# A Scenario-Aware Data Flow Model for Combined Long-Run Average and Worst-Case Performance Analysis*

B.D. Theelen[1], M.C.W. Geilen[1], T. Basten[1], J.P.M. Voeten[1,2], S.V. Gheorghita[1] and S. Stuijk[1]
[1]Eindhoven University of Technology, Department of Electrical Engineering; [2]Embedded Systems Institute
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
b.d.theelen@tue.nl

## Abstract

*Data flow models are used for specifying and analysing signal processing and streaming applications. However, traditional data flow models are either not capable of expressing the dynamic aspects of modern streaming applications or they do not support relevant analysis techniques. The dynamism in modern streaming applications often originates from different modes of operation (scenarios) in which data production and consumption rates and/or execution times may differ. This paper introduces a scenario-aware generalisation of the Synchronous Data Flow model, which uses a stochastic approach to model the order in which scenarios occur. The formally defined operational semantics of a Scenario-Aware Data Flow model implies a Markov chain, which can be analysed for both long-run average and worst-case performance metrics using existing exhaustive or simulation-based techniques. The potential of using Scenario-Aware Data Flow models for performance analysis of modern streaming applications is illustrated with an MPEG-4 decoder example.*

## 1. Introduction

Data flow models are often used for specifying the behaviour of signal processing and streaming applications as a set of tasks, actors or processes with data and control dependencies. The differences between various data flow models can be characterised by their expressive power and the availability of techniques for analysing correctness and performance properties like absence of deadlock and throughput. Such analysis is becoming very important for hardware/software co-design of modern streaming systems. Kahn Process Networks (KPN) [10], for example, can capture many of the dynamic aspects of these systems, but evaluating their correctness and performance is in general undecidable. On the other hand, Synchronous Data Flow (SDF) [12] models do allow analysis of many correctness and performance properties but they lack support for expressing any form of dynamism.

This paper proposes a design-time analysable generalisation of SDF called *Scenario-Aware Data Flow (SADF)*, which can capture several dynamic aspects of modern streaming applications by incorporating the concept of scenarios. Such *scenarios* denote distinct modes of operation (like processing I, P or B frames in MPEG), in which a process may have varying execution times. Furthermore, processes may exchange varying amounts of data in different scenarios. Conforming to terminology for SDF, we use *token* to denote a unit of information that is communicated between processes, whereas *(production/consumption) rate* refers to the number of tokens that is produced/consumed by a process. In SADF, these production and consumption rates can be 0 in certain scenarios to specify that data dependencies are absent or that processes are inactive. The key novelty of SADF is that it expresses the behaviour of all scenarios in a succinct model using a stochastic approach to capture the scenario occurrences in an abstract way. Moreover, analysis of correctness as well as long-run average and worst-case performance are decidable. The advantage of using SADF instead of SDF is therefore the potential to obtain more accurate performance results due to taking the dynamic aspects of the system into account.

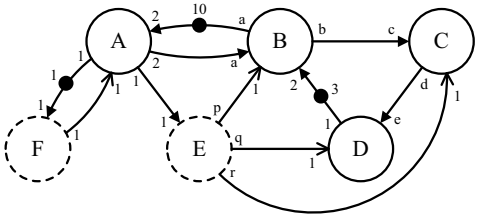



**Figure 1. Scenario-aware data flow.**

To exemplify what we propose, consider the SADF model in Figure 1. We use similar notation as for SDF to depict (the structure of) an SADF model. We distinguish two types of processes; kernels and (scenario) detectors. *Kernels* represent the data processing part of a streaming application, whereas *detectors* model the control part of the application that dynamically detects scenarios. Detectors $E$ and $F$ determine in which scenario the kernels they control ($B$, $C$ and $D$ for detector $E$ and $A$ for detector $F$) operate by sending control tokens via control channels. Such control channels are indicated with open arrowheads, while closed arrowheads denote ordinary (data) channels. When $E$ executes or *fires* (which is the case if $A$ has produced 1 token on the channel to $E$ after firing), it determines the scenario in which $B$, $C$, $D$ and $E$ itself will operate based on a stochastic model. This results in fixing the values for the parameterised rates $p$, $q$ and $r$ and upon firing completion, $E$ sends control tokens indicating the detected scenario to $B$, $C$ and $D$. These kernels fix rates $a$, $b$, $c$, $d$ and $e$ by interpreting the received control tokens before actually processing the data tokens. The execution time of kernels (and detectors) is determined based on the scenario in which they operate, while possible variations within a scenario are taken into account based on a stochastic model.

This paper is organised as follows. After discussing related work in Section 2, Section 3 formally defines SADF and its operational semantics. Section 4 elaborates on useful correctness properties that enable deriving a Markov chain for the evaluation of both long-run average and worst-case performance metrics in Section 5. Section 6 presents experimental results for an MPEG-4 decoder and Section 7 concludes.

## 2. Related Work

When comparing SADF with other data flow models, the most notable differences are the support for varying rates and more specifically rates of 0, the explicit support of correlated execution time distributions for different processes (by means

* This work was partly supported by the European Commission through the Betsy project IST-004042 and partly by the PROGRESS program of the Dutch Technology Foundation STW through the PreMaDoNa project EES.6390.

of the scenarios) and the use of a stochastic approach to model the occurrence of scenarios.

Many generalisations of SDF have been proposed before, often focussing on support for varying rates. For example, Cyclo-Static Data Flow (CSDF) models [3] allow rates to change according to a recurring pattern. SADF can express this behaviour by considering the differences in rates as distinct scenarios and fixing the occurrence sequence in the stochastic model associated with each detector. In Scalable Synchronous Data Flow (SSDF) [16], integer multiples of the rates in a consistent SDF can be used, where these multiples are fixed by an external scheduler. An SADF model can capture the occurrence of the different rates (scenarios) by modelling the scheduler in the detectors. The Parameterised Synchronous Data Flow (PSDF) model is a meta-modelling approach [2] that enables extending dataflow models like SDF with varying rates. PSDF is less expressive than SADF since it requires the parameterised consumption and production rates for a channel to be equal and does not support rates of 0.

Boolean Data Flow (BDF) [4] and Integer Controlled Data Flow [5] follow the approach of restricting KPN models such that the control part of an application is captured by certain predefined process types. Well-behaved forms like the one in [8] even allow analysis of some properties. They can capture scenarios by using subgraphs that are enabled by control tokens, similarly as in control flow graphs. However, these models will, in general, have many more processes than an equivalent SADF model, which avoids including copies of a process that participates differently in multiple scenarios.

Several other authors also proposed to use probabilities for modelling dynamism in an abstract way as well as to support long-run average performance analysis. The suggestion to use (independent) discrete execution time distributions for SDF was declared infeasible for practical applications in [18] because it was considered to suffer too much from state-space explosion problems. SADF combines discrete execution time distributions with scenarios, thereby capturing that execution times of different processes in real-life systems are correlated. Such correlations also limit the state space, which brings performance analysis of practical applications within reach.

To circumvent state-space explosion problems, [18] suggested to use (independent) exponentially distributed execution times for SDF. These distributions have also been extensively studied in the context of stochastic process algebras like PEPA [9] and EMPA [1] and the more control-oriented approach of stochastic Petri Nets [14] as well as queueing networks [11], which all implicitly define Markov chains as SADF does. The generalised form of stochastic Petri Nets in [13] and EMPA are comparable to SADF in the sense that they decouple actions and time. Nevertheless, they support only exponential distributions, which excludes for example worst-case execution time analysis as the sample spaces of exponential distributions are unbounded. On the other hand, there exist several queueing network based approaches that assume general distributions. We are however not aware of approaches that allow taking correlations between such distributions associated to concurrent processes into account.

# 3. Formal Definition

## 3.1. Preliminaries

We start with defining $\mathbb{N} = \{0, 1, \ldots\}$ to indicate the natural numbers and $\mathbb{R}_0^+$ ($\mathbb{R}^+$) to denote the non-negative (positive) real numbers. Similarly as for SDF models, processes (kernels and detectors) in SADF are connected by channels through ports. The finite sets of input and output ports of a process $p$ are denoted by $\mathcal{I}_p$ and $\mathcal{O}_p$ respectively, while the finite set of control ports for a kernel $k$ is indicated by $\mathcal{C}_k$. A channel that connects an output port to an input port of the same process is also called a *self-loop channel*. Without loss of generality, we assume all port sets to be pairwise disjoint and define $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{C}$ to be the union of all input, output and control port sets respectively. For every channel $c$, $\Sigma_c$ denotes the finite set of all possible values of the tokens that it can transfer (channel alphabet). $\Sigma_c^*$ indicates the set of all finite sequences of the tokens in $\Sigma_c$. For a sequence $\sigma = \sigma_1 \ldots \sigma_n$ in $\Sigma_c^*$, the $i^{\text{th}}$ token in $\sigma$ is indicated with $\sigma_i$, whereas $|\sigma|$ denotes the number $n$ of tokens in $\sigma$. For $\sigma, \tau$, $\upsilon \in \Sigma_c^*$, we use $\sigma + \tau$ to indicate the concatenation of $\sigma$ and $\tau$. In case $\upsilon = \sigma + \tau$, we also write $\upsilon - \sigma$ to refer to $\tau$.

The non-empty finite set of scenarios in which a process $p$ can operate is denoted by $\mathcal{S}_p$. Formally, a scenario refers to a set of values for parameterised rates in an SADF and execution time distributions for the processes. For a kernel $k$, the function $R_k : \mathcal{S}_k \times (\mathcal{I}_k \cup \mathcal{O}_k \cup \mathcal{C}_k) \to \mathbb{N}$ assigns the rates to the ports of $k$ for each scenario, whereas the function $R_d : \mathcal{S}_d \times (\mathcal{I}_d \cup \mathcal{O}_d) \to \mathbb{N}$ assigns the rates to the ports of a detector $d$. The discrete random variables specifying the execution times in a scenario $s \in \mathcal{S}_p$ for a process $p$ are denoted by $E_{p,s}$. They range over a finite subset of $\mathbb{R}_0^+$. The probability that $E_{p,s}$ equals a value $e$ in its sample space is denoted by $\mathbb{P}(E_{p,s} = e)$. Using random variables with finite sample spaces ensures the existence of lower and upper bounds, which is required for best/worst-case performance analysis. Future research includes an investigation on using continuous distributions with bounded sample spaces instead.

A discrete-time Markov chain [6] is in this paper defined by a triple $(\mathbb{S}, \iota, \mathbb{P})$, where $\mathbb{S}$ is the non-empty finite state-space of the Markov chain and state $\iota \in \mathbb{S}$ denotes the initial state from which the Markov chain departs with probability 1. The matrix $\mathbb{P}$ is defined such that $\mathbb{P}(S, T) \in [0, 1]$ denotes the one-step transition probability from state $S$ to state $T$, where $\sum_{T \in \mathbb{S}} \mathbb{P}(S, T) = 1$ for all $S \in \mathbb{S}$. An important property of Markov chains is ergodicity [6, 20]. The state space of an ergodic Markov chain includes a single strongly connected component of (positive) recurrent states and possibly several components of transient states. A recurrent state of an ergodic Markov chain is reachable from any other state with probability 1 and has a positive equilibrium probability. Transient states have an equilibrium probability of 0.

## 3.2. Scenario-Aware Data Flow

For a compact formal notation, we assume in this paper that kernels have exactly *one* control port. Kernels without control ports are considered to always operate in the same scenario. Without affecting any performance property, such a kernel can be equipped with a trivial detector for this assumption to hold. Conversely, the results of this paper can be generalised for a kernel $k$ that has multiple control ports by defining $\mathcal{S}_k$ as the Cartesian product of the sets of scenarios of the detectors connected to all its control ports, where all requirements regarding the control port of a kernel that we discuss in this paper should then hold for all control ports of $k$.

Definition 1 introduces functions $\phi$ for an SADF, which enable capturing the status of all its data channels.

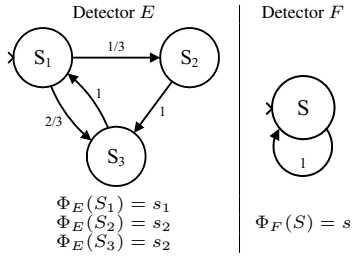



**Figure 2. Specification of detectors.**

**Definition 1 (Channel Status)** *Let $\mathcal{B}$ be the set of all channels, where $\mathcal{B}_c \subseteq \mathcal{B}$ is the set of control channels. A channel status is a function $\phi : \mathcal{B} \setminus \mathcal{B}_c \to \mathbb{N}$ that returns the number of tokens stored in the buffer of each data channel.*

We are now ready to define SADF models.

**Definition 2 (SADF)** *An SADF model is described by a tuple $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*)$, where*

- *$\mathcal{K}$ is the non-empty finite set of kernels and $\mathcal{D}$ is the finite set of detectors with $\mathcal{K} \cap \mathcal{D} = \varnothing$. For each kernel $k \in \mathcal{K}$, $\mathcal{S}_k$ equals the set of scenarios of the detector connected to its unique control port $c$. The rate $R_k(s, c) = 1$ for all scenarios $s \in \mathcal{S}_k$. For each input port $i \in \mathcal{I}_d$ of a detector $d \in \mathcal{D}$, the rate $R_d(s, i)$ is the same positive constant for all $s \in \mathcal{S}_d$. For each output port $o \in \mathcal{O}_d$ that connects $d$ to the control port of a kernel, $R_d(s, o) > 0$ for all $s \in \mathcal{S}_d$. Associated with $d$ is furthermore a Markov chain $(\mathbb{S}_d, \iota_d, \mathbb{P}_d)$ and a function $\Phi_d : \mathbb{S}_d \to \mathcal{S}_d$ that returns the scenario corresponding to each state in $\mathbb{S}_d$;*
- *$\mathcal{B} \subseteq \mathcal{O} \times (\mathcal{I} \cup \mathcal{C})$ is the set of directed channels (each including an unbounded[1] FIFO buffer), where $\mathcal{B}_c = \mathcal{B} \setminus (\mathcal{O} \times \mathcal{I})$ is the set of all control channels. Every port is uniquely connected to one channel and every channel to two ports;*
- *$\phi^* : \mathcal{B} \setminus \mathcal{B}_c \to \mathbb{N}$ is the initial channel status, while the number of tokens initially stored in the control channels in $\mathcal{B}_c$ is equal to $0$.*

A kernel $k \in \mathcal{K}$ starts its firing with reading one token from its control port when it becomes available. That token determines the scenario $s \in \mathcal{S}_k$ in which $k$ will operate. Then $k$ waits until $R_k(s, i)$ tokens are available at every input port $i \in \mathcal{I}_k$. At the moment that sufficient tokens are available, $k$ performs its data processing behaviour, which takes $E_{k,s}$ units of time. The firing of $k$ ends with removing the $R_k(s, i)$ tokens from the input ports and writing $R_k(s, o)$ tokens to each output port $o \in \mathcal{O}_k$.

The firing of a detector $d \in \mathcal{D}$ starts with waiting until $R_d(s, i)$ tokens (which is independent of $s \in \mathcal{S}_d$, see Definition 2) are available at every input port $i \in \mathcal{I}_d$. At the moment that sufficient tokens are available, the Markov chain $(\mathbb{S}_d, \iota_d, \mathbb{P}_d)$ associated with $d$ makes one transition and the scenario $s \in \mathcal{S}_d$ in which $d$ will operate is determined in accordance with the state that is entered due to the transition. After $d$ performed its behaviour, taking $E_{d,s}$ units of time, firing of $d$ ends with removing the $R_d(s, i)$ tokens from its input ports and writing $R_d(s, o)$ tokens to each output port $o \in \mathcal{O}_d$.

1 Like in SDF, a bounded buffer can be modelled by using a reverse channel with a number of initial tokens equal to the buffer size [18].

Notice that in reality, the execution times of a kernel or detector as well as the determination of the scenario by a detector may depend on the values of the consumed tokens. Instead of interpreting data tokens to, for example, trigger transitions in a finite state machine or automaton that represents the behaviour of the involved process, an SADF model abstracts from the actual value of data tokens (similarly as in SDF) to limit its state space. To model data-dependent variations in execution time, SADF relies on the use of scenarios combined with the corresponding execution time distributions, whereas determining the occurrence of scenarios relies on the Markov chains associated to detectors. Such a Markov chain reflects knowledge about the order in which scenarios occur. Consider for example the specification of detector $E$ in Figure 2. Together with $\Phi_E$, the Markov chain specifies that an occurrence of scenario $s_1$ is either followed by one occurrence of scenario $s_2$ with probability $\frac{2}{3}$ or by two occurrences of scenario $s_2$ with probability $\frac{1}{3}$.

Another dynamic aspect of real-life systems is that consuming/producing tokens may actually happen at any time during firing. This could be captured by stochastically distributing the consumption and production of tokens for a process over its execution time. However, in this paper we choose for the distribution where tokens are consumed and produced at the end of a firing. This allows for conservative analysis of the time-average occupancy of the buffers by taking into account the reservation of buffer space at the start of a firing[2].

### 3.3. Operational Semantics

The operational semantics of an SADF is defined in terms of a Timed Probabilistic Labelled Transition System (TPLTS) [17]. To introduce the TPLTS of an SADF, we first give some more definitions to capture the configuration of an SADF.

**Definition 3 (Control Status)** *A control status for an SADF $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*)$ is a function $\psi : \mathcal{B}_c \to \Sigma^*$ with $\Sigma^*$ the union of the sets $\Sigma_c^*$ for all $c \in \mathcal{B}_c$, which returns the sequence of scenarios stored in each control channel.*

**Definition 4 (Kernel Status)** *A kernel status for an SADF $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*)$ is a function $\kappa$ that assigns to each kernel $k \in \mathcal{K}$ a pair in $(\mathcal{S}_k \cup \{-\}) \times (\mathbb{R}_0^+ \cup \{-\})$ denoting the current scenario in which $k$ operates and the remaining execution time for firing $k$ or $(-, -)$ if $k$ is not firing.*

**Definition 5 (Detector Status)** *A detector status for an SADF $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*)$ is a function $\delta$ that assigns to each detector $d \in \mathcal{D}$ a triple in $\mathbb{S}_d \times (\mathcal{S}_d \cup \{-\}) \times (\mathbb{R}_0^+ \cup \{-\})$ denoting the current state $S$ of the Markov chain $(\mathbb{S}_d, \iota_d, \mathbb{P}_d)$, the scenario $\Phi_d(S)$ in which $d$ operates and the remaining execution time for firing $d$ or $(S, -, -)$ if $d$ is not firing.*

Definitions 4 and 5 implicitly exclude the possibility of multiple simultaneous firings of a process (also called auto-concurrency in the context of SDF). The reason is that simultaneous firings of processes operating in different scenarios may "overtake" each other due to the potential differences in execution times (resulting in consuming tokens in another scenario than in which they were produced). This undesirable effect complicates ensuring determinacy and hence, it is excluded. Multiple simultaneous firings can still be captured by

2 Although this is the approach used to obtain the results in Section 6.2, it actually requires a small extension of the semantics in Section 3.3.

| Rate | Scenario $s_1$ | Scenario $s_2$ |
|---|---|---|
| a | 2 | 1 |
| b | 3 | 2 |
| c | 1 | 1 |
| d | 2 | 1 |
| e | 3 | 1 |
| p | 1 | 2 |
| q | 3 | 4 |
| r | 2 | 4 |

| Process | Scenario | $E$ | $\mathbb{P}(E = e)$ |
|---|---|---|---|
| A | $s$ | 2 | 1/2 |
| | | 2.1 | 1/2 |
| B | $s_1$ | 3 | 1/5 |
| | | 2.8 | 4/5 |
| | $s_2$ | 4 | 1 |
| C | $s_1$ | 1 | 1/2 |
| | | 1.2 | 1/2 |
| | $s_2$ | 3 | 1 |
| D | $s_1$ | 4 | 1 |
| | $s_2$ | 2.5 | 1 |
| E | $s_1$ | 0 | 1 |
| | $s_2$ | 0 | 1 |
| F | $s$ | 0 | 1 |

**Figure 3. Rates and execution times.**

using a number of copies of a process and properly distributing and gathering the tokens to be consumed/produced.

**Definition 6 (Configuration)** *A configuration of an SADF is a tuple $(\phi, \psi, \kappa, \delta)$ denoting the channel status, control status, kernel status and detector status respectively. Every SADF $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*)$ has an initial configuration $(\phi^*, \psi^*, \kappa^*, \delta^*)$, where the initial control status $\psi^*$ is defined by $|\psi^*(c)| = 0$ for all $c \in \mathcal{B}_c$, the initial kernel status $\kappa^*$ is defined by $\kappa^*(k) = (-,-)$ for all $k \in \mathcal{K}$ and the initial detector status $\delta^*$ is defined by $\delta^*(d) = (\iota_d, -, -)$ for all $d \in \mathcal{D}$.*

In the remainder of this paper, we use $\Theta$ to denote the set of all reachable configurations of an SADF. Moreover, $D(\Theta)$ indicates the set of probability distribution functions over $\Theta$, which is defined by

$$D(\Theta) = \{\boldsymbol{\pi} : \Theta \to [0,1] \mid \sum_{C \in \Theta} \boldsymbol{\pi}(C) = 1\}$$

We are now ready to define the different transitions for the TPLTS of an SADF, which include five types of action transitions and time transitions. We give the intuition behind their definitions based on our running example of Figure 1. To this end, the tables in Figure 3 complete its specification by giving the values for the parameterised rates and the execution time distributions. Notice that the right-hand table shows the typically small variations in execution times within a scenario and the typically larger variations for different scenarios.

Figure 4 depicts a part of the TPLTS for our running example, departing from the initial configuration $C_0$. A transition is shown as a double directed (multi) arrow, where the first part is labelled with the involved action or time step and the second part denotes the probabilistic fan-out given by the corresponding distribution function in $D(\Theta)$. The only possible action from $C_0$ is a *detect* action for detector $F$, which captures starting its firing (other processes cannot initiate firing since no tokens are available on some of their input ports). Since the Markov chain of $F$ remains in state $S$ (implying detection of default scenario $s$) and its execution time is 0, there is only one possible resulting configuration $C_1$. Hence, $C_1$ is entered with probability 1 in accordance with Definition 7.

**Definition 7 (Detect Action Transition)** *Detect action transitions $\overset{\text{detect}(d)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ refer to detecting the scenario in which a detector $d$ and the kernels controlled by $d$ are going to operate. The relation $(\phi, \psi, \kappa, \delta) \overset{\text{detect}(d)}{\longrightarrow} \boldsymbol{\pi}$ holds if $\delta(d) = (S, -, -)$ for some $S \in \mathbb{S}_d$ and $\phi(x_i) \geq R_d(\Phi_d(S), i)$[3] for each channel $x_i$ connected to an input port $i \in \mathcal{I}_d$. The distribution function $\boldsymbol{\pi}$ is defined by $\boldsymbol{\pi}(\phi, \psi, \kappa, \delta_{T,e}) = \mathbb{P}_d(S,T) \cdot \mathbb{P}(E_{d,\Phi_d(T)} = e)$ for all $T \in \mathbb{S}_d$ and $e$ in the sample space of $E_{d,\Phi_d(T)}$, where $\delta_{T,e} = \delta[d \to (T, \Phi_d(T), e)]$.*

3 Recall that this consumption rate is scenario independent.

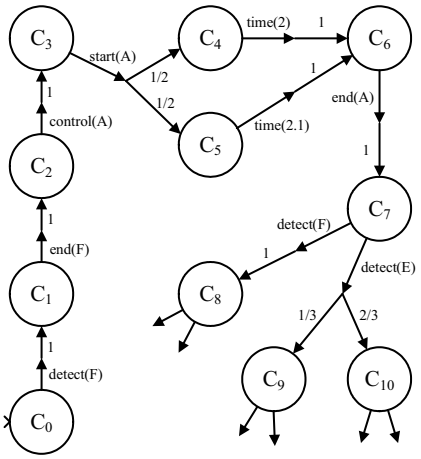



**Figure 4. Partial TPLTS.**

Since $F$ has execution time 0, it can immediately end its firing. This is captured by a *detector end* action. When $F$ finalises its firing, it consumes the token on its input and produces a token to its output. The TPLTS transfers to configuration $C_2$ with probability 1 as specified in Definition 8.

**Definition 8 (Detector End Action Transition)** *Detector end action transitions $\overset{\text{end}(d)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ denote finalising the firing of a detector $d$. Relation $(\phi, \psi, \kappa, \delta) \overset{\text{end}(d)}{\longrightarrow} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi', \psi', \kappa, \delta') = 1$ holds if $\delta(d) = (S, s, 0)$ for some $S \in \mathbb{S}_d$ and $s \in \mathcal{S}_k$. Configuration $(\phi', \psi', \kappa, \delta')$ is defined by $\phi'(x_i) = \phi(x_i) - R_d(s, i)$ for each non self-loop channel $x_i$ connected to an input port $i \in \mathcal{I}_d$ and $\phi'(y_o) = \phi(y_o) + R_d(s, o)$ for each non self-loop channel $y_o \in \mathcal{B} \setminus \mathcal{B}_c$ connected to a corresponding output port $o \in \mathcal{O}_d$, while $\phi'(z) = \phi(z)$ for all other non self-loop channels $z$ and $\phi'(z) = \phi(z) - R_d(s, i) + R_d(s, o)$ for each self-loop channel $z$ connecting an $o \in \mathcal{O}_d$ to an $i \in \mathcal{I}_d$. In addition, for each control channel $c \in \mathcal{B}_c$ connected to a corresponding output port $o \in \mathcal{O}_d$, $\psi'(c) = \psi(c) + \upsilon$ with $\upsilon$ a sequence of tokens (all valued $s$) of length $R_d(s, o)$, while $\psi' = \psi$ for all other control channels. Finally, $\delta' = \delta[d \to (S, -, -)]$.*

In configuration $C_2$, kernel $A$ is ready to fire since a token (valued $s$) is now available on its control port. Starting the firing of a kernel is performed in two steps. A *control* action interprets the control token to determine the scenario, and hence to act according to the appropriate consumption/productions rates and execution time distribution in the second step. As specified in Definition 9, after performing the control action, configuration $C_3$ is entered with probability 1.

**Definition 9 (Control Action Transition)** *Control action transitions $\overset{\text{control}(k)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ reflect fixing the scenario in which a kernel $k$ is going to operate. The relation $(\phi, \psi, \kappa, \delta) \overset{\text{control}(k)}{\longrightarrow} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi, \psi, \kappa', \delta) = 1$ holds if $\kappa(k) = (-,-)$ and $|\psi(c)| \geq 1$ with $c \in \mathcal{B}_c$ the control channel connected to $k$, where $\kappa' = \kappa[k \to (\psi(c)_1, -)]$.*

The second step in starting the firing of a kernel is captured by a *start* action. A start action is enabled for $A$, since

sufficient tokens are available for scenario $s$. In scenario $s$, $A$ can have an execution time of 2 or 2.1, both with probability $\frac{1}{2}$. Hence, there are two resulting configurations possible after the start action as shown in Figure 4. The only difference between $C_4$ and $C_5$ is the remaining execution time for $A$. Definition 10 formalises the start action.

**Definition 10 (Start Action Transition)** *Start action transitions* $\overset{\text{start}(k)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ *indicate starting the processing of data by a kernel* $k$. *Relation* $(\phi, \psi, \kappa, \delta) \overset{\text{start}(k)}{\longrightarrow} \boldsymbol{\pi}$ *holds if* $\kappa(k) = (s, -)$ *for some* $s \in \mathcal{S}_k$ *and* $\phi(x_i) \geq R_k(s, i)$ *for each channel* $x_i$ *connected to an input port* $i \in \mathcal{I}_k$. *Function* $\boldsymbol{\pi}$ *is defined by* $\boldsymbol{\pi}(\phi, \psi, \kappa'_e, \delta) = \mathbb{P}(E_{k,s} = e)$, *where* $\kappa'_e = \kappa[k \rightarrow (s, e)]$ *for all* $e$ *in the sample space of* $E_{k,s}$.

In configurations $C_4$ and $C_5$, no further action transitions can occur since $A$ must first complete its firing. Only then, sufficient tokens become available on its outputs to enable the firing of $E$ and $F$. The completion of firing $A$ requires that its execution time has passed. Consuming the minimal time that enables action transitions again is captured by *time* transitions. Although this minimal amount of time differs for $C_4$ and $C_5$, the resulting configuration after performing the time transition is the same (both only enable $A$ to end its firing while nothing else changes). As specified in Definition 11, configuration $C_6$ is entered with probability 1.

**Definition 11 (Time Transition)** *Time transitions* $\overset{\text{time}(t)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ *denote progress in time with* $t$ *time units. Relation* $(\phi, \psi, \kappa, \delta) \overset{\text{time}(t)}{\longrightarrow} \boldsymbol{\pi}$ *with* $\boldsymbol{\pi}(\phi, \psi, \kappa', \delta') = 1$ *holds if no action transitions are enabled and* $t$ *is the smallest remaining execution time of all processes. Configuration* $(\phi, \psi, \kappa', \delta')$ *is defined by* $\kappa'(k) = (s, n - t)$ *if* $\kappa(k) = (s, n)$ *for some* $s \in \mathcal{S}_k$ *and* $n \in \mathbb{R}^+$, *and* $\delta'(d) = (S, s, n - t)$ *if* $\delta(d) = (S, s, n)$ *for some* $S \in \mathbb{S}_d$, $s \in \mathcal{S}_d$ *and* $n \in \mathbb{R}^+$.

We remark that Definition 11 implicitly assumes maximal progress, which ensures that at most one time transition can be enabled. In configuration $C_6$, kernel $A$ can finalise its firing. This is captured by a *kernel end* action, which includes consuming and producing the appropriate amount of tokens. Definition 12 formalises this last type of action transition.

**Definition 12 (Kernel End Action Transition)** *Kernel end action transitions* $\overset{\text{end}(k)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ *reflect finalising the firing of a kernel* $k$. *Relation* $(\phi, \psi, \kappa, \delta) \overset{\text{end}(k)}{\longrightarrow} \boldsymbol{\pi}$ *with* $\boldsymbol{\pi}(\phi', \psi', \kappa', \delta) = 1$ *holds if* $\kappa(k) = (s, 0)$ *for some* $s \in \mathcal{S}_k$. *Configuration* $(\phi', \psi', \kappa', \delta)$ *is defined by* $\phi'(x_i) = \phi(x_i) - R_k(s, i)$ *for each non self-loop channel* $x_i$ *connected to an input port* $i \in \mathcal{I}_k$ *and* $\phi'(y_o) = \phi(y_o) + R_k(s, o)$ *for each non self-loop channel* $y_o$ *connected to an output port* $o \in \mathcal{O}_k$, *while* $\phi'(z) = \phi(z)$ *for all other non self-loop channels* $z$ *and* $\phi'(z) = \phi(z) - R_k(s, i) + R_k(s, o)$ *for each self-loop channel* $z$ *connecting an* $o \in \mathcal{O}_k$ *to an* $i \in \mathcal{I}_k$. *Moreover,* $\psi' = \psi[c \rightarrow \psi(c) - \psi(c)_1]$ *with* $c \in \mathcal{B}_c$ *the channel connected to the control port of* $k$. *And finally,* $\kappa' = \kappa[k \rightarrow (-, -)]$.

When residing in configuration $C_7$, two detect actions are enabled. There are sufficient tokens available on all inputs of both detectors $E$ and $F$. After non-deterministically choosing between these options, the TPLTS can go to configurations $C_8$, $C_9$ or $C_{10}$. The latter two show the probabilistic fan-out after choosing for performing a detect action for $E$. They represent that the next state of its Markov chain can be $S_2$ (with probability $\frac{1}{3}$) or $S_3$ (with probability $\frac{2}{3}$), where for both there is only one execution time possible. The only difference between $C_9$ and $C_{10}$ is that the Markov chain of $E$ entered a different state. From $C_8$, $C_9$ and $C_{10}$, the system can make further transitions that are not detailed in Figure 4.

## 4. Useful Properties

An execution of an SADF refers to a sequence of transitions through its TPLTS, starting from the initial configuration. To investigate boundedness, absence of deadlock and determinacy for such executions, we first give some additional definitions. Notice that a detector $d$ and all the kernels $k_1, \ldots k_n$ it controls always operate in the same scenario of $\mathcal{S}_d$. Processes $d$ and $k_1, \ldots, k_n$ are therefore said to be mutually *scenario synchronous*, which simplifies defining the repetition vector for an SADF as follows.

**Definition 13 (Repetition Vector)** *A repetition vector for an SADF* $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*)$ *is a function* $\gamma$ *that assigns to all processes* $p_1, p_2$ *in* $\mathcal{P} = \mathcal{K} \cup \mathcal{D}$ *and all their scenarios, an element of* $\mathbb{N}$ *such that if output port* $o \in \mathcal{O}_{p_1}$ *is connected to input or control port* $i \in \mathcal{I}_{p_2} \cup \mathcal{C}_{p_2}$;

- *if* $p_1$ *and* $p_2$ *are scenario synchronous, then* $R_{p_1}(s, o) \cdot \gamma(p_1, s) = R_{p_2}(s, i) \cdot \gamma(p_2, s)$ *for all* $s$ *in* $\mathcal{S}_{p_1} = \mathcal{S}_{p_2}$;
- *if* $p_1$ *and* $p_2$ *are not scenario synchronous, then* $R_{p_1}(s_1, o) \cdot \gamma(p_1, s_1) = R_{p_2}(s_2, i) \cdot \gamma(p_2, s_2)$ *for all* $s_1 \in \mathcal{S}_{p_1}$ *and* $s_2 \in \mathcal{S}_{p_2}$.

*Repetition vector* $\gamma$ *is said to be non-trivial in case* $\gamma(p, s) > 0$ *for all processes* $p \in \mathcal{P}$ *and* $s \in \mathcal{S}_p$.

The existence of a non-trivial repetition vector is important for boundedness and absence of deadlock as clarified below. We remark that SADF models with a non-trivial repetition vector only allow rates of 0 for input and output ports of kernels if both the consumption and production rate for the connecting channel is 0 in the involved scenario. On the other hand, having a non-trivial repetition vector does not necessarily imply that the behaviour is meaningful. Consider for example a detector $d$ with a repetition vector entry larger than 1 for some scenario, while the states of its Markov chain visited in successive firings always imply different scenarios. If in this case, the production and consumption rates between two kernels controlled by $d$ differ in the different scenarios, tokens might be consumed in a scenario different from the one in which they were produced. To avoid this undesirable situation, the property of strong[4] consistency is introduced.

**Definition 14 (Strong Consistency)** *An SADF is called strongly consistent iff it has a non-trivial repetition vector* $\gamma$ *such that for each detector* $d$, $\gamma(d, s) = 1$ *for all scenarios* $s \in \mathcal{S}_d$. *For a strongly consistent SADF, there is a unique smallest non-trivial repetition vector, which is designated as* the *repetition vector of the SADF.*

Recalling Definition 13, it can be observed that strong consistency requires that the rate $R_d(s, o)$ of each output port $o \in \mathcal{O}_d$ of a detector $d$ connected to the control port $c$ of a kernel $k$ must equal the repetition vector $\gamma(k, s)$ of $k$ for all scenarios $s \in \mathcal{S}_d$ (since $R_k(s, c) = 1$). Furthermore, the rate

4 It is possible to define a weaker form of consistency that relaxes the requirement for detectors to have an equivalent repetition vector entry.

$R_d(s, o)$ of each output port $o \in \mathcal{O}_d$ connected to an input port of another detector must be the same for all $s \in \mathcal{S}_d$.

We now continue with another important property affecting boundedness and absence of deadlock. It concerns the persistency of dependencies between processes. Although the structure of how channels connect processes is fixed for an SADF, the possibility of rates equal to 0 allows differences in the dependencies for different scenarios. To enable capturing the persistency of dependencies, we extend the notion of reachability in traditional graphs (including SDF) as follows.

**Definition 15 (Dependability)** *Process $p_n$ depends on process $p_1$ for scenario combination $s_1, \ldots, s_n$ with $s_i \in \mathcal{S}_{p_i}$ for $1 \leq i \leq n$ iff a path of channels exists, which connects $p_1$ to $p_n$ via processes $p_2, \ldots, p_{n-1}$ such that all rates encountered for the ports along the path are positive for $s_1, \ldots, s_n$.*

We now introduce the concept of strong dependency, which is similar to strong connectivity in traditional graphs.

**Definition 16 (Active Process)** *Process $p$ is said to be inactive for a scenario $s \in \mathcal{S}_p$ iff $R_p(x, s) = 0$ for all $x \in \mathcal{I}_p \cup \mathcal{O}_p$. Otherwise, $p$ is active in $s$.*

**Definition 17 (Strong Dependency)** *Let the detectors of an SADF be denoted by $d_1, \ldots, d_n$. The SADF is called strongly dependent iff for each scenario combination $s_1, \ldots, s_n$ with $s_i \in \mathcal{S}_{d_i}$ for $1 \leq i \leq n$, each active process depends on all other active processes.*

The essence of strong dependency is that no cyclic dependencies between active processes are broken when switching scenarios, even though the cyclic dependency between two specific active processes may differ for different scenarios. Notice that detectors are always active and hence, an inactive kernel always depends on the detector controlling it. On the other hand, no other processes depend on inactive kernels, which means that there exists no cyclic dependency. Finally, we can investigate the property of boundedness.

**Definition 18 (Boundedness)** *An SADF is called bounded iff there exists a $B \in \mathbb{N}$ such that for all reachable configurations $(\phi, \psi, \kappa, \delta)$, $\phi(x) \leq B$ for all $x \in \mathcal{B} \backslash \mathcal{B}_c$ and $|\psi(c)| \leq B$ for all $c \in \mathcal{B}_c$. It is unbounded otherwise.*

Executing an unbounded SADF may lead to an unbounded increase in the number of tokens stored in the buffers. To exclude this undesirable situation, the next theorem identifies conditions that ensure boundedness. The detailed proofs of all the theorems in this paper are omitted for space reasons, but sketches of the proofs can be found in the appendix.

**Theorem 1 (Boundedness)** *A strongly consistent and strongly dependent SADF is bounded if for each kernel $k$ the inequality $E_{d,s} \geq \gamma(k, s) \cdot E_{k,s}$ holds with probability 1 for all scenarios $s$ in which $k$ is inactive, where $d$ denotes the detector that controls $k$.*

The last condition in Theorem 1 ensures boundedness of control channels to kernels that may become inactive, see the appendix. If an SADF model is bounded, then the set $\Theta$ is finite (all sets affecting the cardinality of $\Theta$ are finite, including the sample spaces of the execution time distributions).

An SADF may deadlock, which means that there exists a configuration in $\Theta$ without outgoing transitions. Conversely, any execution of a deadlock-free SADF includes an infinite number of control, start and detect action transitions. The next theorem gives conditions for absence of deadlock.

**Theorem 2 (Absence of Deadlock)** *A strongly consistent and strongly dependent SADF is deadlock-free if, for all scenario combinations, there are sufficient initial tokens in each cyclic dependency between active processes such that every process in the cycle can fire a number of times equal to its repetition vector entry.*

From Definitions 2 and 13, it can be derived that the scenario in which a process $p$ operates may only change after firing $\gamma(p, s)$ times for each scenario $s \in \mathcal{S}_p$. Verifying whether an SADF is bounded and deadlock-free could therefore be based on repeated application of existing techniques for analysing these properties in SDF models as mentioned in for example [12, 18] (with a few minor modifications to anticipate for rates of 0) for each scenario combination. Future research includes a detailed investigation on more efficient approaches. By using the SDF techniques, the reader may verify that the SADF in Figure 1 is bounded and deadlock-free.

As shown in Figure 4, the TPLTS of an SADF may include two types of choices; non-deterministic choices originating from the concurrency in the model and probabilistic choices due to variations in scenarios and execution times. The policy for making these choices may affect the functionality (and performance). With the functionality of a dataflow model, one typically considers the sequence of tokens on the channels. Since an SADF abstracts from the actual values of data tokens, we only consider the number of tokens and their timing. The following theorem states that non-determinism does not affect the functionality; only the probabilistic choices do.

**Theorem 3 (Determinacy)** *The functionality represented by an SADF only depends on the probabilistic choices that determine the sequence of scenarios successively detected by each detector and not on the non-deterministic choices.*

## 5. Performance Analysis

Before discussing performance analysis with SADF, we first remark that many of the performance metrics that are of interest (such as throughput and time-average buffer occupancy) have no meaning in an untimed context. An SADF is said to be untimed if for all processes $p$, $E_{p,s} = 0$ with probability 1 for all $s \in \mathcal{S}_p$. It is timed otherwise.

We discuss two approaches for SADF-based performance analysis. The first approach allows evaluation of *any* long-run average or worst-case performance metric, which is based on deriving a Markov chain from the involved TPLTS and relies on a similar approach as in [21]. The first step concerns resolving any non-determinism. In general, the policy used for resolving non-determinism may affect the results for certain performance metrics. An example is the maximum occupancy of a buffer, which depends on the order of scheduling the possibly non-deterministic actions regarding the reading and writing of tokens. Theorem 4 states that many long-run average performance metrics are not affected by the policy used for resolving non-determinism. Examples of such metrics are throughput and time-average buffer occupancy.

**Theorem 4 (Long-Run Equivalence)** *Any long-run average for a timed SADF model that merely depends on the status of the system in configurations of the TPLTS just before and after time transitions converges to the same result regardless how non-determinism is resolved.*

After resolving non-determinism, a new TPLTS is obtained (reflecting a subset of all possible behaviours of the

original one) [21]. In case the SADF is deadlock-free, then this new TPLTS can be interpreted as a discrete-time Markov chain by shifting the information on the occurrence of actions and progress of time into the configurations. The state space of this Markov chain is finite if the SADF is bounded. The final step in this Markov chain based approach concerns defining reward functions to express the metrics of interest [21].

Explicitly constructing the Markov chain from the TPLTS allows for analytical computation of any (complex combination of) long-run average(s) by applying the ergodic theorem [6, 20]. The formal relation between an SADF and its implied Markov chain also provides the foundation for proper simulation-based performance estimation, without the need to explicitly construct the TPLTS or Markov chain. Accuracy analysis of (complex combinations of) long-run averages can in this case be accomplished by using the central limit theorem based techniques proposed in [7, 19]. Nevertheless, to apply the mentioned existing exhaustive and simulation-based techniques, the Markov chain must be ergodic. In the literature this is often simply assumed to be true. With Theorem 5, we identify however an example class of SADF models for which ergodicity is ensured. The reader may verify that the SADF in Figures 1 satisfies the conditions of Theorem 5 (see also Figure 2).

**Theorem 5 (Ergodicity)** *If the Markov chain for each detector of a bounded and deadlock-free SADF is ergodic and has only (positive) recurrent states, then the Markov chain implied by the SADF is ergodic.*

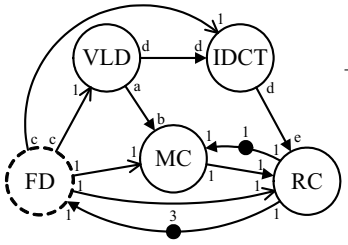
Next to detrmining long-run average performance metrics, the approach of deriving a Markov chain from the TPLTS of an SADF is also suitable for evaluating worst-case performance metrics. Both analytical computation and simulation-based estimation of worst-case metrics are fairly straightforward[5] in this case, though no information on the accuracy of results can be given when using simulation. An alternative approach to evaluate certain *specific* worst-case metrics (like the worst-case execution time or maximum buffer occupancy) is the possibility to convert an SADF model into a set of SDF models; one for each possible scenario combination using the largest possible execution time for each process in that scenario combination. The overall worst-case result can then easily be derived from the results for the individual scenarios (by taking their maximum). Future research includes comparing the efficiency of applying the Markov chain based and SDF based approaches for evaluating worst-case metrics.

# 6. Experimental Results

## 6.1. Prototype Tool Flow

Analysis of modern streaming applications requires efficient tools. Unfortunately, a specialised tool for analysing SADF models according to the approaches discussed in sections 4 and 5 is not yet available. To investigate the potential of using SADF models for performance analysis, we developed however a prototype tool flow that relies on an SADF modelling style for the modelling language POOSL [15]. The operational semantics of a POOSL model also defines a discrete-time Markov chain as a basis for exhaustive and simulation-based performance analysis [19] according to similar concepts as used for SADF models, see [21]. We are however aware of the fact that the TPLTS of a POOSL model has a much smaller grain of execution steps compared to that of an SADF model, which makes this approach inherently inefficient compared to a future specialised SADF analysis tool. Another disadvantage is that currently no tools for exhaustive performance analysis of POOSL models exist.

5 Computation of a worst-case metric based on a Markov chain boils down to identifying the state for which the involved reward is maximal. In simulation, one would track the maximum of all occurring rewards.




| Rate | Scenario $I$ | $P_0$ | $P_x$ |
|---|---|---|---|
| a | 0 | 0 | 1 |
| b | 0 | 0 | $x$ |
| c | 99 | 1 | $x$ |
| d | 1 | 0 | 1 |
| e | 99 | 0 | $x$ |

**Figure 5. SADF model of an MPEG-4 decoder.**

The current tool flow has three phases. The first phase concerns determining the scenarios of a streaming application together with the accompanying rates and execution time distributions. Information on the occurrence of scenarios, which is needed to specify the Markov chains associated with detectors, may originate from standards (for example, common implementations of MPEG use certain fixed sequences of I, P and B frames) or can be obtained using a profiling tool. After organising the information in an XML specification of an SADF model, a tool called `sadf2poosl` automatically constructs a POOSL model according to the SADF modelling style. It also adds performance monitors using the approach of [19]. The final phase relies on the simulator for POOSL models `rotalumis` [15], which obtains performance estimation results with a predetermined accuracy.

## 6.2. MPEG-4 Simple Profile Decoder

In this section, we consider an MPEG-4 decoder for the Simple Profile. This decoder supports video streams consisting of I and P frames. Such frames consist of a number of macro blocks, each requiring operations like Variable Length Decoding (VLD), Inverse Discrete Cosine Transformation (IDCT), Motion Compensation (MC) and Reconstruction (RC). Figure 5 depicts the SADF model of the considered MPEG-4 decoder. The VLD and IDCT kernels in this model fire once per macro block that is decoded for a frame, while the MC and RC kernels fire once per frame. The Frame Detector (FD) represents the part of the actual VLD determining the frame type. The initial token on the channel from RC to MC models the exchange of the previously decoded frame, while the three initial tokens on the channel from RC to FD model that the decoder is capable of performing VLD and IDCT for macro blocks of 3 frames in a pipelined fashion.

When detecting an I frame, all macro blocks must be decoded using VLD and IDCT, while the resulting image is reconstructed by RC straightforwardly. Assuming an image size of $176 \times 144$ pixels (QCIF), there are 99 macro blocks to decode for an I frame, which explains the values of the parameterised rates $c$, $d$ and $e$ in this scenario. Conversely, there are no motion vectors to be taken into account for an I frame and hence, MC does not receive any motion vectors from VLD.

Decoding P frames requires MC to take motion vectors into account for retrieving the correct position of macro

| $k$ | $T_k$ |
|---|---|
| VLD | $0.063 \pm 0.04\%$ |
| IDCT | $0.063 \pm 0.04\%$ |
| MC | $0.00106 \pm 0.19\%$ |
| RC | $0.00106 \pm 0.19\%$ |

| $k$ | av$[L_k]$ | var$[L_k]$ | $\overline{L}_k$ |
|---|---|---|---|
| VLD | $15.99 \pm 0.03\%$ | $75.38 \pm 0.82\%$ | 710 |
| IDCT | $15.99 \pm 0.03\%$ | $56.45 \pm 1.09\%$ | 698 |
| MC | $940.3 \pm 0.02\%$ | $2.4 \cdot 10^5 \pm 3.46\%$ | 3305 |
| RC | $940.3 \pm 0.02\%$ | $1.5 \cdot 10^5 \pm 4.99\%$ | 2216 |

| $b$ | av$[O_b]$ | var$[O_b]$ | $\overline{O}_b$ |
|---|---|---|---|
| VLD - IDCT | $1.910 \pm 0.06\%$ | $0.528 \pm 1.99\%$ | 9 |
| IDCT - RC | $60.19 \pm 0.18\%$ | $671.8 \pm 4.55\%$ | 154 |
| VLD - MC | $34.73 \pm 0.52\%$ | $698.4 \pm 4.39\%$ | 133 |
| MC - RC | $0.577 \pm 0.56\%$ | $0.244 \pm 3.27\%$ | 1 |

All accuracy results are for confidence levels of 0.95.

**Table 1. Performance Results.**

blocks from the previous frame. The resulted image is then corrected, if needed, with the pixel data contained in freshly decoded macro blocks. The number of motion vectors and the number of macro blocks to decode may differ for different P frames. The MPEG-4 decoder in Figure 5 assumes an equal number (0 or $x \in \{30, 40, 50, 60, 70, 80, 99\}$) of motion vectors and macro blocks to be decoded, each implying a different scenario. The possible values for $x$ represent conservative approximations for a range of different numbers of motion vectors that may occur in reality. The special case of 0 motion vectors may reflect decoding still video (in which case the VLD and IDCT kernels are inactive), where MC simply copies the previously decoded frame.

To evaluate the performance of the MPEG-4 decoder, we determined the execution time distributions of the kernels for each scenario using a profiling tool. This tool was also used for determining the occurrence probability of the scenarios. The latter straightforwardly resulted in a fully connected Markov chain with nine states (one for each possible scenario) for detector FD, which has an equilibrium distribution given by the scenario occurrence probabilities.

By applying the prototype tool flow discussed in Section 6.1, we obtained the performance results shown in Table 1. The time unit for the model is a kCycle. The top table shows the throughput $T_k$ of each kernel $k$ (in number of firings per kCycle). It can be observed that RC finalises decoding a frame about every MCycle on average. The middle table gives the average av$[L_k]$, variance var$[L_k]$ and maximum $\overline{L}_k$ of the time $L_k$ between successive firings of $k$. The first observation is that the reciprocal of the average time between successive firings correctly matches with the throughput numbers as expected. Nevertheless, the large variance in the time between successive firings of RC (and MC) shows that decoded frames become available in bursts. Because this is unacceptable in practice, an explicit play-out buffer is needed to communicate frames to a display with fixed rate. This observation may inspire a designer to extend the SADF model correspondingly and investigate how large this play-out buffer should be in order to obtain a certain minimal percentage of deadline misses. This percentage is in fact a long-run average that could be evaluated based on the results of this paper. The bottom table depicts the time-average av$[O_b]$ and time-variance var$[O_b]$ buffer occupancies $O_b$ for some channels $b$ (i.e., the average/variance occupation when taking the duration of each individual occupation into account) as well as their maximum occupancy $\overline{O}_b$. These results can for example be used to dimension the memory that realises these buffers.

# 7. Conclusions

This paper introduces a generalisation of the Synchronous Data Flow model to succinctly express dynamic changes in execution times and rates by means of scenarios. A Scenario-Aware Data Flow (SADF) model can take the correlation between the execution time distributions of different processes into account. With rates of 0, it can express that a data dependency is absent or that a process is inactive in a certain scenario. Another key feature is the use of a stochastic approach to capture the occurrence of scenarios in an abstract way. The SADF model enables design-time analysis of both long-run average and worst-case performance metrics using existing exhaustive or simulation-based techniques. Future work includes a detailed investigation on the properties of SADF models and on an efficient analysis tool.

# References


[1] M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, vol 202 (1–2), pp 1–54, 1998.

[2] B. Bhattacharya and S.S. Bhattacharyya. Parameterized Dataflow Modeling for DSP Systems. *IEEE Transactions on Signal Processing*, vol 49 (10), pp 2408–2421, 2001.

[3] G. Bilsen, M. Engels, R. Lauwereins and J.A. Peperstraete. Cyclo-Static Dtata Flow. *Proceedings of ICASSP'95*, vol 5, pp 3255–3258, IEEE, 1995.

[4] J.T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model*. Ph.D. Thesis, University of California, Berkeley, 1993.

[5] J.T. Buck. Static Scheduling and Code Generation from Dynamic Dataflow Graphs with Integer-Valued Control Streams. *Proceedings of SSC'94*, vol 1, pp 508–513, IEEE, 1994.

[6] K.L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer-Verlag, 1967.

[7] M.A. Cranes and A.J. Lemoine. An Introduction to the Regenerative Method for Simulation Analysis. *Lecture Notes in Control and Information Sciences*, vol 4, 1977.

[8] G.R. Goa, R. Govindarajan and P. Panangaden. Well-Behaved Dataflow Programs for DSP Computation. *Proceedings of ICASSP'92*, vol 5, pp 561–564, IEEE 1992.

[9] J. Hillston. Compositional Markovian Modelling Using a Process Algebra. *Proceedings of NSMC'95*, pp 177–196. Kluwer, 1995.

[10] G. Kahn. The Semantics of a Simple Language for Parallel Programming. *Proceedings of IFIP'74*, pp 471–475, North-Holland, 1974.

[11] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley Interscience, 1975.

[12] E. Lee and D. Messerschmitt. Synchronous Data Flow. *IEEE Proceedings*, vol 75 (9), pp 1235–1245, 1987.

[13] M.A. Marsan, G. Conte and G. Balbo. A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, vol 2 (2), pp 93–122, 1984.

[14] M.K. Molloy. Performance Analysis using Stochastic Petri Nets. *IEEE Transactions on Computers*, vol 31 (9), pp 913–917, 1982.

[15] http://www.es.ele.tue.nl/poosl

[16] S. Ritz, M. Pankart and H. Meyr. High-Level Software Synthesis for Signal Processing Systems. *Proceedings of ASAP'92*, pp 679–693, IEEE, 1992.

[17] R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. Thesis, Massachusetts Institute of Technology, 1995.

[18] S. Siram and S.S. Bhattacharyya. *Embedded Multiprocessors; Scheduling and Synchronization*. Marcel Dekker, 2000.

[19] B.D. Theelen. *Performance Modelling for System-Level Design*. Ph.D. Thesis, Eindhoven University of Technology, 2004.

[20] H.C. Tijms. *Stochastic Models; An Algorithmic Approach*. John Wiley & Sons, 1994.

[21] J.P.M. Voeten. Performance Evaluation with Temporal Rewards. *Performance Evaluation*, vol 50 (2/3), pp 189–218, 2002.


# Appendix: Proofs

This appendix sketches the proofs of the theorems in this paper. Some of these sketches rely on properties of SDF models of which a detailed definition can be found in [12, 18].

## Boundedness

Proving Theorem 1 involves considering three major aspects: boundedness in case all processes always operate in a fixed scenario, the effect of scenario changes on boundedness and boundedness of control channels to kernels that may become inactive. For discussing these aspects, a definition of an *iteration* of an SADF is helpful.

**Definition 19** *For a certain scenario combination, an iteration of an SADF refers to the firing of each process $p$ for $\gamma(p, s)$ times, where $s$ is the scenario in which $p$ operates.*

The conditions of strong consistency and strong dependency in Theorem 1 are identical to those ensuring boundedness for SDF models [12, 18]. Strong dependency for SDF means that all actors are strongly connected, which guarantees that each actor has a cyclic dependency with all other actors. Consistency in SDF ensures that the number of tokens in each channel after an iteration of the SDF is the same as before. The proof that consistency and strong connectivity make an SDF bounded is similar to the one in [25] for computation graphs. Turning back to SADF, strong dependency ensures that active processes always have a cyclic dependency with all other active processes. From Definitions 2 and 13, it follows that the number of tokens in channels between active processes after an iteration of the SADF is the same as before. Based on the proof in [25], we therefore conclude that the number of tokens in channels between active processes is bounded for each individual scenario combination if the SADF is strongly consistent and strongly dependent (for each scenario combination individually, an SADF behaves like an SDF).

An SADF may have control channels to kernels that are inactive in certain scenarios. An inactive kernel does not have a cyclic dependency with the detector controlling it (nor with any other process). Strong consistency is insufficient to ensure that the number of tokens in the control channel does not increase unboundedly for an infinite number of iterations for the involved scenario combination. In other words, firing the kernel does not limit the pace with which the detector produces control tokens. Satisfying the inequality in Theorem 1 ensures however that the kernel is sufficiently fast with removing the tokens from the control channel when it is inactive.

So far, we only showed that Theorem 1 is plausible for each individual scenario combination.

**Lemma 6** *In a strongly consistent SADF, a process $p$ can only change its scenario after firing $\gamma(p, s)$ times if it operates in scenario $s \in \mathcal{S}_p$.*

**Proof** Trivial if $p$ is a detector since then $\gamma(p, s) = 1$ for all $s \in \mathcal{S}_p$. If $p$ is a kernel that is controlled by a detector $d$ via output port $o \in \mathcal{O}_d$, then $p$ operates for $\gamma(p, s) = R_d(s, o)$ firings in scenario $s \in \mathcal{S}_d$ as a consequence of being consistent and Definition 8. ■

Although Lemma 6 proofs that scenario changes can only occur after completing an iteration, such iterations may overlap in a pipelined fashion. The crux for proving Theorem 1 is that the individual firings of processes can be partitioned into individual scenario iterations, each having no nett effect on the number of tokens in the channels, while the number of scenario iterations that is active (in a pipelined fashion) is bounded due to each detector being always dependent on all other detectors.

## Absence of Deadlock

For proving Theorem 2, we follow a similar approach as for Theorem 1. Deadlock in an SADF can only originate from the cyclic dependencies between active processes. The key to absence of deadlock is therefore to ensure that an iteration can always be completed, despite the cyclic dependencies. Recalling Definition 19, this implies that each (active) process can fire a number of times equal to its repetition vector entry. The condition in Theorem 2 ensures that sufficient initial tokens are available to complete an iteration for each individual scenario combination. Because an SADF behaves like an SDF for each scenario combination, absence of deadlock can be verified using a similar approach as for SDF based on the results of [25]. The crux of proving Theorem 2 is that after completing an iteration, the number of tokens in all channels is the same as before and hence, another iteration can always be performed. Using again that scenario changes and the pipelined overlapping of iterations do not affect the nett result of each individual iteration completes the proof.

## Determinacy

The proof of Theorem 3 is based on extending existing proofs for determinacy of other data flow models, like the one for KPN in [26], to accommodate for the probabilistic choices. The essential property that makes the policy for resolving non-determinism irrelevant is that non-determinism only occurs between independent concurrent actions, which leads to satisfying the so-called diamond property. It states that when two actions are enabled, it does not matter for the functionality in which order these two actions are performed. Although transitions for an SADF do not yield configurations as in [26] but rather (finite) distributions, the proof of Theorem 3 still follows a similar approach. To this end, we first lift the transition relation $\rightarrow$ defined in Definitions 7 through 12 to a relation on $D(\Theta)$.

**Definition 20** *Let $\boldsymbol{\pi}_{C,\alpha} \in D(\Theta)$ denote the probability distribution over $\Theta$ after performing an action or time transition $\alpha$ from a configuration $C$ if it exists. Relation $\boldsymbol{\pi}_1 \xrightarrow{\alpha} \boldsymbol{\pi}_2$ holds if for all configurations $C$ with positive $\boldsymbol{\pi}_1(C)$ transition $\alpha$ can be performed and $\boldsymbol{\pi}_2(C') = \sum_{C \in \Theta} \boldsymbol{\pi}_1(C) \cdot \boldsymbol{\pi}_{C,\alpha}(C')$ for all possible resulting $C' \in \Theta$.*

Figure 6 shows an extension of the TPLTS in Figure 4, which focusses on the non-deterministic choice between the detect(E) and detect(F) actions that can be performed from configuration $C_7$. After choosing for one of them, the other can be performed in the next execution step. Let $\boldsymbol{\pi}_1$ denote the distribution where the transition system arrived in configuration $C_7$. Performing the detect(F) action gives $\boldsymbol{\pi}_1 \xrightarrow{\text{detect(F)}} \boldsymbol{\pi}_2$ with $\boldsymbol{\pi}_2(C_8) = 1$. On the other hand, also $\boldsymbol{\pi}_1 \xrightarrow{\text{detect(E)}} \boldsymbol{\pi}_3$ with $\boldsymbol{\pi}_3(C_9) = \frac{1}{3}$ and $\boldsymbol{\pi}_3(C_{10}) = \frac{2}{3}$. The diamond property for SADF models states that there exists some $\boldsymbol{\pi}_4$ such that $\boldsymbol{\pi}_2 \xrightarrow{\text{detect(E)}} \boldsymbol{\pi}_4$ and $\boldsymbol{\pi}_3 \xrightarrow{\text{detect(F)}} \boldsymbol{\pi}_4$. Figure 6 illustrates that such $\boldsymbol{\pi}_4$ with $\boldsymbol{\pi}_4(C_{11}) = \frac{1}{3}$ and $\boldsymbol{\pi}_4(C_{12}) = \frac{2}{3}$ exists indeed.

**Lemma 7 (Diamond Property)** *Let $\alpha$ and $\beta$ be two enabled actions. If $\boldsymbol{\pi} \xrightarrow{\alpha} \boldsymbol{\pi}'$ and $\boldsymbol{\pi} \xrightarrow{\beta} \boldsymbol{\pi}''$ with $\boldsymbol{\pi}, \boldsymbol{\pi}', \boldsymbol{\pi}'' \in D(\Theta)$, then there exists some $\boldsymbol{\pi}^* \in D(\Theta)$ such that $\boldsymbol{\pi}' \xrightarrow{\beta} \boldsymbol{\pi}^*$ and $\boldsymbol{\pi}'' \xrightarrow{\alpha} \boldsymbol{\pi}^*$.*

The diamond property in Lemma 7 states that the policy for resolving the non-deterministic choice between two ac-

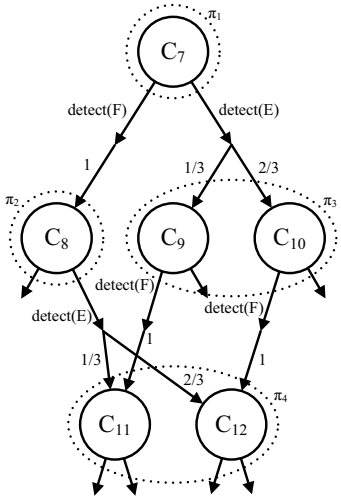



**Figure 6. Diamond property in TPLTS.**

tions that are performed immediately after each other is irrelevant. In a particular execution, two enabled actions may however not be performed immediately after each other because other actions can be enabled as well. For example, from $C_8$ in Figure 6, an end(F) action can be performed (it becomes enabled after the detect(F) action), which postpones performing the detect(E) action. The crux to proving Theorem 3 is that taking one of the enabled actions does not disable any of the others, while the others will be performed before a time transition such that the nett effect is equal to performing any of the others first. This property allows to convert any policy for resolving non-determinism into any other policy using a similar approach as in [26], without changing the sequence of tokens produced on the channels. The conversion involves interchanging the order of executing concurrent actions before the next time transition. At a time transition, no actions are enabled anymore and hence, all differences between policies have been cancelled up to this barrier. Using time transitions as a barrier allows to consider finite pieces of an execution of a timed SADF, thereby simplifying the proof. The same proof can however be extended for executions of untimed SADF models, which do not have time transitions.

## Long-Run Equivalence

Section 5 discussed how non-determinism affects some performance properties. In case non-determinism is not resolved, the TPLTS actually implies a Markov decision process [23] for which a performance metric gives rise to a collection of results [21]. The approach discussed in Section 5 is however to derive a Markov chain by resolving non-determinism explicitly such that a single result is obtained for a metric. Theorem 4 contributes that the result for a long-run average metric that merely depends on the status of the TPLTS in configurations just before and after time transitions is always the same for any policy that can be used to resolve the non-determinism.

The proof of Theorem 4 mainly relies on the result of Theorem 3 and on the way how the involved performance metrics are specified for the Markov chain implied by an SADF using temporal rewards [21] and conditional rewards [19]. The formalism of temporal rewards extends the notion of traditional atomic reward functions to enable specifying delay-type measures [22], which may express an accumulation of atomic rewards over a number of states. An example is the amount of time that has passed between two designated states. Conditional reward functions are atomic rewards indicating whether a certain event has occurred that affects the result of a performance metric. An example is the occurrence of actions regarding the writing and reading of tokens in a buffer, which are events that may affect the time-average buffer occupancy. The property of determinacy ensures that although multiple events affecting a performance metric may occur between two time transitions, the nett effect is equal to taking only the differences between the states just before and after time transitions into account. The time-average buffer occupancy is for example not affected by (the order of) timeless actions regarding the reading/writing of tokens, while only the difference in the occupancy between two time transitions is relevant.

## Ergodicity

The requirement of absence of deadlock (and boundedness) in Theorem 5 mainly ensures that an SADF indeed defines a Markov chain (with finite state space) and not some sub-stochastic process (with infinite state space). The other conditions enable to lift a result for SDF presented in [24] to SADF. [24] proofs that the transition system defined by a bounded and deadlock-free SDF implies a Markov chain consisting of a transient and recurrent part for which all one-step transition probabilities are equal to 1. The transient part originates from starting up the pipelined fashion of executing concurrent iterations, while the recurrent part reflects the periodicity of the behaviour of an SDF. This periodicity (which can be considered as a very strong form of ergodicity) originates from the guaranteed reoccurrence of actor firings and from the assumption of self-timed execution [18]. The latter assumption is equivalent to assuming maximal progress, where the firing of actors is not further postponed after sufficient tokens have become available on all input ports.

Turning back to SADF, which has maximal progress incorporated in its operational semantics, we need to show that the conditions in Theorem 5 imply the state space to include a single strongly connected component of (positive) recurrent states. To this end, we first observe that the state space of a kernel is completely determined by the state space of the detector that controls it. The conditions in Theorem 5 ensure that the Markov chains associated with detectors only consist of a strongly connected component of (positive) recurrent states. Since detectors determine their state independently of each other, the product of the state spaces of all detectors also consists only of such a component. Hence, the behaviour exhibited by all processes together ensures that the states visited after starting up the pipelined fashion of executing concurrent iterations are all (positive) recurrent. Combining this with the result in [24] forms the crux for proving Theorem 5.

## References


[22] G. Clark. Formalising the Specification of Rewards with PEPA. *Proceedings of PAPM'96*, pp 139–160, Carleton Scientific, 1996.

[23] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, New York, 1970.

[24] A.H. Ghamarian, M.C.W. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B.D. Theelen and M.R. Mousavi. Throughput Analysis of Synchronous Dataflow Graphs. In: *Proceedings of ACSD'06*, IEEE, 2006.

[25] R.M. Karp and R.E. Miller. Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing. *SIAM Journal of Applied Mathematics*, vol 14 (6), pp 1390–1411, 1966.

[26] N.A. Lynch and E.W. Stark. A Proof of the Kahn Principle for Input/Output Automata. *Information and Computation*, vol 82(1), pp 81-92, 1989.