Technische Universiteit
Eindhoven
University of Technology

Scenario-Aware Dataflow

B.D. Theelen, M.C.W. Geilen, S. Stuijk, S.V. Gheorghita, T. Basten, J.P.M. Voeten and A.H. Ghamarian

ES Reports
ISSN 1574-9517

ESR-2008-08
10 July 2008

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Scenario-Aware Dataflow

B.D. Theelen · M.C.W. Geilen · S. Stuijk · S.V. Gheorghita · T. Basten · J.P.M. Voeten · A.H. Ghamarian

Released on 10 July 2008

Abstract Dataflow models have proven their usefulness for specifying signal processing and streaming applications. However, traditional dataflow models such as Synchronous Dataflow (SDF) and Kahn Process Networks (KPN) either lack the capability of expressing the dynamic aspects of modern streaming applications or do not support desirable analysis techniques. The dynamism often originates from various modes of operation in which resource requirements differ considerably. Such scenarios cover for example variations in the amount of data to process or variations in the functionality to perform. Neglecting dynamism may lead to unrealistic performance analysis results and therefore to dimensioning the system inefficiently. To enable obtaining more realistic performance results than with traditional design-time analysable dataflow models, a scenario-aware generalisation of SDF was recently introduced. Next to representing the data processing part of an application, this Scenario-Aware Dataflow model (SADF) also captures the control part responsible for determining the various scenarios in which the data processing part may operate. Although improving the expressive power of traditional design-time analysable dataflow models with the possibility to express (more forms of) dynamism, design-time analysis of correctness and performance remains possible. This report presents SADF in detail, including all features for specifying complex correlations between scenario occurrences. The potential of using SADF for modelling and analysing modern streaming applications is illustrated for an MPEG-4 SP decoder and an MP3 decoder.

Keywords Dataflow · Synchronous Dataflow · Dynamism · Verification · Performance Analysis

1 Introduction

Streaming applications are often specified as a set of tasks, actors or processes with data and control dependencies to allow exploiting the parallel and pipelined execution capabilities of hardware platforms. Modern streaming applications are becoming increasingly dynamic, which originates from variations in the functionality to perform and in the amount and/or complexity of the data to process. Such dynamism may lead to large variations in the amount of resources required to execute an application. Various dataflow models have been proposed to specify and analyse streaming applications. They can be characterised by their expressive power and the availability of techniques to analyse correctness and performance properties like absence of deadlock and throughput. Analysing such properties is essential for designing effective and efficient streaming systems. Kahn Process Networks (KPN) [21] can capture many dynamic aspects of modern streaming systems, but evaluating their correctness and performance is in general undecidable. On the other hand, Synchronous Dataflow (SDF) [24] graphs (also known as Weighted Marked Graphs or T-Systems in Petri Net theory) allow analysis of many correctness and performance properties but lack support for expressing any form of dynamism. Cyclo-Static Dataflow (CSDF) [5] is a design-time analysable extension of SDF that can express a fixed periodic pattern in varying resource requirements. When such variations are not according to a fixed periodic pattern but probabilistic information is available on the occurrence of the various situations, the recently introduced *Scenario-Aware*

All authors are with the Eindhoven University of Technology, Department of Electrical Engineering, while J.P.M Voeten is also with the Embedded Systems Institute. The address for both institutes is: P.O. Box. 513, 5600 MB Eindhoven, The Netherlands.

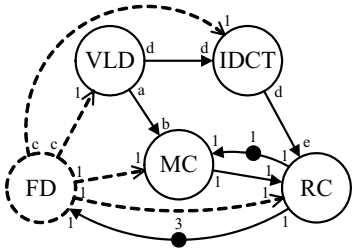
Dataflow (SADF) model [37] can be used. It captures the dynamism in modern streaming applications using *scenarios*. They denote distinct modes of operation (like processing I, P or B frames in MPEG-4), in which a process can have substantially different resource requirements [16]. Although improving the expressive power compared to SDF and CSDF, analysis of correctness and performance properties remains possible at design-time.

The advantage of using SADF instead of SDF or CSDF is the potential of obtaining more realistic performance analysis results. We illustrate the impact of dynamism on the performance with an MPEG-4 decoder for the Simple Profile. Such MPEG-4 SP decoder supports video streams consisting of I and P frames only. These frames include a number of macro blocks, each requiring operations like Variable Length Decoding (VLD), Inverse Discrete Cosine Transformation (IDCT), Motion Compensation (MC) and Reconstruction (RC) of the resulting image. When processing 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 176×144 pixels (QCIF), there are 99 macro blocks to decode for I frames. There are however no motion vectors to be taken into account for I frames. Motion vectors are only an important input to MC in case of processing a P frame that involves determining the new position of macro blocks from the previous frame. In such cases, the resulting image is corrected 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 and ranges from 0 to 99. The case of 0 motion vectors reflects decoding still video, for which VLD and IDCT must not be performed while MC simply copies the previously decoded frame. The MPEG-4 SP decoder clearly shows a lot of dynamism in the functionality to perform and in the amount of data to communicate between the operations. These aspects imply high variations in resource requirements. The order in which the different situations occur strongly depends on the video content and is therefore unlikely to be according to a fixed periodic pattern.

SDF allows constructing conservative performance models by considering the worst-case situation of all possible variations that may occur. For the MPEG-4 SP decoder, this boils down to considering the worst-case amount of macro blocks to process and communicate between processes and the worst-case execution times for each process that can ever occur. When computing the throughput of the MPEG-4 SP decoder based on an SDF graph capturing these worst-case circumstances [14], a mere 0.252525 frames are decoded per 1000 clock cycles on an ARM7TDMI processor. The result basically reflects the throughput achieved when only the worst-case circumstances occur. In reality, the MPEG-4 SP decoder does not always operate in the situation with worst-case resource requirements (actually, it never operates in the captured situation as shown in Section 2). To predict the throughput realised in practice, it is necessary to take the variations in resource requirements into account. Since these variations do not occur according to a fixed periodic pattern, CSDF is not a suitable dataflow model. On the other hand, SADF allows constructing a performance model that captures all dynamism in the MPEG-4 SP decoder to reveal that the throughput for worst-case execution times is 0.425571 frames per 1000 clock cycles (assuming the same scheduling policy and throughput definition as in the previous case of using SDF).

The MPEG-4 SP decoder example showed how the dynamism of an application depends only on one aspect, namely the frame type. In other applications, there may exist different aspects that determine the mode in which they operate. The basic SADF model as defined in [37] allows capturing such cases as long as no correlation exists between the various aspects. To illustrate that this is not always realistic, we consider the example of MP3 decoding. MP3 audio streams consist of frames of three different types (Long, Short and Mixed) [33]. Each of these frame types consists of different numbers and orderings of two block types (Long and Short). Long frames consist of 32 Long blocks, Short frames include 96 Short blocks and Mixed frames are composed of 2 Long blocks succeeded by 90 Short blocks. The frame type and block type together determine the operation mode. Assuming that the frame types and specific block type sequences are not correlated leads to unrealistic analysis results since it allows the occurrence of any of the three possible block type sequences for all frame types. The occurrences of the block types is however a refinement of the frame type occurrences. This observation matches intuitively with the structure of the application's source code. We use the term *hierarchical control* to express this kind of correlations between different aspects determining the scenario. It reflects that the frame type determines the block type orderings, which then give the scenario in which processes operate. Next to introducing hierarchical control, this report relaxes some technical restrictions of [37]. In addition, we discuss key properties of the transition system defined by the formal semantics of an SADF graph and illustrate how SDF and CSDF graphs can be expressed in SADF. Finally, we elaborate on the SADF graph modelling the MPEG-4 SP decoder that is only superficially described in [37], while modelling the MP3 decoder is a new case study.

The remainder of this report is organised as follows. The next Section informally introduces SADF by explaining how the MPEG-4 SP decoder and MP3 decoder can be modelled. It also illustrates how the CSDF



a) Processes and Channels

Rate	(Sub)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

$x \in \{30, 40, 50, 60, 70, 80, 99\}$

b) Parameterised Rates

Process	(Sub)Scenario	E (kCycles)
VLD	P_0	0
	All except P_0	40
IDCT	P_0	0
	All except P_0	17
MC	I, P_0	0
	P_{30}	90
	P_{40}	145
	P_{50}	190
	P_{60}	235
	P_{70}	265
	P_{80}	310
	P_{99}	390
RC	I	350
	P_0	0
	P_{30}, P_{40}, P_{50}	250
	P_{60}	300
	P_{70}, P_{80}, P_{99}	320
FD	All	0

c) Maximum Execution Times

Fig. 1 Modelling the MPEG-4 SP decoder with SADF.

graph for a Channel Equalizer can be captured in SADF. Section 3 discusses related research on dataflow models to complete motivating the introduction of SADF. Readers interested in the formal definition of SADF and theoretical background of the semantics may consider Section 4. In Section 5, we summarise the conclusions.

2 Example SADF Graphs

This section informally introduces SADF by showing how some real-life streaming applications could be modelled. A formal definition of SADF is given in Section 4, which includes all considerations underlying its design.

2.1 MPEG-4 SP Decoder

Figure 1a depicts an SADF graph for the MPEG-4 SP decoder. The vertices denote the tasks or *processes* of the application. Two types of processes are distinguished. *Kernels* (solid vertices) model the data processing part of a streaming application, whereas *detectors* (dashed vertices) represent the part that dynamically determines scenarios and controls the behaviour of processes. The Frame Detector (FD) represents for example that segment of the actual VLD code responsible for determining the frame type and the number of macro blocks to decode. For simplicity, we assume an equal number (0 or $x \in \{30, 40, 50, 60, 70, 80, 99\}$) of motion vectors and macro blocks to decode. The possible values of x denote conservative approximations for a range of different numbers of motion vectors that can occur in reality. The SADF graph in Figure 1 captures the varying resource requirements in 9 different scenarios (one for each possible combination of frame type and number of macro blocks to decode).

The edges in an SADF graph are called *channels*. They denote (potential) dependencies between processes originating from exchanging information. SADF distinguishes *data* channels (solid edges) and *control* channels (dashed edges). A *token* is a unit of information that is communicated via a channel. The availability of tokens in the (in principle unbounded) FIFO buffer corresponding to a channel is indicated with a dot. Control channels communicate tokens that are scenario-valued, while tokens on data channels are non-valued as in SDF and CSDF. Dots on data channels are labelled with the number of tokens that is available, while a dot on a control channel is labelled with the sequence of scenario values of the tokens in the corresponding FIFO buffer. The token on the channel from RC to MC in Figure 1a models for example exchanging the previously decoded frame. The three tokens on the channel from RC to FD capture the capability of decoding up to 3 frames in a pipelined fashion.

The start and end points of channels are labelled with *production* and *consumption rates* respectively. They refer to the number of tokens that is atomically produced/consumed by the corresponding process upon its execution or *firing*. In SADF, rates can be fixed (as in SDF) or parameterised. Fixed rates are shown with positive integer values, while parameterised rates are valued with non-negative integers that may vary for different scenarios. The parameterised rates for the MPEG-4 SP decoder are listed in Figure 1b. It shows that a parameterised rate can be 0 in certain scenarios. Such cases express that data dependencies are absent or that the functionality of a kernel must not be performed in those scenarios. An example is skipping VLD and IDCT when decoding P frames for still video, which is modelled by scenario P_0. Carefully studying the rates in Figure 1 reveals that the VLD and IDCT kernels fire once per decoded macro block, while the MC and RC kernels fire once per frame.

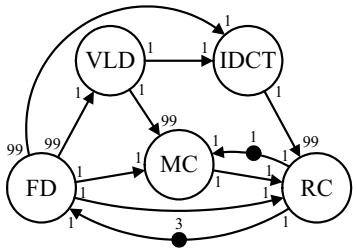
Next to the possibility of skipping operations in certain scenarios, the computation resource requirements may also depend on the data that is being processed. Where the concept of scenarios allows expressing variations in communication resource requirements and coarse-grain changes in computation resource requirements, SADF allows modelling fine-grain variations in computation resource requirements by means of assigning execution time distributions to processes. Table 1c shows that such distributions can be point distributions, which yields a fixed execution time per scenario[1]. We can now summarise that a scenario refers to a set of values for parameterised rates and to a certain execution time distribution. A key feature of the scenario concept is the ability to capture correlations that often exist between the dynamic changes in resource requirements for different processes. In the MPEG-4 SP decoder for example, the frame type coherently affects the rates and execution time distributions of the VLD, IDCT, MC and RC operations as modelled by the SADF graph in Figure 1.

The scenario in which a kernel will operate is determined externally by the detector(s) controlling it. For detectors, the story is somewhat more complicated. The operation mode of a detector is established only partly externally (in case it has control inputs). Next to establishing the scenario of the VLD, IDCT, MC and RC kernels, detector FD also affects its own mode of operation. To distinguish the internal and external forces determining the operation mode of a detector, we use the term *subscenario* to refer to those aspects that are established internally. For detectors, it is actually the subscenario that refers to a set of values for parameterised rates and a certain execution time distribution. The set of subscenarios for detector FD is equivalent to the set of scenarios for the VLD, IDCT, MC and RC kernels. The control tokens that FD produces on its outputs are valued equal to the subscenario in which it operated. This legitimates the use of the summarising tables in Figures 1b and 1c to specify the parameterised rates and execution time distributions. A more detailed explanation of the difference between scenarios and subscenarios for detectors is given in Section 2.2 for the MP3 decoder example.

What remains to be specified for the SADF graph in Figure 1 is how the subscenarios of detector FD are established. In reality, the frame type and number of macro blocks is determined based on the content of the video stream. Because SADF abstracts from such content, an abstract mechanism is needed to capture the possible occurrences of scenarios. Various types of automata can be exploited for this. We follow the proposal of [37] to use a (discrete-time) Markov chain [8] as it facilitates both best/worst-case and average-case performance analysis. Such a Markov chain can be seen as a state machine, where each state refers to a specific subscenario. The transitions occur according to a certain probability, which reflects what subscenario may occur in the next firing and with what probability. For the MPEG-4 SP decoder, we use a Markov chain with 9 states to uniquely determine the subscenario in which detector FD is going to operate and hence, also the scenario in which the VLD, IDCT, MC and RC kernels will operate. Assuming that the frame types and numbers of macro blocks may occur in any order, the Markov chain associated with FD resembles a fully connected graph. The transition probabilities are specified such that the probability of being in a certain state (the equilibrium probability of that state) equals the probability that the involved frame type and number of macro blocks occur. The throughput result reported in the Introduction for the SADF graph in Figure 1 is obtained assuming that the occurrence probabilities of the (sub)scenarios $I, P_0, P_{30}, \ldots, P_{99}$ equals $3/25, 1/50, 1/20, 1/4, 1/4, 9/100, 9/100, 9/100, 1/25$ respectively.

We briefly describe the behaviour of the SADF graph in Figure 1. The firing of detector FD starts with determining the subscenario (independent of the availability of tokens in this case). It is given by the state that is entered after making one transition in the Markov chain. By establishing the subscenario, FD fixes the parameterised production rate c according to Figure 1b. Then, FD waits until at least 1 token is available at its input. When a token has become available, FD performs its behaviour which takes an amount of time given by a sample drawn from the execution time distribution corresponding to the subscenario in which FD operates (being

[1] Table 1c lists the maximum value of the original execution time distributions obtained by profiling the MPEG-4 SP decoder on an ARM7TDMI processor. The results in [35] are based on the mean value of the original distributions, which have been used in [37].



a) Processes and Channels

Process	E (kCycles)
VLD	40
IDCT	17
MC	390
RC	350
FD	0

b) Maximum Execution Times

Fig. 2 Approximating the worst-case behaviour of the MPEG-4 SP decoder in SDF (auto-concurrency is excluded implicitly).

0 in all cases). Firing FD ends with removing 1 token from the input and writing a number of tokens to each output. The number of produced tokens corresponds to the production rates as established by the subscenario. The produced control tokens are valued with the subscenario in which FD has operated.

The firing of a kernel like VLD, IDCT, MC and RC starts with reading 1 token from the control input when it becomes available. The value of that token determines the scenario in which they will operate. It results in fixing the parameterised rate a, b, d and e and the execution time distributions. After establishing the scenario, a kernel waits until sufficient tokens are available at every data input in accordance with the now known consumption rates. At the moment that sufficient tokens have become available, the data processing behaviour is performed. It takes an amount of time given by a sample drawn from the previously established execution time distribution. Firing ends with removing a number of tokens from each data input and writing a number of tokens to each output, where these numbers correspond to the consumption and production rates as established based on the scenario. In addition, one token is removed from the control input. Notice that blocking the firing until sufficient tokens are available implies that processes may be operating in different scenarios at a given point in time.

The example in Figure 1 illustrates that SADF can capture a collection of SDF graphs for a single application operating in different modes, while transitions between scenarios occur in a pipelined fashion. The key difference between SADF and SDF is the establishment of the (sub)scenario in which a process operates as an additional step in firing a process. In fact, an SADF graph with only kernels and point distributions for the execution times is equal to an SDF graph. Conversely, any SDF graph that excludes auto-concurrency is an SADF graph. The phenomenon of auto-concurrency refers to the possibility of firing a process multiple times simultaneously. The operational semantics of SADF excludes such auto-concurrency in order to ensure determinacy, see Section 4.3.

We conclude this Section with the SDF graph in Figure 2. It concerns a straightforwardly constructed SDF graph following the traditional approach to approximate the worst-case resource requirements for the MPEG-4 SP decoder. The worst-case amount of communication resources occurs for scenario P_{99}, in which all processes and channels are used. The worst-case execution time for kernels VLD and IDCT as well as MC are in scenario P_{99}, while for RC it is scenario I. The throughput result mentioned in the Introduction is obtained based on the SDF graph in Figure 2 with these execution times. From the SADF graph in Figure 1, it is however clear that this combination of execution times can never occur. Hence, the SDF approximation in Figure 2 is inadequate.

2.2 MP3 Decoder

MP3 decoding is a frame-based algorithm, which transforms the incoming compressed bitstream into normal pulse code modulated data. Each frame consist of 1152 mono or stereo frequency components. These are divided into two granules of 576 components, which are further divided into 32 subbands of 18 frequency components each. The algorithm includes operations like Huffman Decoding (H), Requantisation (RQ), Reorder (RO), Stereo (S), Alias Reduction (AR), Inverse Modified Discrete Cosine Transform (IMDCT), Frequency Inversion (FI), Synthesis Polyphase Filterbank (SPF), and Write (W) back of decoded sound samples [33]. Most of these operations are to be performed for both left and right sound channels. Figure 3 depicts an SADF graph that models the MP3 decoder, where the subscripts L and R in the process names indicate the involvement in the left

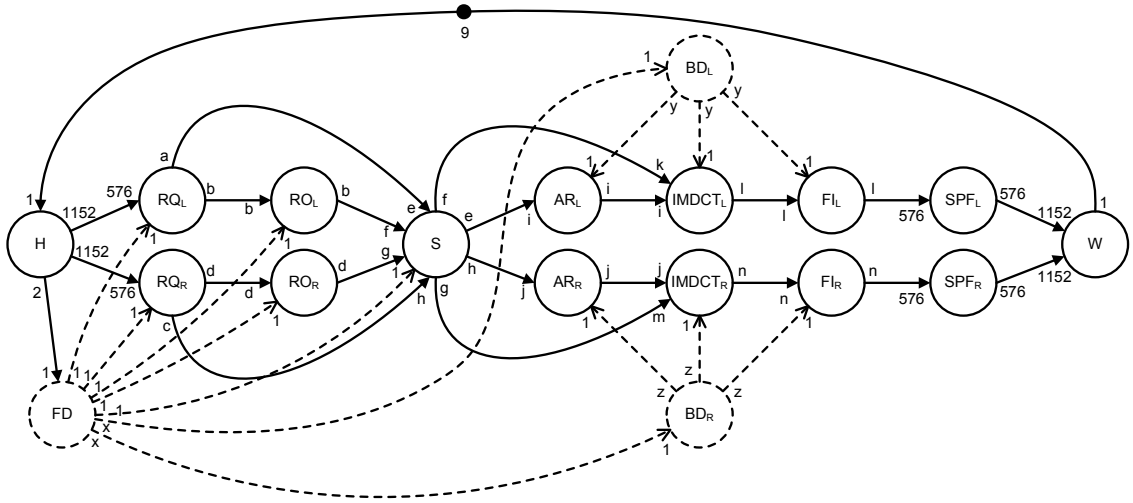


Fig. 3 Modelling an MP3 decoder with SADF using hierarchical control.

or right sound channel respectively. A first observation is that kernels HM, $\mathrm{SPF_L}$, $\mathrm{SPF_R}$ and W have no control inputs, which means that they always operate in the same default scenario (which is indicated with ϵ). The Frame Detector (FD) and Block Detectors (BD) are responsible for determining the scenarios in which the other kernels operate. Observe that $\mathrm{BD_L}$ and $\mathrm{BD_R}$ are controlled by FD. Figure 4 lists the parameterised rates, execution time distributions (which are point distributions[2] in this example) and the properties for detectors FD, $\mathrm{BD_L}$ and $\mathrm{BD_R}$.

As explained in the Introduction, three frame types (Long, Short and Mixed) and two block types (Long and Short) are distinguished. The frame type determines the main characteristics of the MP3 decoder's operation mode while the block type refines those characteristics to identify the operation mode precisely. In practice, only 5 different combinations of frame types occur for the two sound channels. Using a two-letter combination to indicate the fame type for the left and right sound channel respectively, we identify the situations LL, SS, LS and SL. The fifth situation is named M and denotes that a Mixed frame must be processed for both sound channels. Detector FD (which has no control inputs) determines the frame type based on a Markov chain with 5 states. Each of these states uniquely identifies a subscenario in $\{\mathrm{LL}, \mathrm{SS}, \mathrm{LS}, \mathrm{SL}, \mathrm{M}\}$. The transition matrix for the Markov chain associated to FD is given in Figure 4c, which shows that the occurrence of Mixed frames is very rare. Kernel S requires knowledge about the frame types for both sound channels and therefore operates according to a scenario in $\{\mathrm{LL}, \mathrm{SS}, \mathrm{LS}, \mathrm{SL}, \mathrm{M}\}$ that equals the subscenario determined by FD. The scenario of kernels $\mathrm{RQ_L}$, $\mathrm{RO_L}$ and $\mathrm{RQ_R}$, $\mathrm{RO_R}$ is only determined by the frame type for the left and right sound channel respectively. The set of scenarios for these kernels is $\{\mathrm{S}, \mathrm{M}, \mathrm{L}\}$. They basically receive control tokens from FD valued with the left or right letter in the combinations LL, SS, LS, SL or with M. This illustrates that the produced control tokens may be valued differently from the set of subscenarios of a detector (which is another extension of [37]).

The scenario of kernels $\mathrm{AR_L}$, $\mathrm{IMDCT_L}$, $\mathrm{FI_L}$ and $\mathrm{AR_R}$, $\mathrm{IMDCT_R}$, $\mathrm{FI_R}$ merely requires knowledge about the block type for the left and right sound channel respectively. Their set of scenarios is $\{\mathrm{BL}, \mathrm{BS}\}$, where BL and BS denote Long and Short blocks respectively. A Long block contains 18 frequency components, while a Short block includes only 6 components. Detectors $\mathrm{BD_L}$ and $\mathrm{BD_R}$ identify the appropriate number and order of Short and Long blocks based on control tokens valued L, S or M received from FD. From the perspective of $\mathrm{BD_L}$ and $\mathrm{BD_R}$, the two different block types BL and BS are subscenarios of the scenarios S, M and L established externally by detector FD. Multiple Markov chains are associated with detectors that have control inputs, one for each possible scenario in which they can operate. Figure 4d gives graphical representations of the three Markov chains associated with both $\mathrm{BD_L}$ and $\mathrm{BD_R}$. Each of their states implies one of the possible subscenarios in $\{\mathrm{LBL}, \mathrm{SBS}, \mathrm{MBL}, \mathrm{MBS}\}$. The control tokens produced by $\mathrm{BD_L}$ and $\mathrm{BD_R}$ to kernels $\mathrm{AR_L}$, $\mathrm{IMDCT_L}$, $\mathrm{FI_L}$ and $\mathrm{AR_R}$, $\mathrm{IMDCT_R}$, $\mathrm{FI_R}$ respectively in each of the 4 possible subscenarios matches the last two letters of the subscenario name. Although subscenarios LBL and MBL both imply sending control tokens valued BL, the difference between them is the number of such tokens, see Figure 4a. A similar observation holds for subscenarios SBS

[2] Table 4b shows the mean value of the execution time distributions obtained by profiling the MP3 decoder on an ARM7TDMI.

Rate	Scenario L	Scenario S	Scenario M
a, c	576	0	36
b, d	0	576	540

Rate	(Sub)Scenario LL	SS	LS	SL	M
e	0	576	0	576	36
f	576	0	576	0	540
g	0	576	576	0	540
h	576	0	0	576	36
x	1	1	1	1	2

Rate	Scenario BL	BS
i, j	18	0
k, m	0	6
l, n	18	6

Rate	SubScenario LBL	SBS	MBL	MBS
y, z	32	96	2	90

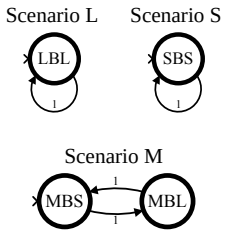
a) Parameterised Rates

Process	(Sub)Scenario	E (Cycles)
H	ϵ_{H}	151977
RQ	L	56431
	S	72695
	M	42911
RO	L	0
	S	34684
	M	17568
S	LL	39763
	SS	38832
	LS, SL	19
	M	53602
AR	BL	409
	BS	0
IMDCT	BL	7414
	BS	5561
FI	BL, BS	4912
SPF	ϵ_{SPF}	1865001
W	ϵ_{WB}	42896
FD	All	0
BD	All	0

b) Mean-Value Execution Times

	LL	SS	LS	SL	M
LL	0.9605	0.0345	0.0029	0.0020	0.0001
SS	0.6267	0.3482	0.0125	0.0125	0.0001
LS	0.7154	0.1428	0.1223	0.0194	0.0001
SL	0.6873	0.1422	0.0282	0.1422	0.0001
M	0.9605	0.0345	0.0029	0.0020	0.0001

c) Markov chain associated with detector FD



d) Markov chains associated with detectors $\mathrm{BD_L}$ and $\mathrm{BD_R}$

Fig. 4 Properties of the SADF graph modelling the MP3 decoder in Figure 3.

and MBS. The behaviour of detectors with control inputs like $\mathrm{BD_L}$ and $\mathrm{BD_R}$ is as follows. Firing starts at the moment that a token becomes available at the control input(s). Their value determines the scenario by means of selecting the corresponding Markov chain. The subscenario is determined by the state visited after making a transition in the selected Markov chain. After establishing the scenario and subscenario, firing continues as described previously. We exemplify the start of a firing by assuming that an M-valued token is available on the control port of detector $\mathrm{BD_L}$. Interpreting this token leads to one transition in the 2-state Markov chain shown in Figure 4d. In case this Markov chain was in state MBS, receiving the M-valued token leads to transiting to state MBL with probability 1 (and hence, fixing MBL as the subscenario). Notice that detector FD will always sent two M-valued tokens. As a result, $\mathrm{BD_L}$ will fire twice according to scenario M. Considering the 2-state Markov chain in Figure 4d, $\mathrm{BD_L}$ establishes subscenario MBL for its first firing and subscenario MBS for the second firing. In subscenario MBL, detector $\mathrm{BD_L}$ submits 2 tokens valued BL to kernel $\mathrm{AR_L}$, $\mathrm{IMDCT_L}$ and $\mathrm{SPF_L}$, while 90 tokens valued BS are produced in subscenario MBS. Hence, the model captures that $\mathrm{AR_L}$, $\mathrm{IMDCT_L}$ and $\mathrm{SPF_L}$ first process 2 Long blocks and subsequently 90 Short blocks for each Mixed frame. This behaviour matches with processing the 92 blocks that constitute a Mixed frame as described in the Introduction. Notice that the link between the value of tokens received on the control port of detector $\mathrm{BD_L}$ and the three Markov chains (through scenarios) represents in fact a choice between mutual exclusive options. This reflects the deterministic decisions made by the actual implementation of the detector based on the value of the received control tokens.

The MP3 decoder example shows that an SADF graph can be more than a collection of SDF graphs with similar structure. Separately considering the scenarios that model decoding Long and Short frames reduces the SADF graph into similarly structured SDF graphs. This is however not true for the scenario of decoding Mixed frames due to processing different block types. It is possible to describe the scenario of decoding Mixed frames

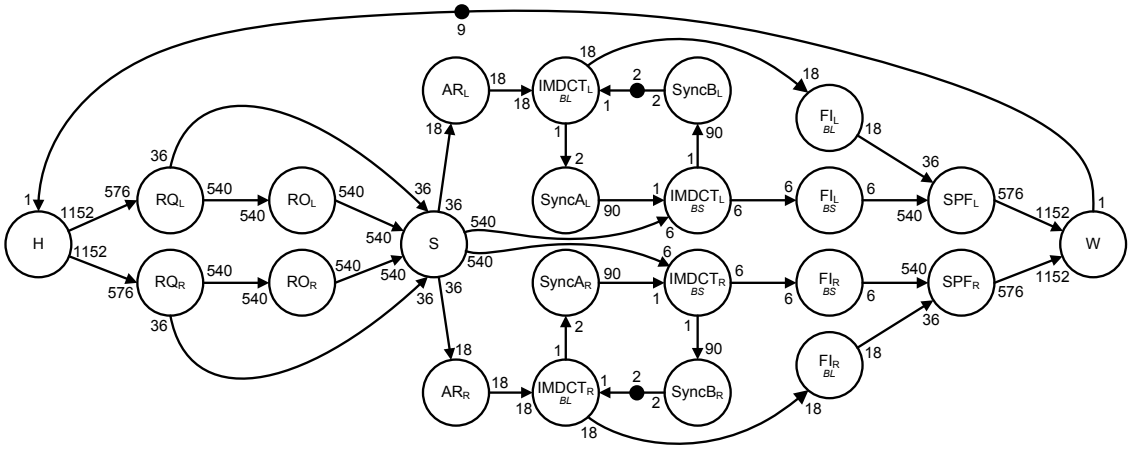


Fig. 5 SDF graph capturing the decoding of Mixed frames by the MP3 decoder (execution times of Sync_L and Sync_R are 0).

by means of an SDF graph, see Figure 5. To capture the processing of the Long and Short block types for Mixed frames correctly, duplicates of IMDCT and FI are needed since they behave differently for these two cases. In addition, processes SyncA_L, SyncB_L, SyncA_R and SyncB_R (all with execution time 0) are introduced to express similar pipelining behaviour as the SADF graph in Figure 3 exhibits. The SyncA processes ensure that performing IMDCT for the first Short block does not start before finishing IMDCT on the last Long block for each granule. On the other hand, the SyncB processes ensure that performing IMDCT for the second granule in a frame does not start before finishing IMDCT for the first granule. Notice that modelling the scenario of decoding Mixed frames in SADF could be based on the SDF graph in Figure 5. However, the necessity of introducing various non-intuitive artifacts would yield an SADF graph with a structure that does not match with the structure of the application's source code. Using hierarchical control instead circumvents this disadvantage.

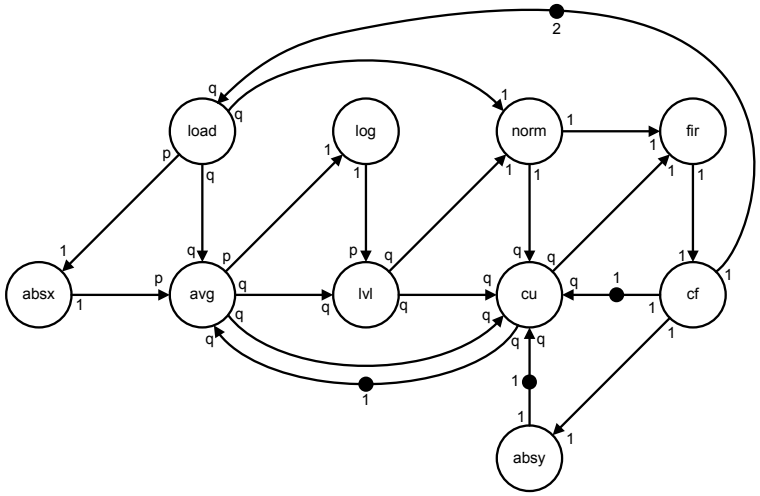
2.3 Channel Equalizer

SADF can be used as an alternative to other dataflow models that allow expressing dynamism. This is useful in case analysis is not or insufficiently supported. Examples of dataflow models that can be expressed succinctly in SADF include Cyclo-Static Dataflow (CSDF) [5] and certain forms of Boolean Dataflow (BDF) [6]. SADF can also be used to develop analysis models of for example Kahn Process Networks (KPN) [21] and Reactive Process Networks (RPN) [13]. A more elaborate comparison of SADF with these and other dataflow models is given in Section 3. This Section discusses how CSDF graphs can be expressed straightforwardly in SADF.

Figure 6 depicts the CSDF graph of a Channel Equalizer application as described in [28] (slightly modified). The processes load, avg, lvl and cu exhibit cyclo-static behaviour consisting of 8 phases, while the other processes do not include dynamism. The abbreviation p refers to the sequence [1, 0, 0, 0, 0, 0, 0, 0] of rates in the subsequent phases, while q is shorthand notation for the sequence [1, 1, 1, 1, 1, 1, 1, 1]. Figure 6b gives the (sequences of) execution times, where for example 8*224 is short for 224, 224, 224, 224, 224, 224, 224, 224.

Converting a CSDF graph into an SADF graph is based on capturing the phases of processes with the concept of scenarios. In this way, the conversion merely involves replacing CSDF processes that exhibit dynamism with a detector. Such detector includes one Markov chain with a state for each phase. The Markov chain uses only transition probabilities of 1 to resemble a strongly connected graph that expresses the cyclo-static sequence of phases. Hence, the phase in which a CSDF process is becomes explicit as state in the Markov chain. Notice that several phases for the CSDF graph of the Channel Equalizer imply the same behaviour. By associating the same subscenario to the involved states of the Markov chain, one only has to specify each different behaviour once.

Although the described procedure converts any CSDF graph into an SADF graph, the correlation between the behaviours of the different detectors remains implicit. In case processes have the same sequence length, it is easy to make the correlation explicit in the SADF graph. This is shown in the SADF graph of Figure 7, where process load is replaced by a detector that coherently controls the behaviour of kernels avg, lvl and cu. Detector



a) Processes and Channels

Process	E
load	[8*224]
absx	114
avg	[704, 7*416]
log	114
lvl	[440, 7*24]
norm	328
cu	[4944, 7648, 6*4966]
fir	144
cf	328
absy	144

b) Execution Times

Fig. 6 CSDF graph modelling a Channel Equalizer [28].

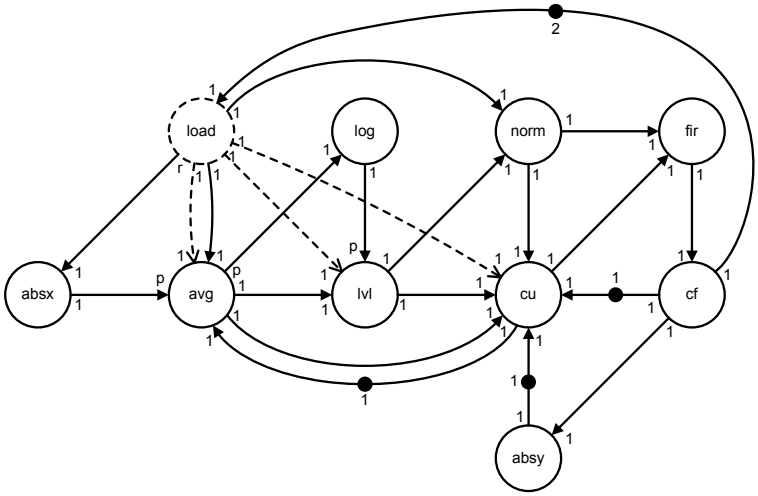
load captures the three different behaviours for the 8 phases in subscenarios P0, P1 and P2-7. Here, P0 represents the first phase, P1 the second phase and P2-7 the 6 other phases. Kernel cu has the same three scenarios in which only its execution times differ. On the other hand, kernels avg and lvl have only two different behaviours that are captured by scenarios P0 and P1-7. These scenarios model the behaviour of the involved kernels in the first phase and the other 7 phases respectively. Detector load values control tokens that are sent to these kernels with P0 in case of operating in subscenario P0 and with P1-7 when operating in subscenarios P1 and P2-7. Notice that processes absx and log only fire in the first phase of the CSDF graph due to not receiving a token on their input in any of the other phases. This behaviour is captured similarly in the SADF graph of Figure 7.

The Channel Equalizer example showed that CSDF graphs can be converted (in various ways) into SADF graphs. A conversion in the other direction is less trivial. Even the subset of SADF that excludes hierarchical control and probabilistic choices (i.e., all execution time distributions are point distributions and all transition probabilities for the Markov chains associated to detectors are 1) is more expressive than CSDF. This originates from the possibility of a behavioural prologue due to initial control tokens and transient states in the Markov chains. Such behavioural prologue may not match with the cyclo-static behaviour that can be captured in CSDF.

3 Related Research

When comparing SADF with traditional dataflow models, the most notable differences are the support for parameterised rates, the explicit support for correlated execution time distributions for different processes (by means of the scenarios and hierarchical control) and the use of a stochastic approach to model the scenario occurrences. In this section, we consider earlier research related to these topics and other work related to semantical aspects.

Many generalisations of SDF have been proposed before, often focussing on support for dynamically changing rates. Section 2.3 already compared SADF with CSDF. In Scalable Synchronous Dataflow (SSDF) [30], integer multiples of the rates in a consistent SDF [24] can be used, where these multiples are determined by an external scheduler. SADF can express the occurrence of the different rates by using scenarios and modelling the external scheduler with a detector. In [4], a meta-modelling approach was introduced that allows extending dataflow models like SDF with parameterised rates. Application of this approach to SDF is known as the Parameterised Synchronous Dataflow (PSDF) model, which is also formalised in [4]. SSDF and PSDF require parameterised rates to be valued with positive integers while the homogeneous version of PSDF additionally requires the parameter value of the consumption and production rates of a specific channel to be equal. SADF (and also CSDF) is more expressive than PSDF by supporting the value 0 for parameterised rates in certain scenarios.



a) Processes and Channels

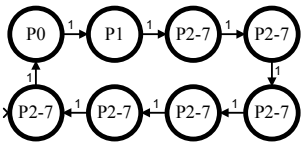
Process	(Sub)Scenario	E
load	All	224
absx	ϵ_{absx}	114
avg	P0	704
	P1-7	416
log	ϵ_{log}	114
lvl	P0	440
	P1-7	24
norm	ϵ_{norm}	328
cu	P0	4944
	P1	7648
	P2-7	4966
fir	ϵ_{fir}	144
cf	ϵ_{cf}	328
absy	ϵ_{absy}	144

b) Execution Times

Rate	SubScenario P0	P1	P2-7
r	1	0	0

Rate	Scenario P0	P1-7
p	1	0

c) Parameterised Rates



d) Detector load

Fig. 7 SADF graph modelling the Channel Equalizer in Figure 6 showing scenario correlations explicitly.

Boolean Dataflow [6] and Integer Controlled Dataflow [7] follow the approach of restricting KPN such that the control part of an application is captured by certain predefined process types. Well-behaved forms like the one in [18] even allow for limited analysis. These dataflow models can capture scenarios and the corresponding correlation between processes for each scenario by using subgraphs that are enabled by control tokens, similarly as how Control Flow Graphs describe possible paths through the application code. Instead of using the mentioned dataflow models, one could also use Conditional Process Graphs [11] to express scenarios by using the same approach. However, the resulting models for this approach will in general have many more processes than an equivalent SADF graph, which avoids duplicating a process that participates differently in multiple scenarios.

Another way to extend SDF for expressing dynamism is described in [17,23], where a Finite State Machine (FSM) is associated to each process of an SDF. The approach is similar to how SADF associates Markov chains to detectors. FSMs require however to specify the internal behaviour of processes in full detail of the implementation, i.e., one cannot abstract from data-dependent conditions based on which choices between alternative behaviours are made. In addition, the approach lacks a way to explicitly indicate correlations between the behaviour of different processes. [17,23] also propose the opposite; the association of an SDF graph to each state of an FSM. This enables representing scenarios with separate SDF graphs that are combined via a detector-like approach described with the FSM. Disadvantages include the need to specify multiple copies of a process that participates in different scenarios and a pipelined transition from one scenario to another is not supported. SADF expresses such pipelined reconfiguration by default. A very general model of computation related to hierarchically combining state machines with processes is that of Reactive Process Networks (RPN) [13]. It concerns a non-deterministic extension of KPN that allows capturing the occurrence of sporadic (control) events. Evaluating the correctness and performance of applications expressed as RPN is in general undecidable. SADF is a deterministic dataflow model [37] that requires continuous communication of control tokens, even if no scenario changes occur. This is similar to the approach in for example [10]. By making adequate abstractions, SADF can straightforwardly serve as a basis for developing analysis models of RPNs.

Several researchers proposed to use probabilities for modelling dynamism in an abstract way as well as to support extensive performance analysis. We are not aware of previous research that uses Markov chains to capture the possible orders of different behaviours in dataflow models. On the other hand, the suggestion of using (independent) discrete execution time distributions for SDF is not new. The approach was however declared infeasible for practical applications in [32] 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 are in practice usually not independent. Although such correlations limit the state space in comparison to using independent distributions, the impact of multiple possible execution times per scenario still contributes considerably to state-space explosion. Nevertheless, for some examples, the state-space explosion is manageable by applying on-the-fly state-space reduction techniques as implemented in the model checking tools for SADF graphs [35]. To circumvent state-space explosion problems, [32] suggested to use (independent) exponentially distributed execution times for SDF. These distributions have been extensively studied in the context of formalisms other than dataflow models such as stochastic process algebras like PEPA [20] and EMPA [3], stochastic Petri Nets [27] (including stochastic forms of Weighted Marked Graphs or T-Systems) as well as queueing networks [2]. Similarly to SADF, all these formalisms implicitly define Markov processes as the means for evaluating performance metrics [37]. However, by supporting exponential distributions only, these formalisms disable for example worst-case timing analysis since the sample spaces of exponential distributions are unbounded. On the other hand, there exist several queueing network based approaches that support general distributions. We are however unaware of approaches that allow taking correlations between such distributions associated to concurrent processes (i.e. different servers in a queueing network) into account.

The semantic model of SADF is that of Timed Probabilistic Systems (TPS) [1,31], which refers to a variant of Markov Decision Processes [9] that explicitly distinguishes actions and time steps. This semantic model forms the mathematical foundation for evaluating correctness and/or performance properties [37,35]. The TPS defined by an SADF graph may include both non-deterministic and probabilistic choices for actions, whereas no such choices exist for time steps (See Section 4.3). EMPA and the generalised form of stochastic Petri Nets in [26] also decouple actions and time, but they associate stochastic (and non-deterministic) choices with progress in time instead. Probabilistic Timed Automata (PTA) [22] and the Parallel Object-Oriented Specification Language (POOSL) [36] use the same semantic model as SADF to found performance evaluation. Performance analysis with PTA as discussed in [22] is limited to metrics that can be described as a (probabilistic or expected) reachability property, while actually performing such analysis involves restricting the time domain of PTA to a finite set of possible values. SADF limits its time domain similarly by definition. On the other hand, SADF allows analysis of a larger class of metrics [35], including delay-type long-run metrics like throughput and jitter. Such metrics can also be evaluated when using POOSL [38,34], which is much more expressive than SADF. Due to this expressive power, POOSL models generally have an exceptionally large or even infinite state space, which renders exact analysis infeasible. On the other hand, their alternative of simulation-based performance analysis relies on assuming ergodicity [34]. For an SADF graph, verifying properties like the finiteness of the state space and ergodicity can be based on structural analysis of its specification as briefly discussed in [37].

In summary, SADF allows for compact models of modern streaming applications with varying resource requirements, which are design-time analysable for a wide range of correctness and performance properties. No other dataflow model combines these capabilities with using stochastic abstractions and the concept of scenarios.

4 Formal Definition

This section formally defines SADF, thereby extending the basic model introduced in [37]. We discuss both the syntactical definition of SADF graphs and how the operational semantics defines Timed Probabilistic Systems.

4.1 Preliminaries

To simplify the mathematical notation, we consider processes (kernels and detectors) to be connected through *ports*. Three types of ports are distinguished; *(data) input ports*, *output ports* and *control (input) ports*. The finite sets of input, output and control ports of a process p are denoted by $\mathcal{I}_p$, $\mathcal{O}_p$ and $\mathcal{C}_p$ respectively. All port sets are pairwise disjoint and we define $\mathcal{I}$, $\mathcal{O}$ and $\mathcal{C}$ to be the union of all input, output and control port sets respectively.

A channel reflects an unbounded FIFO buffer that uniquely connects an output port to either an input port (data channel) or a control port (control channel). A channel that connects ports of the same process p is also called a *self-loop* channel of p. For any channel b, Σ_b indicates the finite set of all possible values of the tokens that b can transfer. Σ_b^* denotes the set of all finite sequences of the tokens in Σ_b. For a sequence $\sigma = \sigma_1 \dots \sigma_n$ in Σ_b^*, the i^{th} token in σ is indicated with σ_i, whereas $|\sigma|$ is the number n of tokens in σ. For sequences $\sigma, \tau, \upsilon \in \Sigma_b^*$, we use $\sigma + \tau$ to represent the concatenation of σ and τ. In case $\upsilon = \sigma + \tau$, we also write $\upsilon - \sigma$ to denote τ.

The non-empty finite set of scenarios in which a process p can operate is denoted by $\mathcal{S}_p$. A detector d also has a non-empty finite set of subscenarios Ω_d. Using $\mathbb{N} = \{0, 1, \dots\}$ to represent the natural numbers, function $R_k^s : \mathcal{I}_k \cup \mathcal{O}_k \cup \mathcal{C}_k \to \mathbb{N}$ specifies for scenario $s \in \mathcal{S}_k$ the rates of a kernel k. For subscenario $\omega \in \Omega_d$ of a detector d, function $R_d^\omega : \mathcal{I}_d \cup \mathcal{O}_d \cup \mathcal{C}_d \to \mathbb{N}$ gives the rates. The sequence of tokens d produces when operating in subscenario ω onto an output port $o \in \mathcal{O}_d$ connected to a control channel b is denoted by $t_d^\omega(o)$ and is in Σ_b^*.

The execution time distribution of a kernel k for scenario $s \in \mathcal{S}_k$ is captured by discrete random variable E_k^s. The probability that E_p^s is equal to a certain value e in its sample space is denoted by $\mathbb{P}(E_p^s = e)$. For a detector d, discrete random variable E_d^ω captures the execution time distribution for subscenario $\omega \in \Omega_d$. Similarly as for kernels, $\mathbb{P}(E_d^\omega = e)$ denotes the probability that E_d^ω equals value e in its sample space. The sample spaces of each random variable E_k^s and E_d^ω is a finite subset of the non-negative real numbers $\mathbb{R}_0^+$. Using random variables with finite sample spaces is a prerequisite for obtaining a finite state space. In addition, it ensures the existence of lower and upper bounds on execution times, which is required to facilitate best/worst-case performance analysis.

As discussed in Section 2, we choose to associate finite-state discrete-time Markov chains [8] to detectors for capturing the occurrences of subscenarios. In this report, we denote a Markov chain with a tuple $(\mathbb{S}, \iota, \mathbb{P})$, where $\mathbb{S}$ is a non-empty finite state-space, $\iota \in \mathbb{S}$ denotes the initial state from which the Markov chain departs with probability 1 and $\mathbb{P}$ is the matrix of one-step transition probabilities such that $\sum_{y \in \mathbb{S}} \mathbb{P}(x, y) = 1$ for all $x \in \mathbb{S}$.

4.2 Scenario-Aware Dataflow

To define SADF, we first introduce the functions ϕ and ψ that capture the status of data and control channels.

Definition 1 (Channel Status) Let $\mathcal{B}$ denote the set of all channels of an SADF graph, where $\mathcal{B}_c \subseteq \mathcal{B}$ is the set of control channels. A *(data) 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.

Definition 2 (Control Status) Let $\mathcal{B}_c$ denote the set of all control channels of an SADF graph. A *control (channel) status* is a function $\psi : \mathcal{B}_c \to \bigcup_{c \in \mathcal{B}_c} \Sigma_c^*$ that returns the sequence of tokens stored in each control channel.

Notice that a channel status abstracts from the values of tokens. Definition 3 extends the definition of the basic SADF model in [37] with the possibilities of having none or multiple control ports, parameterised rates for the input ports of detectors, initial tokens for control channels and hierarchical control.

Definition 3 (SADF Graph) An SADF graph is described by a tuple $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$, where

1. $\mathcal{K}$ and $\mathcal{D}$ are pairwise disjoint finite sets of kernels and detectors respectively such that $\mathcal{K} \cup \mathcal{D} \neq \varnothing$;
2. each kernel $k \in \mathcal{K}$ denotes a tuple $(\mathcal{I}_k, \mathcal{O}_k, \mathcal{C}_k, \mathcal{S}_k, \{(R_k^s, E_k^s) \mid s \in \mathcal{S}_k\})$ with
   - $\mathcal{S}_k = \prod_{c \in \mathcal{C}_k} \Sigma_b$ with b the control channel connected to c
   - for each scenario $s \in \mathcal{S}_k$, $R_k^s(c) = 1$ for all control ports $c \in \mathcal{C}_k$.
3. a detector $d \in \mathcal{D}$ is a tuple $(\mathcal{I}_d, \mathcal{O}_d, \mathcal{C}_d, \mathcal{S}_d, \{(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s, \Phi_d^s) \mid s \in \mathcal{S}_d\}, \Omega_d, \{(R_d^\omega, t_d^\omega, E_d^\omega) \mid \omega \in \Omega_d\})$ with
   - $\mathcal{S}_d = \prod_{c \in \mathcal{C}_d} \Sigma_b$ with b the control channel connected to c
   - for each scenario $s \in \mathcal{S}_d$, $(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s, \Phi_d^s)$ refers to a Markov chain $(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s)$ with non-empty finite state space and a function $\Phi_d^s : \mathbb{S}_d^s \to \Omega_d$ that returns the subscenario associated to each state in $\mathbb{S}_d^s$;
   - for each subscenario $\omega \in \Omega_d$, $R_d^\omega(c) = 1$ for all control ports $c \in \mathcal{C}_d$;
   - for each subscenario $\omega \in \Omega_d$, $R_d^\omega(o) > 0$ and $t_d^\omega(o) \in \{\upsilon \in \Sigma_b^* \mid |\upsilon| = R_d^\omega(o)\}$ for all output ports $o \in \mathcal{O}_d$ that connect d to a control channel b.
4. $\mathcal{B} \subseteq \mathcal{O} \times (\mathcal{I} \cup \mathcal{C})$ is the finite set of channels with $\mathcal{B}_c = \mathcal{B} \setminus (\mathcal{O} \times \mathcal{I})$ the set of control channels. A channel includes an unbounded FIFO buffer and uniquely connects an output port to either an input port or a control port. Furthermore, each port in $\mathcal{I} \cup \mathcal{O} \cup \mathcal{C}$ is uniquely connected to one channel;

5. $\phi^* : \mathcal{B} \setminus \mathcal{B}_c \to \mathbb{N}$ is the initial channel status;
6. $\psi^* : \mathcal{B}_c \to \bigcup_{c \in \mathcal{B}_c} \Sigma_c^*$ indicates the initial control status.

Some observations can be made from Definition 3. The first item specifies that a process is either a kernel or a detector and that an SADF graph consists of at least one such process. It also clarifies that an SADF graph may include only kernels or only detectors. Considering the second and third items, it follows that the scenario in which a process operates is determined based on the combination of tokens available on its control inputs. In case a process p does not have control ports, it always operates according to a default scenario ϵ_p. It means that the behaviour of p is independent of the behaviour of other processes. Furthermore, the rates associated with control channels are always positive and hence, only data dependencies may be absent in a scenario. The consumption rates for control channels are actually fixed to 1. This originates from the observation that control information is usually considered to be indivisible. It also eases the way of establishing the scenario for a process. Where the scenario determines the rates and execution time distribution for a kernel, it is the subscenario that determines the rates and execution time distribution for a detector. The scenario of a detector identifies which Markov chain is used to establish the subscenario (according to some mapping Φ from its state space to the set of subscenarios). Section 2 exemplified how the Markov chains reflect knowledge about the order in which subscenarios are likely to occur. The fourth item in Definition 3 states that channels may be absent and that multiple channels can exist between two processes as shown in Section 2. An SADF graph may also consist of unconnected components. The last two items formalise that tokens might be available in any channel before the application starts running.

Figure 8 depicts the SADF graph that is used throughout the next section to illustrate the definition of the operational semantics. Kernel A has no control inputs and always operates in default scenario ϵ_A. Kernels B, C and D can operate in three different scenarios $\mathcal{S}_\mathrm{B} = \mathcal{S}_\mathrm{C} = \mathcal{S}_\mathrm{D} = \{s_1, s_2, s_3\}$ as controlled by detector E. Since E has no control inputs, $\mathcal{S}_\mathrm{E} = \{\epsilon_\mathrm{E}\}$. Its set of subscenarios Ω_E is equal to the set $\{s_1, s_2, s_3\}$ of scenarios for B, C and D. The tokens E produces on the channels to B, C and D are valued the same as the subscenario in which E operates. Figure 8c exemplifies the use of multiple possible execution times per scenario for A, B, C and D. Figure 8d shows Markov chain $(\mathbb{S}_\mathrm{E}^{\epsilon_\mathrm{E}}, \iota_\mathrm{E}^{\epsilon_\mathrm{E}}, \mathbb{P}_\mathrm{E}^{\epsilon_\mathrm{E}})$ with $\mathbb{S}_\mathrm{E}^{\epsilon_\mathrm{E}} = \{x, y, z\}$ and $\iota_\mathrm{E}^{\epsilon_\mathrm{E}} = x$ and the mapping $\Phi_\mathrm{E}^{\epsilon_\mathrm{E}}$. Figures 4d and 7d only indicate the subscenarios associated to each state. Observe that we also did not specify the initial state of the Markov chain associated with detector FD in Figures 1 and 3; they can be chosen arbitrarily without affecting any correctness and most performance properties (an exception is the expected response delay [35]).

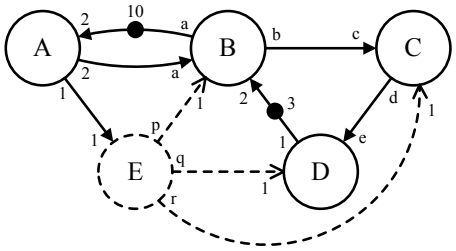
4.3 Operational Semantics

The operational semantics of an SADF graph is defined as a Timed Probabilistic System (TPS) [1,31]. Timed Probabilistic Systems describe behaviour in terms of *configurations* (states) and *transitions*, where timeless actions are explicitly distinguished from advancing time. We use Θ to denote the set of all reachable configurations of an SADF graph and define $\mathcal{D}(\Theta)$ as the set of probability distributions over Θ given by $\mathcal{D}(\Theta) = \{\boldsymbol{\pi} : \Theta \to [0,1] \mid \sum_{c \in \Theta} \boldsymbol{\pi}(c) = 1\}$. The TPS defined by an SADF graph is a labelled transition system of the form $(\Theta, C^*, \mathcal{A}, \mathbb{A}, \mathcal{T}, \mathbb{T})$, where $C^* \in \Theta$ is an initial configuration, $\mathcal{A}$ is a set of actions and $\mathcal{T} \subseteq \mathbb{R}_0^+$ denotes the time domain. $\mathbb{A}$ and $\mathbb{T}$ are two sets of labelled transition relations. $\mathbb{A}$ is a subset of $\Theta \times \mathcal{A} \times \mathcal{D}(\Theta)$ and defines the action transitions. $\mathbb{T}$ denotes the time transitions and is a subset of $\Theta \times \mathcal{T}^+ \times \mathcal{D}(\Theta)$ where $\mathcal{T}^+ = \mathcal{T} \setminus \{0\}$. In the remainder of this section, we define $\Theta, C^*, \mathcal{A}, \mathbb{A}, \mathcal{T}$ and $\mathbb{T}$ concretely for any SADF graph $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$. We start with defining how a configuration in Θ looks like. To this end, we introduce the kernel and detector status.

Definition 4 (Kernel Status) A *kernel status* for an SADF graph $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$ is a function κ 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 graph $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$ is a function δ that assigns to each detector $d \in \mathcal{D}$ a tuple in $\prod_{s \in \mathcal{S}_d} \mathbb{S}_d^s \times (\Omega_d \cup \{-\}) \times (\mathbb{R}_0^+ \cup \{-\})$ denoting the current states $S_{s_1}, \ldots, S_{s_N}$ of all $N = |\mathcal{S}_d|$ Markov chains associated with d, the subscenario in which d operates as well as the remaining execution time for firing d or $(S_{s_1}, \ldots, S_{s_N}, -, -)$ if d is not firing.

Definitions 4 and 5 implicitly exclude the possibility of multiple simultaneous firings of a process. This phenomenon is for SDF also known as *auto-concurrency* [32]. If such auto-concurrency would be allowed for



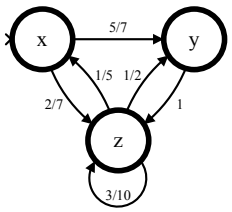
a) Processes and Channels

Rate	(Sub)Scenario s_1	s_2	s_3
a	2	1	1
b	3	3	2
c	1	2	1
d	2	4	1
e	3	3	1
p	1	2	2
q	3	3	4
r	2	4	4

b) Parameterised Rates

Process	(Sub)Scenario	E	$\mathbb{P}(E = e)$
A	ϵ_A	2	1/3
		2.1	2/3
B	s_1	3	1/5
		2.8	4/5
	s_2	4	1
	s_3	1	1
C	s_1	1	1/2
		1.2	1/2
	s_2	3	1
	s_3	5	1
D	s_1	4	1
	s_2	1	1/6
		3.1	5/6
	s_3	2.5	1
E	All	0	1

c) Execution Times



$\Phi_{\mathrm{E}}^{\epsilon_{\mathrm{E}}}(x) = s_1$

$\Phi_{\mathrm{E}}^{\epsilon_{\mathrm{E}}}(y) = s_2$

$\Phi_{E}^{\epsilon_{\mathrm{E}}}(z) = s_3$

d) Detector E

Fig. 8 A small SADF graph to illustrate the definition of the operational semantics.

SADF, then simultaneous firings of a process 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). Even if the execution times of all processes would be fixed, tokens are produced in a non-deterministic order when simultaneous firings end at the same time (this also happens for SDF). In case the value of tokens is relevant (as for control tokens in SADF), allowing multiple simultaneous firings would complicate ensuring determinacy[3]. The desirable property of determinacy ensures that the functional behaviour is independent of how the inherent non-determinism originating from the concurrency in a dataflow model is resolved (See also [37]). Despite excluding auto-concurrency, multiple simultaneous firings could still be captured by 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 graph is a tuple $(\phi, \psi, \kappa, \delta)$ denoting a channel status, control status, kernel status and detector status respectively. SADF graph $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$ has initial configuration $C^* = (\phi^*, \psi^*, \kappa^*, \delta^*)$, where the initial kernel status κ^* is defined by $\kappa^*(k) = (-, -)$ for all $k \in \mathcal{K}$ and initial detector status δ^* is defined by $\delta^*(d) = (\iota_d^{s_1}, \ldots, \iota_d^{s_N}, -, -, -)$ for all $N = |\mathcal{S}_d|$ scenarios of each $d \in \mathcal{D}$.

Before defining the action and time transitions, we give an informal description of the operational semantics that is more detailed and more general compared to the explanations provided for the examples in Section 2.

If a kernel k has control ports, its firing starts with reading one token from all control ports after they have become available. The combination of their values determines the scenario $s \in \mathcal{S}_k$ in which k will operate. If k has no control ports, s always equals ϵ_k and there is no need to wait for tokens to become available on any port. After establishing the scenario, 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 can perform its data processing behaviour, which takes E_k^s units of

[3] One could allow auto-concurrency without consequences in case produced tokens are not valued. Allowing auto-concurrency for kernels without control inputs and fixed execution times (the equivalence of SDF actors) would for example only complicate the semantics but does not invalidate the determinacy result. In this report, we choose however to disallow auto-concurrency completely.

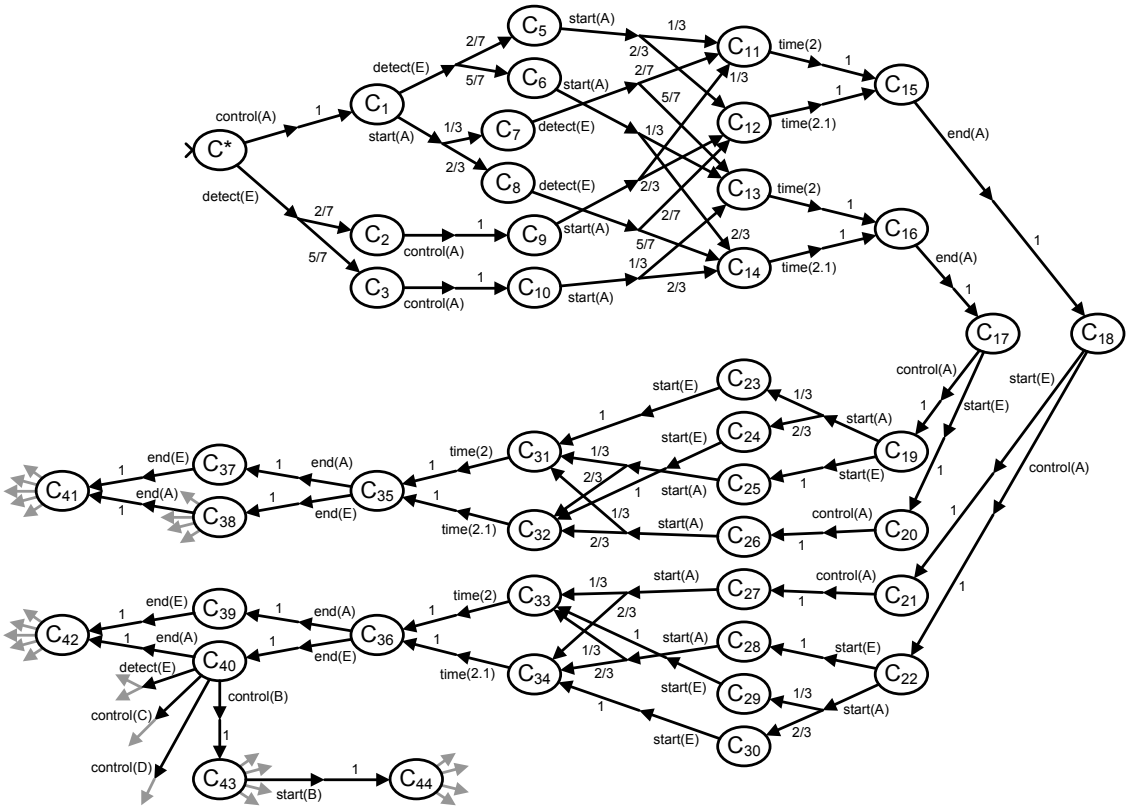


Fig. 9 Fragment of the TPS defined by the example SADF graph in Figure 8.

time. The firing of k ends with removing the $R_k^s(i)$ tokens from the input ports $i \in \mathcal{I}_k$ and writing $R_k^s(o)$ tokens to each output port $o \in \mathcal{O}_k$ as well as removing one token from each control port (if k has control ports).

The firing of a detector d also starts with determining the scenario $s \in \mathcal{S}_d$ in a similar way as for a kernel. In addition, a subscenario $\omega \in \Omega_d$ is established as the result of $\Phi_d^s(x)$ where $x \in \mathbb{S}_d^s$ is the state entered after Markov chain $(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s)$ performed one transition. After establishing the scenario and subscenario, detector d waits until $R_d^\omega(i)$ tokens become available at every input port $i \in \mathcal{I}_d$[4]. When sufficient tokens have become available on all input ports, d performs its behaviour which takes E_d^ω units of time. Firing of d ends with removing the $R_d^\omega(i)$ tokens from its input ports, writing $R_d^\omega(o)$ tokens to each output port $o \in \mathcal{O}_d$ and removing one token from each control port (if d has control ports). The tokens written to an output port connected to a control channel are valued according to t_d^ω such that they can be interpreted by the receiving process for determining its scenario.

Notice that in reality, consumption and production of tokens may occur at any time during firing. This could be captured in an abstract way by stochastically distributing the consumption and production of tokens for a process over its execution time. However, we choose in this report for the distribution where tokens are consumed and produced at the end of a firing. This approach yields conservative results of performance metrics related to the occupancy of the buffers in channels in case reservation of buffer space is taken into account [34] at the start of a firing. Performance analysis of SADF as implemented in [35] is therefore based on an extended version of the operational semantics presented in this report, which also keeps track of buffer space reservation.

Figure 9 shows a part of the TPS defined by the SADF graph in Figure 8 (assuming action urgency or self-timed execution, see below). The nodes represent configurations in Θ. The initial configuration C^*, which is identified with symbol $>$, captures the channel and control status as depicted in Figure 8. Transitions are visualised with double (multi) arrows. The first part of the double (multi) arrow is a directed arrow labelled with

[4] We remark that [37] required rates of input ports for detectors to be fixed, while (sub)scenarios were determined only after sufficient tokens had become available on the input ports. This report relaxes that restriction. Despite the corresponding semantical differences, the analysis results presented in [37] remain valid.

the performed action or the amount of time that advances. The second part is a fan-out of directed arrows labelled with the probabilities $\boldsymbol{\pi}(C')$ for each possible target configuration C'. The gray arrows reflect transitions to parts of the TPS that are omitted. Six different action types are distinguished for $\mathcal{A}$ in Definitions 7 through 12 below.

Definition 7 (Control Action Transition) A *control action transition* $\overset{\text{control}(k)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ represents 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 $|\psi(b_c)| \geq 1$ for each channel b_c connected to a control port $c \in \mathcal{C}_k$, where

$$\kappa' = \begin{cases} \kappa[k \rightarrow (\epsilon_k, -)] & \text{if } \mathcal{C}_k = \varnothing \\ \kappa[k \rightarrow (\prod_{c \in \mathcal{C}_k} \psi(b_c)_1, -)] & \text{otherwise} \end{cases} \tag{1}$$

Definition 7 formalises how *control* actions capture the first step of firing kernels[5]. Notice that the condition regarding the availability of control tokens is trivially satisfied when a kernel does not have control ports. Equation (1) specifies that the scenario in which a kernel will operate is determined by the combination of the first tokens on all control ports if the kernel has control ports or that it equals the default scenario. Furthermore, a control action yields a single target configuration that is entered with probability 1. For the example of Figure 8, only kernel A can immediately perform a control action from the initial configuration C^* leading to configuration C_1. The other kernels cannot start firing because there are no initial tokens available on the control channels.

Determining the scenario and subscenario for a detector is captured by *detect* actions conform Definition 8.

Definition 8 (Detect Action Transition) A *detect action transition* $\overset{\text{detect}(d)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ denotes establishing the scenario and subscenario in which a detector d is going to operate. Transition relation $(\phi, \psi, \kappa, \delta) \overset{\text{detect}(d)}{\longrightarrow} \boldsymbol{\pi}$ holds if $\delta(d) = (S_{s_1}, \ldots, S_{s_{i-1}}, S_{s_i}, S_{s_{i+1}}, \ldots, S_{s_N}, -, -)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with d and if $|\psi(b_c)| \geq 1$ for each channel b_c connected to a control port $c \in \mathcal{C}_d$. The distribution function $\boldsymbol{\pi}$ is given by

$$\boldsymbol{\pi}(\phi, \psi, \kappa, \delta') = \begin{cases} \mathbb{P}_d^{\epsilon_d}(S_{\epsilon_d}, T) \text{ for each } T \in \mathbb{S}_d^{\epsilon_d} & \text{if } \mathcal{C}_d = \varnothing \\ \mathbb{P}_d^{s_i}(S_{s_i}, T) \text{ for each } T \in \mathbb{S}_d^{s_i} \text{ with } s_i = \prod_{c \in \mathcal{C}_d} \psi(b_c)_1 & \text{otherwise} \end{cases}$$

where detector status δ' is defined as

$$\delta' = \begin{cases} \delta[d \rightarrow (T, \Phi_d^{\epsilon_d}(T), -)] \text{ for each } T \in \mathbb{S}_d^{\epsilon_d} & \text{if } \mathcal{C}_d = \varnothing \\ \delta[d \rightarrow (S_{s_1}, \ldots, S_{s_{i-1}}, T, S_{s_{i+1}}, \ldots, S_{s_N}, \Phi_d^{s_i}(T), -)] \text{ for each } T \in \mathbb{S}_d^{s_i} & \text{otherwise} \end{cases}$$

Observe that the condition in Definition 8 regarding the availability of control tokens is trivially satisfied when a detector has no control ports. In such case, the scenario is equivalent to the default scenario. Otherwise, the scenario is established based on the combination of the (first) tokens available on the control ports, similarly as for kernels. The scenario identifies the Markov chain used for determining the subscenario. Given the current state in which the involved Markov chain resides, the subscenario results from the state entered after performing one transition in that Markov chain. Because several target states might be possible (potentially implying different subscenarios), a probabilistic fan-out is obtained in the TPS where the resulting configurations differ in the target state reached for the involved Markov chain (and hence also the subscenario may differ). The probabilistic fan-out corresponds with the outgoing probabilities for the current state of the considered Markov chain. Figure 9 illustrates that detector E can perform a detect action from C^* since it has no control ports. The Markov chain associated with E initially resides in state x and according to Figure 8, there are 2 states to which it can transit, each implying a different subscenario. These 2 options surface in the TPS as a probabilistic fan-out to configurations C_2 and C_3. These configurations merely differ in the new state for the Markov chain and subscenario.

The second step of firing a process involves waiting for the availability of sufficient tokens on the input ports and fixing the execution time. This is formalised by *kernel start* and *detector start* actions in Definitions 9 and 10. Such start actions imply a probabilistic fan-out in the TPS that reflects the characteristics of the execution time distribution corresponding to the previously established scenario (for kernels) or subscenario (for detectors).

[5] Because kernels without control ports always operate in a default scenario, one can actually skip control actions for such uncontrolled kernels. Performance analysis of SADF as implemented in [35] exploits this observation to limit the number of configurations.

Definition 9 (Kernel Start Action Transition) A *kernel start action transition* $\xrightarrow{\text{start}(k)} \subseteq \Theta \times D(\Theta)$ indicates the start of processing data by a kernel k. Relation $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(k)} \boldsymbol{\pi}$ holds if $\kappa(k) = (s, -)$ for some scenario $s \in \mathcal{S}_k$ and if $\phi(b_i) \geq R_k^s(i)$ for each channel b_i connected to input port $i \in \mathcal{I}_k$. Function $\boldsymbol{\pi}$ is defined by $\boldsymbol{\pi}(\phi, \psi, \kappa'_e, \delta) = \mathbb{P}(E_k^s = e)$ with $\kappa'_e = \kappa[k \rightarrow (s, e)]$ for all execution times e in the sample space of E_k^s.

Definition 10 (Detector Start Action Transition) A *detector start action transition* $\xrightarrow{\text{start}(d)} \subseteq \Theta \times D(\Theta)$ completes starting the firing of a detector d. Relation $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(d)} \boldsymbol{\pi}$ holds if $\delta(d) = (S_{s_1}, \ldots, S_{s_N}, \omega, -)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with d, a subscenario $\omega \in \Omega_d$ and if $\phi(b_i) \geq R_d^\omega(i)$ for each channel b_i connected to an input port $i \in \mathcal{I}_d$. The function $\boldsymbol{\pi}$ is defined by $\boldsymbol{\pi}(\phi, \psi, \kappa, \delta'_e) = \mathbb{P}(E_d^\omega = e)$ with $\delta'_d = \delta[d \rightarrow (S_{s_1}, \ldots, S_{s_N}, \omega, e)]$ for all e in the sample space of E_d^ω.

When residing in configuration C_1, kernel A can continue its firing because sufficient tokens are available on the input port (see Figure 8). There are two possible target configurations C_7 and C_8 in accordance with execution time distribution $E_{\mathsf{A}}^{\epsilon_{\mathsf{A}}}$. Conform Definition 9, C_7 and C_8 only differ in the remaining execution time for A (which is 2 time units for C_7 and 2.1 time units for C_8). On the other hand, detector E cannot perform a start action from configurations C_2 and C_3 because there are no tokens on its input port. Such tokens only become available after kernel A has finished firing (see below). Notice however that A can still perform a control action (and subsequently a start action) from C_2 and C_3. Similarly, E can still perform a detect action from C_1. The subgraph covering configurations C^* and C_1 through $C14$ shows that there are 4 different configurations C_{11} through C_{14} that can be entered after E has performed a detect action and A has completed starting its firing. These 4 configurations can be reached via various paths from C^*, where the chosen path reflects how the non-deterministic choices between alternative enabled actions from each intermediate configuration are resolved.

Firing a kernel may finish whenever its remaining execution time reduced to 0. Definition 11 formalises this with a *kernel end* action by consuming the appropriate number of tokens from its input ports and one token from each control port (if it has such ports) as well as producing the proper number of tokens to the output ports. There is one target configuration in which the kernel returns to a state where it is willing to perform the next firing.

Definition 11 (Kernel End Action Transition) A *kernel end action transition* $\xrightarrow{\text{end}(k)} \subseteq \Theta \times D(\Theta)$ reflects finalising the firing of a kernel k. Transition relation $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(k)} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi', \psi', \kappa', \delta) = 1$ holds if $\kappa(k) = (s, 0)$ for some scenario $s \in \mathcal{S}_k$. The resulting configuration $(\phi', \psi', \kappa', \delta)$ is given by

$$\phi' = \begin{cases} \phi[b_i \rightarrow \phi(b_i) - R_k^s(i)] & \text{for each non self-loop channel } b_i \text{ connected to an input port } i \in \mathcal{I}_k \\ \phi[b_o \rightarrow \phi(b_o) + R_k^s(o)] & \text{for each non self-loop channel } b_o \text{ connected to an output port } o \in \mathcal{O}_k \\ \phi[b_l \rightarrow \phi(b_{oi}) + R_k^s(o) - R_k^s(i)] & \text{for each self-loop channel } b_{oi} \text{ connecting an } o \in \mathcal{O}_k \text{ to an } i \in \mathcal{I}_k \end{cases}$$

and $\psi' = \psi[b_c \rightarrow \psi(b_c) - \psi(b_c)_1]$ for each channel b_c connected to a $c \in \mathcal{C}_k$ and $\kappa' = \kappa[k \rightarrow (-, -)]$.

Below, we explain that the remaining execution time of kernel A reduces to 0 in configurations C_{15} and C_{16}, thereby enabling the end action to C_{17} and C_{18} respectively. Observe that finishing the firing of A results in producing tokens on the channel to detector E. Because the subscenario for E is know to be s_2 for C_{17} and s_3 for C_{18}, E can perform a start action conform Definition 10. Since E has only one possible execution time for all its subscenarios, there is one target configuration (C_{20} and C_{21} respectively) for such start action.

Finalising the firing of a detector with a *detector end* action is similar to finalising the firing of a kernel except that it also involves the production of scenario-valued tokens onto control channels. When detector E finishes firing by an end action, it produces tokens valued equal to the subscenario in which it operated. As a result, kernels B, C and D can then start firing with a control action. Figure 9 exemplifies how B can perform a control action from configuration C_{40} where E has produced tokens valued s_3 onto control channels to B, C and D. Definition 12 specifies that the end action returns detectors in a state that may enable a detect action again.

Definition 12 (Detector End Action Transition) A *detector end action transition* $\xrightarrow{\text{end}(d)} \subseteq \Theta \times D(\Theta)$ finalises the firing of a detector d. Transition relation $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(d)} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi', \psi', \kappa, \delta') = 1$ holds if $\delta(d) =$

$(S_{s_1}, \ldots, S_{s_N}, \omega, 0)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with d and subscenario $\omega \in \Omega_d$. The resulting configuration $(\phi', \psi', \kappa, \delta')$ is defined by

$$\phi' = \begin{cases} \phi[b_i \rightarrow \phi(b_i) - R_d^\omega(i)] & \text{for each non self-loop channel } b_i \text{ connected to an } i \in \mathcal{I}_d \\ \phi[b_o \rightarrow \phi(b_o) + R_d^\omega(o)] & \text{for each non self-loop data channel } b_o \text{ connected to an } o \in \mathcal{O}_d \\ \phi[b_l \rightarrow \phi(b_{oi}) + R_d^\omega(o) - R_d^\omega(i)] & \text{for each self-loop channel } b_{oi} \text{ connecting an } o \in \mathcal{O}_d \text{ to an } i \in \mathcal{I}_d \end{cases}$$

and

$$\psi' = \begin{cases} \psi[b_c \rightarrow \psi(b_c) - \psi(b_c)_1] & \text{for each non self-loop channel } b_c \text{ connected to a } c \in \mathcal{C}_d \\ \psi[b_o \rightarrow \psi(b_o) + t_d^\omega(o)] & \text{for each non self-loop control channel } b_o \text{ connected to} \\ \quad \text{with } t_d^\omega(o) \in \Sigma_{b_o}^* \text{ and } |t_d^\omega(o)| = R_d^\omega(o) & \quad \text{an output port } o \in \mathcal{O}_d \\ \psi[b_l \rightarrow \psi(b_{oc}) + t_d^\omega(o) - \psi(b_{oc})_1] & \text{for each self-loop channel } b_{oc} \text{ connecting an output} \\ \quad \text{with } t_d^\omega(o) \in \Sigma_{b_{oc}}^* \text{ and } |t_d^\omega(o)| = R_d^\omega(o) & \quad \text{port } o \in \mathcal{O}_d \text{ to a control port } c \in \mathcal{C}_d \end{cases}$$

and $\delta' = \delta[d \rightarrow (S_{s_1}, \ldots, S_{s_N}, -, -)]$.

Notice that Definition 12 allows specifying the production of any sequence of tokens onto a control channel that has the appropriate length. In this report, we assume that such sequences consist of equally valued tokens.

Next to the six types of action transitions, the TPS may include *time transitions* to represent progress in time. Like in traditional dataflow models, SADF has one notion of time, which means that time advances for all processes together (synchronously) and with the same amount. Definition 13 specifies that this amount equals the smallest remaining execution time of all processes with positive remaining execution time. Alternatively, one could choose to allow advancing time with anything not greater than this smallest time. That would however imply infinite sets of configurations and transitions. This originates from having real valued time, which would enable all possible ways (there are infinitely many) of splitting progress in time with exactly the smallest remaining execution time into (infinitely many) smaller steps, see for example [12]. Choosing to always advance time with the smallest remaining execution time ensures that the TPS defined by any SADF graph excludes the possibility of multiple enabled time transitions from a configuration.

Definition 13 (Time Transition) A *time transition* $\overset{\text{time}(t)}{\longrightarrow} \subseteq \Theta \times D(\Theta)$ denotes progress in time with t units. Transition relation $(\phi, \psi, \kappa, \delta) \overset{\text{time}(t)}{\longrightarrow} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi, \psi, \kappa', \delta') = 1$ holds if there exists a process with a positive remaining execution time and no end action transitions are enabled, where t is the smallest remaining execution time of all processes with a positive remaining execution time. The resulting 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 $n > 0$ for each kernel k, while $\delta'(d) = (S_{s_1}, \ldots, S_{s_N}, \omega, n - t)$ if $\delta(d) = (S_{s_1}, \ldots, S_{s_N}, \omega, n)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with detector d, some $\omega \in \Omega_d$ and $n > 0$ for each detector d.

Next to advancing time with the smallest positive remaining execution time, Definition 13 requires no end actions to be enabled. In other words, finalising the firing of processes is prioritised over advancing time. A process can only perform an end action in case its remaining execution time is 0. Hence, the TPS defined by an SADF graph excludes the possibility of advancing time without intermediately performing (end) actions.

Observe that Definition 13 does not prescribe how non-determinism between advancing time and performing actions other then end actions should be resolved. Consider a configuration C that is entered after some process performed a start action such that its remaining execution time is positive. Then, a time transition is enabled from all configurations reachable from C up to configurations in which an end action is enabled. Although Definition 13 captures all possible scheduling policies, it is often convenient to choose for a specific class of scheduling policies that prescribes when time should advance compared to performing actions. An important class of scheduling policies are those policies that perform actions without delays (i.e., all actions are performed before time advances, which then enables performing actions again). Such *action urgency* [29] matches with *self-timed execution* for SDF graphs [32]. Self-timed execution of SDF graphs ensures obtaining maximal throughput [19]. Assuming action urgency may however be disadvantages for other metrics like latency [15]. The assumption of action urgency emerges in Figure 9 in not considering subgraphs due to time transitions from configurations C_{20}, C_{21} and C_{23} through C_{30}. Observe on the other hand that action urgency does not prescribe how non-determinism between alternative actions should be resolved, thereby leaving several scheduling options open.

5 Conclusions

This report describes the full model of Scenario-Aware Dataflow, covering all considerations encountered during its design. Despite extending the basic model of [37] in several ways, all properties and analysis techniques described in [37] are still valid. The tools for analysing SADF graphs in [35] already support this full SADF model both for model checking and simulation-based estimation of various worst/best-case, (probabilistic/expected) reachability and average-case performance metrics. Future work lies in extending the tool set with additional and more efficient analysis facilities. Approaches to (semi-)automatically derive SADF models from source code of streaming applications and incorporation in automated design flows are other directions of future research.

References

1. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD Thesis. Stanford University, 1997.
2. A.O. Allen. *Probability, Statistics, and Queueing Theory; with Computer Science Applications*. Academic Press, 2nd Edition, 1990.
3. 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.
4. B. Bhattacharya and S.S. Bhattacharyya. Parameterized Dataflow Modeling for DSP Systems. *IEEE Transactions on Signal Processing*, vol 49 (10), pp 2408–2421, 2001.
5. G. Bilsen, M. Engels, R. Lauwereins and J.A. Peperstraete. Cyclo-Static Dtata Flow. *IEEE Transactions on Signal Processing*, vol 44 (2), pp 397–408, IEEE, 1996.
6. J.T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model*. Ph.D. Thesis, University of California, Berkeley, 1993.
7. 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.
8. K.L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer-Verlag, 1967.
9. C. Derman. *Finite State Markovian Decision Processes*. Academic Press, 1970.
10. E. Deprettere, T. Stefanov, S.S. Bhattacharyya and M. Sen. Affine Nested Loop Programs and their Binary Parameterized Dataflow Graph Counterparts. *Proceedings of ASAP'06*, pp 186–190, IEEE, 2006.
11. P. Eles, K. Kuchcinski, Z. Peng, A. Doboli and P. Pop. Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems. *Proceedings of DATE'98*, pp 23–26, IEEE, 1998.
12. M.C.W. Geilen. *Formal Techniques for Verification of Complex Real-Time Systems*. PhD Thesis. Eindhoven University of Technology, 2002.
13. M.C.W. Geilen and T. Basten. Reactive Process Newtorks. *Proceedings of EMSOFT'04*, ACM, 2004.
14. A.H. Ghamarian, M.C.W. Geilen, S. Stuijk, T. Basten, A.J.M. Moonen, M.J.G. Bekooij, B.D. Theelen and M.R.Mousavi. Throughput Analysis of Synchronous Data Flow Graphs. *Proceedings of ACSD'06*, pp 25–34, IEEE, 2006.
15. A.H. Ghamarian, S. Stuijk, T. Basten, M.C.W. Geilen and B.D. Theelen. Latency Minimization for Synchronous Data Flow Graphs. *Proceedings of DSD'07*, pp 189–196, IEEE, 2007.
16. S.V. Gheorghita, T. Basten and H. Corporaal. Application Scenarios in Streaming-Oriented Embedded System Design. *Proceeedings of SoC'06*, pp 175–178, IEEE, 2006.
17. A. Girault, B. Lee, E.A. Lee. Hierarchical Finite State Machine with Multiple Concurrency Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol 18 (6), pp 742–760, 1999.
18. 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.
19. R. Govindarajan and G.R. Gao. Rate-Optimal Schedule for Multi-Rate DSP Computations. *Journal of VLSI Signal Processing*, vol 9, pp 211235, 1995.
20. J. Hillston. Compositional Markovian Modelling Using a Process Algebra. *Proceedings of NSMC'95*, pp 177–196. Kluwer, 1995.
21. G. Kahn. The Semantics of a Simple Language for Parallel Programming. *Proceedings of IFIP'74*, pp 471–475, North-Holland, 1974.
22. M. Kwiatkowska, G. Norman, D. Parker, J. Sproston. Performance Analysis of Probabilistic Timed Automata using Digital Clocks. *Formal Methods in System Design*, vol 29, pp. 33-78, 2006.
23. B. Lee. *Specification and Design of Reactive Systems*. PhD Thesis. University of California, Berkeley, 2000.
24. E. Lee and D. Messerschmitt. Synchronous Data Flow. *IEEE Proceedings*, vol 75 (9), pp 1235–1245, 1987.
25. 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.
26. 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.
27. M.K. Molloy. Performance Analysis using Stochastic Petri Nets. *IEEE Transactions on Computers*, vol 31 (9), pp 913–917, 1982.
28. A. Moonen, M. Bekooij. R. van den Berg and J. van Meerbergen. Practical and Accurate Throughput Analysis with the Cyclo Static Dataflow Model. In: *Proceedings of MASCOTS'07*, pp 238–245, 2007.
29. X. Nicollin and J. Sifakis. An Overview of Synthesis on Timed Process Algebras. *Proceedings of CAV'91*, pp 376–398, Springer-Verlag, 1991.
30. S. Ritz, M. Pankart and H. Meyr. High-Level Software Synthesis for Signal Processing Systems. *Proceedings of ASAP'92*, pp 679–693, IEEE, 1992.

31. R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. Thesis, Massachusetts Institute of Technology, 1995.
32. S. Siram and S.S. Bhattacharyya. *Embedded Multiprocessors; Scheduling and Synchronization*. Marcel Dekker, 2000.
33. S. Shlien. Guide to MPEG-1 Audio Standard. *IEEE Transactions on Broadcasting*, vol 40 (4), pp 206–218, 1994.
34. B.D. Theelen. *Performance Modelling for System-Level Design*. Ph.D. Thesis. Eindhoven University of Technology, 2004.
35. B.D. Theelen. A Performance Analysis Tool for Scenario-Aware Streaming Applications. *Proceedings of QEST'07*, IEEE Computer Society, 2007.
36. B.D. Theelen, O. Florescu, M.C.W. Geilen, J. Huang, P.H.A. van der Putten and J.P.M. Voeten. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. *Proceedings of MEMOCODE'07*, pp 139–148, IEEE Computer Society, 2007.
37. B.D. 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. *Proceedings of MEMOCODE'06*, pp 185–194, IEEE Computer Society, 2006.
38. J.P.M. Voeten. Performance Evaluation with Temporal Rewards. *Performance Evaluation*, vol 50 (2/3), pp 189–218, 2002.