Construction Principles for Well-behaved Scalable Systems

Peter Ochsenschläger* and Roland Rieke*†
*Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
†Philipps-Universität Marburg, Germany
Email: peter-ochsenschlaeger@t-online.de, roland.rieke@sit.fraunhofer.de

***Abstract*—We formally define *scalable systems* as *uniformly monotonic parameterised systems* and motivate this definition. With respect to such scalable systems, we focus on properties, which rely on specific component types and a specific number of individual components for these component types but not on the specific individuality of the individual components. We characterise *well-behaved* scalable systems by those systems which fulfil such a kind of property if already one prototype system (depending on the property) fulfils that property. Self-similar uniformly monotonic parameterised systems have the above desired property. Therefore, we define well-behaved scalable systems as self-similar scalable systems. This paper presents a formal framework that provides construction principles for well-behaved scalable systems. It gives sufficient conditions to specify a certain kind of basic well-behaved scalable systems and shows how to construct more complex systems by the composition of several synchronisation conditions.**

Keywords-uniformly parameterised systems; monotonic parameterised systems; behaviour-abstraction; self-similarity of behaviour.

I. Introduction

Scalability is a desirable property of a system. In [1], four aspects of scalability are considered, i.e., load scalability, space scalability, space-time scalability, and structural scalability. In our paper, we focus on *structural scalability*, which is "the ability of a system to expand in a chosen dimension without major modifications to its architecture" [1]. Examples of systems that need to be highly scalable comprise grid computing architectures and cloud computing platforms [2], [3]. Usually, such systems consist of few different types of components and for each such type a varying set of individual components exists. Component types can be defined in such a granularity that individual components of the same type behave in the same manner, which is characteristic for the type. For example, a client-server system that is scalable consists of the component types *client* and *server* and several sets of individual clients as well as several sets of individual servers. Let us now call a choice of sets of individual components an *admissible choice of individual component sets*, iff for each component type exactly one set of individual components of that type is chosen. Then, a "scalable system" can be considered as a family of systems, whose elements are systems composed of a specific admissible choice of individual component sets.

In this paper, we focus on the dynamic behaviour of systems, which is described by the set of all possible sequences of actions. This point of view is important to define security requirements as well as to verify such properties, because for these purposes sequences of actions of the system have to be considered [4], [5], [6]. For short, we often will use the term *system* instead of *systems behaviour* if it does not generate confusions. With this focus, scalable systems are families of system behaviours, which are indexed by admissible choices of individual component sets. We call such families *parameterised systems*. In this paper, we define *well-behaved scalable systems* as a special class of parameterised systems and develop construction principles for such systems. The main goal for this definition is to achieve that well-behaved scalable systems fulfill certain kind of safety properties if already one prototype system (depending on the property) fulfills that property (cf. Section III). To this end, construction principles for well-behaved scalable systems are *design principles for verifiability* [7].

Considering the behaviour-verification aspect, which is one of our motivations to formally define well-behaved scalable systems, there are some other approaches to be mentioned. An extension to the Murφ verifier to verify systems with replicated identical components through a new data type called RepetitiveID is presented in [8]. A typical application area of this tool are cache coherence protocols. The aim of [9] is an abstraction method through symmetry, which works also when using variables holding references to other processes. In [10], a methodology for constructing abstractions and refining them by analysing counter-examples is presented. The method combines abstraction, model-checking and deductive verification. A technique for automatic verification of parameterised systems based on process algebra CCS [12] and the logic modal mu-calculus [13] is presented in [11]. This technique views processes as property transformers and is based on computing the limit of a sequence of mu-calculus [13] formulas generated by these transformers. The above-mentioned approaches demonstrate that finite state methods combined with deductive methods can be applied to analyse parameterised

 ISBN: 978-1-61208-319-3

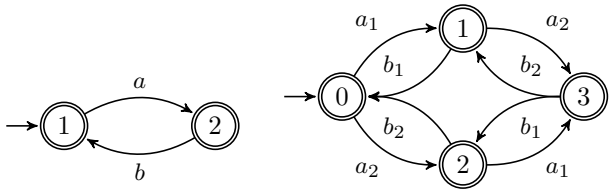
systems. The approaches differ in varying amounts of user intervention and their range of application. A survey of approaches to combine model checking and theorem proving methods is given in [14]. Far reaching results in verifying parameterised systems by model checking of corresponding abstract systems are given in [15], [16]. It is well known that the general verification problem for parameterised systems is undecidable [17], [18]. To handle that problem, we present (a) a formal framework to specify parameterised systems in a restricted manner, and (b) construction principles for well-behaved scalable systems.

In Section II, scalable systems are formally defined. Section III and Section IV give sufficient conditions to specify a certain kind of basic well-behaved scalable systems. Section V shows how to construct more complex well-behaved scalable systems by the composition of several synchronisation conditions. Concluding remarks and further research directions are given in Section VI. The proofs of the theorems in this paper are given in [19].

II. Characterisation of Scalable Systems

The behaviour L of a discrete system can be formally described by the set of its possible sequences of actions. Therefore, $L \subset \Sigma^*$ holds where Σ is the set of all actions of the system, and Σ^* (free monoid over Σ) is the set of all finite sequences of elements of Σ, including the empty sequence denoted by ε. This terminology originates from the theory of formal languages [20], where Σ is called the alphabet (not necessarily finite), the elements of Σ are called letters, the elements of Σ^* are referred to as words and the subsets of Σ^* as formal languages. Words can be composed: if u and v are words, then uv is also a word. This operation is called the *concatenation*; especially $\varepsilon u = u\varepsilon = u$. A word u is called a *prefix* of a word v if there is a word x such that $v = ux$. The set of all prefixes of a word u is denoted by $\mathrm{pre}(u)$; $\varepsilon \in \mathrm{pre}(u)$ holds for every word u. Formal languages which describe system behaviour have the characteristic that $\mathrm{pre}(u) \subset L$ holds for every word $u \in L$. Such languages are called *prefix closed*. System behaviour is thus described by prefix closed formal languages. Different formal models of the same system are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$, $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$. So, they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following, we denote both the mapping h and the homomorphism h^* by h. We consider a lot of alphabetic language homomorphisms. So, for simplicity we tacitly assume that a mapping between free monoids is an alphabetic language homomorphism if nothing contrary is stated. We now introduce a guiding example.

Example 1. *A server answers requests of a family of clients. The actions of the server are considered in the following. We assume with respect to each client that a request will be answered before a new request from this client is accepted. If the family of clients consists of only one client, then the automaton in Fig. 1(a) describes the system behaviour $S \subset \Sigma^*$, where $\Sigma = \{a, b\}$, the label a depicts the request, and b depicts the response.*



(a) Actions at a server with respect to a client (b) Two clients served concurrently by one server

Figure 1. Scalable client-server system

Example 2. *Fig. 1(b) now describes the system behaviour $S_{\{1,2\}} \subset \Sigma^*_{\{1,2\}}$ for two clients 1 and 2, under the assumption that the server handles the requests of different clients non-restricted concurrently.*

For a parameter set I and $i \in I$ let $\Sigma_{\{i\}}$ denote pairwise disjoint copies of Σ. The elements of $\Sigma_{\{i\}}$ are denoted by a_i and $\Sigma_I := \bigcup_{i \in I} \Sigma_{\{i\}}$, where $\Sigma_j \cap \Sigma_k = \emptyset$ for $j \neq k$. The index i describes the bijection $a \leftrightarrow a_i$ for $a \in \Sigma$ and $a_i \in \Sigma_{\{i\}}$.

Example 3. *For $\emptyset \neq I \subset \mathbb{N}$ with finite I, let now $S_I \subset \Sigma^*_I$ denote the system behaviour with respect to the client set I. For each $i \in \mathbb{N}$ $S_{\{i\}}$ is isomorphic to S, and S_I consists of the non-restricted concurrent run of all $S_{\{i\}}$ with $i \in I$.*

It holds $S_{I'} \subset S_I$ for $I' \subset I$.

Let $\mathcal{I}_1$ denote the set of all finite non-empty subsets of $\mathbb{N}$ (the set of all possible clients). Then, the family $(S_I)_{I \in \mathcal{I}_1}$ is an example of a monotonic parameterised system.

If the example is extended to consider several servers, which are depicted by natural numbers, then, e.g.,

$$\mathcal{I}_2 := \{\mathring{I} \times \hat{I} \subset \mathbb{N} \times \mathbb{N} | \mathring{I} \neq \emptyset \neq \hat{I}, \text{ with } \mathring{I}, \hat{I} \text{ finite}\}$$

is a suitable parameter structure.

$\mathcal{I}_2$ used in the example above shows how the component structure of a system can be expressed by a parameter structure using Cartesian products of individual component sets. The following Definition 1 abstracts from the intuition of a component structure.

Definition 1 (parameter structure)**.** *Let N be a countable (infinite) set and $\emptyset \neq \mathcal{I} \subset \mathfrak{P}(N) \setminus \{\emptyset\}$. $\mathcal{I}$ is called a* parameter structure *based on N.*

For scalable systems it is obvious to assume that enlarging the individual component sets does not reduce the corresponding system behaviour. More precisely: let I and K be two arbitrary admissible choices of individual component sets, where each individual component set in I is a subset of the corresponding individual component set in K. If S_I and S_K are the corresponding systems' behaviours, then S_I is a subset of S_K. Families of systems with this property we call *monotonic parameterised systems*. The following definition formalises monotonic parameterised systems.

Definition 2 (monotonic parameterised system)**.** *Let $\mathcal{I}$ be a* parameter structure*. For each $I \in \mathcal{I}$ let $\mathcal{L}_I \subset \Sigma_I^*$ be a prefix closed language. If $\mathcal{L}_{I'} \subset \mathcal{L}_I$ for each $I, I' \in \mathcal{I}$ with $I' \subset I$, then $(\mathcal{L}_I)_{I\in\mathcal{I}}$ is a* monotonic parameterised system*.*

As we assume that individual components of the same type behave in the same manner, S_I and S_K are isomorphic (equal up to the names of the individual components), if I and K have the same cardinality. This property we call *uniform parameterisation*. With these notions we define *scalable systems* as *uniformly monotonic parameterised systems*. Monotonic parameterised systems in which isomorphic subsets of parameter values describe isomorphic subsystems we call *uniformly monotonic parameterised systems*.

Definition 3 (isomorphism structure)**.** *Let $\mathcal{I}$ be a parameter structure, $I, K \in \mathcal{I}$, and $\iota : I \to K$ a bijection, then let $\iota_K^I : \Sigma_I^* \to \Sigma_K^*$ the isomorphism defined by*

$$\iota_K^I(a_i) := a_{\iota(i)} \text{ for } a_i \in \Sigma_I. \tag{1}$$

For each $I, K \in \mathcal{I}$ let $\mathcal{B}(I,K) \subset K^I$ a set (possibly empty) of bijections. $\mathcal{B}_\mathcal{I} = (\mathcal{B}(I,K))_{(I,K)\in\mathcal{I}\times\mathcal{I}}$ is called an isomorphism structure *for $\mathcal{I}$.*

Definition 4 (scalable system)**.** *Let $(\mathcal{L}_I)_{I\in\mathcal{I}}$ a monotonic parameterised system and $\mathcal{B}_\mathcal{I} = (\mathcal{B}(I,K))_{(I,K)\in\mathcal{I}\times\mathcal{I}}$ an* isomorphism structure *for $\mathcal{I}$.*
$(\mathcal{L}_I)_{I\in\mathcal{I}}$ is called uniformly monotonic parameterised with respect to $\mathcal{B}_\mathcal{I}$ *iff*

$$\mathcal{L}_K = \iota_K^I(\mathcal{L}_I) \text{ for each } I, K \in \mathcal{I} \text{ and each } \iota \in \mathcal{B}(I,K).$$

Uniformly monotonic parameterised systems for short are called scalable systems*.*

Example 4. *Let $\mathcal{I} = \mathcal{I}_2$.*

$$\mathcal{B}^2(\mathring{I}\times\hat{I}, \mathring{K}\times\hat{K}) := \{\iota \in (\mathring{K}\times\hat{K})^{(\mathring{I}\times\hat{I})} \mid \text{it exist bijections } \mathring{\iota} : \mathring{I} \to \mathring{K} \text{ and } \hat{\iota} : \hat{I} \to \hat{K} \text{ with } \iota((r,s)) = (\mathring{\iota}(r), \hat{\iota}(s)) \text{ for each } (r,s) \in (\mathring{I}\times\hat{I})\}$$

for $\mathring{I}\times\hat{I} \in \mathcal{I}_2$ and $\mathring{K}\times\hat{K} \in \mathcal{I}_2$ defines an isomorphism structure *$\mathcal{B}^2_{\mathcal{I}_2}$.*

III. Well-behaved Scalable Systems

To motivate our formalisation of *well-behaved*, we consider a typical security requirement of a scalable client-server system: Whenever two different clients cooperate with the same server then certain critical sections of the cooperation of one client with the server must not overlap with critical sections of the cooperation of the other client with the same server. If for example both clients want to use the same resource of the server for confidential purposes, then the allocation of the resource to one of the clients has to be completely separated from the allocation of this resource to the other client. More generally, the concurrent cooperation of one server with several clients has to be restricted by certain *synchronisation conditions* to prevent, for example, undesired race conditions.

According to this example, we focus on properties which rely on specific component types and a specific number of individual components for these component types but not on the specific individuality of the individual components. Now, we want to achieve that a well behaved scalable system fulfils such a kind of property if already one *prototype system* (depending on the property) fulfils that property. In our example, a prototype system consists of two specific clients and one specific server.

To formalise this desire, we consider arbitrary I and K as in the definition of monotonic parameterised system. Then we look at S_K from an abstracting point of view, where only actions corresponding to the individual components of I are considered. If the smaller subsystem S_I behaves like the abstracted view of S_K, then we call this property *self-similarity* or more precisely *self-similarity of scalable systems*, to distinguish our notion from geometric oriented notions [21] and organisational aspects [22] of self-similarity. In [5], it is shown that *self-similar uniformly monotonic parameterised systems* have the above desired property. Therefore, we define *well-behaved scalable systems* as self-similar uniformly monotonic parameterised systems. We now formally look at $\mathcal{L}_I$ from an abstracting point of view concerning a subset $I' \subset I$. The corresponding abstractions are formalised by the homomorphisms $\Pi_{I'}^I : \Sigma_I^* \to \Sigma_{I'}^*$.

Definition 5 (self-similar monotonic parameterised system)**.** *For $I' \subset I$ let $\Pi_{I'}^I : \Sigma_I^* \to \Sigma_{I'}^*$ with*

$$\Pi_{I'}^I(a_i) = \begin{cases} a_i \mid & a_i \in \Sigma_{I'} \\ \varepsilon \mid & a_i \in \Sigma_I \setminus \Sigma_{I'}. \end{cases}$$

A monotonic parameterised system $(\mathcal{L}_I)_{I\in\mathcal{I}}$ is called self-similar *iff $\Pi_{I'}^I(\mathcal{L}_I) = \mathcal{L}_{I'}$ for each $I, I' \in \mathcal{I}$ with $I' \subset I$.*

Definition 6 (well-behaved scalable system)**.** *Self-similar scalable systems for short are called* well-behaved scalable systems*.*

A fundamental construction principle for systems

satisfying several constraints is intersection of system behaviours. This emphasises the importance of the following theorem.

Theorem 1 (intersection theorem)**.** *Let $\mathcal{I}$ be a parameter structure, $\mathcal{B}_\mathcal{I}$ an isomorphism structure for $\mathcal{I}$, and $T \neq \emptyset$.*

i) *Let $(\mathcal{L}_I^t)_{I\in\mathcal{I}}$ for each $t \in T$ be a monotonic parameterised system, then $(\bigcap_{t\in T} \mathcal{L}_I^t)_{I\in\mathcal{I}}$ is a monotonic parameterised system.*

ii) *Let $(\mathcal{L}_I^t)_{I\in\mathcal{I}}$ for each $t \in T$ be a scalable system with respect to $\mathcal{B}_\mathcal{I}$, then $(\bigcap_{t\in T} \mathcal{L}_I^t)_{I\in\mathcal{I}}$ is a scalable system with respect to $\mathcal{B}_\mathcal{I}$.*

iii) *Let $(\mathcal{L}_I^t)_{I\in\mathcal{I}}$ for each $t \in T$ be a self-similar monotonic parameterised system, then $(\bigcap_{t\in T} \mathcal{L}_I^t)_{I\in\mathcal{I}}$ is a self-similar monotonic parameterised system.*

Weak additional assumptions for well-behaved scalable systems imply that such systems are characterised by parametrisation of one well-defined minimal prototype system. More precisely:

Definition 7 (minimal prototype system)**.** *Let $\mathcal{I}$ be a parameter structure based on N. For $I \in \mathcal{I}$ and $n \in N$ let $\tau_n^I : \Sigma_I^* \to \Sigma^*$ the homomorphisms given by*

$$\tau_n^I(a_i) = \begin{cases} a \mid & a_i \in \Sigma_{I\cap\{n\}} \\ \varepsilon \mid & a_i \in \Sigma_{I\setminus\{n\}} \end{cases}.$$

For a singleton index set $\{n\}$, $\tau_n^{\{n\}} : \Sigma_{\{n\}}^* \to \Sigma^*$ is an isomorphism and for each $n \in I \in \mathcal{I}$ holds

$$\Pi_{\{n\}}^I = (\tau_n^{\{n\}})^{-1} \circ \tau_n^I. \tag{2}$$

If now $(\mathcal{L}_I)_{I\in\mathcal{I}}$ is a well-behaved scalable system with respect to $(\mathcal{B}(I,K))_{(I,K)\in\mathcal{I}\times\mathcal{I}}$ with $\{n\} \in \mathcal{I}$ for $n \in I \in \mathcal{I}$ and $\mathcal{B}(I,K) \neq \emptyset$ for all singleton I and K, then because of (2) holds

$$\mathcal{L}_I \subset \bigcap_{n\in I} (\tau_n^I)^{-1}(L) \text{ for each } I \in \mathcal{I},$$

where $L = \tau_n^{\{n\}}(\mathcal{L}_{\{n\}})$ for each $n \in \bigcup_{I\in\mathcal{I}} I$.

L is called the minimal prototype system *of $(\mathcal{L}_I)_{I\in\mathcal{I}}$.*

Definition 8 (behaviour-family $(\dot{\mathcal{L}}(L)_I)_{I\in\mathcal{I}}$ generated by the minimal prototype system L and the parameter structure $\mathcal{I}$)**.** *Let $\emptyset \neq L \subset \Sigma^*$ be prefix closed, $\mathcal{I}$ a parameter structure, and*

$$\dot{\mathcal{L}}(L)_I := \bigcap_{i\in I} (\tau_i^I)^{-1}(L) \text{ for } I \in \mathcal{I}.$$

The systems $\dot{\mathcal{L}}(L)_I$ consist of the "non-restricted concurrent run" of all systems $(\tau_i^{\{i\}})^{-1}(L) \subset \Sigma_{\{i\}}^*$ with $i \in I$. Because $\tau_i^{\{i\}} : \Sigma_{\{i\}}^* \to \Sigma^*$ are isomorphisms, $(\tau_i^{\{i\}})^{-1}(L)$ are pairwise disjoint copies of L.

Theorem 2 (simplest well-behaved scalable systems)**.** *$(\dot{\mathcal{L}}(L)_I)_{I\in\mathcal{I}}$ is a well-behaved scalable system with respect to each isomorphism structure for $\mathcal{I}$ based on N and*

$$\dot{\mathcal{L}}(L)_I = \bigcap_{i\in N} (\tau_i^I)^{-1}(L) \text{ for each } I \in \mathcal{I}.$$

IV. Construction of Well-behaved Systems by Restriction of Concurrency

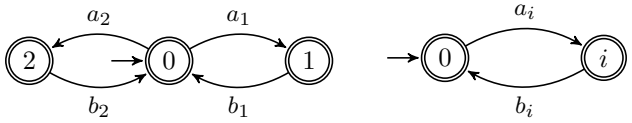
Now, we show how to construct well-behaved systems by restricting concurrency in the behaviour-family $\dot{\mathcal{L}}$. In Example 3, holds $S_I = \dot{\mathcal{L}}(S)_I$ for $I \in \mathcal{I}_1$. If, in the given example, the server needs specific resources for the processing of a request, then - on account of restricted resources - an non-restricted concurrent processing of requests is not possible. Thus, restrictions of concurrency in terms of synchronisation conditions are necessary. One possible but very strong restriction is the requirement that the server handles the requests of different clients in the same way as it handles the requests of a single client, namely, on the request follows the response and vice versa. This synchronisation condition can be formalised with the help of S and the homomorphisms Θ^I as shown in the following example.

Example 5. *Restriction of concurrency on account of restricted resources: one "task" after another. All behaviours with respect to $i \in I$* influence *each other. Let*

$$\bar{S}_I := S_I \cap (\Theta^I)^{-1}(S) = \bigcap_{i\in I} (\tau_i^I)^{-1}(S) \cap (\Theta^I)^{-1}(S)$$

for $I \in \mathcal{I}_1$, where generally, for each index set I, $\Theta^I : \Sigma_I^* \to \Sigma^*$ is defined by $\Theta^I(a_i) := a$, for $i \in I$ and $a \in \Sigma$.

From the automaton in Fig. 1(b), it is evident that $\bar{S}_{\{1,2\}}$ will be accepted by the automaton in Fig. 2(a).



(a) Automaton accepting $\bar{S}_{\{1,2\}}$ (b) Automaton accepting $\bar{S}_I$

Figure 2. Automata accepting $\bar{S}_{\{1,2\}}$ and $\bar{S}_I$

Given an arbitrary $I \in \mathcal{I}_1$, then $\bar{S}_I$ is accepted by an automaton with state set $\{0\} \cup I$ and state transition relation given by Fig. 2(b) for each $i \in I$.

From this automaton, it is evident that $(\bar{S}_I)_{I\in\mathcal{I}_1}$ is a well-behaved scalable system, with respect to each isomorphism structure $\mathcal{B}_{\mathcal{I}_1}$ for $\mathcal{I}_1$.

Example 6. *A restriction of concurrency in the extended example where a family of servers is involved is more complicated than in the case of $(\bar{S}_I)_{I\in\mathcal{I}_1}$. The reason for that is that in the simple example the restriction of*

concurrency can be formalised by a restricting influence of the actions with respect to all parameter values (i.e., the entire Σ_I). When considering the restriction of concurrency in the extended example, the actions influence each other only with respect to the parameter values, which are bound to the same server.

Let the first component of the elements from $\mathbb{N}\times\mathbb{N}$ in the parameter structure $\mathcal{I}_2$ denote the server, then the actions from $\Sigma_{\{r\}\times\hat{I}}$ influence each other for given $r\in\mathring{I}$ with $\mathring{I}\times\hat{I}\in\mathcal{I}$ and thus restrict the concurrency.

For the formalisation of this restriction of concurrency, we now consider the general case of monotonic parameterised systems $(\dot{\mathcal{L}}(L)_I)_{I\in\mathcal{I}}$. As already observed in (2), for each well-behaved scalable system $(\mathcal{L}_I)_{I\in\mathcal{I}}$ there exists (under weak preconditions) a system $(\dot{\mathcal{L}}(L)_I)_{I\in\mathcal{I}}$ with $\mathcal{L}_I\subset\dot{\mathcal{L}}(L)_I$ for each $I\in\mathcal{I}$, where $L=\tau_n^{\{n\}}(\mathcal{L}_{\{n\}})$ for each $n\in I\in\mathcal{I}$. Moreover, in context of Definition 8 it was observed that $\dot{\mathcal{L}}(L)_I$ consists of the non-restricted concurrent run of pairwise disjoint copies of L.

In conjunction, this shows that an adequate restriction of concurrency in $(\dot{\mathcal{L}}(L)_I)_{I\in\mathcal{I}}$ can lead to the construction of well-behaved scalable systems. Therefore, the restricting influence of actions with respect to specific parameter values described above shall now be formalised.

Definition 9 (influence structure)**.** *Let $T\neq\emptyset$ and $\mathcal{I}$ a parameter structure. For each $I\in\mathcal{I}$ and $t\in T$ a* sphere of influence *is specified by $E(t,I)\subset I$. The family*

$$\mathcal{E}_{\mathcal{I}}=(E(t,I))_{(t,I)\in T\times\mathcal{I}}$$

is called influence structure *for $\mathcal{I}$ indexed by T.*

The non-restricted concurrent run of the pairwise disjoint copies of L will now be restricted in the following way: For each $t\in T$ the runs of all copies k with $k\in E(t,I)$ influence each other independently of the specific values of $k\in E(t,I)$. With respect to our extended example (several servers) with $\mathcal{I}_2$, the spheres of influence $E(t,I)$ are generalisations of the sets $\{r\}\times\hat{I}$, where $I=\mathring{I}\times\hat{I}$ and $t=(r,s)\in\mathring{I}\times\hat{I}$.

Generally, for each $t\in T$ the intersection

$$\dot{\mathcal{L}}(L)_I\cap(\tau^I_{E(t,I)})^{-1}(V) \tag{3}$$

formalises the restriction of the non-restricted concurrent run of the copies of L within $\dot{\mathcal{L}}(L)_I$ by the mutual influence of each element of $E(t,I)$.

Definition 10 (behaviour of influence and influence homomorphisms)**.** *In* (3)*, the* behaviour of influence *V is a prefix closed language $V\subset\Sigma^*$, and for $I,I'\subset N$ the homomorphism $\tau^I_{I'}:\Sigma^*_I\to\Sigma^*$ is defined by:*

$$\tau^I_{I'}(a_i)=\begin{cases} a \mid & a_i\in\Sigma_{I\cap I'}\\ \varepsilon \mid & a_i\in\Sigma_{I\setminus I'}\end{cases}. \tag{4}$$

The homomorphisms $\tau^I_{E(t,I)}$ are called the influence homomorphisms *of $\mathcal{E}_{\mathcal{I}}$.*

Definition 11 (behaviour-family $(\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I)_{I\in\mathcal{I}}$ generated by the minimal prototype system L, the influence structure $\mathcal{E}_{\mathcal{I}}$, and the behaviour of influence V)**.** *Because the restriction* (3) *shall hold for all $t\in T$, the restricted systems $\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I$ are defined by the prefix closed languages*

$$\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I:=\dot{\mathcal{L}}(L)_I\cap\bigcap_{t\in T}(\tau^I_{E(t,I)})^{-1}(V)\ \text{for } I\in\mathcal{I}.$$

Definition 11 shows how synchronisation requirements for the systems $\dot{\mathcal{L}}(L)_I$ can be formalised by influence structures and behaviour of influence in a very general manner. Since, similar to the well-behaved scalable systems $(\dot{\mathcal{L}}(L)_I)_{I\in\mathcal{I}}$, in the systems $(\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I)_{I\in\mathcal{I}}$ each $\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_{\{i\}}$ shall be isomorphic to L for each $\{i\}\in\mathcal{I}$, $V\supset L$ has to be assumed. Therefore, in general we assume for systems $(\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I)_{I\in\mathcal{I}}$ that $V\supset L\neq\emptyset$. Note that $\tau^I_{I'}$ are generalisations of τ^I_n and Θ^I, because

$$\tau^I_n=\tau^I_{\{n\}}\text{ and }\Theta^I=\tau^I_I=\tau^I_N \tag{5}$$

for each $I\subset N$ and $n\in N$.

Further requirements, which assure that $(\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I)_{I\in\mathcal{I}}$ are well-behaved scalable systems, will now be given with respect to $\mathcal{E}_{\mathcal{I}}$, $\mathcal{B}_{\mathcal{I}}$, L and V. Assuming $T=N$ and $\varepsilon\in V$ the scalability property is assured by the following technical requirements for $\mathcal{E}_{\mathcal{I}}$ and $\mathcal{B}_{\mathcal{I}}$:

Theorem 3 (construction condition for scalable systems)**.** *Let $\mathcal{I}$ be a parameter structure based on N, $\mathcal{E}_{\mathcal{I}}=(E(n,I))_{(n,I)\in N\times\mathcal{I}}$ be an influence structure for $\mathcal{I}$, and let $\mathcal{B}_{\mathcal{I}}=(\mathcal{B}(I,I'))_{(I,I')\in\mathcal{I}\times\mathcal{I}}$ be an isomorphism structure for $\mathcal{I}$. Let $\varepsilon\in V\subset\Sigma^*$, for each $I\in\mathcal{I}$ and $n\in N$ let $E(n,I)=\emptyset$, or it exists an $i_n\in I$ with $E(n,I)=E(i_n,I)$, and for each $(I,I')\in\mathcal{I}\times\mathcal{I}, \iota\in\mathcal{B}(I,I')$ and $i\in I$ holds*

$$\iota(E(i,I))=E(\iota(i),I').$$

Let $E(t,I')=E(t,I)\cap I'$ for each $t\in T$ and $I,I'\in\mathcal{I}, I'\subset I$. Then $(\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I)_{I\in\mathcal{I}}$ is a scalable system with respect to $\mathcal{B}_{\mathcal{I}}$ and

$$\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I=\dot{\mathcal{L}}(L)_I\cap\bigcap_{n\in I}(\tau^I_{E(n,I)})^{-1}(V)).$$

Example 7. *Let $\mathcal{I}$ be a parameter structure based on N, and for $I\in\mathcal{I}$ let $\bar{E}(i,I):=I$ for $i\in N$.*

$\bar{\mathcal{E}}_{\mathcal{I}}:=(\bar{E}(i,I))_{(i,I)\in N\times\mathcal{I}}$ *satisfies the assumptions of Theorem 3 for each isomorphism structure $\mathcal{B}_{\mathcal{I}}$.* (6)

It holds $(\Theta^I)^{-1}(V)=(\tau^I_{\bar{E}(i,I)})^{-1}(V)$ for each $i\in N, I\in\mathcal{I}$, and $V\subset\Sigma^*$.
Therefore, $\mathcal{L}(L,\bar{\mathcal{E}}_{\mathcal{I}},V)_I=\dot{\mathcal{L}}(L)_I\cap(\Theta^I)^{-1}(V)$ for $I\in\mathcal{I}$. Especially, $\bar{S}_I=\mathcal{L}(S,\bar{\mathcal{E}}_{\mathcal{I}_1},S)_I$ for each $I\in\mathcal{I}_1$.

 ISBN: 978-1-61208-319-3

Example 8. *For the parameter structure $\mathcal{I}_2$, and for $\mathring{I}\times\hat{I}\in\mathcal{I}_2$ let*

$$E^2((\mathring{n},\hat{n}),\mathring{I}\times\hat{I}) := \begin{cases} \{\mathring{n}\}\times\hat{I} \mid & \mathring{n}\in\mathring{I} \\ \emptyset \mid & \mathring{n}\in\mathbb{N}\setminus\mathring{I} \end{cases} .$$

$$\mathcal{E}^2_{\mathcal{I}_2} := (E^2((\mathring{n},\hat{n}),\mathring{I}\times\hat{I}))_{((\mathring{n},\hat{n}),\mathring{I}\times\hat{I})\in(\mathbb{N}\times\mathbb{N})\times\mathcal{I}_2} \quad (7)$$

satisfies the assumptions of Theorem 3 for the isomorphism structure $\mathcal{B}^2_{\mathcal{I}_2}$.

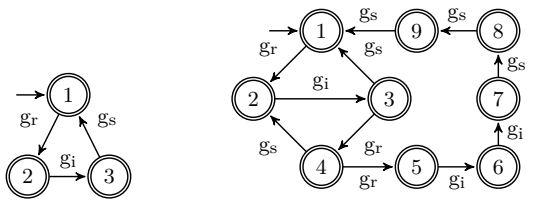
$(\mathcal{L}(S,\mathcal{E}^2_{\mathcal{I}_2},S)_I)_{I\in\mathcal{I}_2}$ is the formalisation of the extended example (several servers) with restricted concurrency.

In order to extend Theorem 3 with respect to self-similarity, an additional assumption is necessary. This is demonstrated by the following counter-example.

Example 9. *Let $G\subset\{\mathrm{g_r},\mathrm{g_i},\mathrm{g_s}\}^*$ the prefix closed language, which is accepted by the automaton Fig. 3(a). Let $H\subset\{\mathrm{g_r},\mathrm{g_i},\mathrm{g_s}\}^*$ the prefix closed language, which is accepted by the automaton in Fig. 3(b). It holds $\emptyset\neq G\subset H$ but $(\mathcal{L}(G,\bar{\mathcal{E}}_{\mathcal{I}_1},H)_I)_{I\in\mathcal{I}_1}$ is not self-similar, e.g.,*

$$\Pi^{\{1,2,3\}}_{\{2,3\}}(\mathcal{L}(G,\bar{\mathcal{E}}_{\mathcal{I}_1},H)_{\{1,2,3\}}) \neq (\mathcal{L}(G,\bar{\mathcal{E}}_{\mathcal{I}_1},H)_{\{2,3\}}$$

because $\mathrm{g_r}\mathrm{g_i}_1\mathrm{g_r}_2\mathrm{g_r}_3\in\mathcal{L}(G,\bar{\mathcal{E}}_{\mathcal{I}_1},H)_{\{1,2,3\}}$, and hence $\mathrm{g_r}_2\mathrm{g_r}_3\in\Pi^{\{1,2,3\}}_{\{2,3\}}(\mathcal{L}(G,\bar{\mathcal{E}}_{\mathcal{I}_1},H)_{\{1,2,3\}})$, but $\mathrm{g_r}_2\mathrm{g_r}_3\notin(\mathcal{L}(G,\bar{\mathcal{E}}_{\mathcal{I}_1},H)_{\{2,3\}}$.



(a) Automaton accepting G (b) Automaton accepting H

Figure 3. Counterexample

Definition 12 (closed under shuffle projection)**.** *Let $L,V\subset\Sigma^*$. V is closed under shuffle projection with respect to L, iff*

$$\Pi^{\mathbb{N}}_K[(\bigcap_{n\in\mathbb{N}}(\tau^{\mathbb{N}}_n)^{-1}(L))\cap(\Theta^{\mathbb{N}})^{-1}(V)]\subset(\Theta^{\mathbb{N}})^{-1}(V) \quad (8)$$

for each subset $\emptyset\neq K\subset\mathbb{N}$. We abbreviate this by $\mathrm{SP}(L,V)$.

Remark 1. *It can be shown that in $\mathrm{SP}(L,V)$ $\mathbb{N}$ can be replaced by each countable infinite set.*

Remark 2. *If L and V are prefix closed with $\emptyset\neq L\subset V$, then it is easy to show that $\mathrm{SP}(L,V)$ follows from self-similarity of $(\mathcal{L}(L,\bar{\mathcal{E}}_{\mathcal{I}_1},V)_I)_{I\in\mathcal{I}_1}$.*

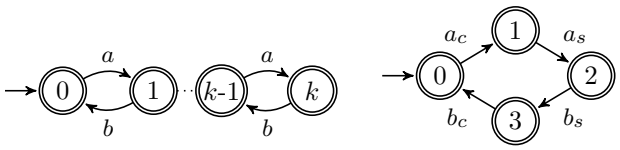
With Definition 12 we are now able to formulate our main result for constructing well-behaved scalable systems defined by a single synchronisation condition.

Theorem 4 (construction condition for well-behaved scalable systems)**.** *By the assumptions of Theorem 3 together with $\mathrm{SP}(L,V)$*

$$(\mathcal{L}(L,\mathcal{E}_{\mathcal{I}},V)_I)_{I\in\mathcal{I}}$$

is a well-behaved scalable system.

Example 10. *For $k\in\mathbb{N}$ let the prefix closed language $F_k\subset\{a,b\}^*$ be defined by the automaton in Fig. 4(a).*



(a) Automaton for $F_k\subset\{a,b\}^*$ (b) One client, one server

Figure 4. Automata at different abstraction levels

With respect to Example 1, $F_1=S$ holds. It can be shown that $\mathrm{SP}(S,F_k)$ holds for each $k\in\mathbb{N}$. With Theorem 4 now, by (6) and (7) especially, the systems $(\mathcal{L}(L,\bar{\mathcal{E}}_{\mathcal{I}_1},F_k)_I)_{I\in\mathcal{I}_1}$ and $(\mathcal{L}(L,\mathcal{E}^2_{\mathcal{I}_2},F_k)_I)_{I\in\mathcal{I}_2}$ are uniformly monotonic parameterised and self-similar. These are the two cases of the guiding example where the concurrency of the execution of requests is bounded by k.

Theorem 4 is the main result for constructing well-behaved scalable systems defined by a single synchronisation condition. The following section shows how this result together with the Intersection Theorem can be used for constructing more complex well-behaved scalable systems defined by the combination of several synchronisation conditions, as for example well-behaved scalable systems consisting of several component types.

V. Well-behaved Scalable Systems Generated by a Family of Influence Structures

Up to now, the examples were considered at an abstraction level, which takes into account only the actions of the server (or the servers, depending on the choice of the parameter structure).

Example 11. *For a finer abstraction level, which additionally takes into account the actions of the clients, a finer alphabet, e.g., $\check{\Sigma}=\{a_c,b_c,a_s,b_s\}$ and a prefix closed language $\check{S}\subset\check{\Sigma}^*$ is needed, which, e.g., is defined by the automaton in Fig. 4(b).*

In general, a finer relation for system specifications at different abstraction levels can be defined by alphabetic language homomorphisms.

Definition 13 (abstractions)**.** *In general, let $\check{L}\subset\check{\Sigma}^*$ and $L\subset\Sigma^*$ be prefix closed languages. We call $\check{L}$* finer than *L or L* coarser than *$\check{L}$ iff an alphabetic homomorphism $\nu:\check{\Sigma}^*\to\Sigma^*$ exists with $\nu(\check{L})=L$.*

For each parameter structure $\mathcal{I}$ and $I \in \mathcal{I}$ ν defines an homomorphism $\nu^I : \check{\Sigma}_I^* \to \Sigma_I^*$ by $\nu^I(a_i) := (\nu(a))_i$ for $a \in \check{\Sigma}$ and $i \in I$, where $(\varepsilon)_i := \varepsilon$.

Let now $\mathcal{E}_\mathcal{I}$ be an influence structure for $\mathcal{I}$ indexed by N which is the base of $\mathcal{I}$, and let $\emptyset \neq L \subset V \subset \Sigma^*$ be prefix closed. $(\mathcal{L}(L,\mathcal{E}_\mathcal{I},V)_I)_{I\in\mathcal{I}}$ induces a restriction of the concurrency in $(\dot{\mathcal{L}}(\check{L})_I)_I$ by the intersections

$$\dot{\mathcal{L}}(\check{L})_I \cap (\nu^I)^{-1}[\bigcap_{t\in N} (\tau^I_{E(t,I)})^{-1}(V)] \text{ for each } I \in \mathcal{I}. \quad (9)$$

If $\check{\tau}^I_{I'} : \check{\Sigma}_I^* \to \check{\Sigma}^*$ is defined analogously to $\tau^I_{I'}$ for $I, I' \subset N$ by

$$\check{\tau}^I_{I'}(a_i) = \begin{cases} a \mid & a \in \check{\Sigma} \text{ and } i \in I \cap I' \\ \varepsilon \mid & a \in \check{\Sigma} \text{ and } i \in I \setminus I' \end{cases},$$

then holds $\tau^I_{I'} \circ \nu^I = \nu \circ \check{\tau}^I_{I'}$. From this it follows that

$$(\nu^I)^{-1}[\bigcap_{t\in N} (\tau^I_{E(t,I)})^{-1}(V)] = \bigcap_{t\in N} (\check{\tau}^I_{E(t,I)})^{-1}(\nu^{-1}(V))$$

and therewith

$$\dot{\mathcal{L}}(\check{L})_I \cap (\nu^I)^{-1}[\bigcap_{t\in N} (\tau^I_{E(t,I)})^{-1}(V)] = \mathcal{L}(\check{L},\mathcal{E}_\mathcal{I},\nu^{-1}(V))_I \quad (10)$$

for each $I \in \mathcal{I}$. Notice that $\emptyset \neq \check{L} \subset \nu^{-1}(V) \subset \check{\Sigma}^*$ is prefix closed. So if $(\mathcal{L}(L,\mathcal{E}_\mathcal{I},V)_I)_{I\in\mathcal{I}}$ fulfils the assumptions of Theorem 3, then this holds for $(\mathcal{L}(\check{L},\mathcal{E}_\mathcal{I},\nu^{-1}(V))_I)_{I\in\mathcal{I}}$ as well and the system

$$(\dot{\mathcal{L}}(\check{L})_I \cap (\nu^I)^{-1}[\bigcap_{t\in N} (\tau^I_{E(t,I)})^{-1}(V)])_{I\in\mathcal{I}}, \quad (11)$$

which is defined by the intersections (9), is a scalable system. The following general theorem can be used to prove self-similarity of such systems.

Theorem 5 (inverse abstraction theorem)**.** *Let $\varphi : \Sigma^* \to \Phi^*$ be an alphabetic homomorphism and $W, X \subset \Phi^*$, then*

$$\mathrm{SP}(W,X) \textit{ implies } \mathrm{SP}(\varphi^{-1}(W),\varphi^{-1}(X)).$$

Generally, by (8), $\mathrm{SP}(\nu^{-1}(L),\nu^{-1}(V))$ implies $\mathrm{SP}(X,\nu^{-1}(V))$ for each $X \subset \nu^{-1}(L)$. Especially $\mathrm{SP}(\check{L},\nu^{-1}(V))$ is implied by $\mathrm{SP}(L,V)$ on account of Theorem 5. So, by Theorem 5, if $(\mathcal{L}(L,\mathcal{E}_\mathcal{I},V)_I)_{I\in\mathcal{I}}$ fulfils the assumptions of Theorem 4, then

$$(\mathcal{L}(\check{L},\mathcal{E}_\mathcal{I},\nu^{-1}(V))_I)_{I\in\mathcal{I}} = (\dot{\mathcal{L}}(\check{L})_I \cap (\nu^I)^{-1}[\bigcap_{t\in N} (\tau^I_{E(t,I)})^{-1}(V)])_{I\in\mathcal{I}} \quad (12)$$

is a well-behaved scalable system.

The intersections in (9) formalise restriction of concurrency in $(\dot{\mathcal{L}}(\check{L})_I)_{I\in\mathcal{I}}$ under *one specific aspect* (one specific synchronisation condition), which is given by ν, $\mathcal{E}_\mathcal{I}$, and V. Restriction of concurrency under *several aspects* (several synchronisation conditions) is formalised by the intersections

$$\dot{\mathcal{L}}(\check{L})_I \cap \bigcap_{r\in R} (\nu^I_r)^{-1}[\bigcap_{t\in N} (\tau^I_{E_r(t,I)})^{-1}(V_r)]$$

for each $I \in \mathcal{I}$ based on N, $R \neq \emptyset$ is the index set of the aspects. The family of aspects restricting concurrency is given by

- a family $(\nu_r)_{r\in R}$ of *alphabetic homomorphisms* $\nu_r : \check{\Sigma}^* \to \Sigma^{(r)*}$ for $r \in R$,
- a family $(\mathcal{E}^r_\mathcal{I})_{r\in R}$ of *influence structures* $\mathcal{E}^r_\mathcal{I} = (E_r(t,I))_{(t,I)\in N\times\mathcal{I}}$ indexed by N for $r \in R$, and
- a family $(V_r)_{r\in R}$ of *influence behaviours* $V_r \subset \Sigma^{(r)*}$ for $r \in R$.

From (10) it follows now

$$\dot{\mathcal{L}}(\check{L})_I \cap \bigcap_{r\in R} (\nu^I_r)^{-1}[\bigcap_{t\in N} (\tau^I_{E_r(t,I)})^{-1}(V_r)] = \bigcap_{r\in R} \mathcal{L}(\check{L},\mathcal{E}^r_\mathcal{I},\nu_r^{-1}(V_r))_I$$

for each $I \in \mathcal{I}$. Because of the intersection theorem, the uniform monotonic parameterisation and self-similarity of the system

$$(\dot{\mathcal{L}}(\check{L})_I \cap \bigcap_{r\in R} (\nu^I_r)^{-1}[\bigcap_{t\in N} (\tau^I_{E_r(t,I)})^{-1}(V_r)])_{I\in\mathcal{I}} \quad (13)$$

can be inferred from respective properties of the systems

$$(\mathcal{L}(\check{L},\mathcal{E}^r_\mathcal{I},\nu_r^{-1}(V_r))_I)_{I\in\mathcal{I}} \text{ for each } r \in R.$$

Using (11) and (12), this requires the verification of the assumptions of Theorem 4 for

$$(\mathcal{L}(\nu_r(\check{L}),\mathcal{E}^r_\mathcal{I},V_r)_I)_{I\in\mathcal{I}}$$

for each $r \in R$. If $\mathcal{I}$ is based on $N = \bigtimes_{k\in K} N_k$, where K is a finite set and each N_k is countable, then along the lines of $\mathcal{I}_2$, a parameter structure $\mathcal{I}_K$ can be defined for this domain. Such $\mathcal{I}_K$ fit for systems consisting of finitely many component types. Each subset $K' \subset K$ with $\emptyset \neq K' \neq K$ defines a bijection between N and $(\bigtimes_{k\in K'} N_k) \times (\bigtimes_{k\in K\setminus K'} N_k)$. By this bijection, for each of these K' an influence structure $\mathcal{E}^{K'}_{\mathcal{I}_K}$ is defined like $\mathcal{E}^2_{\mathcal{I}_2}$ that satisfies the assumptions of Theorem 3 with respect to an isomorphism structure $\mathcal{B}^K_{\mathcal{I}_K}$ defined like $\mathcal{E}^2_{\mathcal{I}_2}$.

VI. Conclusions and Further Work

This paper presented a formal framework to construct well-behaved scalable systems. The basic parts of that framework are formalisations of parameter structures, influence structures and isomorphisms structures. Together with so-called prototype systems and behaviours of influence these structures formally define scalable systems, if certain conditions are fulfilled. Scalable systems

 ISBN: 978-1-61208-319-3

are called well-behaved, iff their behaviour is self-similar. A sufficient condition for such self-similarity is given in terms of prototype systems and behaviours of influence. A deeper analysis of this condition is subject of a forthcoming paper of the authors. One of our motivations for the formal definition of well-behaved scalable systems was the verification of behaviour properties. Usually, behaviour properties of systems are divided into two classes: *safety* and *liveness* properties [23]. Intuitively, a safety property stipulates that "something bad does not happen" and a liveness property stipulates that "something good eventually happens". In [5], it is shown, that for well-behaved scalable systems a wide class of safety properties can be verified by finite state methods. To extend this verification approach to reliability or general liveness properties, additional assumptions for well-behaved scalable systems have to be established. In [24], such assumptions have been developed for uniformly parametrised two-sided cooperations. To generalise these ideas to a wider class of well-behaved scalable systems is subject of further work.

Acknowledgement

Roland Rieke developed the work presented here in the context of the projects MASSIF (ID 257475) being co-funded by the European Commission within FP7 and the project ACCEPT (ID 01BY1206D) being funded by the German Federal Ministry of Education and Research.

References

[1] A. B. Bondi, "Characteristics of scalability and their impact on performance," in Workshop on Software and Performance, 2000, pp. 195–203.

[2] S. Bullock and D. Cliff, "Complexity and emergent behaviour in ICT systems," Hewlett-Packard Labs, Tech. Rep. HP-2004-187, 2004.

[3] J. Weinman, "Axiomatic Cloud Theory," http://www.joeweinman.com/Resources/Joe_Weinman_Axiomatic_Cloud_Theory.pdf, July 2011, [retrieved: Dec, 2013].

[4] P. Zegzhda, D. Zegzhda, and A. Nikolskiy, "Using graph theory for cloud system security modeling," in Computer Network Security, ser. LNCS, I. Kotenko and V. Skormin, Eds. Springer, 2012, vol. 7531, pp. 309–318.

[5] P. Ochsenschläger and R. Rieke, "Security properties of self-similar uniformly parameterised systems of cooperations," in Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, 2011, pp. 640–645.

[6] S. Schneider, "Security Properties and CSP," in IEEE Symposium on Security and Privacy. IEEE Computer Society, 1996, pp. 174–187.

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans. Dependable Sec. Comput., vol. 1, no. 1, 2004, pp. 11–33.

[8] C. N. Ip and D. L. Dill, "Verifying Systems with Replicated Components in Murφ," Formal Methods in System Design, vol. 14, no. 3, 1999, pp. 273–310.

[9] F. Derepas and P. Gastin, "Model checking systems of replicated processes with SPIN," in Proceedings of the 8th International SPIN Workshop on Model Checking Software (SPIN'01), ser. LNCS, M. B. Dwyer, Ed., vol. 2057. Toronto, Canada: Springer, 2001, pp. 235–251.

[10] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre, "Incremental verification by abstraction," in TACAS, ser. Lecture Notes in Computer Science, T. Margaria and W. Yi, Eds., vol. 2031. Springer, 2001, pp. 98–112.

[11] S. Basu and C. R. Ramakrishnan, "Compositional analysis for verification of parameterized systems," Theor. Comput. Sci., vol. 354, no. 2, 2006, pp. 211–229.

[12] R. Milner, Communication and Concurrency, ser. International Series in Computer Science. NY: Prentice Hall, 1989.

[13] J. C. Bradfield, "Introduction to modal and temporal mu-calculi (abstract)," in CONCUR, ser. Lecture Notes in Computer Science, L. Brim, P. Jancar, M. Kretínský, and A. Kucera, Eds., vol. 2421. Springer, 2002, p. 98.

[14] T. E. Uribe, "Combinations of Model Checking and Theorem Proving," in FroCoS '00: Proceedings of the Third International Workshop on Frontiers of Combining Systems. London, UK: Springer, 2000, pp. 151–170.

[15] E. M. Clarke, M. Talupur, and H. Veith, "Environment abstraction for parameterized verification," in VMCAI, ser. Lecture Notes in Computer Science, E. A. Emerson and K. S. Namjoshi, Eds., vol. 3855. Springer, 2006, pp. 126–141.

[16] M. Talupur, "Abstraction techniques for parameterized verification," Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, 2006, CMU-CS-06-169.

[17] K. R. Apt and D. C. Kozen, "Limits for automatic verification of finite-state concurrent systems," Inf. Process. Lett., vol. 22, no. 6, May 1986, pp. 307–309.

[18] I. Suzuki, "Proving properties of a ring of finite-state machines," Inf. Process. Lett., vol. 28, no. 4, Jul. 1988, pp. 213–214.

[19] P. Ochsenschläger and R. Rieke, "Proofs for: Construction Principles for Well-behaved Scalable Systems," http://private.sit.fraunhofer.de/~rol/Proofs-Well-behaved-Scalable-Systems.pdf, 2013, [retrieved: Dec, 2013].

[20] J. Sakarovitch, Elements of Automata Theory. Cambridge University Press, 2009.

[21] K. Falconer, Fractal Geometry: Mathematical Foundations and Applications. Wiley, 2003.

[22] N. Agoulmine, Autonomic Network Management Principles: From Concepts to Applications. Elsevier Science, 2010.

[23] B. Alpern and F. B. Schneider, "Defining liveness," Information Processing Letters, vol. 21, no. 4, October 1985, pp. 181–185.

[24] P. Ochsenschläger and R. Rieke, "Reliability Aspects of Uniformly Parameterised Cooperations," in ICONS 2012, The Seventh International Conference on Systems, Reunion Island. IARIA, 2012, pp. 25–34.

 ISBN: 978-1-61208-319-3