Dependability modelling with the stochastic process algebra tool CASPA

Johann Schuster
Universität der Bundeswehr München
johann.schuster@unibw.de

Markus Siegle
Universität der Bundeswehr München
markus.siegle@unibw.de

ABSTRACT

This note describes CASPA, a stochastic process algebra tool for performance and dependability modelling, analysis and verification. Internally, the tool works with the symbolic data structure MTBDD (multi-terminal binary decision diagram), which allows for handling models with very large state space. The paper illustrates how CASPA can be used for dependability analysis, by modelling and analysing a phased mission system from the literature.

Keywords

dependability modelling, stochastic process algebra, MTBDD

1. INTRODUCTION

The tool CASPA, which employs a stochastic process algebra as its input language, can be used for performance and dependability modelling, analysis and verification. The development of CASPA started in 2003, and since then it has been developed to a highly efficient tool for the generation and analysis of Markov chains with very large state space [3, 1]. In addition to actions with exponential delay (Markovian actions), CASPA offers immediate actions which, together with an efficient elimination algorithm [1], allow for convenient model specification and Markov chain generation.

The paper is organised as follows: Sec. 2 recapitulates a case study and Sec. 3 sketches its CASPA model. In Sec. 4 the experimental results are presented and Sec. 5 concludes the paper.

2. PMS SYSTEM

As a case study we use the Phased Mission System (PMS) from [2], sketched in Fig. 1. It consists of two non-repairable components A and B and five switches *K1* to *K5*. The aim is to keep the connection between S and T over the lifetime of the system. The lifetime consists of two phases: In phase 1 two parallel branches, namely *BRANCH A* (A-K2-K4) and *BRANCH B* (K1-K3-B) are used. On failure of A or B both switches of its branch have to be opened. After an exponentially distributed time the phase is changed to phase 2. The phase change has to be done by first opening *K1* and *K4* and then closing *K5* thus leading to a serial connection (A-K2-K5-K3-B). After an exponentially distributed time phase 2 is left and the mission is accomplished. The following errors with exponential failure rate $\lambda_{\mathtt{err}}$ have to be taken into account: Failure of A, B, and of currently closed switches. Furthermore, on-demand failures (probability $p_{\mathtt{err}}$) can occur when trying to close or open the switches.

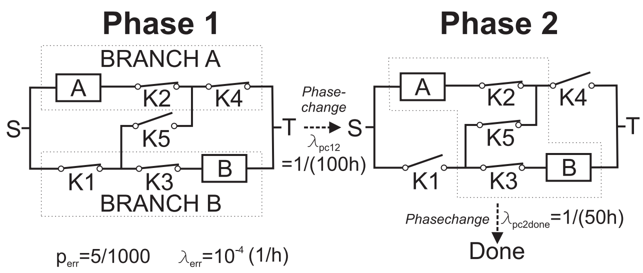

Figure 1: PMS scheme and mission

```
System :=
  ( (K1(CLOSED) |[broken]| K2(CLOSED) |[broken]|
     K3(CLOSED) |[broken]| K4(CLOSED) |[broken]|
     K5(OPEN))
     |[broken]|
     (A(OK) |[broken]| B(OK)) )
 |[SYNCSET]|
PMS(PARALLEL, OK, OK)
```

Figure 2: System process

3. COMPOSITIONAL MODEL

The PMS system can be modelled with CASPA in a natural compositional way: One sequential process (i.e. submodel) for each switch and each non-repairable component. A monitoring process *PMS* keeps track of the errors and the current phase of the system. Fig. 2 shows the overall model, which is the synchronised parallel composition of the submodels. The synchronising action `broken` is used to notify all submodels when the system has failed, whereas SYNCSET is an abbreviation for all actions allowing for a bi-directional communication between the PMS process and the other components (e.g. `failK2`, `open2`). The submodels will be defined in the following sections. Note that the initial states of the switches are according to phase 1 in Fig. 1.

```
(1) K2(state[1]) :=
(2) [state=CLOSED]  -> (failureK2, SWITCHFAILRATE);
                        (*failK2,1*); K2(OPEN)
(3) [state=CLOSED]  -> (*open2,1*); (
                        (*fail_k2_open, FAILUREPROB*);
                          (*openfailed,1*); K2(CLOSED)
                       +(*k2_open, SUCCESSPROB*);
                          (*open, 1*);  K2(OPEN)  )
(4) [state!=CLOSED] -> (*open2,1*);(*open,1*);K2(state)
(5) [state=CLOSED]  -> (*use2,1*); K2(CLOSED)
(6) [state!=CLOSED] -> (*use2,1*);
                        (*unavailable,1*); K2(state)
(7) [*]             -> (*broken, 1*); stop
```

Figure 3: Model of K2

3.1 Switch process

In Fig. 3 we show the CASPA model of switch *K2*. The switches are in some sense *memoryless*, that means if an inadvertent open occurs, one can simply close it again and it will work as before. So a switch only has two states, *OPEN* and *CLOSED*. Line (1) of the model description means that the parameterised process *K2* has a parameter range of $\{0, 1\}$. In line (2), the inadvertent opening of a switch is reflected: The action *failureK2* is Markovian with rate *SWITCHFAILRATE*. It is followed by an immediate action *failK2* (which indicates to the PMS process that the switch has failed) and ends up at process *K2* with *state=OPEN*. The on-demand failure is covered in line (3), namely when by the immediate action *open2* the switch is requested to open, a choice between stuck-at-closed (*fail_k2_open*) and successful open (*k2_open*) must be made. Again, the process is continued in an open or closed state respectively. Line (4) covers the case where an open request occurs and the switch is already open. Lines (5) and (6) are used by the *PMS* process to check if all components for phase 2 still work. If at least one component answers with *unavailable*, the mission has failed. Finally, to keep the state space small, we use line (7) to avoid further activity after the system is broken (the broken event is broadcast by the *PMS* process).

3.2 The PMS process

The monitoring process of the PMS system is sketched in Fig. 4. As seen in line (1) the process has three parameters: The current phase, the state of *BRANCH A* and the state of *BRANCH B*. Line (2) shows an example of a switch failure when both branches are working in parallel. Again, the other switches have to be covered, as well. More interesting, line (3) shows the case when *K1* fails but *BRANCH A* does not work any more. Then the system breaks and *PMS* changes to the *FAIL* state. Line (4) shows the detection of a failure of component *A* during phase 1 and does the reconfiguration: *K2* and *K4* are requested to open (actions *open2* and *open4*). If both switches get stuck at closed (PMS process receives *openfailed* twice), *PMS* stops all switches and components by the immediate *broken* action and sets *phase* and both branches to *FAIL*. If at least one switch can be opened, the system continues in the *PARALLEL* phase but with *brancha=FAIL*. A similar line has to be given for the case of a failure of component *B*. Of course, some other cases, the phase change and the entire serial phase are missing, but we think this should be enough to get the idea.

4. EXPERIMENTAL RESULTS

For determining the probability that the mission will be accomplished successfully, we used both CASPA's path-based solver and its uniformisation solver on an Intel Xeon 3.06 GHz machine with 2 GB of main memory running SUSE Linux version 9.1. CASPA directly generates an MTBDD representation of the model specification. The model has 117 reachable states, and path-based analysis leads in 1.1 seconds (including model generation and reachability analysis, $\simeq 0.08$ seconds per path) to three fulfilling paths (two of them given in Tab. 1). The third path is the dual to the second one with K1 and K4 interchanged. The probability for a successful completion of the mission is $9.24006 \cdot 10^{-1}$. The same probability can be calculated in CASPA specifying the measure `statemeasure survival PMS(phase=DONE)` and using the uniformisation algorithm. For this approach CASPA reduces the model to only 19 tangible states. The result for a sufficiently large T (e.g. T>41000 hours) is $9.24006 \cdot 10^{-1}$ and the calculation takes 0.91 seconds (again, including model generation, elimination and reachability analysis).

```
(1)PMS(phase[PHASES], brancha[1], branchb[1]) :=
(2) [phase=PARALLEL, brancha=OK, branchb=OK] -> (*failK2,1*);
                              PMS(PARALLEL,FAIL,branchb)
(3) [phase=PARALLEL, brancha!=OK, branchb=OK] -> (*failK1,1*);
                            (*broken,1*); PMS(FAIL,FAIL,FAIL)
(4) [phase=PARALLEL] -> (*failA,1*);(*open2,1*); (
      (*openfailed,1*);(*open4,1*);(
            (*openfailed,1*);(*broken,1*); pms(FAIL,FAIL,FAIL)
          + (*open,1*); pms(PARALLEL,FAIL,branchb))
     +(*open,1*);(*open4,1*);(
            (*openfailed,1*);pms(PARALLEL,FAIL,branchb)
           +(*open,1*); pms(PARALLEL,FAIL,branchb)))
```

Figure 4: Administration process (sketch)

Path	Probability
phasechange12->k1_open->k4_open->k5_close->phasechange2done	$9.066497 \cdot 10^{-1}$
failureK1->phasechange12->k4_open->k5_close->phasechange2done	$8.678150 \cdot 10^{-3}$

Table 1: Successful paths

5. CONCLUSION

In this note we have shown how to model a phased mission system with the performance and dependability analysis tool CASPA. We have used both path-based analysis and uniformisation for the calculations.

Acknowledgements: We would like to thank Deutsche Forschungsgemeinschaft (DFG) who supported this work under grants SI 710/2 and SI 710/3.

6. REFERENCES

[1] J. Bachmann, M. Riedl, J. Schuster, and M. Siegle. An Efficient Symbolic Elimination Algorithm for the Stochastic Process Algebra tool CASPA. In *SOFSEM 2009*, pages 485–496. Springer, LNCS 5404, 2009.

[2] M. Bouissou, Y. Dutuit, and S. Maillard. Reliability analysis of a dynamic phased mission system: Comparison of Two Approaches. *Modern Statistical and Mathematical Methods in Reliability*, pages 87–104, 2005.

[3] M. Kuntz, M. Siegle, and E. Werner. Symbolic Performance and Dependability Evaluation with the Tool CASPA. In *Europ. Perf. Engineering Workshop*, pages 293–307. LNCS 3236, 2004.