A Toolbox for Functional and Quantitative Analysis of DEDS

Falko Bause, Peter Buchholz and Peter Kemper

Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany

e-mail: {bause,buchholz,kemper}@ls4.informatik.uni-dortmund.de

Abstract

This paper presents a toolbox for the construction of modular tools for functional and quantitative (performance) analysis of discrete event dynamic systems (DEDS). The intention is to simplify the usage of appropriate analysis algorithms, thus supporting the development of appropriate tools.

We describe concepts and contents of the toolbox together with an interface for cooperation of tools. This interface is based on a general interchange format for model description, which covers modelling formalisms, like e.g., a wide class of Queueing Networks and Stochastic Petri Nets. Furthermore currently available tool components are described in detail, where the stress is on state based functional and quantitative analysis techniques for Markovian models.

Keywords: Discrete Event Dynamic Systems, Toolbox, Functional Analysis, Quantitative Analysis, Performance Analysis, Interchange Formats, Abstract Petri Net Notation, Numerical Analysis, Markov Models

1 Introduction

Today a variety of tools exist supporting the functional and quantitative analysis of systems (e.g., [25, 29, 38, 45, 46]). In the industrial area primarily tools have been developed which simulate the system under consideration and support the system analyst by providing convenient graphical user interfaces (GUIs). These GUIs serve as front-ends for the simulator programs and often offer modelling elements which are close to the system description standards. A typical example are tools for the evaluation of flexible manufacturing systems [47].

Also in the scientific area, where more abstract modelling formalisms have been established, several tools with nearly the same features have been developed. E.g., about 27 tools for the analysis of time-augmented Petri Nets are reported in [25, 45]. Most of them offer a GUI for slightly different modelling formalisms and have several analysis algorithms in common. Unfortunately, in most cases, it is difficult to integrate further analysis algorithms or to adapt the offered analysis features to other modelling formalisms. For this purpose, up to now, model descriptions often have to be transformed by hand when using a tool from another research group. Surely it is considered to be an advantage if researchers can share, e.g., their model descriptions and thereby exploit the existing variety of analysis programs and GUIs amongst themselves. Furthermore it is desirable that a researcher can incorporate analysis algorithms from other tools into his or her own tool. Thus it would be possible to offer a reasonable set of analysis features, e.g., for teaching purposes. Also the reimplementation of certain standard algorithms could easily be avoided this way.

In this paper we describe a toolbox which supports a modular construction of tools for functional and quantitative analysis (F&QA tools). An (extendible) interface is presented which allows the description of a variety of modelling formalisms including a wide class of Queueing Networks, Generalized Stochastic Petri Nets, Coloured Petri Nets and combinations of them. Furthermore several functional and quantitative analysis approaches for Markovian models are described. A toolbox which has been developed with a similar intention is described in [43], but concentrates on performance and reliability issues and only implicitly considers functional aspects. In contrast to our approach it does not offer an interchange format for high level modelling formalisms. The PN editor PED described by Heiner [39] also follows a similar direction trying to exploit functional analysis features of different Petri net tools under the roof of a common GUI. This approach can be viewed as an interim step, since it suffers from difficulties when integrating further analysis features (of other tools) due to the lack of a common interface description. Similar to PED also CPN-AMI [20] offers several functional analysis algorithms by exploiting the analysis features of other tools.

The outline of the article is as follows. In Sect. 2 we describe the various tools offered by the toolbox followed by an illustration of the key interface in Sect. 3. Sect. 4 gives some hints on the implementation of GUIs and Sect. 5 is dedicated to analysis issues.

2 Modular Structure

Fig. 1 shows the various components and interfaces of the proposed toolbox. The main components are a GUI and the different analysis programs. All cooperate via a textual interface, called Abstract Petri Net Notation (APNN; [4]), see Sect. 3. This textual interface is complemented by the description of analysis specific parameters, e.g., the definition of quantitative measures like rewards (cf. [43]), and by the description of the result format.

This interface is the basis for the cooperation of all other tool components: the GUI which transforms the user-specified model to the APNN, and the various analysis programs which read the APNN together with further analysis specific parameters and calculate the desired results. Our toolbox provides several analysis modules for functional (qualitative) and quantitative (performance) analysis. The state based analysis tools split up into a module generating an automata description (e.g. one describing the reachability graph (cf. Sect 5.1)), and a part performing core analysis steps (e.g., calculating the steady state distribution by vector/matrix iterations). The mentioned automata description is a further interface which simplifies the incorporation of state based analysis features into our toolbox, since the modules for automata generation can be reused. The analysis programs export results in a specific output format which can be used by the GUI or report generators for result presentation.

Furthermore the toolbox also comprises transformers, which translate other formats into the APNN. E.g., software translating GreatSPN [17] net specifications or PEP [40] model descriptions to the APNN. Furthermore Queueing Networks and different forms of Stochastic Petri Nets can be described in APNN format, cf. Sect. 3. Also the analysis of high-level modelling formalisms has been developed, although not sufficiently implemented: [3] presents a transformation of SDL to Queueing Petri Nets [1], which are covered by the APNN.

3 APNN-Format

This subsection gives an impression on the interchange format, the APNN. The APNN covers several modelling formalisms including Place/Transition Nets, Coloured Petri Nets (CPNs [30]), hierarchical forms of CPNs as proposed in [28], (Generalized) Stochastic Petri Nets [16], Queueing Networks [32], and combinations of these formalisms, like e.g. Hierarchically combined Queueing Petri Nets (HQPNs [2]). HQPNs are a superset of a wide class of Queueing Networks and Stochastic Petri nets. The net of Fig. 2, which comprises all HQPN elements, should serve as an example illustrating the APNN format, which is given in Fig. 3.

A net description is enclosed in a `\beginnet`-`\endnet` environment where the parameter in brackets is the unique identifier of the net. The element `queuedplace` is the specification of a queue with PS scheduling strategy, 2 servers and its colour set contains only the element `job`. The `rank` attribute determines priorities for bulk arrivals and the `weight` attribute specifies the service time distribution by its mean and coefficient of variance. The element `subnetplace` is linked to a subnet description (`obj_25`) via the `substitute` attribute thus supporting hierarchical models. The notion of ports [28] is used to describe the gluing of the subnet with the upper level net, see, e.g., the APNN specification of arcs `obj_7` and `obj_8`.

Before starting a quantitative analysis the measures of interest have to be defined. Generally such measures can be described by rewards, which have

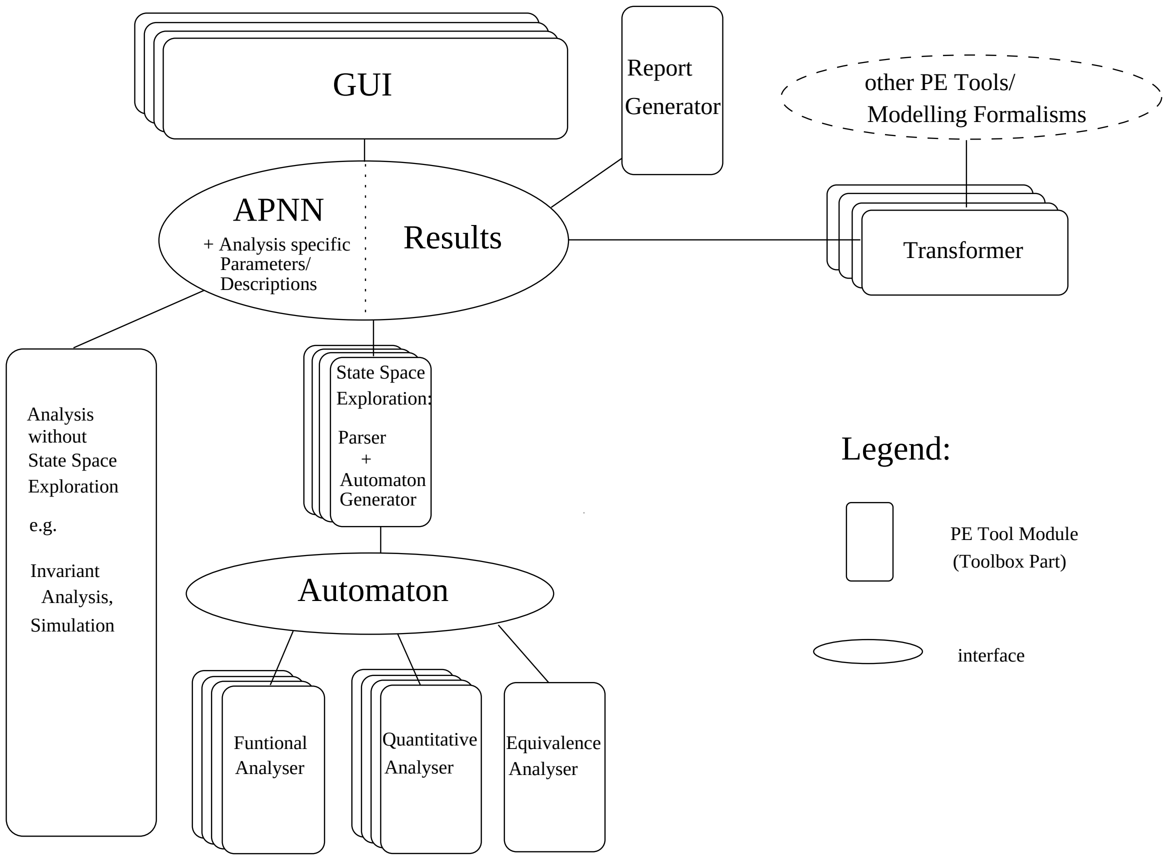

Figure 1: Structural overview of the toolbox

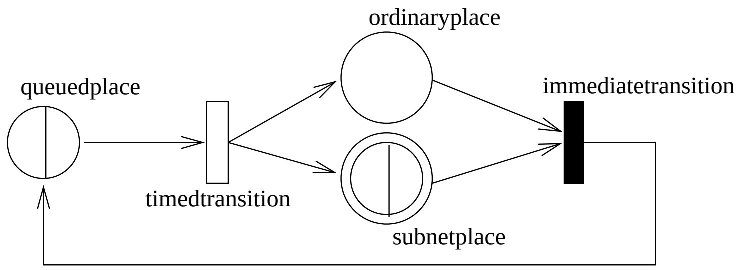

Figure 2: Example net

been inserted into the APNN. E.g., if one wants to calculate the variance of the population of colour `job` in the queue of the queued place, the corresponding reward can be encoded in APNN format as follows:

```
\begin{reward}
  \name{VARIANCE_OF_POPULATION_OF_COLOUR_job_OF_QUEUE_IN_PLACE_obj_1}
  \type{variance}
  \correspond{obj_1}
  \condition{obj_1.queue.job}
  \value{obj_1.queue.job}
\end{reward}
```

where the `correspond`-expression links this reward to the specific model element.

The example illustrates that the APNN covers a wide range of functional and quantitative modelling formalisms. Surely it can be extended to incorporate further related formalisms.

4 Graphical User Interface

The implementation of GUIs is a tedious and often time consuming task, but today an appropriate GUI is a must for every DEDS analysis tool. The usual way of implementation is to build the GUI from scratch using appropriate programming languages and libraries, like Tcl/Tk. We currently try to reduce the effort of GUI implementation testing the following two directions.

One possibility is to extend graphic libraries such that GUIs especially for DEDS modelling can easily be implemented. A student group at our department is currently developing classes for Java implementations.

Another possibility is to develop a GUI generator for DEDS. Here the tool developer benefits from the fact that most GUI operations are standard ones, like e.g. opening/closing model description files, connecting nodes, specifying attributes, and copying subgraphs of the graphical net description. A student group has developed a first version of such a graphic generator supporting the development of GUIs for graph-based modelling formalism. The GUI developer has to specify the modelling elements, their attributes

```
\beginnet{examplenet}

   \place{obj_1}{
      \name{queuedplace}
      \init{2'job}
      \capacity{1000'job}
      \colour{ with  job }
      \queue{timed}
        \sched{PS}
        \noserver{2}
        \rank{case colour of  job => 1}
        \prio{case colour of  job => 1}
        \weight{case colour of  job => (3,1.25)}
   }

   \place{obj_2}{
      \name{subnetplace}
      \init{0'fork1}
      \capacity{1000'fork1}
      \colour{ with  fork1 }
      \substitute{obj_25}
   }

   \place{obj_3}{
      \name{ordinaryplace}
      \init{0'fork2}
      \capacity{1000'fork2}
      \colour{ with  fork2 }
   }

   \transition{obj_4}{
      \name{timedtransition}
      \prio{0}
      \weight{case colour of  forking => 1}
      \guard{ mode = forking }
   }

   \transition{obj_5}{
      \name{immediatetransition}
      \prio{1}
      \weight{case colour of  joining => 1}
      \guard{ mode = joining }
   }

   \arc{obj_6}{
      \from{obj_1}
      \to{obj_4}
      \weight{case mode of forking => 1`job}
   }

   \arc{obj_7}{
      \from{obj_4}
      \to{obj_2}
      \weight{case mode of forking => 1`fork1}
      \bind{obj_4}\cont{obj_4_obj_25_port_in}\with{obj_2}
   }

   \arc{obj_8}{
      \from{obj_2}
      \to{obj_5}
      \weight{case mode of joining => 1`fork1}
      \bind{obj_2}\with{obj_5}\cont{obj_5_obj_25_port_out}
   }

   \arc{obj_9}{
      \from{obj_4}
      \to{obj_3}
      \weight{case mode of forking => 1`fork2}
   }

   \arc{obj_10}{
      \from{obj_3}
      \to{obj_5}
      \weight{case mode of joining => 1`fork2}
   }

   \arc{obj_12}{
      \from{obj_5}
      \to{obj_1}
      \weight{case mode of joining => 1`job}
   }

\endnet

\beginnet{obj_25}
   ...  //specification of subnet
\endnet
```

Figure 3: APNN of example net

and interconnection restrictions (e.g. for bipartite graphs) to create a core GUI. Only few extensions, like, e.g., functions for exporting the APNN have to be specified in C code.

The development of both possibilities is currently in progress and we hope to be able to evaluate in the near future which approach will better facilitate the development of application-oriented GUIs.

5 Analysis

Analysis of DEDS requires a formal semantics of the employed modeling formalisms, this is typically defined in terms of the underlying state space. Let $\mathcal{S}$ denote the state space of a DEDS and n its cardinality. A large value of n with respect to the model description is referred to as 'state space explosion'. At this level, a DEDS is an automaton and it is represented as an edge- and node-labelled directed multi-graph, where nodes correspond to states and edges to state transitions. Labels are assigned with respect to the goal of analysis. We distinguish functional analysis from quantitative analysis. The former basically asks for correctness, i.e. logical behaviour, while the latter considers model behaviour with respect to time.

5.1 Functional Analysis

For functional analysis one focuses on logical, causal dependencies, timing aspects are usually neglected. For PNs, invariant analysis is a well known technique at net level which is orthogonal to state based analysis. It does not suffer from the state space explosion for the price of weaker results. Our toolbox contains a tool for invariant computation like most PN analysis tools. The resulting set of invariants is stored in a format which allows reuse of these results for other analysis tools, e.g. place invariants are used for the computation of place capacities in SupGSPN, described in Sect. 5.2.2.

Certain questions arise frequently in functional analysis, e.g. to decide existence of a home space, liveness and boundedness of a given PN. For this problem, a specific state based algorithm is implemented as well. The notion of home space is closely related to ergodicity for Markov chain analysis, such that functional analysis of a given model preceding any quantitative analysis is recommended [1].

For more general properties, the state based functional analysis results in proving presence or absence of certain nodes or paths in $\mathcal{S}$. The relevant information at the level of $\mathcal{S}$ is a characterisation of the state at nodes and the cause/reason for a state transition at edges. The property of interest is specified by temporal logic formulas. Among the many temporal logics considered in literature, the computational tree logic (CTL) is a well known example of a branching time logic, which is supported by many model checking tools. Our toolbox contains a model checker for CTL formulas as well [35], which profits from modular or hierarchically structured state spaces like analysis tools for quantitative analysis as described in Sect. 5.2. Equivalences [21] are frequently employed in functional analysis to relief the state space explosion problem. The available model checker supports bisimulation to reduce $\mathcal{S}$. Equivalences are considered to some extend for quantitative analysis as well, see Sect. 5.2.2.

5.2 Quantitative Analysis

Performance characteristics presuppose a notion of time, such that a DEDS requires appropriate timing information, which enhances the representation at state space level by information about the duration of activities. Here we focus on models with Markovian timing. In case of HQPNs, rates of exponential distributions are assigned to transitions in $\mathcal{S}$. If immediate transitions are possible, the resulting Semi-Markov chain is transformed into a CTMC by the common elimination of vanishing states [13]. The resulting state space contains only tangible states and transitions are labelled with rates multiplied with the probability of subsequent immediate activities. Additionally rewards are assigned to states and transitions.

Such models can be analysed by simulation or with numerical analysis techniques, which are discussed in the following.

5.2.1 Simulation

Simulation is the most general analysis method which, theoretically, can be applied for any DEDS model, Markovian or not, structured or not, finite-state or infinite-state. However, simulation also has the drawback of statistical results which require long simulation runs to get small confidence intervals. This especially holds for detailed measures which are based on states with a small probability or rare events. Thus simulation should only be used in cases where state based analysis, as introduced below, cannot be applied at all or requires too much effort. Here we only outline specific points for the integration of simulators in a toolbox. In particular, we outline some new concepts related to the analysis of modular or hierarchically structured systems.

Since the APNN has a well formalised syntax and semantics in terms of the dynamic behaviour of nets, it is straightforward to implement a simulator for models described via the APNN. Currently we have a prototype implementation in C/C++. The advantage of using the APNN as a simulator interface is that a single simulation engine can be applied for a wide variety of modeling formalisms, namely all formalisms for which a mapping on the APNN description can be defined (cf. Sect. 3).

A practical problem of simulative analysis is the adequate representation of results. For point estimates of typical measures like mean populations or throughputs, the problem is, of course, identical to the representation of results computed with any other analysis technique. However, simulation allows us to animate the dynamics of a system, which makes a realisation much more complex. Since the simulator is only aware of the APNN description of a model, dynamic behaviour can only be described at this level. Our current approach is to define a APNN description for traces. The simulator writes a trace file describing the dynamics of those elements for which a trace has been requested (e.g., the firing of a transition, change of the marking of a place). The trace file has to be interpreted on-line or off-line by the GUI to animate the dynamics.

Finally we consider simulative analysis of structured models. The state based analysis techniques described below (cf. Sect. 5.2.2) support the structure of a model for analysis purposes. We support two different kinds of structure, namely synchronously interacting automata and hierarchical models. In both cases, a model consists of communicating automata and this structure can be exploited in a simulator. Object oriented concepts are exploited by implementing automata as objects in the simulator and by defining interfaces via exported methods. A first C++ realisation of an object oriented simulator supporting HQPNs shows that the approach is a convenient way of realising parallel simulations [31]. The idea is to associate a logical simulator process to each object and to implement communication via message passing or remote procedure calls. Additionally, the object structure allows us to combine aggregation and simulation by representing some components by less complex aggregates (see Sect. 5.2.2).

5.2.2 Numerical analysis

CTMCs are a popular model type which is commonly used for quantitative analysis. CTMC based analysis is performed in two steps. First, the model, in our case specified via the APNN, has to be transformed into a CTMC. Afterwards, the stationary or transient distribution of the CTMC is computed by solving a set of linear or differential equations. From the stationary or transient distribution performance measures can be derived. Practical problems of the approach arise due to the well known state-space explosion. Thus, highly sophisticated solution methods are required.

In this section, we consider different approaches to analyse large but finite CTMCs numerically. In particular, we consider recently developed structured analysis approaches which are an essential part of our toolbox. We start with a brief introduction of conventional numerical analysis techniques to solve the set of balance equations of a CTMC. Then we present modular structured CTMCs which result from stochastic automata models communicating via synchronised transitions. Afterwards, hierarchical descriptions of CTMCs are introduced. Although modular and hierarchical descriptions have been developed independently during the last decade, it can be shown that the hierarchical approach is in fact a generalisation of the modular idea [7].

At the end of this section we consider briefly performance equivalence allowing exact aggregation and approximate analysis based on decomposition and aggregation.

Conventional Markovian Analysis The term conventional analysis is used for the established numerical analysis approach for CTMCs. The tool implementing conventional analysis first reads the APNN description and generates state space, generator matrix and reward vectors. For large state spaces we recommend to try to structure the model in a modular or hierarchical way and use this structure for state space generation (cf. the following paragraphs), which becomes more efficient if the number of states exceeds about $100,000$ states.

After the generation phase, the CTMC is described by the generator matrix, a set of reward vectors and possibly an initial vector (see below). Generator matrices are interpreted as a special case of an automata description and may have a block structure. An automaton is specified by a number of states and a number of matrices to describe transitions of different types. All these parts are generated and stored in files using a standardised format.

The block structure of a generator matrix results from conditions defined for the model. States are grouped in the same block, if they observe the same conditions. Conditions depend on the used modeling formalism. In PNs, the token population of some places or sets of places is used to structure the state space. The block structure of a generator matrix is exploited in solution techniques using aggregation/disaggregation steps. Ideally blocks should be built in a way that transition rates between states in different blocks are small compared with transition rates between states in the same block. Such a structure need not exist for arbitrary models.

Let $\mathbf{Q}$ be the generator matrix of some finite CTMC. If $\mathbf{Q}$ is irreducible, then $\mathbf{pQ} = \mathbf{0}$ subject to $\mathbf{pe}^T = 1.0$ has to be solved to obtain the stationary solution vector $\mathbf{p}$. A large number of solution methods exist to obtain $\mathbf{p}$ (see [48] for an excellent overview). Because matrix $\mathbf{Q}$ is usually large and sparse, iterative methods are commonly employed. Depending on the size and structure of $\mathbf{Q}$ and the required accuracy, available methods show a different behaviour. Up to now a "black box" algorithm to choose an optimal or nearly optimal method for a given matrix $\mathbf{Q}$ is not known. In consequence, a toolbox has to include several algorithms. The current implementation contains more than 10 algorithms including point iteration methods like JOR or SOR, projection techniques like GMRES or Arnoldi, aggregation/disaggregation methods like KMS (see [48] for details) and the multi-level algorithm [27]. All methods have been implemented in C. The integration of different methods in a toolbox with a well defined interface has several advantages:

- New methods can be easily integrated, because data structures and basic routines are available and can be reused by sharing source code.
- Methods become comparable, because they all use the same basic routines allowing a fair comparison of time and space requirements due to the method and not due the programming language or style.
- The solution methods are independent of the modeling formalism, since they only use an automata/a matrix format.
- The solution vector of one method can be used as initial vector for another method to improve accuracy.

Analogous argumentation applies for transient analysis, although the number of techniques used for transient analysis is smaller. In our toolbox we implemented the randomization approach which has several times shown to be an efficient and stable method for transient CTMC analysis [44]. After the stationary or transient solution vector has been computed, result measures are generated by combining reward and solution vector.

Analysis of Modularly Structured Markov Chains This kind of Markov chains results from a set of N stochastic automata. An automaton i for $0 < i \leq N$ has a state space $\mathcal{S}_i$ with cardinality n_i and transitions which can either be local, such that they can occur asynchronously and independently from the state of other automata, or synchronised, such that they can occur only in conjunction with the corresponding transition of other automata. This can be interpreted as synchronous interaction (rendezvous), and is observed in modeling formalisms like stochastic process algebras [26, 24], stochastic automata networks [41, 42] and superposed GSPNs [22, 23]. The state space of the overall system is then contained in the cross product over all automata state spaces $\mathcal{S}_i$.

The primary problem of conventional numerical analysis is the space required to store $\mathbf{Q}$. This problem is solved in modular representations by implicitly representing $\mathbf{Q}$ in a Kronecker term which consists of a set of smaller matrices $\mathbf{Q}_t^i$. A matrix $\mathbf{Q}_t^i$ contains only transitions t of an automaton i and has dimension $n_i \times n_i$. This matrix representation is called structured representation. The reduction of space requirements is drastic since a sum of small matrices is stored to represent a generator matrix in the dimension of the cross product of $\mathcal{S}_i$.

Hence a space-time tradeoff is observed, when iterative solution methods are applied with a structured representation. The crucial step in iterative solution methods is a matrix vector multiplication involving $\mathbf{Q}$. Due to the implicit representation, nonzero matrix entries of $\mathbf{Q}$ are computed from the structured representation to perform the required multiplication. Originally proposed advantages [48] in multiplication algorithms with structured representations are only observed for nonsparse matrices [12]. Nevertheless, structured representations increase the size of solvable CTMCs by about an order of magnitude compared to conventional methods and space for iteration vectors becomes the main bottleneck.

Structured representations implicitly consider the cross product of $\mathcal{S}_i$ as the state space of the overall system. Due to the restricting effects of synchronisation this often yields many unreachable states. Unreachable states can be recognized by a state space exploration based on a structured representation [34], and an additional mapping (TRS-permutation in [33]) allows to avoid the consideration of unreachable states for iterative solution methods. However this mapping implies additional space and time requirements. In [12] various algorithms are discussed which reduce this overhead by a space efficient tree type data structure to a factor $\sum_i log(n_i)$ in time. This data structure can be further minimised in space [35] without changing the time for a matrix vector multiplication. Note that the amount of unreachable states is only relevant in the initial state space exploration phase. Once the numerical iterative solution is applied, the ratio between the state space and $\times\mathcal{S}_i$ is of minor importance.

Algorithms based on modularly structured CTMCs have been integrated in SupGSPN, a numerical analyser, which consists of a state space exploration module and a variety of iterative solution methods for modular structured Markov chains for steady and transient solution. The set of solution algorithms includes Gauss-Seidel and the method of Jacobi with or without overrelaxation and randomization in several variations. If immediate events/transitions are allowed, as e.g. for GSPNs, we support only models where synchronised transitions are timed, however a solution with immediate synchronised transition is feasible for the price of an enlarged state space as well, as Ciardo and Tilgner demonstrate in [19].

Analysis of Hierarchically Structured Markov Chains The idea of hierarchically structured CTMCs is an extension of the modular structure. We will first introduce the basic ideas by a simple example. Afterwards, we briefly describe the general approach and its integration in the F&QA tool.

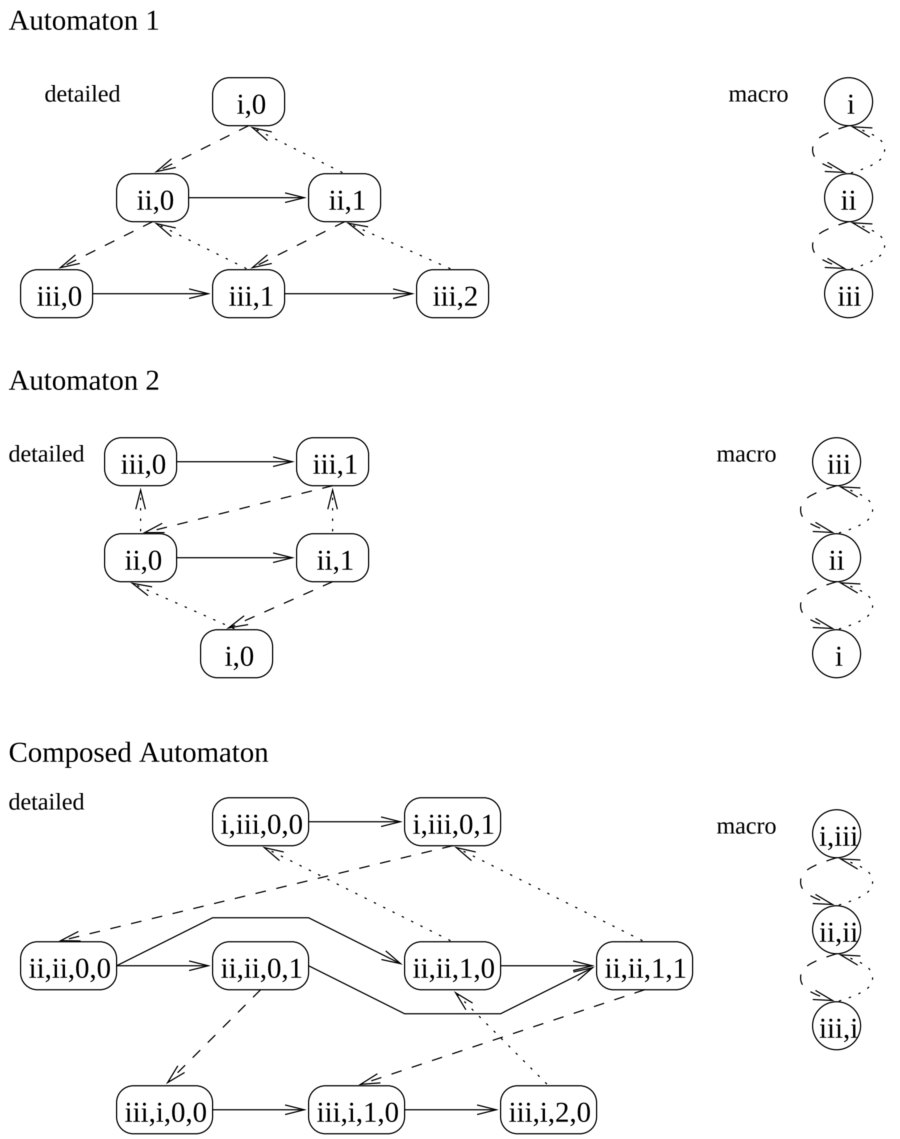

Figure 4: Simple example for the hierarchical approach.

The hierarchical approach starts, like the modular approach, with a set of automata numbered 1 through N. In contrast to the modular composition, we now assume that the state spaces $\mathcal{S}_i$ of the automata are partioned into sets of macro states. For each $\mathcal{S}_i$ we have m_i $(0 < m_i \leq n_i)$ macro states. Each state $s \in \mathcal{S}_i$ belongs to exactly one macro state a. $\mathcal{S}_i[a]$ is the set of states belonging to macro state a. All sets $\mathcal{S}_i[a]$ are non-empty and, since the macro states define a partition of $\mathcal{S}_i$, we have $\mathcal{S}_i = \cup_{a=1}^{m_i}\mathcal{S}_i[a]$. Fig. 4 shows a simple example of two automata with 3 macro states each. Local transitions are denoted as solid arcs. Additionally, two synchronised transitions are defined which are represented by dashed and dotted arcs, respectively. For notational convenience, we use roman numbers for macro states and number macro states from 1 to m_i. Furthermore, we number detailed states inside each macro state from 0 through $n_i^a - 1$ for macro state a. Consequently, the relation $n_i = \sum_{a=1}^{m_i} n_i^a$ holds. Each state from $\mathcal{S}_i$ is characterised by the number of the macro state and the number of the micro state inside the macro state. By choosing m_i equal to 1 we obtain a unstructured description of $\mathcal{S}_i$. From the partition into a macro states, a macro transition system can be defined. The macro transition system contains for each macro state a state and a transition $a - t \rightarrow b$ if $r - t \rightarrow s$ for some $r \in \mathcal{S}_i[a]$, $s \in \mathcal{S}_i[b]$. Local transitions starting and ending in the same macro state need not be considered in the macro transition system. The macro transition systems for the automata in the example can be seen on the right side of Fig. 4. We assume that the initial state of automaton 1 is $(i, 0)$ and the initial state of automaton 2 is $(iii, 0)$.

Macro states can be easily integrated in the matrices describing an automaton. After macro states have been defined, states are reordered such that states belonging to the same macro state follow consecutively. Then a block structure can be defined on the automata matrices according to the macro states (i.e., states 0 through $n_i^1 - 1$ belong the first block, states n_i^1 through $n_i^1 + n_i^2 - 1$ belong to the second block and so on).

Afterwards a macro transition system for the global system is generated, which is shown for the example at the bottom of Fig. 4. The macro transition system results from the composition of macro transition systems for all automata. Since each macro transition system of an automaton is also an automaton, the composition of macro transition systems is well defined. A state in the macro transition system is characterised by a vector $\mathbf{a}$, where $\mathbf{a}(i)$ is the macro state of automaton i. Usually not all combinations of macro states are reachable, which implies that the corresponding detailed states in the global system are also unreachable.

Now we can proceed similarly to the modular approach. The set of detailed states belonging to a global reachable macro state $\mathbf{a}$ is defined as

$$\mathcal{S}[\mathbf{a}] = \times_{i=1}^{N}\mathcal{S}_i[\mathbf{a}(i)] \ .$$

Generator matrix $\mathbf{Q}$ is decomposed into submatrices

$$\mathbf{Q}[\mathbf{a}, \mathbf{b}] = \sum_{t \in T} \mathbf{Q}_t[\mathbf{a}, \mathbf{b}] \ ,$$

where $\mathbf{Q}_t[\mathbf{a},\mathbf{b}]$ is a matrix including transitions due to model transition t which start in a state from $\mathcal{S}[\mathbf{a}]$ and end in a state from $\mathcal{S}[\mathbf{b}]$. Each of these matrices can be described as the sum of Kronecker products of small automata matrices $\mathbf{Q}_t^i[a,b]$, where $\mathbf{Q}_t^i[a,b]$ is a matrix containing transitions between states from $\mathcal{S}_i[a]$ and $\mathcal{S}_i[b]$ due to model transition t.

In summary, the hierarchical matrix description is an extension of the modular description, where the ideas of the modular description are used at submatrix instead of matrix level. If the number of macro states is one for each automaton, we obtain exactly the modular description described above. The advantage of the introduction of macro states is that we can reduce the number of unreachable states and avoid overhead due to the handling of unreachable states. Ideally we obtain a hierarchical description without unreachable states. This effect can be nicely seen in the example. The hierarchical description defines exactly the set of reachable states. However, if we use a modular description for the example, then the potential reachability includes 30 states, but only 9 of these states are reachable.

Fortunately, a lot of progress has been made recently in defining appropriate macro structures for automata. The first idea is to define a macro structure at the model level. This can be done easily for closed QN models. In this case, submodels are defined as sets of queues and macro states are defined as the population (vector) of the submodel [5]. Since the population keeps constant in closed QNs, we have an invariant which determines the set of reachable macro states. In fact, the example automata have been derived from a simple single class QN with population 2. The first automaton describes a processor sharing queue with Erlang 2 service time distribution and the second automaton a first come first served queue with Erlang 2 service time distribution. Macro states are defined with respect to the population in the queues. Macro state i describes the empty queue, macro state ii the queue with one customer and macro state iii the queue with two customers.

In a similar way macro states may be defined for PN models, by forcing specific environments. This idea has been realised in the QPN tool [2], where the macro state is defined by the marking of a distinguished place per subnet. Alternative approaches introduce redundant places (semi-) automatically into PNs and define macro states according to the marking at the redundant places [10, 11, 14]. With some exceptions, like simple QNs and marked graph PNs, the approaches cannot assure that the introduction of macro states completely eliminates unreachable states from the modular description. However, experience shows that the number of unreachable states is usually reduced drastically, if an appropriate invariant, like the submodel population in closed QNs, has been found. Additionally, a state level algorithm has been proposed recently [7] which determines for an arbitrary network of composed automata a macro state structure which allows a hierarchical description without unreachable states and which generates a minimal number of macro states. If this algorithm generates too many macro states, then the model or the chosen decomposition is not appropriate for a hierarchical description.

In the hierarchical analyser, we define a macro structure at the net-level for HQPNs by automatically defining an appropriate frame for subnets. Furthermore the algorithm to generate a hierarchical structure for a network of automata has been completely integrated. Both hierarchical descriptions produce input matrices for a tool including algorithms exploiting the Kronecker structure to determine the stationary or transient distribution. Several algorithms have been integrated in the F&QA tool including point iterations methods (Power, JOR, SOR) and projection methods (GMRES, Arnoldi) for stationary analysis. Most methods can be combined with aggregation/disaggregation steps or new forms of preconditioners [8, 9]. The algorithms allow the analysis of CTMCs with more than 5 million states on contemporary workstations with 128 Megabyte of main memory without using secondary memory.

A Note on Performance Equivalence Modular or hierarchical descriptions of CTMCs can be exploited in time and space efficient algorithms for state space generation and iterative numerical analysis. Additionally, the compositional structure allows one to use equivalence relations to reduce component state spaces. Equivalence is a concept that is well established in functional analysis. Starting with the well known language equivalence of finite automata, a large variety of equivalence relations has been proposed. Of particular importance are the different forms of bisimulation equivalence [21]. The largest bisimulation equivalence relation can be computed for an automaton and a reduced automaton is built by substituting each equivalence class by a single state. Since bisimulation equivalence is a congruence according to modular composition, the reduced instead of the original automaton can be used in compositional analysis. Different forms of bisimulation equivalence exist for various functional analysis purposes [21].

More recently the idea of bisimulation equivalence has been extended to stochastic automata [6]. The extension is based on a combination of bisimulation for labeled transition systems and lumpability for CTMCs and is denoted as performance bisimulation. As in the untimed case, performance bisimulation allows the generation of a reduced stochastic automaton which can be used in numerical analysis, either modular or hierarchical. It is especially possible to extend the algorithm for computing bisimulations such that performance bisimulations are computed [7].

Our toolbox contains a tool to generate bisimulations and performance bisimulations. It reads the automata descriptions, computes the required bisimulation, generates the reduced automaton according to the bisimulation and writes the description of the reduced automaton in a file. The reduced automaton can afterwards be used instead of the original one in compositional analysis techniques. The approach is an exact aggregation approach since results are not modified if the right bisimulation has been chosen (e.g., to preserve quantitative results a performance bisimulation is required).

Approximate Analysis Although the computing power of modern workstations increases permanently as well as the development of more sophisticated analysis techniques allows the analysis of much larger state spaces, state based analysis is still restricted to relatively small models. This situation obviously will not change in the future, since state based analysis is often an exponentially growing problem in terms of model complexity. Thus alternative analysis techniques have to be considered. Apart from simulation, approximate analysis based on decomposition and aggregation can be used.

For these solution techniques, the model is decomposed into parts and in each analysis step one part is represented in detail and the remaining parts are described by aggregates. Results of an analysis step determine performance results for the detailed part and are used to built an appropriate aggregate for the detailed part. Usually the approach is iterated. Different realisations of the outlined idea have been published in the literature [15, 18, 36, 37]. In our toolbox, decomposition and aggregation is integrated for all models which are described in a modular and hierarchical way. Thus the technique can be used for a wide class of models.

6 Conclusion

We have presented a toolbox for the combined functional and quantitative analysis of DEDS models on the basis of a common model interchange format, the APNN. This format covers a fair amount of Petri net like formalisms as well as certain Queuing network concepts. At a lower level the toolbox uses a well defined format for communicating stochastic automata, an established format to describe DEDS at a state transition level.

A modular tool design around common interfaces reduces the effort of implementing new modules or tools drastically. Based on the APNN interface, model level algorithms, especially for Petri net analysis, are integrated. At the automata level, state level algorithms for functional and quantitative analysis are implemented. The possibility to reuse software components facilitates the construction of modular tools. This increases the productivity and the stability.

The goal behind the development of the toolbox was twofold. First, research activities should be supported by providing a platform for a fast implementation of new analysis methods. The second goal was the support of classroom work. Small projects can be defined, which concentrate on specific aspects, but the resulting modules can be integrated in the toolbox and then applied as one step in the analysis of complex models. Examples for such student projects are the development of specific GUIs, the implementation of parallel simulators and the realisation of stubborn set methods for Petri net analysis.

Modeling and analysis of real world applications will drive future developments of the toolbox. The set of available algorithms and GUIs will certainly grow. Our main focus is currently the integration of distributed analysis algorithms.

Acknowledgement We like to thank S. Römer from TU Munich for providing a PEP-parser and S. Donatelli from the university of Torino for

contributing a GreatSPN to APNN translator module.

References

[1] F. Bause. Queueing Petri Nets — A formalism for the combined qualitative and quantitative analysis of systems *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse (France), pp. 14–23, 1993.

[2] F. Bause, P. Buchholz, P. Kemper. QPN-tool for the specification and analysis of hierarchically combined queueing Petri nets. *H. Beilner, F. Bause (eds.); Quantitative Evaluation of Computing and Communication Systems*, Springer LNCS 977, pp. 224–238.

[3] F. Bause, H. Kabutz, P. Kemper, P Kritzinger. SDL and Petri net performance analysis of Communicating systems *Proc. of the 15th International Symposium on Protocol Specification, Testing and Verification*, Warsaw (Poland), June 1995.

[4] F. Bause, P. Kemper, P. Kritzinger. Abstract Petri Net Notation. *Petri Net Newsletters*, No. 49, pp. 9–27, 1995.

[5] P. Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems 15 (1994) 59-80.*

[6] P. Buchholz. Equivalence relations for stochastic automata networks; *W. J. Stewart (ed.), Computation with Markov Chains: Proc. of the 2nd Int. Workshop on the Numerical Solution of Markov Chains, Kluwer Int. Publishers (1995) 197-216.*

[7] P. Buchholz. Hierarchical structuring of superposed GSPNs. *Proc. of the 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97), IEEE CS-Press, 1997, 81-90.*

[8] P. Buchholz. An aggregation-disaggregation algorithm for stochastic automata networks. *Probability in the Engineering and Informational Sciences 11 (1997) 229-253.*

[9] P. Buchholz. Structured analysis approaches for large Markov chains. *(submitted for publication).*

[10] P. Buchholz, P. Kemper. Numerical analysis of stochastic marked graph nets. *Proc. of the 6th Int. Workshop on Petri Nets and Performance Models (PNPM'95), IEEE CS-Press, 1995, S. 32-41.*

[11] P. Buchholz, P. Kemper. Hierarchical reachability graph generation for Petri nets; *Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 660, 1997 (submitted for publication).*

[12] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models. Technical report, ICASE Report No. 97-66 NASA/CR-97-206274, 1997. submitted for publication.

[13] A. Blakemore. The cost of eliminating vanishing markings from generalized stochastic Petri nets. In *Proc. of the 3rd Int. Workshop on Petri Nets and Performance Models*, pages 85–92. IEEE Computer Society, 1989.

[14] J. Campos, M. Silva, S. Donatelli. Structured solution of DSSP systems. *Proc. of the 7th Int. Workshop on Petri Nets and Performance Models (PNPM'97), IEEE CS-Press, 1997, 91-100.*

[15] J. Campos, J. M. Colom, H. Jungnitz, M. Silva. A general iterative technique for approximate throughput computation of stochastic marked graphs. *Proc. of the 5th Int. Work. on Petri Nets and Performance Models, IEEE CS-Press (1993) 138-147.*

[16] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: a definition at the net level and its implications. *IEEE Trans. Software Engineering*, 19(2):89–107, Feb 1993.

[17] G. Chiola, G. Franceschinis, R. Gaeta, M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation, special issue on Performance Modeling Tools*, 24(1&2), November 1995, pp. 47–68.

[18] G. Ciardo, K. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation 18 (1994) 3 7-59.*

[19] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Technical report, ICASE report, 1996.

[20] see http://www-src.lip6.fr/cpn-ami

[21] R. Cleaveland, J. Parrow, B. Steffen. The concurrency workbench: a semantics based tool for the verification of concurrent systems. *ACM trans. on Prog. Lang. and Sys. 15 (1993) 36-72.*

[22] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–26, 1993.

[23] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Application and Theory of Petri nets 1994*, Berlin, 1994. Springer.

[24] S. Donatelli, M. Ribaudo, and J. Hillston. A comparison of performance evaluation process algebra and generalized stochastic Petri nets. In *Proc. 6th Int. Workshop Petri Nets and Performance Models*, pages 158–168. IEEE Computer Society Press, 1995.

[25] F. Feldbrugge. Petri Net Tool Overview 1992. in *Advances in Petri Nets*, Springer-Verlag, pp. 169–209, 1993.

[26] J. Hillston. Compositional Markovian modelling using a process algebra. In W.J. Stewart, editor, *Computations with Markov Chains: Proc. 2nd Int. Work. Numerical Solution of Markov Chains*, pages 177–196. Kluwer Academic Publishers, 1995.

[27] G. Horton, S. Leutenegger. A multi-level solution algorithm for steady state Markov chains. *Performance Evaluation Review 22 (1994) 191-200.*

[28] P. Huber, K. Jensen, and R. Shapiro. Hierarchies in coloured Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets*, LNCS 483. Springer, Berlin, pp. 313–341, 1990.

[29] International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. held every 1.5 years (1984 Paris, 1985 Sophia Antipolis, 1987 Paris, 1989 Palma, 1991 Torino, 1992 Edingburgh, 1994 Vienna, 1995 Heidelberg, 1997 St. Malo, 1999 Palma).

[30] K. Jensen. Coloured Petri Nets Vol. 1. *Springer EATCS Monographs on Theoretical Computer Science*, 1992.

[31] T. Jürgens. Verteilte Simulation von HQPNs auf einem Netzwerk von Workstations. *Diplomarbeit, Fachbereich Informatik, Universität Dortmund (1997).*

[32] K. Kant. Introduction to computer system performance evaluation; *Mc Graw Hill*, 1992.

[33] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep 1996.

[34] P. Kemper. Reachability analysis based on structured representations. In *Application and Theory of Petri Nets*, LNCS 1091. Springer, 1996.

[35] P. Kemper and R. Lübeck. Model checking based on Kronecker algebra. *Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 669, 1998.*

[36] Y. Li, C. M. Woodside. Complete decomposition of stochastic Petri nets representing generalized service networks. *IEEE Trans . on Comp. 44 (1995) 577-592.*

[37] V. Mainkar, K. S. Trivedi. Fixed point iteration using stochastic reward nets. *IEEE Trans. on Softw. Eng. 22 (1996).*

[38] K.M. Nichols, D. Dubois, et al. Performance Tools. *IEEE Software*, 7(3), pp. 21–30, May 1990.

[39] PED – A Hierarchical Petri Net Editor, see
http://www-dssz.Informatik.TU-Cottbus.DE/~wwwdssz/ped.html

[40] see http://www.informatik.uni-hildesheim.de/~pep/

[41] B. Plateau and K. Atif. Stochastic automata network for modelling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, 1991.

[42] B. Plateau and J.M. Fourneau. A methodology for solving Markov models of parallel systems. *Journal of Parallel and Distributed Computing*, 12, 1991.

[43] R.A. Sahner, K.S. Trivedi, A. Puliafito. Performance and reliability analysis of computer systems – An example-based approach using the SHARPE software package. *Kluwer Academic Publishers*, 1996.

[44] A. Reibman and K.S. Trivedi. Numerical transient analysis of Markov models *Comput. Opns Res.*, 15:19–36,1988.

[45] see http://www.daimi.aau.dk/PetriNets/tools

[46] see http://www.cs.du.edu/sigmetrics/tools.html

[47] see Simulation Software Survey: http://lionhrtpub.com/orms/surveys/Simulation/Simulation.html

[48] W. J. Stewart. Introduction to the numerical solution of Markov chains. *Princeton University Press (1994).*