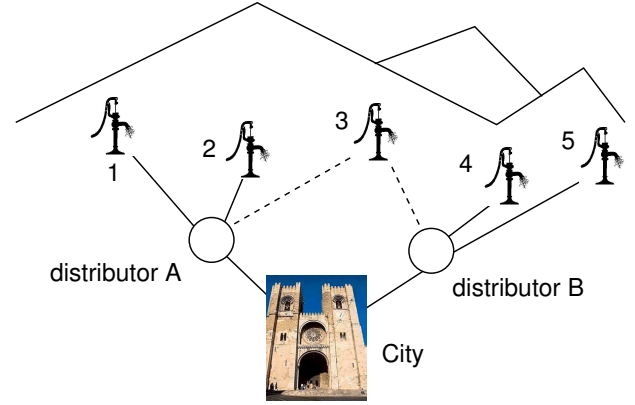


Figure 5: Water supply system of a city comprising up to five pumps and two water distributors.

4 EXAMPLE

For an illustrative example, please refer to Figure 5, showing a water supply systems of a city. The system consists of up to five pumps (1, 2, ..., 5) and two distributors (A and B). The water of pumps 1 and 2 flows into A whereas the water of pumps 4 and 5 flows into B. The water of pump 3 can flow into both A or B. We assume that there is enough water for the city as long as it is supplied by water from at least two pumps flowing through A, B, or a combination of both.

We assume that the pumps and distributors have a mean time to failure of one year. For the sake of simplicity, we assume that all failure and repair times are exponentially distributed. All pumps and distributors fail independently from each other. Immediately after a failure, pumps and distributors are being repaired, which takes 24 hours in average.

Pump 3 is special in terms of its failure behavior: after a failure, with probability p , the pump delivers dirty water which spoils any distributor to which it is delivered. Consequently, the distributor must be shut down until the pump is repaired. This arises the question which distributor should be connected to pump 3. In the following, we will compare three possible configurations C1, C2, and C3:

- C1: To avoid any possible contamination, pump 3 is not used at all.
- C2: Pump 3 is connected to both distributors to achieve the highest degree of redundancy.
- C3: As a compromise, pump 3 is connected to distributor A only.

The left reliability block diagram (RBD) of Figure 6 models configuration C1. If both distributors are available, the middle path can be chosen. This path is available, if at least 2 of the 5 pumps are available. If distributor B is failed, the upper path must be taken. This path requires the pumps 1 and 2 to be available. Similarly the lower path requires the pumps 4 and 5 to be available and must be taken, if distributor B is failed. The RBD can be analyzed with any modeling

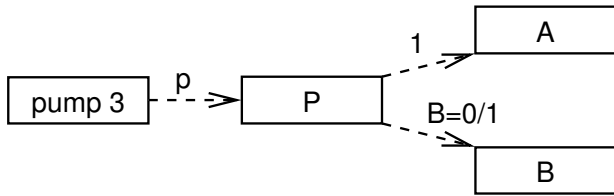


Figure 7: Failure dependency diagram. If pump 3 fails, with probability p the failure propagation is either propagated to distributor A ($B=0$) or distributor A and B ($B=1$).

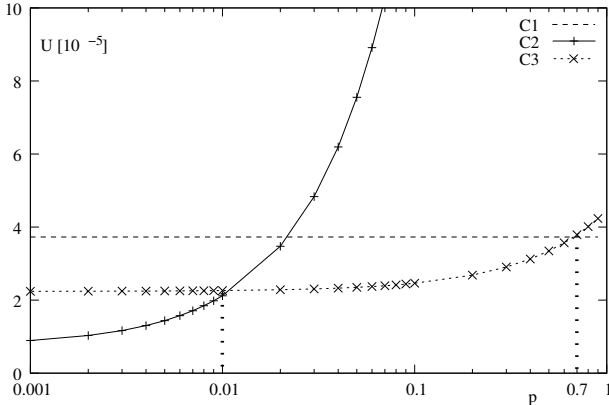


Figure 8: Unavailability (U) of the system vs. the failure propagation probability of pump 3. For configuration 1, U has the constant value $3.73 \cdot 10^{-5}$. This configuration is the best if $p > 0.7$. If $p < 0.01$, the best result is obtained using configuration 2. For all other cases ($0.01 < p < 0.7$) configuration 3 should be used.

tool for reliability block diagrams which supports repeated events. For example, the evaluation with OpenSESAME yields an unavailability of the city's water supply of $3.73 \cdot 10^{-5}$. In average, the city is without water for 19.6 minutes per year.

The block diagrams for C2 and C3 are also shown in Figure 6. These models cannot be evaluated using traditional solution methods due to the failure propagation from pump 3 to the distributors. Thus, the corresponding OpenSESAME-model is enriched with the failure dependency diagram shown in Figure 7. For C2, the parameter B takes the value 1: a failure of pump 3 will first propagate to the pseudo-component P with probability p , and from there to both distributors A and B . The pseudo-component P cannot fail by itself and cannot be repaired (Its MTTF and MTTR are both infinite). It ensures that A and B fail simultaneously in case of a failure propagation. The failure propagation is of the blocking type: as soon as pump 3 is repaired, the pseudo component as well as both A and B become available immediately. For C3, the parameter B takes the value 0. Thus, a pump failure can affect distributor A only.

An evaluation of configuration C2 and C3 using OpenSESAME yields the results shown in Figure 8. It shows the unavailability of the three configurations with respect to the parameter p .

The unavailability of C1 is constant as it does not depend on p . For very small values of p , C2 yields the best unavailability (between 3.96 and 11.1 min-

utes per year). However, the unavailability increases rapidly when p becomes larger. In this case, pump 3 acts as a single point of failure, as it can spoil the overall system. Thus, if p is larger than 1 %, C2 is the better configuration. Finally, for large values of p , i.e. $p > 0.7$, it is better to not use pump 3 at all which results in the above-mentioned unavailability of 19.6 minutes per year.

5 RELATED WORK

5.1 SAVE

The System AVailability Estimator (SAVE, (Goyal, Carter, de Souza e Silva, Lavenberg, and Trivedi 1986)) was one of the first tools which were able to automatically generate Markovian Chains. However, this tool does not satisfy today's expectations regarding manageability and universality. For instance, the tool does not provide a graphical input model and does not support dynamic redundancy.

5.2 SHARPE

This work provides an environment which allows the combination of several different models like block diagrams, fault trees, Markovian Chains, Stochastic Reward Nets etc. (see (Sahner, Trivedi, and Puliafito 1996)). In this way, the modeler can choose a different model type for different parts of the system and can therefore choose an appropriate level of manageability and accuracy for each part of the model. However, the tool does not allow to incorporate inter-component dependencies between two components which are modeled in two different diagrams. Thus, the set of components has to be divided into independent subsets at the very beginning of the modeling process. This division cannot be changed later, which hampers a stepwise refinement and the modifiability of the model.

5.3 MEADEP

The software package MEADEP (Tang, Hecht, Miller, and Handal 1998) follows a similar approach than SHARPE. Here, Reliability Block Diagrams and Markovian Chains can be combined into one availability model which is evaluated bottom up, i.e. the evaluation results of the lower models are used as parameters for the other models until the top-level model has been resolved. However, MEADEP does not allow to integrate inter-components dependencies to span over several diagrams.

5.4 HIDE

In contrast to SAVE and MEADEP, the program HIDE (Bondavalli, Dal Cin, Latella, Majzik, Pataricza, and Savoia 2001) and its successors (Majzik, Pataricza, and Bondavalli 2003) are not based on combining several modeling techniques but on converting a high level input model into state-based models. In other terms, HIDE uses a similar principle than

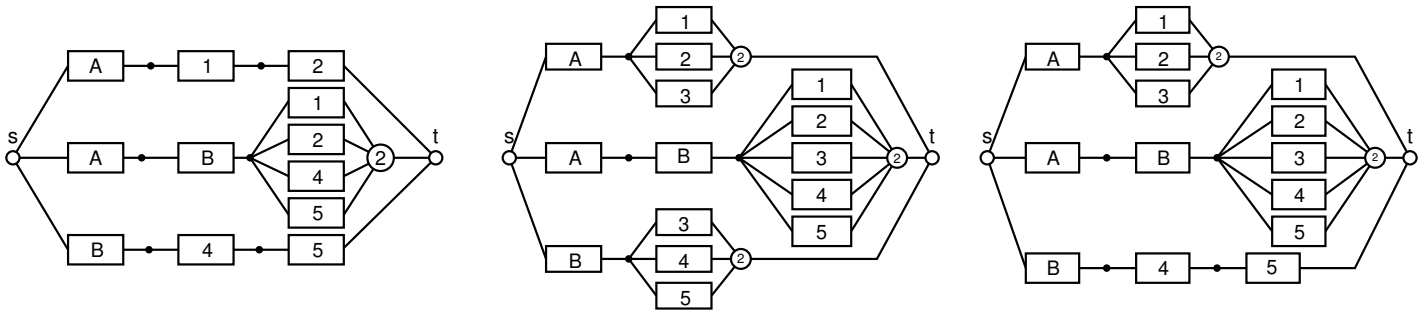


Figure 6: Reliability block diagram for configuration C1 (left), C2 (middle), and C3 (right).

the work presented in this paper. However, there are significant differences in the type of input diagrams. Whereas in our work the goal was to create simple, intuitive and manageable diagrams for modelers which are familiar with traditional combinatorial modeling methods, the HIDE project uses UML diagrams for input. This has the advantage that availability estimations can be gained “for free”, if such UML diagrams already exist, e.g. if they were created during the design phase of the system. However, if a model has to be created from scratch, many modelers will feel more comfortable with modeling methods which resemble traditional modeling methods for highly available systems, like the widely used Reliability Block Diagrams.

5.5 DIFtree

Another tool which transforms high-level input diagrams into state-spaced models is the software tool DIFtree (Doyle, Dugan, and Patterson-Hine 1995; Dugan, Sullivan, and Coppit 2000), which has been incorporated into the modeling environment Galileo (Coppit and Sullivan 2000). The input diagrams are so called Dynamic Fault Trees, which extend traditional fault trees by a set of new gates that can handle different kinds of redundancy (e.g. cold, warm, and hot redundancy). However, all dependencies have to be modeled by using these gates. Therefore, a failure propagation between two components in two different branches of the tree cannot be introduced without a major redesign of the model. Furthermore, many modelers prefer Reliability Block Diagrams to fault trees when modeling High Availability systems, as they are more closely related to the system’s schematic layout.

5.6 UWG3

More recently, a new component concept for fault trees was introduced and implemented in the tool UWG3 (Kaiser, Liggesmeyer, and Mäckel 2003). In contrast to traditional fault trees, the tool is based on so-called cause effect graphs (CEG). These graph do not contain repeated events which are often a source of modeling errors in traditional fault trees, especially if one wants to modularize a large tree. In addition, with CEG, several top events can be modeled in one diagram. To achieve this usability without losing the

generality of fault trees including repeated events, dependencies between the modules of the tree can be modeled using so-called ports. However, in contrast to OpenSESAME, these mechanisms model strong dependencies between parts of the model which represent the same components of the system rather than weak dependencies between the failure and repair behavior of different components of the system.

5.7 Boolean Logic Driven Markov Processes

In a recent article (Bouissou and Bon 2003), an innovative approach for combining fault trees and Markov models is presented. Each leaf of the fault tree represents a component of the system which can be described in more detail by a Markov process. Switching between the states of the chain can be triggered by the failure or repair of other components, which allows for modeling inter-component dependencies. To simplify the task of the modeler, several predefined standard cases can be reused. These cases include warm standby redundancy, on demand failures, and components with an increasing failure rate (i.e. aging components). The applicability of this technique to real world problems has been shown in the context of modeling fault-tolerant electrical high voltage apparatus.

6 CONCLUSIONS & FUTURE WORK

This paper shows how the availability modeling tool OpenSESAME can be used to quantitatively evaluate complex fault tolerant systems with dependent components. In its current implementation (version 1.0), OpenSESAME supports arbitrary monotone redundancy structures which can be enriched with the following dependencies: different kinds of failure propagation, failures with a common cause, imperfect coverage, fail-over times, and limited repair personnel. Variables can be used for all aspects of the model which allows for an automatic variation of the model.

OpenSESAME cannot model non-monotone or non-Boolean systems and is not suited to compute other measures like reliability, MTTF or MTTR of a system. It can furthermore not be used to model phased-mission systems. In its current implementation, OpenSESAME assumes constant failure and repair rates, only. However, we plan to extend the modeling power of OpenSESAME in our future work.

Like any state-based method, OpenSESAME suffers from the state-space explosion problems if the number of components becomes large. This problem was already partially solved by an divide and conquer approach as described in Section 3. However, this techniques only works for models where several sets of interdependent components (SIC) can be identified. Therefore, we are currently extending the back-end of OpenSESAME to support other modeling tools which are able to cope with larger state space sizes. For example, the tool Möbius (Clark, Courtney, Daly, Deavours, Derisavi, Doyle, Sanders, and Webster 2001) can evaluate Petri nets by simulation, avoiding the explicit generation of the state space. Additionally, we are currently implementing a generator for stochastic process algebras instead of Petri nets. For this modeling paradigm tools like CASPA (Kuntz, Siegle, and Werner 2004) exists, which can store the state space implicitly in a symbolic representation (Lampka and Siegle 2006), which greatly reduces the memory requirements.

7 AVAILABILITY

A copy of OpenSESAME can be obtained from <http://OpenSESAME.in.tum.de/> free of charge.

REFERENCES

Ajmone Marsan, M., G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis (1995). *Modelling with Generalized Stochastic Petri Nets*. Wiley and Sons.

Bondavalli, A., M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, and G. Savoia (2001). Dependability Analysis in the Early Phases of UML Based System Design. *Journal of Computer Systems Science and Engineering* 16, 265ff.

Bouissou, M. and J. L. Bon (2003, 11). A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering and System Safety*, 149–163.

Clark, G., T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster (2001). The Möbius modeling tool. In *Proc. 9th International Workshop on Petri Nets and Performance Models*. IEEE Computer Society Press.

Coppit, D. and K. J. Sullivan (2000). Galileo: A tool built from Mass-Market Applications. In *Proceedings of the 22nd International Conference on Software Engineering*, pp. 750–753. IEEE Computer Society Press.

Doyle, S., J. Dugan, and F. Patterson-Hine (1995, March). A Combinatorial Approach to Modeling Imperfect Coverage. *IEEE Transaction on Reliability* 44(1), 87ff.

Dugan, J., K. Sullivan, and D. Coppit (2000, March). Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transaction on Reliability* 49(1), 49ff.

German, R., C. Kelling, A. Zimmermann, and G. Hommel (1995). TimeNet: A toolkit for evaluating non-Markovian stochastic Petri nets. *Performance Evaluation* 24, 69ff.

Goyal, A., W. Carter, E. de Souza e Silva, S. Lavenberg, and K. S. Trivedi (1986, July). The System Availability Estimator (SAVE). In *Proc. Sixteenth International Symposium on Fault-Tolerant Computing*. IEEE Computer Society Press.

Hermanns, H., U. Herzog, and J.-P. Katoen (2002). Process algebra for performance evaluation. *Theor. Comput. Sci.* 274(1-2), 43–87.

Kaiser, B., P. Liggesmeyer, and O. Mäkel (2003). A new component concept for fault trees. In *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*, Volume 33.

Kuntz, M., M. Siegle, and E. Werner (2004). CASPA - A Tool for Symbolic Performance and Dependability Evaluation. In *Proceedings of EPEW'04 (FORTE co-located workshop)*, pp. 293 – 307. Springer, LNCS 3236.

Kuo, S.-K., S.-K. Lu, and F.-M. Yeh (1999). Determining Terminal-Pair Reliability Based On Edge Expansion Diagrams Using OBDD. *Transaction on Reliability* 48(3).

Lampka, K. and M. Siegle (2006). Activity-local symbolic state graph generation for high-level stochastic models. In *Proceedings of the 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2006)*.

Lindemann, C. (1998). *Performance Modelling with Deterministic and Stochastic Petri Nets*. Wiley and Sons.

Majzik, I., A. Pataricza, and A. Bondavalli (2003). Stochastic Dependability Analysis of System Architecture Based on UML Models. *Lecture Notes in Computer Science* 2677, 219–244.

Sahner, R., K. Trivedi, and A. Puliafito (1996). *Performance and Reliability Analysis of Computer Systems*. Kluwer Academic Publishers.

Tang, D., M. Hecht, J. Miller, and J. Handal (1998, December). MEADep: A dependability evaluation tool for engineers. *IEEE Transaction on Computers* 47(4), 443ff.

Walter, M. and W. Schneeweiss (2005). *The modeling world of Reliability/Safety Engineering*. LiLoLe Verlag.

Walter, M. and C. Trinitis (2004). How to Integrate Inter-Component Dependencies into Combinatorial Availability Models. In *Proc. Ann. Reliability and Maintainability Symp. (RAMS 2004), Los Angeles, USA*, pp. 226–231. IEEE Computer Society Press.

Walter, M. and C. Trinitis (2005). OpenSESAME: Simple but Extensive Structured Availability Modeling Environment. In *Proc. 2nd International Conference on the Quantitative Evaluation of Systems (QEST) 2005*. IEEE Computer Society Press.