Scenario-Aware Dataflow

Bart Theelen

1 Scenario-Aware Dataflow

This section\(^1\) discusses Scenario-Aware Dataflow (SADF), which is a generalization of dataflow models with strict periodic behavior. Like many dataflow models, SADF is primarily a coordination language that highlights how actors (which are potentially executed in parallel) interact. To express dynamism, SADF distinguishes data and control explicitly, where the control-related coherency between the behavior (and hence, the resource requirements) of different parts of a signal processing application can be captured with so-called *scenarios* [6]. The scenarios commonly coincide with dissimilar (but within themselves more static) modes of operation originating, for example, from different parameter settings, sample rate conversion factors, or the signal processing operations to perform. Scenarios are typically defined by clustering operational situations with similar resource requirements [6]. The scenario-concept in SADF allows for more precise (quantitative) analysis results compared to applying traditional SDF-based analysis techniques. Still, common subclasses of SADF can be synthesized into efficient implementations [20, 9].

1.1 SADF Graphs

We introduce SADF by some examples from the multi-media domain. We first consider the MPEG-4 video decoder for the Simple Profile from [25, 21]. It supports video streams consisting of intra (I) and predicted (P) frames. For an image size of \(176 \times 144\) pixels (QCIF), there are 99 macro blocks to decode for I frames and no

---

\(^1\) Bart Theelen thanks Marc Geilen (m.c.w.geilen@tue.nl) and Sander Stuijk (s.stuijk@tue.nl), both from the Eindhoven University of Technology, for their contribution to this section.
motion vectors. For P frames, such motion vectors determine the new position of certain macro blocks relative to the previous frame. The number of motion vectors and macro blocks to process for P frames ranges between 0 and 99. The MPEG-4 decoder clearly shows variations in the functionality to perform and in the amount of data to communicate between the operations. This leads to large fluctuations in resource requirements [13]. The order in which the different situations occur strongly depends on the video content and is generally not periodic.

Figure 1 depicts an SADF graph for the MPEG-4 decoder in which 9 different scenarios are identified. SADF distinguishes two types of actors: kernels (solid vertices) model the data processing parts, whereas detectors (dashed vertices) control the behavior of actors through scenarios\(^2\). Moreover, data channels (solid edges) and control channels (dashed edges) are distinguished. Control channels communicate scenario-valued tokens that influence the control flow. Data tokens do not influence the control flow. The availability of tokens in channels is shown with a dot. Here, such dots are labeled with the number of tokens in the channel. The start and end points of channels are labeled with production and consumption rates respectively. They refer to the number of tokens atomically produced respectively consumed by the connected actor upon its firing. The rates can be fixed or scenario-dependent, similar as in PSDF. Fixed rates are positive integers. Parameterized rates are valued with non-negative integers that depend on the scenario. The parameterized rates for the MPEG-4 decoder are listed in Table 1b. A value of 0 expresses that data dependencies are absent or that certain operations are not performed in those scenarios.

\(^2\) In case of one detector, SADF literature may not show the detector and control channels explicitly.
Studying Table 1b reveals that for any given scenario, the rate values yield a consistent SDF graph. In each of these scenario graphs, detector FD has a repetition vector entry of 1 [25], which means that scenario changes as prescribed by the behavior of detectors may occur only at iteration boundaries of each such scenario graph. This is not necessarily true for SADF in general as discussed below.

SADF specifies execution times of actors (from a selected time domain, see Subsection 1.2) per scenario. Table 1c lists the worst-case execution times of the MPEG-4 decoder for an ARM7TDMI processor. Tables 1b and 1c show that the worst-case communication requirements occur for scenario \( P_{99} \), in which all actors are active and production/consumption rates are maximal. Scenario \( P_{99} \) also requires maximal execution times for VLD, IDCT, and MC, while for RC it is scenario \( I \) in which the worst-case execution time occurs. Traditional SDF-based approaches need to combine these worst-case requirements into one (unrealistically) conservative model, which yields too pessimistic analysis results.

An important aspect of SADF is that sequences of scenarios are made explicit by associating state machines to detectors. The dynamics of the MPEG-4 decoder originate from control-flow code that (implicitly or explicitly) represents a state-machine with video stream content dependent guards on the transitions between states. One can think of if-statements that distinguish processing I frames from processing P frames. For the purpose of compile-time analysis, SADF abstracts from the content of data tokens (similar to SDF and CSDF) and therefore also from the concrete conditions in control-flow code. Different types of state machines can be used to model the occurrences of scenarios, depending on the compile-time analysis needs as presented in Subsection 1.2. The dynamics of the MPEG-4 decoder can be captured by a state-machine of 9 states (one per scenario) associated to detector FD.

The operational behavior of actors in SADF follows two steps, similar to the switch and select actors in BDF. The first step covers the control part which establishes the mode of operation. The second step is like the traditional data flow behavior of SDF actors in which data is consumed and produced. Kernels establish their scenario in the first step when a scenario-valued token is available on their control inputs. The operation mode of detectors is established based on external and internal forces. We use subscenario to denote the result of the internal forces affecting the operation mode. External forces are the scenario-valued tokens available on control inputs (similar as for kernels). The combination of tokens on control inputs for a detector determine its scenario, which (deterministically) selects a corresponding state-machine. A transition is made in the selected state machine, which establishes the subscenario. Where the scenario determines values for parameterized rates and execution time details for kernels, it is the subscenario that determines these aspects for detectors. Tokens produced by detectors onto control channels are scenario-valued to coherently affect the behavior of controlled actors, which is a key feature of SADF. Actor firings in SADF block until sufficient tokens are available. As a result, the execution of different scenarios can overlap in

---

3 Execution of the reflected function or program is enabled when sufficient tokens are available on all (data) inputs, and finalizes (after a certain execution time) with producing tokens on the outputs.

4 If a detector has no control inputs, it operates in a default scenario \( \varepsilon \) and has one state machine.
a pipelined fashion. For example, in the MPEG-4 decoder, IDCT is always ready
to be executed immediately after VLD, which may already have accepted a control
token with a different scenario value from FD. The ability to express such so-called
pipelined reconfiguration is another key feature of SADF.

We now turn our attention to the MP3 audio decoder example taken from [21]
depicted in Figure 2. It illustrates that SADF graphs can contain multiple detectors,
which may even control each other's behavior. MP3 decoding transforms a com-
pressed audio bitstream into pulse code modulated data. The stream is partitioned
into frames of 1152 mono or stereo frequency components, which are divided into
two granules of 576 components structured in blocks [14]. MP3 distinguishes three
frame types: Long (L), Short (S) and Mixed (M) and two block types: Long (BL) and
Short (BS). A Long block contains 18 frequency components, while Short blocks in-
clude only 6 components. Long frames consist of 32 Long blocks, Short frames of
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. Neglecting that the frame types and specific block type sequences are corre-
lated leads to unrealistic models. The sequences of block types is dependent on the
frame type, as is reflected in the structure of source code of the MP3 audio decoder.
SADF supports hierarchical control to intuitively express this kind of correlation
between different aspects that determine the scenario.

Figure 3a lists the parameterized rates for the MP3 decoder. Only five combina-
tions of frame types occur for the two audio channels combined. We use a two-letter
abbreviation to indicate the combined frame type for the left and right audio channel
respectively: LL, SS, LS and SL. Mixed frames M cover both audio channels simulta-
neously. Detector FD determines the frame type with a state machine of 5 states,
each uniquely identify a subscenario in \{LL, SS, LS, SL, M\}. The operation mode of
kernel S depends on the frame types for both audio channels together and therefore
it operates according to a scenario from this same set. The scenario of kernels RQL,
ROL and RQR, RO_R is only determined by the frame type for either the left or right
audio channel. They operate in scenario $S$, $M$ or $L$ by receiving control tokens from FD, valued with either the left or right letter in $LL$, $SS$, $LS$, $SL$ or with $M$.

Detectors $BD_L$ and $BD_R$ identify the appropriate number and order of Short and Long blocks based on the frame scenario, which they receive from FD as control tokens valued $L$, $S$ or $M$. From the perspective of $BD_L$ and $BD_R$, block types $BL$ and $BS$ are refinements (subscenarios) of the scenarios $L$, $S$ and $M$. Figure 3b shows the three state machines associated with $BD_L$ as well as $BD_R$. Each of their states implies one of the possible subscenarios in \{LBL, SBS, MBL, MBS\}. The value of the control tokens produced by $BD_L$ and $BD_R$ to kernels $AR_L$, IMDCT$_L$, FI$_L$ and $AR_R$, IMDCT$_R$, FL$_R$ in each of the 4 possible subscenarios matches the last two letters of the subscenario name (i.e., $BL$ or $BS$). Although subscenarios LBL and MBL both send control tokens valued $BL$, the difference between them is the number of such tokens (similarly for subscenarios SBS and MBS).

Consider decoding of a Mixed frame. It implies the production of two $M$-valued tokens on the control port of detector $BD_L$. By interpreting each of these tokens, the state machine for scenario $M$ in Figure 3b makes one transition. Hence, $BD_L$ uses subscenario MBL for its first firing and subscenario MBS for its second firing. In subscenario MBL, $BD_L$ sends 2 $BL$-valued to kernels $AR_L$, IMDCT$_L$ and SPF$_L$, while 90 $BS$-valued tokens are produced in subscenario MBS. As a result, $AR_L$, IMDCT$_L$ and SPF$_L$ first process 2 Long blocks and subsequently 90 Short blocks as required for Mixed frames.

The example of Mixed frames highlights a unique feature of SADF: reconfigurations may occur during an iteration. An iteration of the MP3 decoder corresponds to processing frames, while block type dependent variations occur during processing Mixed frames. Supporting reconfiguration within iterations is fundamentally different from assumptions underlying other dynamic dataflow models, including for example PSDF. The concept is orthogonal to hierarchical control. Hierarchical control is also different from other dataflow models with hierarchy such as Heterogeneous Dataflow [7]. SADF allows pipelined execution of the controlling and controlled behavior together, while other approaches commonly prescribe that the controlled behavior must first finish completely before the controlling behavior may continue.
1.2 Analysis

Various analysis techniques exist for SADF, allowing the evaluation of both qualitative properties (such as consistency and absence of deadlock) and best/worst-case and average-case quantitative properties (like minimal and average throughput). We briefly discuss consistency of SADF graphs. The MPEG-4 decoder is an example of a class of SADF graphs where each scenario is like a consistent SDF graph and scenario changes occur at iteration boundaries of these scenario graphs (but still pipelined). Such SADF graphs are said to be strongly consistent [25], which is easy to check as it results from structural properties only. The SADF graph of the MP3 decoder does not satisfy these structural properties (for Mixed frames), but it can still be implemented in bounded memory. The required consistency property is called weak consistency [21, 4]. Checking weak consistency requires taking the possible (sub)scenario sequences as captured by the state machines associated to detectors into account, which complicates a consistency check considerably.

Analysis of quantitative properties and the efficiency of the underlying techniques depend on the selected type of state machine associated to detectors as well as the chosen time model. For example, one possibility is to use non-deterministic state machines, which merely specify what sequences of (sub)scenarios can occur but not how often. This typically enables best/worst-case analysis. Applying the techniques in [2, 4, 5] then allows computing that a throughput of processing 0.253 frames per kCycle can be guaranteed for the MPEG-4 decoder. An alternative is to use probabilistic state machines (i.e., Markov chains), which also capture the occurrence probabilities of the (sub)scenario sequences to allow for average-case analysis as well. Assuming that scenarios $I, P_0, P_{90}, P_{60}, P_{70}, P_{80}$ and $P_{99}$ of the MPEG-4 decoder may occur in any order and with probabilities 0.12, 0.02, 0.05, 0.25, 0.25, 0.09, 0.09 and 0.04 respectively, the techniques in [22] compute that the MPEG-4 decoder processes on average 0.426 frames per kCycle.

The semantics of SADF graphs where Markov chains are associated to detectors while assuming generic discrete execution time distributions$^5$ has been defined in [21] by using Timed Probabilistic Systems (TPS) as formal semantic model. Such transition systems operationalize the behavior with states and guarded transitions that capture events like the begin and end of each of the two steps in firing actors and progress of time. In case an SADF graph yields a TPS with finite state space, it is amenable to analysis techniques for (Priced) Timed Automata or Markov Decision Processes and Markov Chains by defining reward structures as also used in (probabilistic) model checking. In [22], specific properties of dataflow models in general and SADF in particular are discussed that enable substantial state-space reductions during such analysis. The underlying techniques have been implemented in [23] in the SDF$^3$ tool kit [17], covering the computation of worst/best-case and average-case properties for SADF including throughput and various forms of latency and buffer occupancy metrics [23]. Other variants of Scenario-Aware Dataflow have been proposed that are supported by exact analysis techniques using formal semantic

---

$^5$ This covers the case of constant execution times as so-called point distributions [21, 22].
models. The techniques presented in [26, 10, 11] exploit Interactive Marchov Chains (IMC) to combine the association of Markov chains to detectors with exponentially distributed execution times, which allows for example computing the response time distribution of the MPEG-4 decoder to complete processing the first frame [26]. A further generalisation of the time model for Scenario-Aware Dataflow with Markov Chains associated to detectors is proposed in [8]. This generalisation is based on the formal semantic model of Stochastic Timed Automata (STA) and further allows for scenario-dependent cost annotations to compute for example energy consumption. In case exact computation is hampered by state-space explosion, [25, 23] exploit an automated translation into process algebraic models expressed in the Parallel Object-Oriented Specification Language (POOSL) [24], which supports statistical model checking (simulation-based estimation) of various average-case properties.

In case of abstracting from the stochastic aspects of execution times and scenario occurrences, SADF is still amenable to worst/best-case analysis. Since SADF graphs are timed dataflow graphs, they exhibit linear timing behavior [12, 27, 2], which facilitates network-level worst/best-case analysis by considering the worst/best-case execution times for individual actors. For linear timed systems this is know to lead to the overall worst/best-case performance. For the class of SADF graphs with a single detector (often called FSM-based SADF), very efficient performance analysis can be done based on a \((\max, +)\)-algebraic interpretation of the operational semantics. It allows for worst-case throughput analysis, some latency analysis and can find critical scenario sequences without explicitly exploring the underlying state-space. Instead, the analysis is performed by means of state-space analysis and maximum-cycle ratio analysis of the equivalent but much smaller \((\max, +)\)-automaton [2, 4, 5]. Reference [4] shows how this analysis can be extended for weakly-consistent SADF graphs. An alternative to using \((\max, +)\)-algebra is proposed in [16], where the formal semantic model of Timed Automata (TA) is exploited to enable analyzing various qualitative and quantitative properties.

1.3 Synthesis

FSM-based SADF graphs have been extensively studied for implementation on (heterogeneous) multi-processor platforms [19, 9]. Variations in resource requirements need to be exploited to limit resource usage without violating any timing requirements. The result of the design flow for FSM-based SADF implemented in the SDF³ tool kit [17] is a set of Pareto optimal mappings that provide a trade-off in valid resource usages. For certain mappings, the application may use many computational resources and few storage resources, whereas an opposite situation may exist for other mappings. At run-time, the most suitable mapping is selected based on the available resources not used by concurrently running applications [15]. We highlight two key aspects of the design flow of [19, 17]. The first concerns mapping channels onto (possibly shared) storage resources. Like other dataflow models, SADF associates unbounded buffers with channels, but a complete graph
may still be implemented in bounded memory. FSM-based SADF allows for effi-
cient compile-time analysis of the impact that certain buffer sizes have on the timing
of the application. Hence, a synthesized implementation does not require run-time
buffer management, thereby making it easier to guarantee timing. The design flow
in [19] dimensions the buffer sizes of all individual channels in the graph sufficiently
large to ensure that timing (i.e., throughput) constraints are met but also as small as
possible to save memory and energy. It exploits the techniques of [18] to analyze
the trade-off between buffer sizes and throughput for each individual scenario in
the FSM-based SADF graph. After computing the trade-off space for all individual
scenarios, a unified trade-off space for all scenarios is created. The same buffer
size is assigned to a channel in all scenarios. Combining the individual spaces is
done using Pareto algebra [3] by taking the free product of all trade-off spaces and
selecting only the Pareto optimal points in the resulting space. Figure 4 shows the
trade-off space for the individual scenarios in the MPEG-4 decoder. In this appli-
cation, the set of Pareto points that describe the trade-off between throughput and
buffer size in scenario $P_{99}$ dominate the trade-off points of all other scenarios. Uni-
fying the trade-off spaces of the individual scenarios therefore results in the trade-off
space corresponding to scenario $P_{99}$. After computing the unified throughput/buffer
trade-off space, the synthesis process in [19] selects a Pareto point with the smallest
buffer size assignment that satisfies the throughput constraint as a means to allocate
the required memory resources in the multiprocessor platform.

A second key aspect of the synthesis process is the fact that actors of the same or
different applications may share resources. The set of concurrently active applications
is typically unknown at compile-time. It is therefore not possible to construct a single static-order schedule for actors of different applications. The design flow
in [19] uses static-order schedules for actors of the same application, but sharing
of resources between different applications is handled by run-time schedulers with
TDMA policies. It uses a binary search algorithm to compute the minimal TDMA
time slices ensuring that the throughput constraint of an application is met. By minimizing the TDMA time slices, resources are saved for other applications. Identification of the minimal TDMA time slices works as follows. In [1], it is shown that the timing impact of a TDMA scheduler can be modeled into the execution time of actors. This approach is used to model the TDMA time slice allocation it computes. Throughput analysis is then performed on the modified FSM-based SADF graph. When the throughput constraint is met, the TDMA time slice allocation can be decreased. Otherwise it needs to be increased. This process continues until the minimal TDMA time slice allocation satisfying the throughput constraint is found.

References


