Combining Decomposition and Lumping to Evaluate Semi-hierarchical Systems

Nils Müllner, Oliver Theel and Martin Fränzle
Carl von Ossietzky University of Oldenburg, Germany, and
OFFIS Institute for Computer Science
Email: nils.muellner|theel|fraenzle@informatik.uni-oldenburg.de

Abstract—**Determining performance and fault tolerance properties of distributed systems is a challenging task. One common approach to quantify such properties is to construct the state space and a transition model of the distributed system that is to be evaluated. The challenge lies in the state space being exponentially large in the size of the system. One popular approach to tackle this challenge is to combine decomposition and lumping. The system is decomposed, transition models of the subsystems are constructed and minimized by lumping bisimilar states under an equivalence relation, and the intermediate marginal transition systems are composed to construct the minimal aggregate transition model. The approach allows to circumvent the necessity to construct a full transition model while preserving the ability to compute precise measures. The decomposition yet hinges on the structure of the communication within the system. When processes do not influence each other, decomposition is trivial as it is arbitrary. On the contrary, when all processes are influenced by all other processes — known as heterarchical structure — systems cannot be decomposed at all. Between systems of independent and heterarchical processes are i) hierarchically structured systems and ii) systems that are globally hierarchical, but contain locally heterarchical subsystems. The hierarchical type has been addressed elsewhere. This paper targets the second type — referred to as *semi-hierarchically structured* —, thus expanding the frontier from decomposing purely hierarchically structured systems to decomposing semi-hierarchically structured systems. Furthermore, this paper points out the role of different types of execution semantics regarding the decomposition.**

I. Introduction

The scope of this paper is the evaluation of system properties like fault tolerance and performance properties. We consider deterministic systems that are exposed to and influenced by probabilistic environments to determine, *how well* they operate within that environment. More specifically, we focus on systems that can recuperate from the effects of sporadic faults. The measure of interest in this concern is the convergence rate, that is, the probability flow from illegal towards legal states in the state space. The problem is that the number of states in the state space is exponential in the number of processes of the underlying system. One method to cope with this problem is to slice the system, construct the transition models of the subsystems, and minimize the subsystems by lumping bisimilar states under an equivalence relation. Eventually, the lumped transition models of the subsystems are recomposed. Similar earlier approaches targeted *unstructured* systems, in which processes do not influence each other, and later, hierarchically structured systems, in which the influence of one superior subsystem into another inferior subsystem had to be accounted for in the transition models. While unstructured systems commonly operate under maximal parallel execution semantics — meaning that all processes that can execute actually do execute —, hierarchical structured systems have been discussed for serial execution semantics. With such semantics, exactly one process is selected to execute a computation step per time step. This paper proposes advances in two directions. First, it includes heterarchical subsystems to be regarded during the decomposition. Second, it provides a discussion regarding semi-parallel execution semantics and how it affects the decomposition.

a) Related work: This paper continues our previous approach [1, 2] by i) extending the method to further account for semi-hierarchically structured systems and by ii) discussing different types of execution semantics despite serial and maximal parallel execution semantics. A small example points out the issues arising with these extensions and how they can be tackled.

Markov chains, as introduced by Kemeny and Snell in 1969 [3],[1] can be lumped to reduce the state space. Lumping is a well-known technique of coalescing states under an equivalence relation, based on the definition of probabilistic bisimulation by Larsen and Skou from 1989 [4], presented by Buchholz in 1994 [5]. It allows to speed up model checking as discussed by Katoen et al. in 2007 [6] and is implemented in popular tools like the Caesar/Aldebaraan Development Package (CADP) [7] to carry out formal verification and performance analysis with the non-stochastic process algebra LOTOS [8].

Classifying distributed systems into independent, hierarchical, semi-hierarchical and heterarchical ones regarding their decomposability is proposed in our previous work [1]. Decomposition of *independent* systems — with the intention to apply lumping on the transition models of the subsystems — is discussed under various angles by Boudali et al. [9, 10, 11, 12, 13, 14] for independent processes. Hermanns and Katoen further contribute a case study regarding a *plain old telephone system* [15] in that context. Contrary to systems comprising *independent* processes, hierarchically structured systems are intrinsically more challenging to decompose as the influence that one subsystem has on another subsystem must be accounted for.

The decomposition of semi-hierarchically structured systems proposed in this paper exploits lumping and system decomposition as discussed for systems of independent and

[1] We refer to revised version from 1976.

hierarchically structured processes before. The main contribution is to expand this concept to cover for semi-hierarchically structured systems, too. A major challenge that is attacked in this paper and has not surfaced for systems of independent and hierarchical processes concerns execution semantics.

b) Structure: The paper is organized as follows. Section II presents the system model and its conversion to a transition model. Section III explains how the aggregate transition model is constructed and applied to quantify the risk of blackouts. Section IV provides a brief example to demonstrate the practical value of the approach. Section V concludes the paper.

II. Model

As setup we consider a system providing deterministic execution dynamics and probabilistic environmental influence. The deterministic part is specified first. Consider a distributed system $\mathfrak{S} = \{\Pi, E, \mathfrak{A}\}$ comprising a set of processes $\Pi = \{\pi_1, \ldots, \pi_n\}$ with each process π_i having a register R_i of volatile memory storing data that is prone to sporadic faults[2]. The set E defines unilateral neighboring relations between the processes. It contains tuples of processes $E = \{\langle \pi_1, \pi_i \rangle, \ldots\}$ such that the second process of a tuple has read access from the first process. The algorithm $\mathfrak{A}$ specifies what the processes execute. It is considered to be stored in non-volatile memory, thus being immune to sporadic faults. To focus on the decomposition-and-lumping method, we consider a deterministic algorithm. A deterministic algorithm can be formulated as set of guarded commands for which — for each state — exactly one guard evaluates to true.

Each register has a domain: a set of values it can possibly store. The state space $\mathcal{S}$ contains all combinations of possible values stored in the registers. A state $s \in \mathcal{S}$ contains the values allocated to the registers of all the processes. Consider each process to have exactly one register. Then, the *length* of each state coincides with the number of processes n. The cardinality of the state space, which is the number of possible states, is labeled $|\mathcal{S}|$. For instance, a system with two processes π_1 and π_2 each containing one register with cardinality $|R_1| = |R_2| = 3$ has a state space of cardinality $|\mathcal{S}| = |R_1| \cdot |R_2| = 9$.

We now specify the entities of the probabilistic environment. A probabilistic scheduler selects processes to execute an atomic execution step, which is one atomic guarded command of the algorithm. A scheduler is probabilistic in the context of this approach if at each time step each process has a non-zero probability to be selected. Hence, a process can possibly be continuously barred from execution. The execution semantics determine the parallelism of the execution. Under serial execution semantics, at each time step one process is selected to execute one computation step, that is, one guarded command of its algorithm. Under maximal parallel execution semantics, all processes for which at least one guard evaluates to true, execute. With deterministic algorithms, every process executes at every time step under maximal parallel execution semantics. When execution steps are synchronized among the processes under maximal parallel execution semantics, the scheduler is redundant. Semi-parallel execution semantics admit more than one, but not all n processes to execute an atomic step in parallel. The second part of the probabilistic environment subsumes transient sporadic faults. In our fault model, the perturbed process stores an arbitrary value from the domain of the affected register in that register. We consider that only the executing process is exposed to transient faults.

A state-based *safety predicate* $\mathcal{P}$ is a Boolean expression partitioning the state space into legal states $\mathcal{S}_{legal}$ and illegal states $\mathcal{S}_{illegal}$. As discussed in the previous section, the classification of systems into

- unstructured (no dependencies among the processes),
- hierarchically structured (*one-way* dependencies among the processes, no circular dependencies),
- semi-hierarchically structured (globally *one-way* dependencies among the sub-systems with local mutual dependencies) and
- heterarchically structured (fully meshed dependencies)

is important. When processes (or subsystems) do not influence each other mutually and are running in parallel, they can be evaluated individually. Since each process executes at each time step and is probably perturbed by a fault, the system can reach every state from every other state. This means, that at most n registers can change per time step, considering every process has exactly one register as discussed above. The number of registers that can change in parallel per time step is referred to as the Hamming distance, which is n in this case. This holds analogously true for all systems executing under maximal parallel execution semantics. On the contrary, consider a system under serial execution semantics. When only one process executes per time step and all other processes are immune to faults and every process contains one register, then the Hamming distance is 1.

Hazards occur when processes executing in parallel read each others' registers *in the wrong order*. To exclude hazards from our approach, we consider all executing processes at each time step to first read each others' registers and to write to their own register after all processes finished reading. This can for instance be accomplished with a mutual exclusion function that allows processes to enter the storing phase only when all other processes acknowledged they have left the reading phase. For simplicity, we do not consider this as given.

III. Method

This section informally describes the general approach. A semi-hierarchically structured system is decomposed such that

- each subsystem is tractable,
- subsystems overlap in as few processes as possible,
- processes with the same distance to the root process are within the same subsystems if possible, and
- heterarchical processes are within the same subsystems.

This implies that the approach does not work for systems containing intractably large heterarchical subsystems. Let $j, 1 <$

[2]We consider the algorithm to be stored safely in non-volatile memory. Faults occurring during the transmission and communication phase can be accounted to the probability of a process register storing a faulty value.

$j < n$ be the number of processes that the scheduler selects to execute a computation step at each time step in parallel. Let k be the minimum of the number of processes within a subsystem $\mathfrak{S}_i$ and j: $k = \min(|\mathfrak{S}_i|, j)$. For each subsystem and each $l, 0 \leq l \leq k$ one transition model has to be constructed, for the case that no process within a subsystem is selected to execute, up to the case in which all processes that can maximally execute within a subsystem are selected to execute. Subsequently, all transition models are lumped and then composed as follows: Considering the execution semantics, each case of process selection is combined via the transition models applying the Kronecker product for parallel composition, thus, constructing each possible case of scheduling selection as transition model. Finally, all case transition models are composed, again applying the Kronecker product. The example in the following section figuratively demonstrates and explains the approach.

IV. Example

The example introduced in this section comprises mutually independent components that are semi-hierarchically structured locally. We focus on evaluating just *one* component, as the counting abstraction for independent components is well-known as discussed in the introduction.

A. The setup

This section describes how a real world system is mapped onto the theoretical system description. A greater plain, like a desert or a vineyard [16], is covered evenly with small local networks such as depicted in Figure 1. Each sensor mote — modeled as one of five processes π_1 to π_5 — in such a local wireless sensor network (WSN) measures either humidity or temperature, exclusively one of them at a time. A radio station broadcasts the type of value that shall be stored at that time. To minimize costs, only one root sensor mote is equipped with a radio receiver. This distinguished root sensor mote π_1 propagates the type to be recorded to the remaining motes.

Each process contains both sensors (for temperature and humidity) and enough memory to store measured data for the duration of the desired mission time. The sensors can only measure one type of data per time step. The root process reads the broadcasted value and propagates it to all neighboring processes. The switch between measuring temperature or humidity is arbitrary and modeled probabilistically. Consider for instance an observer that wants to evaluate temperature and humidity of a region. That observer can switch the type of data to be recorded. With a probability pr_{switch} they want *the other* measure to be recorded and with probability $1 - pr_{switch}$ they continue with the current value.

Process π_1 is the root process and reads only the probabilistic broadcast. The non-root processes behave similar to the self-stabilizing broadcast algorithm (BASS) presented in [1, 2] but without priorities. Processes π_2 and π_3 read from π_1 and from each other. Processes π_4 and π_5 read from both π_2 and π_3. A central scheduler demon that is not shown in the figure, probabilistically selects two processes per time step to execute in parallel. Each computation step starts with a read phase in which the executing processes inquire which type of data is to be stored. Afterwards, they store the corresponding measure. Forcing the processes to maintain a strict sequence of reading and writing allows to exclude read-after-write hazards as discussed by Patterson and Hennessy [17, 18]. For instance, assume that π_1 and π_2 execute. Without a strict sequence, it is undetermined whether first π_1 updates according to the radio broadcast and also updates π_2 in the same step, or if first π_2 inquires the *old* status of π_1 before both processes write. A strict sequence implies the latter constellation.

We abbreviate temperature with 0 and humidity with 2. When a process cannot determine which type is intended — that is, when there is no majority for one of the two types —, it stores nothing to save memory, and propagates 1 until it executes again. The fault model forces a process to store 0 when it should store 2 and vice versa. In case a process is supposed to store 1 and is perturbed by a fault, the effect of the fault is undetermined. We pessimistically assume that it stores the currently inappropriate value that is not broadcasted at that time to solve this non-determinism, thus computing the lower bound.

With five processes and two processes executing in parallel per time step, it requires at least five steps for every processes to have executed. Hence, after a switch on the broadcast, the system must execute at least three steps to propagate the new type of data to be stored. The system thus cannot continuously store the desired data in every process and every time step. With $pr_{switch} \geq \frac{1}{3}$, meaning that a switch occurs in average every three or less time steps, it is unlikely that — even without transient faults — the consistency is very high, since the mean switching interval is lower than the minimal time required for convergence.

B. The goal

The goal is to determine the *consistency* of the measured data in a given probabilistic environment. A *set of data* contains the data stored by each process. That set is consistent if it coincides with the broadcasted type at that time. The system has not only to cope with transient faults propagating through the system, but also with probabilistic switches. The system probabilistically converges to the currently broadcasted type of data and is thereby probabilistically self-stabilizing [19]. The question 'What is the probability that each process stores the desired type of data in each time step?' determines the desired fault tolerance property which concerns the consistency. Furthermore, we consider that all processes initially store 0 and 0 is broadcasted in the first time step. Therefore, we are interested in the *instantaneous window availability* (IWA) [20] with window size 1 for each time step.

C. The motivation

This example allows to highlight important properties. The first point concerns resolving non-determinism. In the fault model, it is undetermined which value the perturbed process stores when it is supposed to store 1. This example computes the *lower boundary* to solve this non-determinism. The same computation can be repeated with an optimistic assumption — that is, storing the currently broadcasted value — leading to the *upper boundary*. The upper and lower boundaries demarcate the corridor of possible legal execution traces.

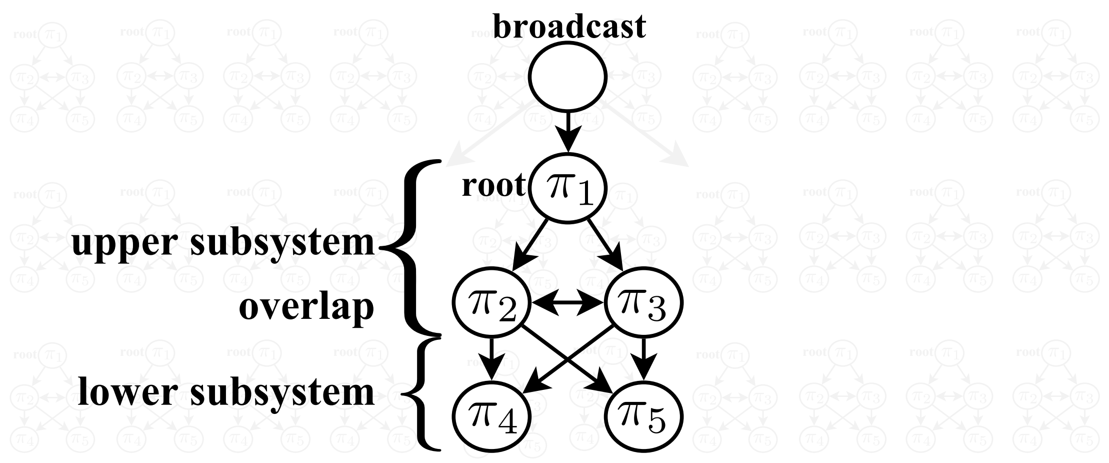


Figure 1. Wireless sensor network component

The second important point are semi-parallel execution semantics. Previous work on *hierarchically* structured systems [1, 2] introduced an adapted variant of the Kronecker product to account for exclusively one of the subsystems one process executed. *Semi-parallel execution semantics* — that is, not *all* processes but more than *one* process can execute per time step — are a special challenge as a case distinction is required that is neither necessary for strict serial nor for maximal parallel execution semantics. The present case study allows to address this challenging issue.

Third, the system contains a heterarchical subsystem with processes π_2 and π_3. The challenge here is to slice the system through the heterarchical set.

This also allows to discuss the fourth challenge: Slicing through multiple processes. This example allows to discuss why slicing through multiple processes is not more complex than slicing through one process as discussed in [1, 2]. The slicing in this example further demonstrates that the processes in subsystems — in this case π_4 and π_5 — need not even necessarily be connected.

D. The input parameters

The input parameters contain

1) fault probabilities,
2) switching probabilities and
3) scheduling probabilities.

We consider the fault probability of an executing process storing the wrong type of data to be $q = 0.01$ and the switching probability to be $pr_{switch} = 0.03$ for both switching directions, from 0 to 2 and vice versa. The numerical values[3] can be adapted as desired. The scheduler selects two processes randomly with a uniform probability distribution.

E. The safety predicate

The system is in a safe state — that is, the data set being recorded is consistent — when all processes record the value that is broadcasted at that time:

$$s_t \models \mathcal{P} \begin{cases} s_t = \langle 0,0,0,0,0 \rangle \wedge \text{broadcasted value is } 0 \\ s_t = \langle 2,2,2,2,2 \rangle \wedge \text{broadcasted value is } 2 \end{cases} \quad (1)$$

The quantification method can easily be adapted such that safety is also satisfied when not all, but only a subset of the processes stores the broadcasted type of value.

F. The state spaces of the subsystems

The first process can store either 0 or 2 and all other processes can derive 1 as well. Furthermore, the broadcasted value determines whether $s_t \models \mathcal{P}$ and must be accounted for as well. For instance, the system can be in state $s_t = \langle 0,0,0,0,0 \rangle$ when 0 is broadcasted, thus satisfying $\mathcal{P}$, or it can be in the same state when 2 is broadcasted, thus not satisfying $\mathcal{P}$. The full product transition model hence contains $|\mathcal{S}| = 2 \cdot 2 \cdot 3^4 = 324$ states — that is, number of possibly broadcasted values times the number of possible values in π_1 times the number of possible states to the power of processes these values are being stored in — as pictured in Figure 2(a). Coalescing states results in a state space of $|\mathcal{S}'| = 2 \cdot 2 \cdot 6 \cdot 6 = 144$ states as pictured in Figure 2(b).

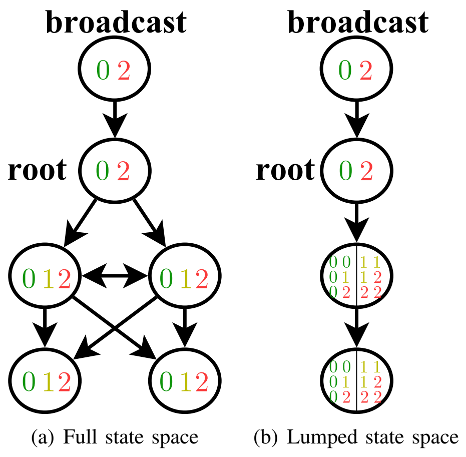


Figure 2. State space reduction

[3]The source code is available at http://www.informatik.uni-oldenburg.de/~phoenix/WSN.zip. The tables in the source code contain symbolic DTMCs such that the input parameters can be easily adapted.

G. The decomposition

The system is sliced in π_2 and π_3. The upper subsystem comprises processes π_1, π_2 and π_3, the lower subsystems π_2, π_3, π_4 and π_5. Both subsystems overlap in the heterarchical processes π_2 and π_3. The scheduler selects two processes. If the two processes were to deterministically execute both within the same subsystem, then the decomposition could be carried out like for serial execution semantics. Here, two processes, one in each subsystem, can execute in parallel. Therefore, each case must be accounted for with its own transition matrix. The first case is that both processes selected for execution belong to the upper subsystem. The second case is that both selected processes belong to the lower subsystem. The third case is that one process belongs to each of the two subsystems.

We label the sub-Markov chain for the upper subsystem $\mathcal{D}_1$ and $\mathcal{D}_2$ for the lower subsystem. A second index is added labeling the case that no process in the corresponding subsystem is selected $\mathcal{D}_{1,0}$, that one process is selected $\mathcal{D}_{1,1}$ or that both selected processes are within the subsystem $\mathcal{D}_{1,2}$ (analogously for $\mathcal{D}_2$). Figure 3 shows the decomposition schema. The upper subsystem — being hierarchically superior — is tackled first. The transition matrices describe what can happen in one execution step with two processes executing simultaneously. The three probabilistic influences are switch, fault and scheduler selection. The latter comprises the events $\mathfrak{s}_{1,2}$, i.e. the probability that processes π_1 and π_2 are selected, $\mathfrak{s}_{1,3}$, $\mathfrak{s}_{1,4}$, $\mathfrak{s}_{1,5}$, $\mathfrak{s}_{2,3}$, $\mathfrak{s}_{2,4}$, $\mathfrak{s}_{2,5}$, $\mathfrak{s}_{3,4}$, $\mathfrak{s}_{3,5}$ and $\mathfrak{s}_{4,5}$. With uniformly distributed scheduling probabilities, each combination is likely to be selected with a probability of 0.1. The probability that exactly one process in the upper subsystem is selected, is hence $\mathfrak{s}_{1,4} + \mathfrak{s}_{1,5} + \mathfrak{s}_{2,4} + \mathfrak{s}_{2,5} + \mathfrak{s}_{3,4} + \mathfrak{s}_{3,5} = \mathfrak{s}_{\text{both}} = 0.6$. The probability that both selected processes belong to the upper subsystem is $\mathfrak{s}_{1,2} + \mathfrak{s}_{1,3} + \mathfrak{s}_{2,3} = 0.3 = \mathfrak{s}_{\text{up}}$. The probability that none of them is selected, is $\mathfrak{s}_{4,5} = 0.1 = \mathfrak{s}_{\text{low}}$.

For each case, the transition matrix is constructed. Next, all three matrices $\mathcal{D}_{1,0}$, $\mathcal{D}_{1,1}$ and $\mathcal{D}_{1,2}$ are lumped to $\mathcal{D}'_{1,0}$, $\mathcal{D}'_{1,1}$ and $\mathcal{D}'_{1,2}$, shown in Figure 4. The colors are encoded on a standard MatLab spectrum from blue being 0 to red being 1 as shown in Figure 7. These matrices are required again later. We label a lumped state with the doubly lined alphabet according to the sum of the stored register values. For instance, the lump of states $\langle 0, 2\rangle$ and $\langle 2, 0\rangle$ is labeled $\langle \mathbb{2}\rangle$. Afterwards, the three matrices $\mathcal{D}'_{1,0}$, $\mathcal{D}'_{1,1}$ and $\mathcal{D}'_{1,2}$ are added and $\mathcal{D}'_{\pi_2,\pi_3}$ is uncoupled. Both π_4 and π_5 store

- 0 when reading $\langle 0, 0\rangle$ or $\langle \mathbb{1}\rangle$, which is the lumped state of states $\langle 0, 1\rangle$ and $\langle 1, 0\rangle$,
- 1 when reading $1, 1$ or $\langle \mathbb{2}\rangle$ which is the lumped state of states $\langle 0, 2\rangle$ and $\langle 2, 0\rangle$, and
- 2 when reading $\langle 2, 2\rangle$ or $\langle \mathbb{3}\rangle$ which is the lumped state of states $\langle 1, 2\rangle$ and $\langle 2, 1\rangle$.

H. A minor simplification

At this stage, we exploit a minor simplification that is helpful when computing the *instantaneous* window availability instead of a limiting property. When computing a limiting property, the limiting probability that a certain input is propagated from superior to inferior subsystem does not change over time. It is the same for time step Ω (i.e. the limit) as it is for time step $\Omega + 1$. For *instantaneous* properties on the other hand, the probability that a certain value is propagated *does* change, until reaching the stationary distribution in the limit. For a precise quantification, it would be necessary to construct the transition model for the lower subsystem for each time step.

The important question is, how this simplification influences the result. In the beginning, the input vector (i.e. the initial distribution) differs maximal from the stationary distribution. With each time step it differs less. When the stationary distribution replaces the current distribution as input parameter, convergence is sped up. The next question is, how much the convergence is sped up. Provided that the initial distribution assigns the complete probability mass to state $\langle 0, 0, 0, 0, 0\rangle$ and that 0 is broadcasted in the beginning, then the probability that this status changes is less 0.08. With each additional computation step the computation error gets smaller until becoming zero in the limit. The simplification is considered to be acceptable since i) the convergence to the stationary distribution is fast and the introduced error is only relevant in the first few computation steps, ii) the error converges to zero, and iii) the gain for accepting this minor error is a great simplification in the computation. The lower sub-Markov chain needs not to be computed (and lumped subsequently) for each time step.

I. Continuing the construction of $\mathcal{D}'$

Thus, matrices $\overline{\mathcal{D}}'_{2,0}$, $\overline{\mathcal{D}}'_{2,1}$ and $\overline{\mathcal{D}}'_{2,2}$ are constructed. In these matrices, the states where π_2 and π_3 are responsible for bisimilarities have been lumped but the states where π_4 and π_5 are responsible for bisimilarities have not. The overline notation marks the DTMCs as intermediate stage that possibly carries further potential for lumping. Processes π_2 and π_3 are uncoupled subsequently. Lumping further reduces the state space and $\mathcal{D}'_{2,-,0}$, $\mathcal{D}'_{2,-,1}$ and $\mathcal{D}'_{2,-,2}$ are constructed as shown in Figure 5. Finally, $\mathcal{D}'_{\text{low}} = \mathcal{D}'_{1,0} \otimes_K \mathcal{D}'_{2,-,2}$, $\mathcal{D}'_{\text{both}} = \mathcal{D}'_{1,1} \otimes_K \mathcal{D}'_{2,-,1}$ and $\mathcal{D}'_{\text{up}} = \mathcal{D}'_{1,2} \otimes_K \mathcal{D}'_{2,-,0}$ are computed by applying the Kronecker product. In this case — contrary to [1, 2] — the Kronecker product is applicable, since two processes execute in parallel. As the cases of parallel executions have been distinguished from the beginning, the full DTMC $\mathcal{D}'$ is the accumulated effort of all three case DTMCs: $\mathcal{D}' = \mathcal{D}'_{\text{low}} + \mathcal{D}'_{\text{both}} + \mathcal{D}'_{\text{low}}$, shown in Figures 6 and 7.

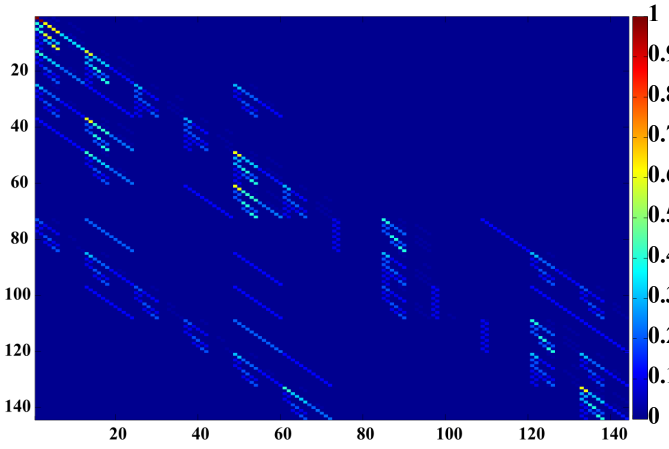


Figure 7. The reduced DTMC $\mathcal{D}'$

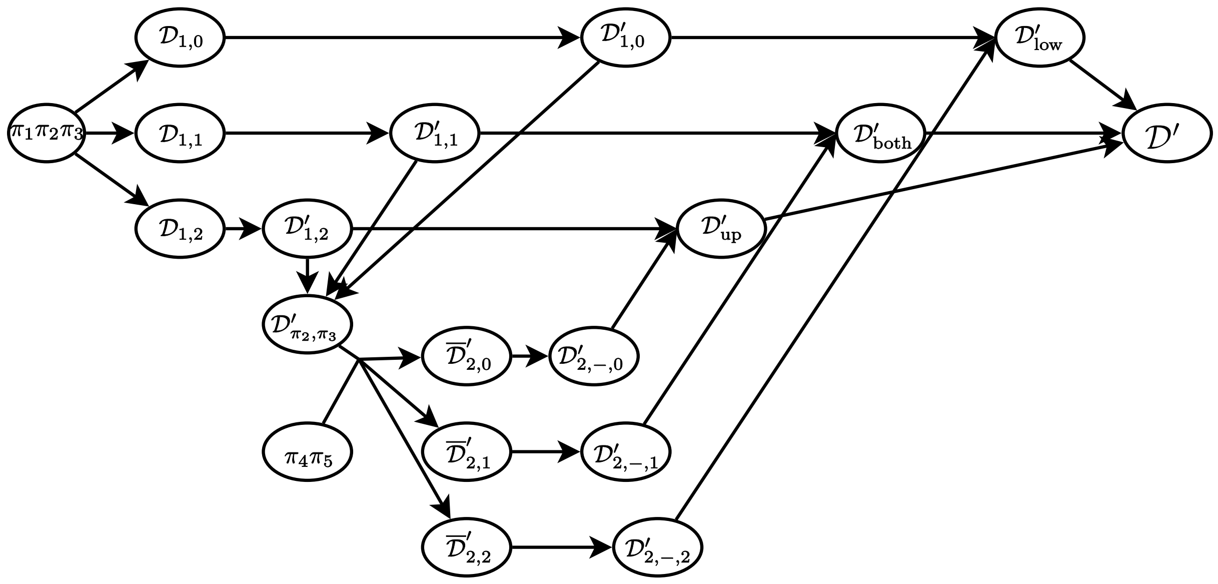
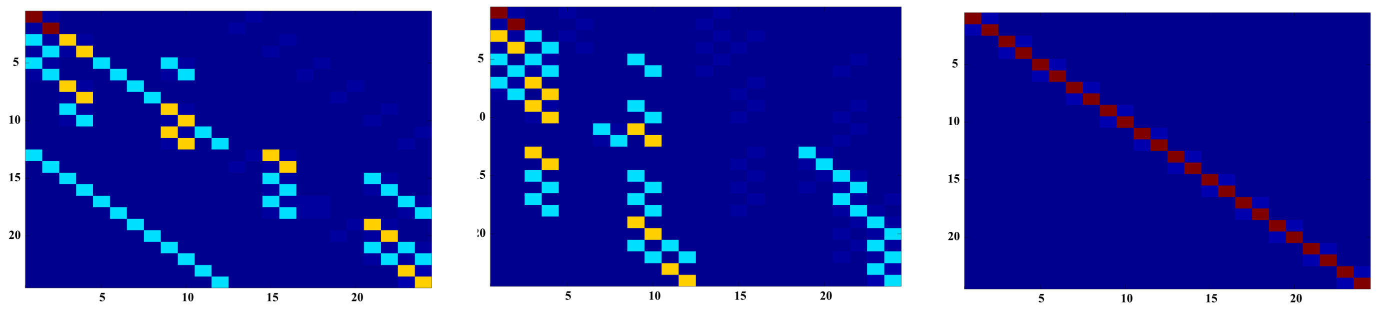
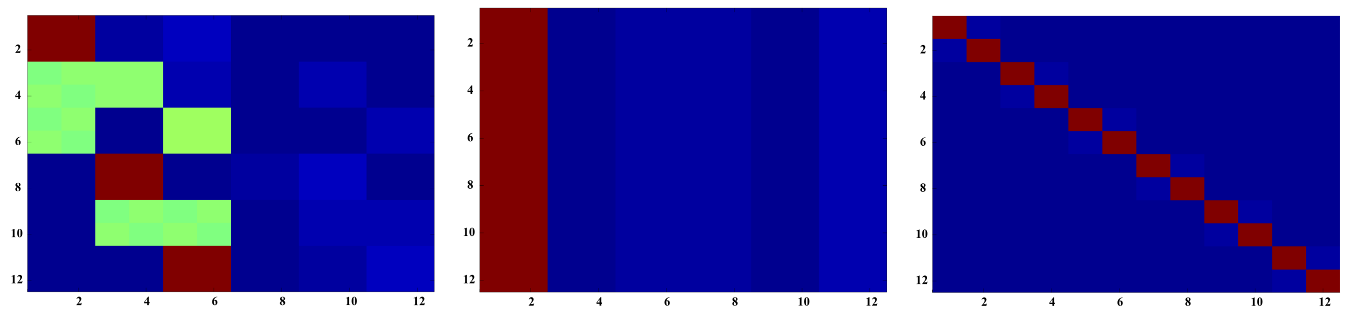


Figure 3. Decomposing schema for the WSN transition system



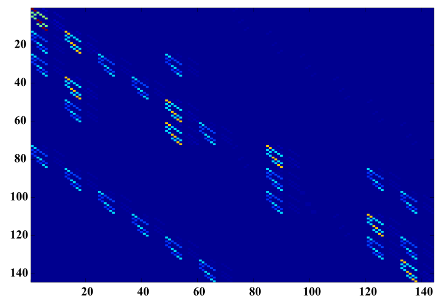
(a) $\mathcal{D}'_{1,0}$ (b) $\mathcal{D}'_{1,1}$ (c) $\mathcal{D}'_{1,2}$

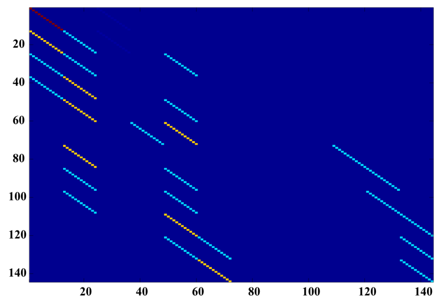
Figure 4. Reduced upper subsystem DTMCs



(a) $\mathcal{D}'_{2,-,0}$ (b) $\mathcal{D}'_{2,-,1}$ (c) $\mathcal{D}'_{2,-,2}$

Figure 5. Reduced and uncoupled lower subsystem DTMCs


(a) $\mathcal{D}'_{\text{low}}$


(b) $\mathcal{D}'_{\text{both}}$

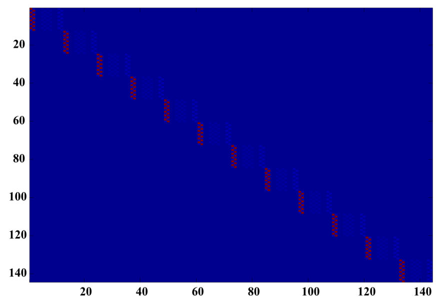

(c) $\mathcal{D}'_{\text{up}}$

Figure 6. Recomposed DTMCs

J. The result

Figure 8 shows the probability mass in states $\langle 0,0,0,0,0\rangle$ when 0 is propagated (i.e. the green line converging from above) and $\langle 2,2,2,2,2\rangle$ when 2 is propagated (i.e. the red line converging from below) for the first thousand time steps. It merely takes a little more than a hundred steps until both lines meet and the system *converges* to its stationary distribution. The numerical values at this time step are $pr(s_{100} = \langle 0,0,0,0,0\rangle \wedge \text{propagated value} = 0) = 0.4151$ and $pr(s_{100} = \langle 2,2,2,2,2\rangle \wedge \text{propagated value} = 2) = 0.4033$. With equal switching and fault probabilities, it was expected that both predicate satisfaction probabilities converge to the same value. With switching at 0.03 and a minimum of three computation steps for convergence and a fault probability of 0.01, it seems plausible that the consistency is about 0.82. The

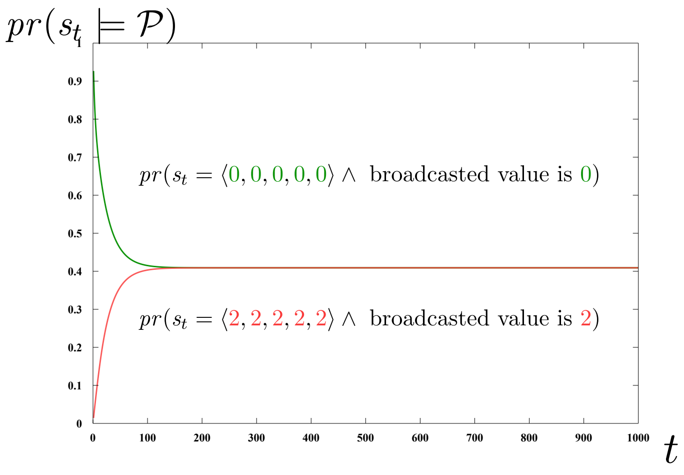


Figure 8. Result of the WSN example – converging consistency

average consistency of the measured data is about 0.82 in the limit. The *convergence inertia* — which is the time spent for convergence due to both switching and recovery — is shown in Figure 9 for the first 100 time steps. The term *inertia* is chosen as the system requires time to cope with switching and the effects of faults. The convergence inertia is the probability with regards to the current time step that the system is *between* the legal states.

K. Interpretation

While the *limiting windows availability*, which was in the focus of our previous work, is a measure on stop times, the current example demonstrates how such a measure on stop times can continuously be exploited by measuring the desired probability for *each* time step in contrast to just its first occurrence. The example further demonstrates how non-determinism, local heterarchies, semi-parallel execution semantics and slicing among multiple — even heterarchical lumpable — processes can be achieved. The notion of *switching* introduced a further challenge by demanding dynamic predicates and doubling the size of the state space. Furthermore, it extended pure *recovery liveness* during times of error to a more general *convergence inertia* that accounts for both switching as well as effects of faults. A system designer has now the opportunity to test various settings, alter the fault and switching probabilities as well as the topology, until a suitable system providing the desired consistency of the measured data is found.

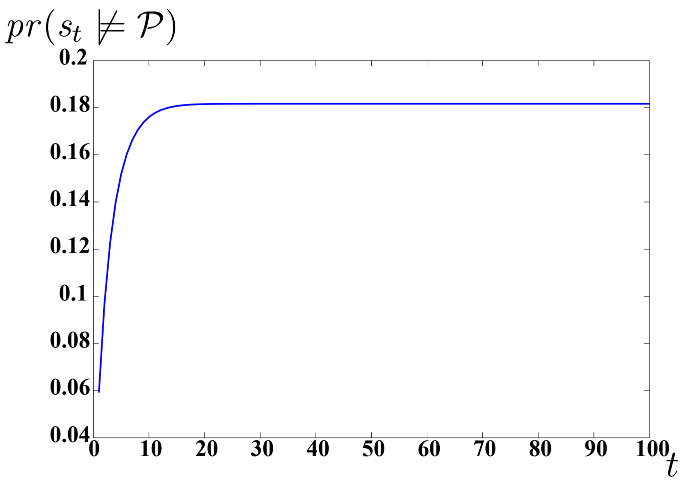


Figure 9. Result of the WSN example – convergence inertia

V. Conclusion and Future Work

We presented an extension to our previous approach. Related work has been presented and advances in its light were pointed out. These advances include: a novel fault tolerance measure for consistency, discussing how non-determinism can be regarded, extending the prior method to account for local heterarchies, semi-parallel execution semantics via case distinction and decomposing via slicing through multiple processes. An example scenario evaluated a wireless sensor network component in that light in order to demonstrate the practical application of the concepts and methods. Although wireless sensor networks are a practical application, we will focus fitting the proposed methods and concepts on further real world applications. A second direction for the future is to extend the concepts to cover for security, too. In that sense, the probabilistic notions of fault tolerance and performance will be combined with a rather deterimistic view of security and interactive Markov chains will be employed as model to determine the safe and secure co-design of a distributed system.

Acknowledgment

This work was (partially) supported by the German Research Council under grant SFB/TR 14 *AVACS*, the EU Commission under grant FP7-ICT-2009-257005 *MoVeS* and by the funding initiative *Niedersächsisches Vorab* of the Volkswagen Foundation and the Ministry of Science and Culture of Lower Saxony (as part of the *Interdisciplinary Research Center on Critical Systems Engineering for Socio-Technical Systems*).

References

[1] Nils Müllner, Oliver Theel, and Martin Fränzle. Combining Decomposition and Reduction for State Space Analysis of a Self-Stabilizing System. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA2012)*, pages 936 – 943, Fukuoka-shi, Fukuoka, Japan, March 2012. IEEE Computer Society Press. Best Paper.

[2] Nils Müllner, Oliver Theel, and Martin Fränzle. Combining Decomposition and Reduction for the State Space Analysis of Self-Stabilizing Systems. In *Journal of Computer and System Sciences (JCSS)*, volume 79, pages 1113 – 1125. Elsevier Science Publishers B. V., November 2013. The paper is an extended version of a publication with the same title.

[3] John G. Kemeny and James L. Snell. *Finite Markov Chains*. University Series in Undergraduate Mathematics. New York, NY, USA, 2, 1976 edition, 1976.

[4] Kim G. Larsen and Arne Skou. Bisimulation Through Probabilistic Testing. In *Conference Record of the 16th ACM Symposium on Principles of Programming Languages (POPL1989*, pages 344 – 352, 1989.

[5] Peter Buchholz. Exact and Ordinary Lumpability in Finite Markov Chains. *Journal of Applied Probability*, 31(1):59–75, 1994.

[6] Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen. Bisimulation Minimisation Mostly Speeds up Probabilistic Model Checking. In *Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'07, pages 87–101, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Hubert Garavel, Frédéric Lang, and Radu Mateescu. An overview of CADP 2001. Research Report RT-0254, INRIA, 2001.

[8] LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Standard, September 1989. Information Processing Systems, Open Systems Interconnection.

[9] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. Dynamic Fault Tree analysis using Input/Output Interactive Markov Chains. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007*, pages 708–717, Los Alamitos, CA, USA, June 2007. IEEE Computer Society Press.

[10] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains. In *Proceedings of the 5th international conference on Automated technology for verification and analysis*, ATVA'07, pages 441–456, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] Hichem Boudali, Pepijn Crouzen, Boudewijn R. Haverkort, Matthias Kuntz, and Mariëlle Stoelinga. Arcade - A Formal, Extensible, Model-Based Dependability Evaluation Framework. In *ICECCS*, pages 243–248. IEEE Computer Society, 2008.

[12] Hichem Boudali, Pepijn Crouzen, Boudewijn R. Haverkort, Matthias Kuntz, and Mariëlle Stoelinga. Architectural Dependability Evaluation with Arcade. In *DSN*, pages 512–521, 2008.

[13] Hichem Boudali, Hasan Sözer, and Mariëlle Stoelinga. Architectural Availability Analysis of Software Decomposition for Local Recovery. In *SSIRI*, pages 14–22, 2009.

[14] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis. *IEEE Trans. Dependable Sec. Comput.*, 7(2):128–143, 2010.

[15] Holger Hermanns and Joost-Pieter Katoen. Automated Compositional Markov Chain Generation for a Plain-Old Telephone System. In *Science of Computer Programming*, pages 97–127, 1999.

[16] Jenna Burrell, Tim Brooke, and Richard Beckwith. Vineyard Computing: Sensor Networks in Agricultural Production. *IEEE Pervasive Computing*, 3:38 – 45, 2004.

[17] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 2nd Edition*. Morgan Kaufmann Publishers Inc., 1996.

[18] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., 2005.

[19] Stéphane Devismes, Sèbastien Tixeuil, and Masafumi Yamashita. Weak vs. Self vs. Probabilistic Stabilization. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS2008)*, pages 681 – 688, Washington, DC, USA, 2008. IEEE Computer Society Press.

[20] Nils Müllner, Abhishek Dhama, and Oliver Theel. Deriving a Good Trade-off Between System Availability and Time Redundancy. In *Proceedings of the Symposia and Workshops on Ubiquitous, Automatic and Trusted Computing*, number E3737 in Track "International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies (Ufirst-09)", pages 61 – 67, Brisbane, QLD, Australia, July 2009. IEEE Computer Society Press.