An easy-to-use, efficient tool-chain to analyze the availability of telecommunication equipment

Kai Lampka[1], Markus Siegle[1] and Max Walter[2]

[1] Universität der Bundeswehr München, Institut für Technische Informatik,
{kai.lampka,markus.siegle}@unibw.de
[2] Technische Universität München, Lehrstuhl für Rechnertechnik und Rechnerorganisation, max.walter@in.tum.de

Abstract. The tool OpenSESAME offers an easy-to-use modeling framework which enables realistic availability and reliability analysis of fault-tolerant systems. Our symbolic engine, which is based on an extension of binary decision diagrams (BDDs), is capable of analyzing Markov reward models consisting of more than 10^8 system states. In this paper, we introduce a tool chain where OpenSESAME is employed for specifying models of fault-tolerant systems, and at the back end our symbolic engine is employed for carrying out numerical Markov reward analysis. For illustrating the applicability of this approach, we analyze a model of a fault-tolerant telecommunication service system with N redundant modules, where the system is available as long as at least K modules are available. Based on this model, it is shown, that the suggested tool chain has more modeling power than traditional combinatorial methods, e.g. simple reliability block diagrams or fault trees, is still easy-to-use if compared to other high-level model description techniques, and allows the analysis of complex system models where other tools fail.

Key words: Reliability and Availability Analysis,
Markov Reward Model, State Space Explosion
Binary Decision Diagram, Reliability Block Diagrams

1 Introduction

Motivation: Obtaining measurement data in order to quantify the reliability and availability (RA) of a system is often very difficult in practice, or even impossible. Thus one is restricted to analyzing a system (or high-level) model, rather than analyzing the system directly. Reliability block diagrams (RBD) are an adequate technique for describing systems, when RA-issues are emphasized. Furthermore, RBDs are a well accepted method in industrial practice. However, using RBDs assumes that, firstly, all failure and repair events in the system are stochastically independent, and secondly, that each component can be in two states only (active or failed). In contrast, Markov Reward models (MRMs) provide a powerful mathematical framework for computing system state probabilities and thus quantifying a system under study. The modeling power of MRMs is much higher than that of RBDs: Each component can be described by an arbitrary number of states (e.g. active, passive, and several failed states), and arbitrary inter-component dependencies (such as failure propagation, failures with a common cause, or limited repair capacities) can be specified. In contrast to empirical evaluation as provided by simulation studies, which is the most accepted technique in industry, MRM-based studies are restricted to models where events occur with an exponential or zero delay. However, this restriction comes at the benefit that MRMs allow an extensive (!) analysis, such that rare events of fatal impact can also be assessed, where simulation studies may fail to do so. Consequently, MRMs are an adequate formal model for analyzing in particular industrial critical systems.

In this work we consider a tool chain, in which the tool OpenSESAME (simple but extensive structured availability modeling environment) is used as the user interface. In this tool, systems are modeled using RBDs, which can be enriched with intercomponent dependencies. Thus, the traditional limitations of these easy-to-use models were overcome. OpenSESAME automatically converts these diagrams into a high-level model specification (e.g. a stochastic Petri net (SPN)). The interleaving semantics of standard high-level model description methods, such as SPN among others, applied for transforming the obtained high-level model into its low-level representation (commonly denoted as state graph (SG)), may lead to an exponential blow-up in the number of system states. This phenomenon, commonly addressed as *state space explosion* problem, often hampers the analysis of complex and large systems, if not making it impossible at all. Here symbolic methods have shown to ease the problem, such that system models consisting of more than, say, 10^8 system states are still treatable and their RA-measure are still obtainable on commodity computers. Therefore, in the approach presented here, OpenSESAME diagrams are first converted into stochastic activity networks (SAN) as accepted by the tool Möbius [DCC$^+$02]. Internally, the generated SANs are analysed by the *zero-suppressed multi-terminal decision diagram* (ZDD)-based symbolic framework as presented in [LS06a,LS06b]. In this paper we show, that

1. using OpenSESAME, it is much easier to create sophisticated availability models than by e.g. directly creating the corresponding SAN manually, and
2. the proposed tool chain, which is based on ZDDs, is much more efficient in terms of time and space compared to traditional solution methods. The advantages stem from the fact that the traditional methods require the explicit generation of the system's complete state space and the storage of its transition matrix in a sparse matrix format.

Organization: The paper is organized as follows: Sec. 2 introduces the employed tool infrastructure. The model world is introduced in Sec. 3 by giving basic definitions. The general idea of symbolically representing and numerically solving MRMs is introduced in Sec. 4. An industrial case study for evaluating our framework is presented in Sec. 5. Sec. 6 concludes the paper by indicating

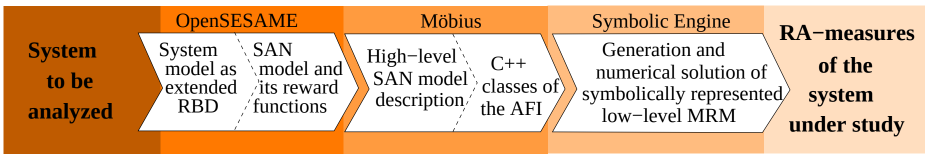
System to be analyzed
OpenSESAME
System model as extended RBD
SAN model and its reward functions
Möbius
High−level SAN model description
C++ classes of the AFI
Symbolic Engine
Generation and numerical solution of symbolically represented low−level MRM
RA−measures of the system under study

Fig. 1. Tool chain

some related work, by summarizing the achieved innovations and mentioning future steps.

2 Tool Chain

For analyzing systems we employ the tool chain as illustrated in Fig. 1. Via the process of abstraction and simplification one builds a system model, specified as an extended RBD within the tool OpenSESAME. The obtained RA-model is then mapped onto a Stochastic Activity Network (SAN). A SAN is a form of extended Generalized Stochastic Petri Net (GSPN), which also contains reward functions, employed here for describing the RA-measures of interest. The modeling tool Möbius supports the specification of high-level models of that kind. Our new symbolic engine, which possesses an interface to Möbius, can then be employed for generating a symbolic representation of the specified MRM (SAN + reward functions). The numerical solution of the symbolically represented low-level MRM allows finally the computation of the RA-measures of interest. In the following we briefly introduce the different components as employed in the suggested tool chain.

OpenSESAME [WT04,WT05]: The graphical user interface of OpenSESAME allows for the creation of traditional combinatorial availability models, which can be enriched with inter-component dependencies. In these models, a system is defined by its *components*, each specified by a *Mean Time To Failure (MTTF)* and *Mean Time To Repair (MTTR)*. In addition, *reliability block diagrams* specify the redundancy structure of the system, i.e. they determine which components have to be available at the same time to make the overall system available. Several kinds of inter-component dependencies can be specified in an OpenSESAME input model which greatly increases the modeling power without compromising its usability as shown in Sec. 3.1 of this paper.

Möbius modeling tool: Möbius is a software tool for performance and reliability evaluation of discrete event systems. Currently, Möbius supports several model specification formalisms [DCC+02], including *Stochastic Activity Networks* (SAN), an extension of GSPNs. Since OpenSESAME can generate GSPNs out of its input diagrams, Möbius can be used in our tool chain.

Within Möbius, the SANs are mapped onto the *Abstract Functional Interface* (AFI), which is implemented in C++ and constitutes the interface between the state graph generator and the (high-level) SAN model specification. Each place of the SAN is hereby mapped onto a *state variable* (SV). Consequently, during state graph exploration, a state of the system model is represented by the values of the SVs, where the ordered tuple of n SVs is commonly denoted as the *state vector*. However, rather than specifying now the RA-measures directly on the level of the state graph, one may define reward functions on the level of the SAN model specification. In the tool chain presented here, these rate rewards are created automatically by OpenSESAME.

Symbolic Engine: The new symbolic engine for analyzing Möbius models with very large state graphs is based on ZDDs, where the implementation consists of the following four modules:

1. A module for the explicit generation of states, which make uses of Möbius' AFI and thus constitutes the interface between the symbolic engine and Möbius.
2. The symbolic state graph generation engine, which generates a symbolic representation of the CTMC of the low-level MRM.
3. A ZDD-library, which is based on the CUDD-package [Som]. This library mainly contains the C++ class definition of ZDDs, the new recursive algorithms for manipulating them and their operator-caches.
4. A library for computing the desired RA-measures on the basis of the symbolically represented MRM. This module contains:
   (a) steady state and transient numerical solvers for computing the state probabilities.
   (b) algorithms for efficiently generating symbolic representations of rate reward functions and for computing the first and second moment of their probability distributions.

3 Model world

3.1 OpenSESAME input model

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 of which the system consists. 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

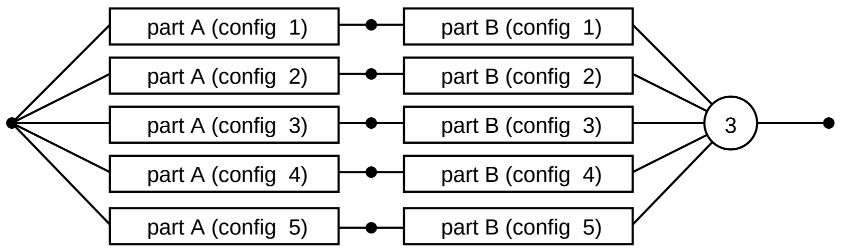

Fig. 2. *"3-out-of-5:G"* system. The system is available, if at least 3 out of 5 configurations are available. In this case, each configuration comprises two components: part A and part B.

same type, the table also lists the number of components of this type. Furthermore, 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-out-of-N:G"* edges are supported.

As an example, Figure 2 shows a *"3-out-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 arbitrarily large sub-diagram. Alternatively, the system could be modeled using regular RBDs without *"k-out-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 masking: 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.

A detailed description of all features of OpenSESAME is outside the scope of this paper. Instead, the interested reader is referred to previous publications [WS05,WT05]. For an overview on the transformation process of OpenSESAME input models into GSPNs and SANs one may refer to [WT04].

3.2 Properties of high-level MRMs

Via state graph generation a high-level model description and its set of user-defined rate rewards can be mapped to a continuous time Markov chain (CTMC), where each system state is equipped with a rate reward[3]. This yields what is commonly denoted as (low-level) Markov reward model (MRM). In the following we define some properties of high-level model descriptions, as required for our symbolic framework for efficiently generating symbolic representations of MRMs.

Static properties: A high-level model M consists of a finite ordered set of discrete state variables (SVs) $s_i \in \mathcal{S}$, where each can take values from a finite subset of the naturals. Each state of the model is thus given as a vector $\vec{s} \in \mathbb{S} \subset \mathbb{N}^{|S|}$. A model has a finite set of activities ($\mathcal{A}ct$), where the enabling and execution of an activity l depends on a set of SVs ($\mathcal{S}_l^{\mathcal{D}}$). Two activities are defined to be dependent if their sets of dependent SVs are not disjoint. We also define a projection function $\chi\colon\ (\mathcal{S}_l^{\mathcal{D}}, \mathbb{N}^{|S|}) \longrightarrow \mathbb{N}^{|\mathcal{S}_l^{\mathcal{D}}|}$ which yields the sub-vector consisting of the dependent SVs only. We use the shorthand notation $\vec{s}_{d_l} := \chi(\mathcal{S}_l^{\mathcal{D}}, \vec{s}\,)$, where $\vec{s}_{d_l}$ is called the activity-local marking of state $\vec{s}$ with respect to activity l.

Dynamic properties: When an activity is executed, the model evolves from one state to another. For each activity $l \in \mathcal{A}ct$ we have a transition function $\delta_l : \mathbb{S} \longrightarrow \mathbb{S}$, whose specific implementation depends on the model description method. Concerning the target state of a transition, we use the superscript of a state descriptor to indicate the sequence of activities leading to that state. It is assumed that the computation of δ_l depends solely on those positions of $\vec{s}$ referring to the SVs contained in $\mathcal{S}_l^{\mathcal{D}}$. By state graph exploration one can construct the successor-state relation as a set of quadruples $T \subseteq (\mathbb{S} \times \mathcal{A}ct \times \mathbb{R}^{>0} \times \mathbb{S})$, which is the set of transitions of a stochastic labeled transition system (SLTS), i.e. the

[3] The presented methodology can also take care of impulse rewards, but these are not used for the considered application case study.

underlying activity-labeled CTMC. If activity labels are removed, transitions between the same pair of states are aggregated via summation of the individual rates.

Rate rewards: Rate rewards enable the modeler to define complex performance measures on the basis of the high-level model, rather than on the level of the underlying CTMC [SM91]. A rate reward defines the reward gained by the model in a specific state. This gives us the following setting: A rate reward r defined on a high-level model is specified by the rate reward returning function $\mathcal{R}_r : \mathbb{S} \rightarrow \mathbb{R}_+$, and where $\mathcal{S}_r^{\mathcal{D}} \subseteq \mathcal{S}$ is the set of SVs on which the computation of r actually depends. Analogously to activity-local markings we will also employ the shorthand notation $\vec{s}_{d_r} := \chi(\mathcal{S}_r^{\mathcal{D}}, \vec{s}\,)$. The set of all rate rewards defined for a given high-level model, will be denoted as $\mathcal{R}$.

4 Symbolic Representation and Solution of MRMs

4.1 Symbolic representation of low-level Markov reward models

In this section, zero-suppressed multi-terminal binary DDs (ZDDs) are introduced, and it will be shown how this symbolic data structure can be employed for representing CTMCs and their reward functions.

The ZDD data structure: Different types of symbolic data structures have been employed successfully for compactly representing very large labeled Markov chains. In a Zero-suppressed BDD (ZBDD) [Min93], the skipping of a variable means that this variable takes the value 0. Thus, ZBDDs are more compact than the original BDDs [Bry86] when representing Boolean functions whose satisfaction set is small and contains many 0-assignments. A previous paper [LS06a] introduced the multi-terminal version of ZBDDs, which we call zero-suppressed multi-terminal binary decision diagrams (ZDD). Analogously to algebraic decision diagrams (ADDs) [ADD97], ZDDs permit the representation of pseudo-Boolean functions. It has been found that, for our area of application, the ZDD-based representation is more compact than the ADD-based representation by a factor of approximately two to three, which has the positive effect that the construction and manipulation times of the symbolic representations, as well as the times for the numerical solution of the represented MRM are reduced by about the same factor [LS06a,LS06b].

ZDD-based representation of SLTS: By state graph exploration one can construct the set of transitions of the stochastic labeled transition system (SLTS). Each transition within an activity-labeled SLTS T can then be encoded by applying a binary encoding scheme which represents the transition $(\vec{s} \xrightarrow{l,\lambda} \vec{s}^{\,l})$ as the bit-vector $\left(\mathcal{E}_{\mathcal{A}ct}(l), \mathcal{E}_S(\vec{s}\,), \mathcal{E}_S(\vec{s}^{\,l})\right)$. The rate λ is hereby unaccounted, since it will be stored in a terminal node of the ZDD. The individual bit positions of the obtained vectors correspond to the Boolean variables of the ZDD. Given a

ZDD-based representation of a SLTS, one simply has to abstract over the binary encoded activity labels, in order to obtain a symbolic representation of the corresponding transition rate matrix. Hereby the boolean variables holding the binary encoded row and column indices are ordered in an interleaved way. Such an ordering is a commonly accepted heuristics for obtaining small BDD sizes, and it also works well for ZDDs.

Symbolic representation of rate reward functions: Having pairs of binary encoded system states and rate rewards, one obtains a pseudo-boolean function for each reward specification. This function can once again be represented by means of a ZDD.

4.2 Generating and solving the low-level Markov reward model

The top-level algorithm for generating and solving low-level Markov reward models can be divided into three main phases: At first one derives a symbolic representation of the CTMC from the high-level model. Secondly one computes steady-state or transient state probabilities. In the third phase, the symbolically represented CTMC enables one to generate symbolic representations of the rate reward functions. Their different stochastic moments can be efficiently computed via BDD-traversal. The main idea of our approach is to limit the explicit exploration and explicit execution of reward functions only to fractions of the low-level MRM, where the missing parts are generated via ZDD manipulations. In contrast to standard methods, this strategy leads to significant runtime-benefits, where the employment of ZDDs yields significant reductions in memory space.

Phase 1: Constructing a symbolic representation of a CTMC [LS06a]: The main idea of the activity-local state graph generation scheme is the partitioning of the CTMC or the SLTS T to be generated into sets of transitions with label $l \in \mathcal{A}ct$, where each state is reduced to the activity-dependent markings:

$$T^l := \{(\vec{s}_{d_l}, l, \lambda, \vec{s}^{\,l}_{d_l}) \mid \ \vec{s}_{d_l} = \chi(\mathcal{S}^{\mathcal{D}}_l, \vec{s}) \wedge \vec{s}^{\,l}_{d_l} = \chi(\mathcal{S}^{\mathcal{D}}_l, \vec{s}^{\,l}) \wedge (\vec{s}, l, \lambda, \vec{s}^{\,l}) \in T\} \quad (1)$$

During state graph generation the activity-local transitions T^l are successively generated, where each is encoded by its own (activity-local) ZDD Z_l, which solely depends on the Boolean variables encoding the dependent SVs of activity l. The overall transition relation is then obtained by executing a symbolic composition scheme:

$$\mathsf{Z}_T := \sum_{l \in \mathcal{A}ct} \mathsf{Z}_l \cdot \mathbb{1}_l,$$

where in the above equation $\mathbb{1}_l$ represents the pairwise identity over the Boolean variables encoding activity l's set of independent SVs ($\mathcal{S}^{\mathcal{I}}_l = S \setminus \mathcal{S}^{\mathcal{D}}_l$). One may note, that due to the `Apply`-algorithm of [Bry86] and derivatives thereof, that $\mathsf{Z}_l \cdot \mathbb{1}_l$ may in general not yield the Kronecker-product of the encoded matrices. The ZDD Z_T thus constructed encodes a set of potential transitions, therefore at this point it is necessary to perform symbolic reachability analysis. On the other hand symbolic composition might also result in states triggering new model behavior. In case where such states exist, a new round of explicit state graph exploration, encoding, composition and symbolic reachability analysis follows. Several rounds may be required until a global fix point is reached and a complete representation of the user-defined CTMC is constructed.

The advantages of the activity-local scheme can be summarized as follows:

1. In general, only a small fraction of the transitions of the Markov chain needs to be generated explicitly, whereas the bulk of the transitions is generated during symbolic composition.
2. The scheme does not require any particular model structure. In particular, the method is not restricted to structures that can be represented by a Kronecker descriptor.
3. The model is partitioned automatically at the level of the individual activities, i.e. a user-defined partitioning is not necessary.
4. The composition of the individual "activity-local" portions of the Markov chain is carried out efficiently at the level of the symbolic data structure.

Phase 2: ZDD-based solution [LS06b]: Once the symbolic representation of the CTMC, i.e. its transition rate matrix, is generated, probabilities for each system state are computed. The solvers considered in this paper employ an approach in which the generator matrix is represented by a symbolic data structure and the probability vectors are stored as arrays [Par02]. If n Boolean variables are used for state encoding, there are 2^n potential states, of which only a small fraction may be reachable. Allocating entries for unreachable states in the vectors would be a waste of memory space and would severely restrict the applicability of the algorithms (as an example, storing probabilities as doubles, a vector with about 134 million entries already requires 1 GByte of RAM). Therefore a dense enumeration scheme for the reachable states has to be implemented. This is achieved via the concept of offset-labeling. In an offset-labeled ZDD, each node is equipped with an offset value. While traversing the ZDD encoding the matrix, in order to extract a matrix entry, the row and column index in the dense enumeration scheme can be determined from the offsets, basically by adding the offsets of those nodes where the `then`-Edge is taken.

The space efficiency of symbolic matrix representation comes at the cost of computational overhead, caused by the recursive traversal of the ZDD during access to the matrix entries. For that reason, Parker [Par02] introduced the idea of replacing the lower levels of the ADD by explicit sparse matrix representations, which works particularly well for block-structured matrices. In the context of our work, we call the resulting data structure *hybrid offset-labeled* ZDD. The level at which one switches from symbolic representation to sparse matrix representation, called *sparse level* s, depends on the available memory space, i.e. there is a typical time/space tradeoff.

For numerical analysis, it is well-known that the Gauss-Seidel (GS) scheme and its over-relaxed variants typically exhibit much better convergence than the Jacobi, Jacobi-Over-relaxation or Power method. However, Gauss-Seidel requires row-wise access to the matrix entries, which, unfortunately, cannot be realized efficiently with ZDD-based representations. As a compromise, Parker [Par02] developed the so-called pseudo-Gauss-Seidel (PGS) iteration scheme, where the matrix is partitioned into blocks (not necessarily of equal size). Within each block, access to matrix entries is in arbitrary order, but the blocks are accessed in ascending order. PGS requires one complete iteration vector and an additional vector whose size is determined by the maximal block size. Given a ZDD which represents the matrix, each inner node at a specific level corresponds to a block. Pointers to these nodes can be stored in a sparse matrix, which means that effectively the top levels of the ZDD have been replaced by a sparse matrix of block pointers. The ZDD level at which the root nodes of the matrix blocks reside is called *block level* b. Overall, this yields a memory structure in which some levels from the top and some levels from the bottom of the ZDD have been replaced by sparse matrix structures. We call such a memory structure a block-structured hybrid offset-labeled ZDD. The choice of an adequate s and an adequate b is an optimization problem. In general, increasing b improves convergence of the PGS scheme (but also increases the time per iteration), and replacing more ZDD levels by sparse structures improves speed of access.

Phase 3: Generating symbolic representations of rate reward functions: After the system state probabilities are computed, one needs to generate the symbolic representations of the rate reward functions. Hereby the main idea is once again to exploit locality, so that the explicit evaluation of each reward function is limited to a fraction of states of the CTMC, rather than evaluating the reward functions for each state. I.e. similar to activity-local transition systems one restricts oneself to processing rate-reward-local states. The symbolic representation R_r of the characteristic (pseudo-boolean) function of the set:

$$\mathbb{S}_r := \{\vec{s}_{d_r} \in \mathbb{S} \mid \mathcal{R}_r(\vec{s}_{d_r}) \neq 0\}$$

gives hereby a rate-reward-local reward function, such that $\mathsf{R}_r \cdot \mathbb{S}$ yields the rate reward for each system state concerning rate reward definition r. Once state probabilities and also symbolic representations of all rate reward functions have been constructed, their moments can be computed via BDD-traversal. Due to the nature of the traversal, one only visits hereby those states individually whose reward value is not zero. The obtained stochastic moments are the desired RA measure, e.g. unavailability.

5 Case Study: Fault-Tolerant Adjunct Processor

In the digital telephone network, so-called adjunct processors translate easy-to-remember, location-independent virtual phone numbers (used e.g. by emer-

parameter	default value	description
N	6	number of configurations
K	4	number of initially active configurations
MTTF-SBC$_i$	$5 \cdot 10^4$ hours	mean time to failure of SBC i
MTTF-RTB$_i$	$1 \cdot 10^5$ hours	mean time to failure of RTB i
MTTR-SBC$_i$	1 hour	mean time to repair of SBC i
MTTR-RTB$_i$	1 hour	mean time to repair of RTB i
FOT$_{ij}$	0.1 hours	fail-over-time from configuration i to configuration j

Table 1. Default parameters of the I/O-unit submodel investigated in this paper.

gency departments) into their location-dependent physical equivalent.[4] Because adjunct processors play a crucial role in the network, they must be highly available. Typically, an availability of 99.999% is demanded for such a system which corresponds to a mean downtime of less than 5 minutes per year. In a previous work, we investigated the availability of an adjunct processor implementation [GLW00] by a SPN-based model. We will now apply the proposed method to this model to point out its benefits.

5.1 System Description

From a top-level view, the adjunct processor is a series system comprising host units, I/O-units, hot-swap controllers, power supplies, a RAID system and so on. Due to place restrictions, we will evaluate the I/O-subsystem only, as it is the most complex part of the system. The other parts can be evaluated in a similar way which is not shown here.

The I/O-unit consists of N configurations, each comprising a single board computer (SBC) and a so-called rear transition board (RTB). All cabling is connected to the RTB which allows for a quick replacement of the SBC in case of a failure. We assume, that $K <= N$ configurations have to be available at the same time to make the I/O-unit available. Furthermore, we assume that each configuration can be in three states: active, failed, or passive. A passive (or stand-by) configuration does not perform any work but waits until an active configuration fails. After failure detection and localization, the I/O-unit is reconfigured which means that one of the passive configurations becomes activated. The overall time interval which lies between a failure and the completion of the reconfiguration is called the fail-over-time.

A configuration fails, if either its SBC or RTB fails. This can happen to both the active and the passive configurations. Modern architectures in the telecommunication systems are open systems and may contain boards from different vendors. Thus, in general, all components of the system will have different failure and repair rates and also the fail-over times may vary. The parameters of our model are given in Table 1. For the sake of simplicity, we assume exponentially

[4] For example, if one calls 112 in Germany, one will be connected to the closest fire department.

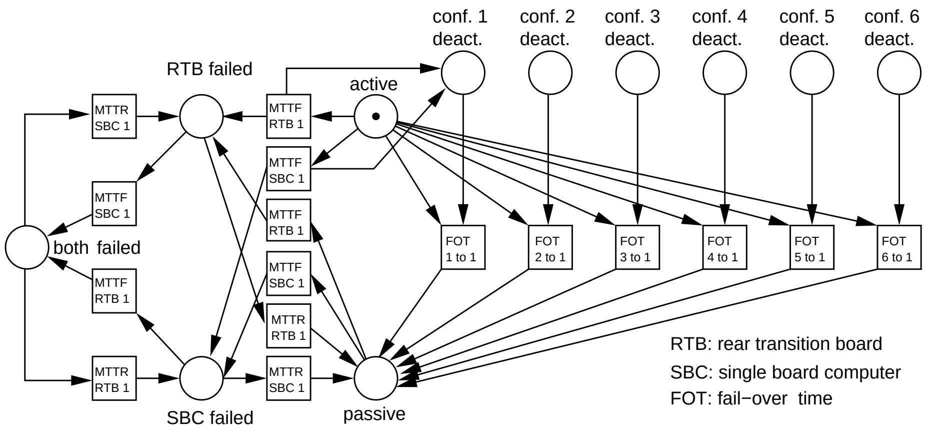

Fig. 3. Single component specified as SPN

distributed time intervals in the following evaluations. This is acceptable for the mean time to failures, because the effects of aging can be neglected in electronic devices. Analytic evaluation of models with non-exponentially distributed time-intervals is an active area of research.
Using OpenSESAME, the I/O-unit can be modeled using the RBD shown in Fig. 2 for the case $N = 5$. This model is enriched with information on the respective fail-over-times which are necessary to switch from one configuration to another. We think that the OpenSESAME model is quite intuitive and easy to modify. In contrast, a SPN-based model as sketched in Fig. 3 is much more complex. The figure shows only one sixth of the overall SPN structure for the case $N = 6$. It represents configuration 1 which is one of the configurations which are active after system startup. The remaining five configurations are modeled with equivalent subnets, however, all subnets share the six places `conf. 1-6 deact`. As it can be seen, even a small fraction of the overall net is much less readable than the RBD from Fig. 2. Moreover, changing the parameters N or K requires a work-intensive and error-prone modification of the SPN structure. This exemplifies the benefit of the proposed method, where OpenSESAME is used to generate the SPN from an OpenSESAME input model which comprises the RBD from Fig. 2, attributed with the respective fail-over-times.

5.2 Model Evaluation

Table 2.A shows the evaluation results in terms of the I/O unit's unavailability for the default values given in Tab. 1 and different fail-over-times. As can be seen, the fail-over time has a significant impact on the result. Please note that if traditional combinatorial methods like a simple RBD or fault tree analysis were used, this would imply the assumption that the fail-over-time is zero. Thus, the result would be highly over-optimistic as it is several orders of magnitude lower than the correct results even for small fail-over-times.

(A) Model specific RA-measures (system unavailability)

	mean fail-over time				
	0 sec	10 sec	1 min	5 min	6 min
"4-out-of-6"	$< 5.99 \cdot 10^{-13}$	$1.67 \cdot 10^{-7}$	$9.97 \cdot 10^{-7}$	$4.93 \cdot 10^{-6}$	$5.91 \cdot 10^{-6}$
"6-out-of-8"	$< 1.60 \cdot 10^{-12}$	$2.49 \cdot 10^{-7}$	$1.49 \cdot 10^{-6}$	$7.40 \cdot 10^{-6}$	$8.86 \cdot 10^{-6}$

(B) Model specific data

	states	*trans*	$trans_e$
"4-out-of-6"	$9.48720 \cdot 10^5$	$1.45607 \cdot 10^7$	240
"6-out-of-8"	$2.61671 \cdot 10^8$	$5.86973 \cdot 10^9$	544

(C) Solution times

	DSPNexpress				Symbolic Approach			
	peak mem.	SG time	iter. time	# iter.	peak mem.	SG time	iter. time	# iter.
"4-out-of-6"	7.4 GByte	123.014	0.7315	12	36 MByte	2.50562	0.093181	46
"6-out-of-8"	xxx	xxx	xxx	xxx	4105 MByte	15.8026	40.533267	49

Table 2. Data as obtained for analyzing the case study

For evaluating the efficiency of our symbolic framework, we analyzed the adjunct processor for two different parameter sets. In the first set, we investigated a *"4-out-of-6"* system (i.e. $N = 6$ and $K = 4$) whereas in the second set a *"6-out-of-8"* system was analyzed ($N = 8$ and $K = 6$). The numerical values were equal to the ones presented in Tab. 1. We stress that the sub-units were not assumed to be symmetric, which resulted in Markov reward models of substantial size to be analyzed. If sub-units were symmetric, lumping techniques for state space reduction could be exploited.

Table 2.B shows the size of the analyzed low-level MRM models as derived from the OpenSESAME input model, via the translation to a SAN and finally applying the symbolic state graph generation scheme. Consequently, Table 2.B contains the number of system states (*states*), the number of transitions among these system states (*trans*) and the number of transitions explicitly established by our activity-local ZDD-based state graph generation scheme ($trans_e$). The latter is extremely small, which is the main source of efficiency of the approach.

Table 2.C shows the memory and CPU-times as required for generating and solving the MRM. Hereby the two different configurations were analyzed on a 64-bit Opteron system with 8 GByte of RAM and a Linux OS. For demonstrating the effectiveness of our ZDD-based framework, we also exported and analyzed the system models with the GSPN-based tool *DSPNexpress* [Lin98]. Table 2.C gives the *peak memory* consumption, the CPU time in secs. required for generating the CTMC (*SG time*), the CPU times in secs. required for each numerical iteration for computing the state probabilities (*iter. time*) and their number (# iter.) as executed under the respective numerical solution method. As numerical solution method, we decided to employ the pseudo-Gauss-Seidel method of [Par02] in case of ZDDs, whereas *DSPNexpress* employs the generalized minimal residual method (GMRES). As convergence criteria a relative convergence of 10^{-6} was taken. The data of Table 2.C indicates that the ZDD-based framework is much more efficient than the standard sparse matrix approach employed within the *DSPNexpress*-tool. In case of the *"6 out 8"*-configuration, *DSPNexpress* was unable to analyze the system model, due to a lack of RAM. Even for the smaller configuration (*"4-out-of-6"*), our ZDD-based framework is more efficient. Hereby, and in contrast to sparse matrix techniques, the bottleneck is the state-probability vector, since even for very large systems the block-structured hybrid ZDD-based representation of the transition rate matrix of the Markov reward model is still very compact. E.g. for the *"4-out-of-6"*-configuration the ZDD-based representation of the transition rate matrix requires 0.226 MByte only, whereas the probability vector, the iteration vector and the vector holding the diagonal entries of the generator matrix require 14.5 MByte of RAM. Consequently the ZDD-based methodology clearly eases the restriction on Markov reward analysis. Thus it is not surprising that the suggested ZDD-based tool chain is still capable of computing the desired RA-measures in approx. 34 minutes, where standard methods as employed within the *DSPNexpress*-tool fail to do so.

6 Conclusion

Related Work: Several techniques have been proposed to simplify the creation of state-based dependability models. One possibility is to combine several modeling methods in one user-interface (see, e.g. [STP96,THMH98]). Another possibility is to extend Boolean methods (e.g. [DSC00]). The approach favored in our work, i.e. automatically creating the models from a high-level input can also be found (see [MPB03,BB03]). Please refer to our previous work [WT04] for a detailed comparison with OpenSESAME. However, none of these methods uses symbolic data structures for the representation of the state space.

Based on the original Binary Decision Diagram (BDD) data structure [Bry86], several extensions have been developed for representing not only Boolean but also pseudo-Boolean functions, i.e. functions of the type $f : \mathbb{B}^n \mapsto \mathbb{R}$, where Algebraic Decision Diagrams (ADDs) [ADD97] are one of the best known types. In recent years, powerful state space generation algorithms based on symbolic data structures have been developed and implemented in software tools such as PRISM [KNP05] and SMART [CJMS03]. While PRISM is based on ADDs, SMART employs multi-valued decision diagrams and matrix diagrams. Using these techniques, generating the state space and transition structure of the underlying Markov model from the high-level specification is extremely fast, and the resulting symbolic representation can be very memory efficient. Numerical analysis based on the symbolic representation is still an active area of research. Using the approach of offset-labeling, combined with hybrid matrix representation, as first proposed for ADDs in [Par02], it has been demonstrated that iterative solution techniques based on symbolic data structures can be almost as fast as sparse matrix approaches, while being much more memory efficient and therefore able to solve much larger systems.

Summary: In this paper we presented a tool chain which takes a set of high-level diagrams as its input. These diagrams, which comprise component tables, reliability block diagrams, failure dependency diagrams, repair group tables, and variable tables, yield an input model for the tool OpenSESAME. The tool automatically converts this high-level model specification into a SAN as accepted by the tool Möbius. In a second step, the obtained SAN is converted by our ZDD-based symbolic engine into a symbolic representation of a MRM. On the basis of this symbolic representation numerical analysis is carried out, finally yielding the desired reliability- and availability measures of the system under study.
For illustrating the advantages of such an approach, we presented a model of a fault-tolerant telecommunication service system with N redundant modules, where the system is available as long as at least K modules are available. Each module comprises two components and can be either in failed, stand-by or active mode. Reconfiguring the system after a failure takes some time during which the system is not available. All failure-, repair- and reconfiguration-rates can be different. Considering a *"4 out of 6"* and *"6 out of 8"* configuration, where in case of the latter the obtained MRM already consist of more than 2.61×10^8 system states, we illustrate, that our tool chain is capable of computing the relevant measures of interest without problems, where standard techniques, such as included in the tool *DSPNexpress*, are less efficient or even fail.

Future work: Since we develop our implementations in the context of Möbius, we are currently implementing an efficient symbolic realization of the "Replicate" feature [SM91], so that modeled symmetries lead to much smaller Markov reward models to be solved. Furthermore, an adaptation of aggregation methods for the approximate solution of CTMCs to the case of ZDD-represented MRMs seems to be a promising starting point for future research.

References

[ADD97] *Formal Methods in System Design: Special Issue on Multi-terminal Binary Decision Diagrams*, Volume 10, No. 2-3, April - May 1997.

[BB03] M. Bouissou and J. L. Bon. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. *Reliability Engineering and System Safety*, pages 149–163, November 2003.

[Bry86] R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE ToC*, C-35(8):677–691, August 1986.

[CJMS03] G. Ciardo, R.L. Jones, A.S. Miner, and R. Siminiceanu. Logical and stochastic modeling with SMART. In *Proceedings of Tools 2003*, pages 78 – 97. Springer, LNCS 2794, 2003.

[DCC+02] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W.H. Sanders, and P. Webster. The Moebius Framework and Its Implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.

[DSC00] J.B. Dugan, K.J. Sullivan, and D. Coppit. Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transaction on Reliability*, 49(1):49–59, March 2000.

[GLW00] G. Graf, M. Leberecht, and M. Walter. High Availability Commodity Computing - A CompactPCI-System Evaluation. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 4. CSREA Press, 2000.

[KNP05] M. Kwiatkowska, G. Norman, and D. Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, 2005.

[Lin98] C. Lindemann. *Performance Modelling with Deterministic and Stochastic Petri Nets*. Wiley and Sons, 1998.

[LS06a] K. Lampka and M. Siegle. Activity-Local Symbolic State Graph Generation for High-Level Stochastic Models. In *Proc. of MMB Conference 2006*, pages 245–264, 2006.

[LS06b] K. Lampka and M. Siegle. Analysis of Markov Reward Models using Zero-suppressed Multi-terminal Binary Decision Diagrams, 2006. To appear in *Int. Conf. Valuetools 2006*.

[Min93] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proc. of DAC*, pages 272–277, Dallas (Texas), USA, June 1993. ACM Press.

[MPB03] I. Majzik, A. Pataricza, and A. Bondavalli. Stochastic Dependability Analysis of System Architecture Based on UML Models. *Lecture Notes in Computer Science*, 2677:219–244, 2003.

[Par02] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[SM91] W. H. Sanders and J. F. Meyer. A unified Approach for specifying Measures of Performance, Dependability, and Performability. In *Dependable Computing for Critical Applications*, Vol. 4, pages 215–237. Springer-Verlag, 1991.

[Som] F. Somenzi. CUDD Package, Release 2.4.x. http://vlsi.colorado.edu/~fabio.

[STP96] R.A. Sahner, K.S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems*. Kluwer Academic Publishers, 1996.

[THMH98] D. Tang, M. Hecht, J. Miller, and J. Handal. MEADEP: A dependability evaluation tool for engineers. *IEEE Transaction on Reliability*, 47(4):443–450, 12 1998.

[WS05] M. Walter and W. Schneeweiss. *The modeling world of Reliability/Safety Engineering*. LiLoLe Verlag, 2005.

[WT04] M. Walter and C. Trinitis. How to Integrate Inter-Component Dependencies into Combinatorial Availability Models. In *Proc. Ann. Reliability and Maintainability Symp. (RAMS 2004), Los Angeles, USA*, pages 226–231. IEEE, 2004.

[WT05] M. Walter and C. Trinitis. 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, 2005.