Towards Translating FSM-SADF to Timed Automata

Mladen Skelin
Department of Engineering Cybernetics,
Norwegian University of Science and Technology
mladen.skelin@itk.ntnu.no

Erik Ramsgaard Wognsen, Mads Chr. Olesen,
René Rydhof Hansen, Kim Guldstrand Larsen
Department of Computer Science, Aalborg University
{erw,mchro,rrh,kgl}@cs.aau.dk

***Abstract*—Dataflow formalisms play a significant role in the areas of design and analysis of embedded streaming applications. These formalisms can roughly be split into static and dynamic ones. Static dataflow formalisms are highly analyzable, but due to their static nature are not able to capture the dynamism inherent to modern embedded streaming applications. Dynamic dataflow formalisms on the other hand provide a sufficient level of expressiveness to capture the application dynamism at the cost of reduced analyzability. The recently introduced finite state machine-based scenario aware dataflow (FSM-SADF) formalism provides a good trade-off between expressiveness and analyzability. This paper reports on the translation of the FSM-SADF formalism to timed automata (TA). In short, we propose a compositional translation from FSM-SADF to TA that enables computation of some quantitative and qualitative properties of the model not supported by the existing tools, in the UPPAAL model checker. We demonstrate our approach on an MPEG-4 case study which is a typical example of a streaming application from the multi-media domain.**

I. Introduction

Dataflow formalisms are widely used to design and analyze embedded streaming applications running on distributed platforms such as MPSoCs (Multi-Processor System on Chips). In general, dataflow formalisms take the form of a directed graph which consists of *actors* as vertices and *channels* as edges. Actors are computational entities that usually represent application sub-tasks, while channels are communicational entities used to communicate application, control and synchronisation data between actors. In dataflow, an actor *firing* is an indivisible quantum of computation during which an actor consumes a certain number of data values from its input channels and produces a certain number of data values on its output channels. These data values are abstracted into *tokens* and the consumption and production numbers are called *rates*. In timed dataflow formalisms, it takes some time for the actor firing to complete and this time duration is called the actor *firing duration*.

Modern embedded streaming applications exhibit a high level of dynamism. The consequence is that the workload of such applications changes over time. How difficult it is to model such a dynamic application will depend on the particular dataflow formalism used. Not all dataflow formalisms provide us with a sufficient level of expressiveness needed to capture the dynamism of the application. On the other hand, those that do, pay the price in terms of significantly reduced analyzability. A good comparison between expressiveness and analyzability for various dataflow formalisms can be found in [6].

This work is supported by the 7th EU Framework Program under grant agreement 318490 (SENSATION).

The scenario aware dataflow (SADF) formalism [8] models an application as a collection of different behaviours called *scenarios* in which consumption and production rates and the actor firing durations change within one scenario and from one scenario to the other. A stochastic approach is used to model variance in firing durations and scenario ordering. The SADF formalism can therefore model dynamic applications and comes equipped with algorithms able to decide on the its qualitative and quantitative properties. These algorithms are implemented in the SDF^3 tool [5].

Finite state machine-based SADF (FSM-SADF) [7], [3] is a subset of SADF that abstracts from the stochastic aspects of firing durations and scenario ordering. In FSM-SADF every scenario is represented by a synchronous dataflow (SDF) graph [4], while scenario occurrence patterns are given by a finite state machine. These restrictions render the FSM-SADF more analyzable and implementation efficient than the general SADF model and very well positioned on the expressiveness vs. analyzability trade-off chart [6]. All FSM-SADF analysis and implementation algorithms can be found in the SDF^3 tool.

However, tools such as SDF^3 can be too specialized in the sense that they can only handle predefined properties, thus lacking support for user-defined properties. Although work has been done to check general properties by translating to model checkers [10], [9], this has only been done for the probabilistic SADF formalism. To circumvent this limitation, in this paper we propose a translation of the FSM-SADF formalism to timed automata (TA) as the first step to enable more general verification. Using TA has a number of advantages, in that very efficient abstractions exist. For example, temporal logics can express many of the properties common in reasoning about timed systems with concurrency. Furthermore, TA models of dataflow specifications can be easily extended to add costs such as energy and include the underlying implementation fabric models. This would in the future give us the possibility of using FSM-SADF for reachability analysis of embedded dynamic streaming applications through an optimal control formulation using model-checking techniques. We demonstrate our approach using an MPEG-4 case study modeled as an FSM-SADF graph for which we compute important quantitative and qualitative properties, some of which are not supported by the SDF^3 tool. We use the UPPAAL [2] state-of-the-art tool. The closest related work is the work of Ahmad et al. [1], although this only tackles the SDF formalism and is more concerned with modelling lower-level details of the scheduling on a given execution platform.

II. Definition of FSM-SADF

Here we give a more concise definition of FSM-SADF than the one given in [7]. Specifically, since the sets of ports and detectors have a simpler structure than in the general SADF, it is not necessary to represent them explicitly.

Definition 1 (FSM-SADF graph)**.** *An FSM-SADF graph is a tuple* $G = (\mathcal{S}, \mathcal{K}, \mathcal{B}, E, R_p, R_c, \mathbb{S}, \mathbb{T}, \iota, \Phi, t, \phi_\iota, \psi_\iota)$*, where*

1) $\mathcal{S}$ *is the nonempty finite set of scenarios,*
2) $\mathcal{K}$ *is the nonempty finite set of kernels,*
   - $\mathcal{P} = \mathcal{K} \cup \{d\}$*, where* $d \notin \mathcal{K}$ *denotes the unique detector, is the set of processes,*
3) $\mathcal{B} \subseteq \mathcal{K} \times \mathcal{P}$ *is the set of buffers,*
4) $E : \mathcal{P} \times \mathcal{S} \to \mathbb{N}_0$ *is the execution time for each process in each scenario,*
5) $R_p, R_c : \mathcal{B} \times \mathcal{S} \to \mathbb{N}_0$ *is the production (consumption) rate of the kernel producing to (process consuming from) each buffer in each scenario,*
6) $(\mathbb{S}, \mathbb{T}, \iota, \Phi)$ *is the FSM of the detector, where* $\mathbb{S}$ *is the nonempty set of states,* $\mathbb{T} : \mathbb{S} \to 2^{\mathbb{S}}$ *is the transition function,* $\iota \in \mathbb{S}$ *is the initial state, and* $\Phi : \mathbb{S} \to \mathcal{S}$ *associates each state with a scenario,*
7) $t : \mathcal{K} \times \mathcal{S} \to \mathcal{S}^+$ *is the string of scenarios sent to the FIFO of each kernel in each scenario of the detector,*
8) $\phi_\iota : \mathcal{B} \to \mathbb{N}_0$ *is the initial buffer status,*
9) $\psi_\iota : \mathcal{K} \to \mathcal{S}^*$ *is the initial control status.*

The detector is connected to every kernel by an explicitly ordered (FIFO) control channel. We further define $In(p) = \{b \in \mathcal{B} \mid \pi_r(b) = p\}$, where π_r is the right projection function, to be the set of buffers that process p consumes from (that input into p). Similarly, $Out(k) = \{b \in \mathcal{B} \mid \pi_l(b) = k\}$.

In anticipation of the next section we define $\varnothing$ to be the empty multiset, $\mathbb{P}$ to be the set of all submultisets of its input set, $\uplus$ to be the multiset sum, and $\setminus$ to be the zero-truncated asymmetric multiset difference. For example let $A = \{1, 1\}$ and $B = \{1, 2\}$. Then $A \cup B = \{1, 1, 2\}$ (maxima of multiplicities), $A \uplus B = \{1, 1, 1, 2\}$ (sums of multiplicities), $A \setminus B = \{1\}$, and $B \setminus A = \{2\}$. For strings $\sigma, \tau, \nu \in \mathcal{S}^*$ we define σ_i to be the ith element of σ, $\sigma + \tau$ to be the concatenation of σ and τ, and, if $\nu = \sigma + \tau$, then $\nu - \sigma = \tau$.

A. Operational Semantics

The behavior of an FSM-SADF graph is defined as a transition system where states are configurations.

Definition 2 (Configuration)**.** *A configuration of an FSM-SADF graph* $G = (\mathcal{S}, \mathcal{K}, \mathcal{B}, E, R_p, R_c, \mathbb{S}, \mathbb{T}, \iota, \Phi, t, \phi_\iota, \psi_\iota)$ *is a tuple* $(\phi, \psi, \kappa, \delta)$*, where* ϕ *is a buffer status,* ψ *a control status,* κ *a kernel status, and* δ *a detector status:*

- *A buffer status is a function* $\phi : \mathcal{B} \to \mathbb{N}_0$ *from each buffer to the number of tokens it stores,*
- *A control status is a function* $\psi : \mathcal{K} \to \mathcal{S}^*$ *from each kernel to the string of scenarios (control tokens) its FIFO stores,*
- *A kernel status is a function* $\kappa : \mathcal{K} \to \mathbb{P}(\mathcal{S} \times \mathbb{N}_0)$ *that to each kernel assigns a multiset of ongoing firings and their remaining execution times,*
- *A detector status is a pair* $\delta \in \mathbb{S} \times (\mathbb{N}_0 \cup \{-\})$ *that represents the state of the FSM and the remaining execution time of the ongoing firing, or, if there is no ongoing firing, the value* $-$*.*

The initial configuration of G *is* $(\phi_\iota, \psi_\iota, \kappa_\iota, \delta_\iota)$*, where* ϕ_ι *and* ψ_ι *are defined in* G*,* $\kappa_\iota = \mathcal{K} \times \{\varnothing\}$ *and* $\delta_\iota = (\iota, -)$*.*

Five types of configuration transitions are distinguished.

Definition 3 (Kernel Start Action)**.** *A kernel start action transition* $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(k)} (\phi', \psi', \kappa', \delta)$ *represents the start of a firing of kernel* k*. Let* $s = \psi(k)_1$ *denote the scenario of the firing (if it is defined). The transition is enabled if* $|\psi(k)| \geq 1$ *and* $\forall b \in In(k) : \phi(b) \geq R_c(b, s)$*. The resulting statuses are defined as*

$$\phi' = \phi[b \mapsto \phi(b) - R_c(b, s)] \quad \textit{for all } b \in In(k)$$
$$\psi' = \psi[k \mapsto \psi(k) - s]$$
$$\kappa' = \kappa[k \mapsto \kappa(k) \uplus \{(s, E(k, s))\}]$$

Definition 4 (Kernel End Action)**.** *A kernel end action transition* $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(k)} (\phi', \psi, \kappa', \delta)$ *is the end of a firing of kernel* k*. It is enabled if* $\exists s \in \mathcal{S} : (s, 0) \in \kappa(k)$*. The resulting buffer and kernel statuses are*

$$\phi' = \phi[b \mapsto \phi(b) + R_p(b, s)] \quad \textit{for all } b \in Out(k)$$
$$\kappa' = \kappa[k \mapsto \kappa(k) \setminus \{(s, 0)\}]$$

Definition 5 (Detector Start Action)**.** *A detector start action transition* $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(d)} (\phi', \psi, \kappa, \delta')$ *represents the start of a firing of the detector,* d*. It is enabled if there is no ongoing firing* $\exists s \in \mathbb{S} : \delta = (s, -)$ *and all inputs are available* $\forall b \in In(d) : \phi(b) \geq R_c(b, \Phi(s))$*. The resulting statuses are*

$$\phi' = \phi[b \mapsto \phi(b) - R_c(b, \Phi(s))] \quad \textit{for all } b \in In(d)$$
$$\delta' = (s, E(d, \Phi(s)))$$

Definition 6 (Detector End Action)**.** *A detector end action transition* $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(d)} (\phi, \psi', \kappa, \delta')$ *is enabled if* $\exists s \in \mathbb{S} : \delta = (s, 0)$*, and the resulting statuses are*

$$\psi' = \psi[k \mapsto \psi(k) + t(k, \Phi(s))] \quad \textit{for all } k \in \mathcal{K}$$
$$\delta' = (s', -) \quad \textit{for some } s' \in \mathbb{T}(s)$$

In [7] time transitions are defined very generally, such that to account for given scheduling/resource constraints one needs to instantiate the time transitions needed. In the following we will assume a unconstrained execution, namely that all ongoing firings advance at the same pace.

Definition 7 (Time Transition)**.** *A time transition* $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{time}(t)} (\phi, \psi, \kappa', \delta')$ *represents time progressing* t *time units. It is enabled if no kernel end or detector end transition is enabled, and* t *is the smallest remaining execution time of any ongoing firing. The resulting kernel status is*

$$\kappa' = \kappa[k \mapsto \{(s, n - t) | (s, n) \in \kappa(k)\}] \quad \textit{for all } k \in \mathcal{K}$$

using multiset comprehension. The detector status $\delta = (s, n)$ *is updated as* $\delta' = (s, n - t)$*, unless* $n = -$ *in which case it is unchanged,* $\delta' = \delta$*.*

B. Overtaking Problem

A closer look at the definition of the kernel status and the kernel start action in Section II-A reveals that the operational semantics of FSM-SADF allows the possibility of multiple simultaneous firings of a kernel, i.e. auto-concurrency. If these simultaneous firings of a kernel occur in different scenarios, due to the potential difference in kernel execution times in different scenarios, tokens may "overtake" each other. The result of that is that some kernel might consume tokens in a different scenario than the one these were produced in. This phenomenon makes it hard to ensure determinacy. One way of ensuring determinacy under auto-concurrency is considered in [3] by the introduction of the (max,+) algebraic semantics of FSM-SADF. In this paper, we assure determinacy by prohibiting auto-concurrency in the TA translation introduced in the following section.

III. Translation of FSM-SADF to TA

To be able to model check an FSM-SADF specification, we encode the operational semantics of Section II-A in the UPPAAL model checker. We refer to [2] for the full formalism and introduce it only briefly here due to space constraints. The correctness of the translation (with auto-concurrency being prohibited) follows from the construction itself as explained in the remainder of this section. In UPPAAL, a system is modeled as a network of TA that is extended with bounded discrete variables that are part of the state. We recall the definition of TA where we use $\mathcal{B}(\mathcal{C})$ to denote the set of constraints defined over a finite set of real-valued variables $\mathcal{C}$ called *clocks* and where $\Sigma = \{a!, a?, \ldots\}$ is a finite alphabet of synchronization actions.

Definition 8 (Timed automaton (TA))**.** *A timed automaton $\mathcal{A}$ is a tuple (L, l^0, E, I), where L is a finite set of locations (nodes), l^0 is the initial location, $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times L$ is the set of edges and $I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations. We shall write $l \xrightarrow{g,a,r} l'$ when $(l, g, a, r, l') \in E$.*

The configuration is modelled such that the kernel and detector statuses are encoded in the states of the TA, while the buffer and control statuses are modelled explicitly using discrete variables. These do not add to the expressive power of the formalism, and for presentation purposes, we do not encode their use in the following, but we show the token consumptions and productions in the UPPAAL models in Fig. 1.

Given an FSM-SADF graph G, we generate a parallel composition of TA $System = \mathcal{A}_{k_1} \parallel \ldots \parallel \mathcal{A}_{k_n} \parallel \mathcal{A}_d$, where $k_i \in \mathcal{K}$ and $n = |\mathcal{K}|$. As there is only one instance of each kernel, no auto-concurrency exists, and determinacy is ensured. Fig. 1a shows the UPPAAL model of any kernel and Fig. 1b shows the detector of an FSM-SADF graph with $\mathcal{S} = \{A, B\}$ and the FSM defined to generate the language $(AB)^*$.

Every kernel $k_i \in \mathcal{K}$ is translated to the TA $\mathcal{A}_{k_i} = (L_i, l_i^0, E_i, I_i)$ where $L_i = \{\texttt{Initial}, \texttt{Configure}, \texttt{Fire}\}$, $l_i^0 = \texttt{Initial}$, and E_i and I_i are given as follows. The path

$$\texttt{Initial} \xrightarrow{|\psi(k_i)|\geq 1,\texttt{asap!},\emptyset} \texttt{Configure} \xrightarrow{\forall b \in In(k_i): \phi(b) \geq R_c(b,s),\emptyset,\{x_i\}} \texttt{Fire}$$

corresponds to the kernel start action. It is split into two TA edges because the kernel must first receive its configuration from the detector to know, depending on the current scenario s, how many tokens must be available in its input buffers. The edge $(\texttt{Initial}, \ldots, \texttt{Configure})$ synchronizes by sending on the urgent broadcast channel `asap` (which has no receivers), thus ensuring that the transition is taken as soon as a scenario is available. The kernel end action is encoded by

$$\texttt{Fire} \xrightarrow{x_i = E(k_i,s),\emptyset,\emptyset} \texttt{Initial}$$

and the invariant $I(\texttt{Fire}) = x_i \leq E(k_i, s)$ which together assure that the system stays in the location `Fire` for exactly the execution time $E(k_i, s)$ of the kernel k_i in scenario s. Thus, time transitions are encoded implicitly in the operation of the network of TA, for which time progresses in unison.

The detector TA uses the structure of its FSM, but embeds in each transition a firing location wherein time can pass between the events of consuming the input tokens and producing the output tokens. We encode it as $\mathcal{A}_d = (L_d, l_d^0, E_d, I_d)$, where $L_d = \mathbb{S} \cup \{(s, s') \mid s, s' \in \mathbb{S} \wedge s' \in \mathbb{T}(s)\}$ and $l_d^0 = \iota$. The edge set E_d is defined such that each transition $s_i \to s_j$ described by $\mathbb{T}$ is translated into a detector start edge followed by a detector end edge:

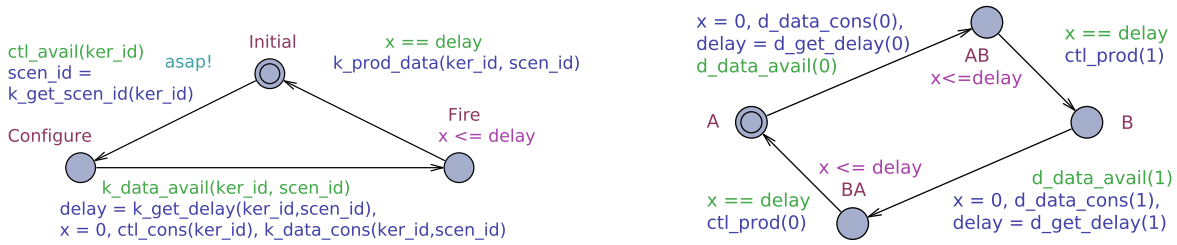
$$s_i \xrightarrow{\forall b \in In(d): \phi(b) \geq R_c(b,\Phi(s)),\emptyset,\{x_d\}} (s_i, s_j) \xrightarrow{x_d = E(d,\Phi(s)),\emptyset,\emptyset} s_j$$

The invariant function I_d is defined such that each firing location (s_i, s_j) maps to the invariant $x_d \leq E(d, \Phi(s_i))$.

IV. Model Checking of TA Model

In this section we demonstrate examples of qualitative and quantitative analysis of the MPEG-4 FSM-SADF specification from [6] using the UPPAAL model checker. The types of analysis, the associated time and memory usage and whether or not the respective type of analysis can be found in the SDF^3 tool are shown in Table I. The experiments were performed on an Intel Core i7-3520M CPU running UPPAAL 4.1.18 64-bit on Linux. The default settings were used and UPPAAL was restarted between each query.

SDF^3 can analyze deadlock freedom, buffer occupancy, inter-firing latency, response delay and throughput of an FSM-SADF specification. Table I reveals that using UPPAAL we can analyze all these properties, except throughput which is a subject of future work. Using UPPAAL we obtained the same results as SDF^3 on the analysis types supported by both frameworks. Analyzing for deadlock freedom is achieved by the UPPAAL query (`A[] not deadlock`). Maximum buffer occupancy analysis is performed using the UPPAAL supremum operator, e.g. (`sup:` b_i). Maximum inter-firing latencies can be obtained as clock suprema. For example, maximum latency of the detector process can be computed using the query (`sup:` x_d). We can check the relationship between the maximum response delay of a process p and a constraint r using the query (`E<>` p`.bFirstFirCompleted and` y_p `>=` r), where y_p is a clock that is never reset, and `bFirstFirCompleted` is a variable set to `true` when p completes its first firing within a scenario. In this experiment, p = MC, and r = 3510. We can also check interleaving patterns of process firings, e.g. "between two consecutive firings of the process p, process q fires at least n times", etc.



(a) Kernel in UPPAAL

(b) Example detector in UPPAAL

Fig. 1: FSM-SADF UPPAAL model

TABLE I: MPEG-4 verification time and virtual memory usage

Analysis	SDF^3	Time [s]	Mem [MB]
Deadlock freedom	Yes	5.87	452
Maximum buffer occupancy, all buffers	Yes	2.44	452
Maximum inter-firing latency, detector	Yes	3.71	455
Maximum response delay, MC	Yes	1.44	252
Interleaving patterns of process firings	No	4.89	460
Maximum delay between process firings	No	4.18	455

For this we use a *leads to* query (whenever a eventually b) and a counter variable: (p`.Fire --> `q`.FireCount >=` n). In the experiment, p = MC, q = RC, and $n = 1$. We can also check whether the maximum delay between the end times of firings of two processes within a scenario is greater than, less than or equal to a predefined value by constructing a query monitor TA that synchronizes with the events of firing completions of the processes it monitors. In the case of kernels, this synchronization takes place when the edges `(Fire,...,Initial)` are taken. In the experiment we verify that the MC-RC delay is always smaller than 5000.

UPPAAL allows us to check the model against various properties, many of which are not supported by the SDF^3 tool-set, therefore justifying the use of a general verification tool such as UPPAAL as a complement to specialized tools. The flexibility of the UPPAAL's TCTL based query language and the possibility of construction of various query monitor automata allows the user to easily and in almost no time compute various qualitative and quantitative properties of the model. In contrast to UPPAAL, doing the same in a specialized tool like SDF^3 would involve the user into a process of software development.

V. Conclusion and Future Work

FSM-SADF is a powerful dataflow formalism that is able to capture the dynamic behaviour of modern streaming applications while offering a good trade-off between expressiveness, analyzability and implementation efficiency. However, the formalism is currently only supported by the SDF^3 tool-set which implements a predefined set of properties that can be analysed/verified. In this paper we propose a translation of FSM-SADF to TA, thereby enabling the use of the UPPAAL model checker for analysing and verifying user-defined properties in a straightforward manner. As future work we plan to develop methods for worst-case throughput analysis, investigate the scalability of our translation and also to give performance comparison with SDF^3. Furthermore, we wish to further investigate auto-concurrency and the overtaking problem and explore policies that would assure determinacy, such as the (max,+) one [3]. As our translation also sets the first milestone towards enabling the use of FSM-SADF in a wider context, e.g. cost-optimal analysis, we also plan to investigate reachability analysis of applications modeled by FSM-SADF through an optimal control formulation using the UPPAAL family of model-checkers.

References

[1] W. Ahmad, E. d. Groote, P. K. F. Hölzenspies, M. I. A. Stoelinga, and J. C. v. d. Pol. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. Technical Report TR-CTIT-13-17, University of Twente, Enschede, January 2014.

[2] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*. Springer, 2004.

[3] M. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, Oct 2010.

[4] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9), Sept 1987.

[5] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF for free. In *Sixth International Conference on Application of Concurrency to System Design (ACSD 2006), 28-30 June 2006, Turku, Finland*, 2006.

[6] S. Stuijk, M. Geilen, B. Theelen, and T. Basten. Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, July 2011.

[7] S. Stuijk, A. Ghamarian, B. Theelen, M. Geilen, and T. Basten. FSM-based SADF. Technical report, Eindhoven University of Technology, Department of Electrical Engineering, 2008.

[8] B. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings. Fourth ACM and IEEE International Conference on*, July 2006.

[9] B. D. Theelen, M. Geilen, and J. Voeten. Performance Model Checking Scenario-Aware Dataflow. In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 6919 of *Lecture Notes in Computer Science*, Aalborg, Denmark, 2011. Springer.

[10] B. D. Theelen, J.-P. Katoen, and H. Wu. Model checking of Scenario-Aware Dataflow with CADP. In W. Rosenstiel and L. Thiele, editors,

Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2012. IEEE.