A survey of main dataflow MoCCs for CPS design and verification

Guillaume Roumage, Selma Azaiez, Stephane Louise

A survey of main dataflow MoCCs for CPS design and verification

Guillaume Roumage, Selma Azaiez, Stéphane Louise
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
{guillaume.roumage, selma.azaiez, stephane.louise}@cea.fr

***Abstract*—The automotive industry has recently emphasized reducing the number of Electronic Control Units (ECUs) installed in vehicles for economic and ecological reasons. This reduction means that the design and verification must be independent of the vehicle's final choice of (MC)SoCs, knowing they will evolve as time passes. To that end, dataflow Models of Computation and Communication (MoCCs) are powerful tools for maintaining this independence. A subclass of dataflow MoCCs –deterministic dataflow MoCCs– is of particular interest since it allows designers to derive safety and security properties at compile-time. This work proposes a short survey of the existing deterministic dataflow MoCCs. We describe the properties of each dataflow MoCC and present an expressiveness hierarchy of dataflow MoCCs adjustable to designers' needs.**

***Index Terms*—MoCC, CPS design, dataflow graph, survey**

I. Introduction

The current trend in automotive, avionics, and aeronautics embedded systems is to accelerate the transition from several dozen or hundreds of Electronic Control Units (ECUs) to a few Multi -and Many- core SoCs (MC-SoCs). ECUs are simple and generally low-profile processors and Systems-on-Chip (SoC). MC-SoCs provide a software-dominated integration of functions and are usually partially redundant for safety. A transition is also occurring for communication systems in the automotive industry. The constraints in data communications (e.g., several high-resolution cameras, radars, even lidars) have led to the evolution of the old-fashioned CAN bus to Ethernet and PCI buses. Several factors have driven this evolution, such as new functionalities (e.g., ADAS –Advanced Driver Assistance System– or semi-autonomous vehicles), low-cost high-profile MC-SoCs available on mass-market smartphones, and a higher conscience of the ecological impact of embedded electronics. The current supply chain crisis in electronics has also contributed to this evolution.

Embedded systems are now referred to as Cyber-Physical System (CPS). Mathematically sound methods enable an agile and versatile evolution of CPS design without compromising safety, security, and cost (particularly concerning time-constrained CPSs). A mathematically grounded approach to CPS design allows a system to be adapted to a new line of MC-SoC components without re-qualifying the whole software architecture. A mathematical software architecture model is sometimes called a Model of Computation and Communication (MoCC). MoCCs rank from "simple" ones, e.g., the Von Neumann model, to higher-level models on which we focus in this paper.

CPSs must fulfill various constraints such as scheduling order, latency, throughput, memory footprint, and deadline. Some dataflow (DF) Models of Computation and Communication (dataflow MoCCs/DF MoCCs) ensure the satisfiability of those constraints at compile-time if no run-time fault occurs. A MoCC applied to a CPS functions as a set of rules that defines the behavior of the CPS's entities, individually and collectively. This work surveys many DF MoCCs in the literature. Each DF MoCC provides different degrees of freedom to abstract a real-life system.

The paper is organized as follows: section II presents the benefits of designing a CPS with a subclass of DF MoCCs, the deterministic DF MoCCs. Section III presents the standard foundations of all DF MoCCs. We refine those foundations and submit a short survey in section IV. Throughout this survey, DF MoCCs are described according to *features*, which are elements that describe the system's behavior and functioning. In section V, we evaluate those features resulting in an expressiveness hierarchy. We conclude the paper in section VI.

II. Deterministic dataflow MoCC-based system design

Deterministic DF MoCCs is an interesting subclass of DF MoCCs. Static analyses and safety properties of a CPS can be derived through the prism of such MoCCs. They allow designers to ensure critical safety criteria of an embedded system at compile-time. Deterministic DF MoCCs permit the prediction of the run-time system's behavior, static sizing of data buffers, communication channels process, some scheduling aspects, and proof of the system's correctness.

The need to closely model various real-life systems has led to the development of many DF MoCCs. A trade-off must be found between three aspects that characterize a MoCC:

- The expressiveness and compactness define which system can be modeled and how cumbersome the model can be.
- The implementation efficiency is influenced by the code size or the complexity of the run-time scheduling issues.
- The analyzability is the ability to derive safety and security properties from the model.

III. Shared background for dataflow MoCCs

The wide variety of DF MoCCs shares the same background, as they all represent CPSs with a directed graph called the Dataflow Graph (DFG). The DFG models the CPS, and the

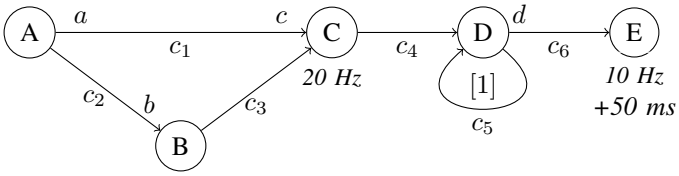

Fig. 1. A generic dataflow graph denoted G. The letters at both channels' endpoints model the production/consumption rate between the actors. Rates of 1 are omitted for clarity. Rates a to d take their value in the range rate of the DF MoCC that rule the DFG. Optional initial tokens –between brackets– might be in the channel, as for channel c_5: there is one initial token for the self-loop of actor D. Actor C has a firing frequency of 20 Hz, and actor E has both frequency and phase constraints: it must fire at 10 Hz after a delay of 50 ms. Frequencies and phases are optional.

DF MoCC interprets the behavior of the CPS through the DFG. The nodes of a DFG are actors that perform computations, and the arcs are channels through which the actors at both endpoints communicate by exchanging data tokens. A token is the atomic data object. The internal functioning of the actors is usually a black box.

A. Formal definition of the Dataflow Graph

An actor a is a tuple (I_a, O_a) with I_a (resp. O_a) the set of input (resp. output) ports –possibly empty– of a such that $I_a \cap O_a = \emptyset$. A channel c is a tuple $(a_p, a_c, prod, cons, init)$ that connects an output port of a producer actor a_p to an input port of a consumer actor a_c. A DFG denoted G is a tuple (A, C) where A is a set of actors, and C is a set of channels such that G is not the union of disjoint graphs.

Actors of DFGs usually share standard rules as follows. Let us consider a channel c that connects a producer actor to a consumer actor. Whenever the producer (resp. consumer) fires (i.e., is invoked), it writes (resp. reads) an amount of $prod$ (resp. $cons$) tokens to (resp. from) the buffer of c that usually has a bounded FIFO structure. An actor fires by executing an execution function. Some initial tokens $init$ might be in the buffer before any fire. An actor is enabled to fire if a set of firing rules is fulfilled. A necessary condition (but not always sufficient) for an actor to be enabled is a sufficient number of tokens in its input channels (i.e., more significant than the consumption rate). The actor's firing is an atomic process unless Best/Worst-Case Execution (BCET/WCET) Time are provided.

B. Static analyses of the Dataflow Graph

Deterministic DF MoCCs allow designers to derive static analyses of a CPS. The consistency and liveness analysis are arguably the most important. Consistency asserts that a CPS can run indefinitely in finite memory (prevent channel overflow), and liveness asserts that a CPS does not deadlock (prevent channel underflow). Valid consistency and liveness analysis imply that a schedule (i.e., a partial ordering of actors' firing) can be built at compile-time. Table I summarizes the properties that define the static analyzability of a CPS.

A static DFG is represented with its topology matrix Γ where γ_{ij} is the rate of actor j on the channel i, positive if it represents a production of data tokens and negative otherwise. Other entries are null, and the self-loop channels are not written. The topology matrix is crucial for analyzing a DFG through algebraic manipulations. The following matrix represents the topology of the DFG in Fig. 1:

$$\Gamma_G = \begin{array}{c} \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{pmatrix} a & 0 & -c & 0 & 0 \\ 1 & -b & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & d & -1 \end{pmatrix} \end{array} \begin{array}{l} c_1 = (A, C) \\ c_2 = (A, B) \\ c_3 = (B, C) \\ c_4 = (C, D) \\ c_6 = (D, E) \end{array}$$

The consistency is equivalent to the boundedness when the underlying DF MoCC that interprets the DFG has no parametric rates. Consistency and boundedness are two interpretations of whether a DFG executes in bounded memory. An *iteration* of a DFG is a set of actors' fires that keep the distribution token unchanged. The *repetition vector* of a DFG is the column vector that associates the number of times each actor fires within a single iteration. A DFG is bounded if an iteration needs a finite amount of memory, and a DFG is consistent if a non-trivial repetition vector exists (i.e., Γ_G has a non-trivial kernel). BCET and WCET enable the computation of more refined analyses, such as end-to-end latency or throughput.

TABLE I
CPS'S STATIC ANALYSES DERIVED WITH A DETERMINISTIC DATAFLOW MOCC

Property	Acronym	Definition
Consistency	*Co*	The DF MoCC ensures that there exists at least one unbounded execution in bounded memory
Liveness	*Li*	The DF MoCC ensures that there exists at least one deadlock-free execution
Memory	*Me*	The DF MoCC can statically compute channels' memory footprint of an execution
Schedule	*Sc*	The DF MoCC can statically schedule the DFG
Quasi-static Schedule	*QSc*	The DF MoCC can statically generate a quasi-static schedule
Determinism	*De*	Sequences of data produced in channels do not depend on the actors' scheduling
Latency	*La*	The DF MoCC can compute the time between two consecutive firings of an actor
Throughput	*Th*	The DF MoCC can compute the number of actor's firings during a period

C. Timing constraints and Dataflow Graph

The main distinctive property of CPS is timeliness and enforcing time constraints. This property is especially true when individual real-time tasks have periodic behaviors, which is a usual case. Typically, the period and the phase shift of each task/actor would be part of the specifications and initial design because they are rooted in physical and hardware constraints (e.g., maximum speed, inertia, camera frame rate, lidar, and radar output rates, ADC frequencies, servomotors or stepper motors sampling, etc.). All the frequencies rarely match. Even if they would, as sources are different, a small amount of clock drift should be expected and allowed to have a robust system. In practice, data-fusion between captors before calculating decisions for actuators needs to accommodate the

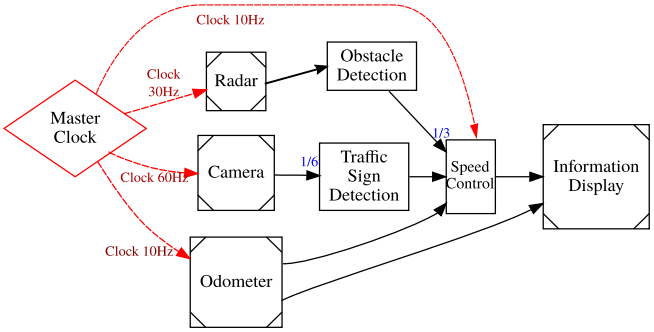

Fig. 2. An ADAS application excerpt with several real-time clocks shows the necessity of under-sampling on some channels. The PolyGraph DF formalism model the ADAS application excerpt.

discrepancy between clock rates in some way. Thus, for such applications, a mechanism is required for sub-sampling some channels in data-fusion actors. In practice, such sub-sampling can be modeled by periodically changing rates (e.g., CSDF, section IV-B) or rational rates (e.g., PolyGraph, section IV-C).

A typical excerpt of an ADAS application is shown in Fig. 2. It presents several captors with their own clocks and two actuators (*Speed control* and *Information Display*). *Speed Control* is a fusion actor with several input channels. Thus, some sub-sampling is required either at its input channel or upstream.

D. Features of deterministic dataflow MoCCs

The DFG is the basic brick of systems modeled with a dataflow perspective. A DF MoCC interprets the system behavior through its DFG. Each DF MoCC emphasizes a set of elements called features. A feature is an element that describes the system's behavior and functioning. The proposed survey evaluates the static analyzability (defined in table I) and the features (defined in table II) of the main DF MoCCs.

IV. Classification of Dataflow MoCCs

We propose to classify the main DF MoCCs into nine categories as follows: the Synchronous Dataflow and its extensions (section IV-A and table III), the Phase-based DF MoCCs (section IV-B and table IV), the DF MoCCs with timing constraints (section IV-C and table V), the Boolean-based DF MoCCs (Section IV-D and table VI), the Scenario-based DF MoCCs (section IV-E and table VII), the DF MoCCs with Enable and Invoke capabilities (section IV-F and table VIII), the DF MoCCs with unique features (section IV-G and table IX), the process network-based dataflow MoCCs (section IV-H and table X) and the meta-models for DF MoCCs (section IV-I and table XI).

A. Synchronous Dataflow and its extension

1) SDF: The principles of dataflow models first appeared in [1]. However, the analyzability of that first model is limited,

TABLE II
Features emphasized by dataflow MoCCs

Feature (Acronym)	Definition	Evaluation
Auto-concurrency (AC)	Actors can have multiple firings simultaneously	Presence or absence
Phases (Ph)	Actors can have rates with a cyclic pattern. An element of such a cycle is a phase.	
Initial and Steady Sequences (IniSteS)	Actors can have initialization phases	
Parameters (Pa)	Actors and channels can have parametric rates	
Hierarchy (Hi)	The DF MoCC can describe an actor with a DFG ruled with the same DF MoCC as its parent DFG	
Meta-Model (MM)	A meta-model DF MoCC adds additional rules on top of a DFG modeled with a non-meta-model DF MoCC	
Initial Tokens (IT)	Tokens can be present in the channels' buffer before the start of an execution	
Global State (GS)	Shared data for actors of the DFG	
Multi-Dimensional FIFO (MDF)	The DF MoCC can describe the channels' FIFO as multi-dimensional lattices	
Timing Constraints (TiCons)	Actors can have delay/frequency constraints	
Consumption Threshold (CT)	A necessary condition for an actor's firings is a number of tokens in their input channels above a threshold	
Multiple Execution Modes (MEM)	A mechanism that may affect an actor's behavior without necessarily affecting its rates	
Sliding Window (SWi)	Token consumption with a sliding window	
Initialization and discard of initial token (IniDisIT)	A mechanism that control the initialization/discard of initial tokens at the start/end of each iteration of the DFG	
Range rate	The domain in which rates attain their values	$\{1\}, \mathbb{N}^*, \mathbb{N}, \mathbb{N}^* \times ... \times \mathbb{N}^*, \mathbb{Q}^*, \Omega$
Rate and topology	Periodicity of rates/topology updates. Rates/topology are dynamic if the MoCC's rules and/or internal actor actions allow an update of rates/topology at run-time.	{fix, between, within} iterations

and Synchronous Dataflow ([2]) has laid the foundations of the dataflow paradigm we use today. SDF models a CPS with a DFG where the rates belong to $\mathbb{N}^*$. Reference [3] provides consistency and liveness checking.

Despite its low expressiveness, SDF and its variants have been extensively studied because they can model many applications. Besides, researchers have created implementations of SDF. Thus, many works have previously researched memory consumption but with notably distinct techniques. For example, a shared buffer memory model is studied in [4]. A trade-off between buffer requirements and throughput constraints with a non-shared buffer model is explored in [5]. The authors of [6] provide arithmetic manipulations to compute minimum buffers size that yield a deadlock-free schedule. The authors of [7] chose a model-checking approach. In [8], a linear programming formulation computes the buffer size with optimal throughput without degrading storage constraints.

The latency is an important performance indicator explored in [9]. Regarding implementation efficiency, Scalable Synchronous Dataflow (SSDF, [10]) is a specific SDF implementation that minimizes code size and context-switch overhead.

2) HSDF: The restriction of rates' values of SDF to $\{1\}$ yields Homogeneous Synchronous Dataflow (HSDF, [2]). Many static analyses of SDF apply to HSDF.

3) HSDF^a: Homogeneous Synchronous Dataflow with auto-concurrency (HSDF^a, [11]) determines the consumption order of tokens with static indices independently of the pro-

duction order, dismissing the channels' FIFO policy. HSDF^a provides an end-to-end latency using a timed automata model.

4) BDDF: Bounded Dynamic Dataflow (BDDF, [12]) extends SSDF by allowing a set of actors to have dynamic and upper-bounded ports. The topology of the network might change at run-time. A Finite State Machine (FSM) models topology updates. Each state defines a set of connected actors.

5) CG: The Computation Graphs (CGs, [13]) are more general than SDF. The CGs associate a consumption threshold with each channel of the DFG. Thus, an actor can fire if the number of tokens in its input channel is more significant than that threshold. The authors of [13] develop properties that structure the determinism of the CG and provide a set of conditions that deadlock a CG.

6) SPDF: Schedulable Parametric Dataflow (SPDF, [14]) is a parametric extension of SDF. The parameters range in $\mathbb{N}^*$ and are communicated through a dedicated network inserted at the top of the DFG. The parameters are allowed to change within an iteration. SPDF can statically analyze a DFG concerning boundedness and liveness and computes a quasi-static schedule (i.e., a schedule made at compile-time that depends on parameter values known at run-time).

7) MDSDF: Multi-Dimensional Synchronous Dataflow (MDSDF, [15]) specifies the number of tokens produced/consumed as a multi-dimensional lattice. Thus, MDSDF is suitable to model signal processing applications (e.g., image processing). MDSDF can schedule a DFG at compile-time. Reference [15] gives a method to compute the repetition matrix (i.e., the repetition vector with many dimensions) and conditions to ensure deadlock-freeness.

8) WSDF: Windowed Synchronous Dataflow (WSDF, [16]) extends MDSDF by allowing token consumption with sliding windows. A token is consumed with a specific sampling pattern through a set of windows of predefined size. WSDF provides a boundedness checking.

9) IBSDF: Interface-Based Synchronous Dataflow (IBSDF, [17]) is a hierarchical extension of SDF. A source and sink node surround the DFG. They both behave as an interface to the environment. Each level of the hierarchy is analyzable.

10) ILDF: Interval-rate, Locally-static Dataflow (ILDF, [18]) models DFG's rates as a finite natural integers interval and fixes the value at the beginning of the execution. ILDF statically analyzes the DFG regarding consistency, buffer sizing, and latency, assuming a valid schedule is possible.

B. Phase-based dataflow MoCCs

1) CSDF: Cyclo-Static Dataflow (CSDF, [19]) models actors' execution function, production, and consumption rates as cyclic patterns defined at compile-time. The value changes periodically following that cycle. An element of such a cycle is a phase. The rates take their values in $\mathbb{N}$. Thus, some channels may be disabled for a few phases. CSDF provides consistency and liveness checkings using a conversion algorithm from CSDF to HSDF. The authors of [20] provide a trade-off between throughput and buffer size for the CSDF.

TABLE III
FEATURES AND STATIC ANALYZABILITY OF SYNCHRONOUS DATAFLOW AND ITS EXTENSIONS.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability	Turing Complete
SDF [2]–[9]	fix	fix	$\mathbb{N}^*$		*Co, Li, Me, Sc, La, Th*	○
HSDF [2]	fix	fix	$\{1\}$		*Co, Li, Me, Sc*	○
HSDF^a [11]	fix	fix	$\{1\}$	*AC*	*La*	○
BDDF [12]	within iterations	within iterations	$\mathbb{N}^*$		*Me*	○
CG [13]	fix	fix	$\mathbb{N}^*$	*CT*	*Li, De*	○
SPDF [14]	within iterations	fix	$\mathbb{N}^*$	*Pa*	*Co, Li, QSc*	○
MDSDF [15]	fix	fix	$\mathbb{N}^*$	*MDF*	*Co, Li, Sc*	○
WSDF [16]	fix	fix	$\mathbb{N}^*$	*MDF, SWi*	*Co*	○
IBSDF [17]	fix	fix	$\mathbb{N}^*$	*Hi*	*Co, Li, Sc*	○
ILDF [18]	fix	fix	$\mathbb{N}^* \times ... \times \mathbb{N}^*$		*Co, Me, La*	○

2) CSDF^a: Cyclo-Static Dataflow with Auto-concurrency (CSDF^a, [21]) allows the auto-concurrency in CSDF with a static token order; as in HSDF^a, the channels consequently no longer have a FIFO policy. The authors provide a buffer sizing computation. A set of mechanisms maintain the overhead independent of the replication factor (i.e., the number of simultaneous execution of an actor), e.g., predefined buffer's accesses pattern for read/write operations.

3) CDDF: Cyclo-Dynamic Dataflow (CDDF, [22]) is a dynamic version of CSDF. The execution function, token ratios, and firing sequence length can vary at run-time. The needed information of a previous actor execution for subsequent ones must be conveyed through a self-edge: a control token containing all firings information is read at each fire, including the code segment executed by the actor. Thus, CDDF has the feature "multiple execution modes". The data-dependent behavior of CDDF limits the static analyzability and implies run-time scheduling depending on the control token values.

4) PCG: The Phased Computation Graphs (PCGs, [23]) extend CSDF with consumption thresholds. The rates of PCGs are divided into initial and steady sequences. The initial sequence is performed at the beginning of the execution. The steady sequence, which is cyclic, takes over for the rest of the execution. The authors of [24] create a PCG generator and provide consistency and liveness checking and a lower bound for buffer sizing.

5) FRDF: Fractional Rate Dataflow (FRDF, [25]) is the first DF MoCC with rational rates. The semantic of rational rates is the following. An actor produces/consumes either a fraction $\frac{p}{q}$ of a token every firing or p tokens every q firings. In contrast with other DF MoCCs studied in this paper, FRDF does not have initial tokens and cannot derive a rate into a unique sequence of firings. Thus, the same rate may imply multiple production/execution patterns of data tokens. Therefore, rates may vary within iterations.

6) VPDF: Variable-rate Phased Dataflow (VPDF, [26]) extends VRDF, described further in section IV-G. VPDF inherits the structural constraints of VRDF, e.g., every parameter defines a single phase of a single actor. Actors' phases have two parameters: the number of repetitions and the rate of that phase. VPDF provides a buffer capacity computation for a throughput-constrained DFG.

TABLE IV
FEATURES AND STATIC ANALYZABILITY OF THE PHASE-BASED DATAFLOW MOCCS.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability	Turing Complete
CSDF [19], [20]	fix	fix	$\mathbb{N}$	*Ph*	*Co, Li, Me, Sc, Th*	○
CSDF$^{\alpha}$ [21]	fix	fix	$\mathbb{N}$	*AC, Ph*	*Me*	○
CDDF [22]	within iterations	within iterations	$\mathbb{N}$	*MEM, Ph*		○
PCG [23], [24]	fix	fix	$\mathbb{N}$	*CT, IniSteS, Ph*	*Co, Li, Me, Sc*	○
FRDF [25]	within iterations	fix	$\mathbb{Q}^*$	Absence of *IT, Ph*	*Co*	○
VPDF [26]	within iterations	within iterations	$\mathbb{N}$	*Pa, Ph*	*Me*	○

TABLE V
FEATURES AND STATIC ANALYZABILITY OF THE DATAFLOW MOCCS WITH TIMING CONSTRAINTS.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability	Turing Complete
TPDF [27]	within iterations	within iterations	$\mathbb{N}$	*MEM, Pa, Ph, TiCons*	*Co, Li, Sc*	○
PolyGraph [28], [29]	within iterations	within iterations	$\mathbb{Q}^*$	*MEM, Ph, TiCons*	*Co, Li, Sc*	○

C. Dataflow MoCCs with timing constraints

1) TPDF: Transaction Parameterized Dataflow (TPDF, [27]) variously extends CSDF. Rates can be parametric. TPDF has a select-duplicate and a transaction actor. Select-duplicate actor replicates its single entry into any combinations of its outputs, and the transaction actor is the symmetric process. Besides, TPDF provides clock constraints and actors' execution modes. A clock actor sends a control token periodically to (an)other actor(s). The control token defines the execution mode of the actor which consumes that control token, e.g., waiting for all input data to be available before fires or selecting the data with the highest priority. TPDF provides consistency, liveness checking, and a scheduling strategy.

2) PolyGraph: PolyGraph ([28]) enhances the semantic of rational rates of FRDF. A rate of $\frac{p}{q}$ means p tokens are produced/consumed every q firings. An actor's fire increases/decreases by $\frac{p}{q}$ the fractional number of tokens in the channels involved. The natural number of tokens in a channel is the fractional number of tokens rounded down. In contrast with FRDF, initials tokens permit to derive a unique firing sequence from a rational rate. An actor may have a frequency constraint and a delay. Thus, it must fire at that frequency, and its first fire occurs after the delay.

Reference [29] present a dynamic extension of PolyGraph. Actors of dynamic Polygraph label tokens with a mode. The mode of tokens consumed by an actor defines its behavior, e.g., modifying the algorithm that processes the tokens. PolyGraph and dynamic Polygraph both have a high expressiveness and are analyzable regarding consistency and liveness.

D. Boolean-based dataflow MoCCs

1) BDF and IDF: Boolean-controlled Dataflow (BDF, [30]) is the first dataflow MoCC focusing on topological modifications. BDF provides an if-then-else structure using two actors. The switch (resp. select) actor has one (resp. two) input port(s) and two (resp. one) output port(s). A boolean control token decides which port is used. BDF is Turing complete and weakly analyzable. A consistency analysis based on the proportion of true tokens provides only a probabilistic analysis.

TABLE VI
FEATURES AND STATIC ANALYZABILITY OF THE BOOLEAN-BASED DATAFLOW MOCCS.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability	Turing Complete
BDF [30]	fix	within iterations	$\mathbb{N}^*$	*MEM*		•
IDF [31]	fix	within iterations	$\mathbb{N}^*$	*MEM*		•
BPDF [32]	between iterations	within iterations	$\mathbb{N}^*$	*Pa*	*Co, Li, Sc*	○

Integer-controlled Dataflow (IDF, [31]) is a generalization of BDF where control tokens are any integer. Thus, the switch and select actors become case and end-case actors with many output/input ports. The behavior of switch/select and case/end-case actors induce the feature "multiple execution modes".

2) BPDF: Boolean Parametric Dataflow (BPDF, [32]) combines two types of parameters. Integer parameters express dynamic rates and boolean parameters on the channel of the DFG. Those latter dynamically (des)activate the channels. BPDF provides boundedness and liveness static checking.

E. Scenario-based dataflow MoCCs

1) SADF and SADF^T: Scenario-Aware Dataflow (SADF, [33]) models a system with a set of scenarios. A scenario is an assignation to parameterized rates. Some actors broadcast the current scenario to their followers. Scenarios are known at compile-time with a stochastic execution time distribution.

SADF has the determinism property. The behavior of a DFG model with SADF only depends on the probabilistic choices that determine the sequence of scenarios successively detected by each detector and not on the non-deterministic choices originating from the concurrency in the model.

SADF's scenarios can model complex control structures, including switch/select of the BDF model. Thus, SADF is Turing complete. SADF might be restricted to be non-Turing complete. In that case, SADF provides conditions for consistency, liveness, and determinism. We denote SADF^T as the non-restricted version of SADF, i.e., the Turing complete version.

2) FSM-SADF: Finite State Machine-based - Scenario-Aware Dataflow (FSM-SADF, [34]) restricts SADF. A set of scenarios captures the dynamic behavior. Each scenario is modeled with SDF. An FSM specifies the order in which the scenarios occur and the rates of the current scenario. The SDF graphs, together with the FSM, model the application. In contrast with SADF, the execution time of a scenario is fixed, and the auto-concurrency is enabled. FSM-SADF provides throughput, latency, and buffer size analyses.

3) FSM-PSADF: The Finite State Machine-based - Parameterized Scenario-Aware Dataflow (FSM-PSADF, [35]) uses parameters to improve the compactness of FSM-SADF. The scenario and the parameter configuration in that scenario are both non-deterministically chosen at the end of an iteration. FSM-PSADF develops latency and throughput analysis.

TABLE VII
FEATURES AND STATIC ANALYZABILITY OF THE SCENARIO-BASED DATAFLOW MOCCS.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability	Turing Complete
SADF [33]	within iterations	within iterations	$\mathbb{N}$	*Pa*	*Co, De, Li*	○
SADF^T [33]	within iterations	within iterations	$\mathbb{N}$	*Pa*		•
FSM-SADF [34]	between iterations	between iterations	$\mathbb{N}^*$	*AC*	*Co, Li, Me, La, Th*	○
PFSM-SADF [35]	between iterations	between iterations	$\mathbb{N}^*$	*AC, Pa*	*La, Th*	○

F. Dataflow MoCCs with Enable & Invoke capabilities

1) EIDF and CFDF: Enable-Invoke Dataflow (EIDF, [36]) endows actors with two capabilities and a set of modes. A mode defines the number of tokens consumed and produced. The enable capability asserts if an actor can fire in a given mode while the invoke capability performs a fire in that mode. The mode used for a fire is called the execution mode.

The invoke capability results in both the output tokens and the set of enabled modes for the subsequent firing. The Core Functional Dataflow (CFDF, [36]) behaves the same as EIDF, except that the invoke capability returns a single mode. The following mode description of EIDF is available for CFDF.

In contrast with PolyGraph's deciding mode procedure, EIDF's mode choice relies on a function, not token labeling. Besides, a mode in PolyGraph is finer-grained, e.g., it may influence the produced data type or the algorithm used.

The enable and invoke capabilities can be formulated to describe switch/select actors of BDF. Thus, EIDF and CFDF are Turing-complete. We classify enable and invoke capabilities as the "multiple execution modes" feature.

2) PSM-CFDF: Parameterized Set of Modes - Core Functional Dataflow (PSM-CFDF, [37]) is tailored for CFDF when the number of modes grows significantly. Actors have a set of parameters, and a configuration is an assignation to those parameters. Modes with related functionalities are clustered together and denoted as Parameterized Set of Modes (PSM). The active PSM and the active configuration uniquely determine the mode for the actor firing.

3) CF-PSDF: Core Functional - Parameterized Synchronous Dataflow (CF-PSDF, [38]) is a mix between PSDF (described further in section IV-I) and CFDF. A CF-PSDF actor has a set of modes and three graphs: the ctrl graph, the subctrl graph, and the body graph. The ctrl and subctrl graphs have the same role as the init and subinit graphs of PSDF. The ctrl graph decides the execution mode and transmits the mode information to the ctrl graph of subsequent CF-PSDF actors. Two distinct actors can control a CF-PSDF actor. The first sends mode information to the ctrl graph, and the second sends data to the body graph.

4) HCFDF: Hierarchical Core Functional Dataflow (HCFDF, [39]) views its actors as CFDF actors with a set of nested DFGs. Let H be an HCFDF actor. The nested DFGs match a subset of ports of H. A firing of H might be an invocation of a subset of the nested graphs, given that the dataflow interface defined by the mode is unchanged.

TABLE VIII
FEATURES OF THE DATAFLOW MOCCS WITH ENABLE & INVOKE CAPABILITIES.

MoCC	Rate	Topology	Range rate	Features	Turing Complete
EIDF [36]	within iterations	within iterations	$\mathbb{N}$	*MEM*	•
CFDF [36]	within iterations	within iterations	$\mathbb{N}$	*MEM*	•
PSM-CFDF [37]	within iterations	within iterations	$\mathbb{N}$	*Pa, MEM*	•
CF-PSDF [38]	within iterations	within iterations	$\mathbb{N}$	*Hi, MEM, Pa*	•
HCFDF [39]	within iterations	within iterations	$\mathbb{N}$	*Hi, MEM*	•

TABLE IX
FEATURES AND STATIC ANALYZABILITY OF THE DATAFLOW MOCCS WITH UNIQUE FEATURES.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability	Turing Complete
SPBDF [40]	fix	fix	$\mathbb{N}^*$	*GS*	*Co*	○
HDF [41]	between iterations	fix	$\mathbb{N}^*$	*Hi*	*Co, Li, Sc*	○
VRDF [42]	within iterations	within iterations	$\mathbb{N}$	*Pa*	*Me*	○

G. Dataflow MoCCs with unique features

1) SPBDF: Synchronous PiggyBacked Dataflow (SPBDF, [40]) provides a global state for the DFG. SPBDF provides a method for the memory requirements and consistency of the global state. In this context, consistency applies to paths of the DFG. A path is state-consistent if, for each firing of every actor on that path, consumed (resp. produced) tokens from (resp. in) the global table have the same value.

2) HDF: Heterochronous Dataflow (HDF, [41]) studied the combination between FSM and DF MoCC. For instance, an SDF actor can be an FSM, and conversely. The authors have also studied the combination of FSM with synchronous/reactive model and discrete events models. The liveness, consistency, and schedulability are decidable at compile-time.

3) VRDF: Variable Rate Dataflow (VRDF, [42]) is a parametric dataflow model which imposes many restrictions on parameter usage and strong structural constraint. For instance, the repetition vector solution for two actors using the same parameter must be equal. VRDF presents an algorithm for computing the required memory capacity.

H. Process network-based dataflow MoCCs

Actors of dataflow MoCCs we studied previously are functional. The output tokens of a firing are purely a function of the input tokens of that firing. Besides, the firing rules are sequential, i.e., they can be tested in a predefined order using only blocking reads ([43]). A sequence of actors' firings is a dataflow process, and a network of such processes is a dataflow process network.

1) KPN: The dataflow process networks are a particular case of Kahn Process Networks (KPNs, [44]). An actor of a KPN is a process that maps one or more (possibly infinite) input sequences to one or more output sequences. In contrast with the dataflow process network, actors of a KPN may have a state.

TABLE X
FEATURES OF THE PROCESS NETWORK-BASED DATAFLOW MOCCS.

MoCC	Rate	Topology	Range rate	Turing Complete
KPN [44]	within iterations	within iterations	Ω	•
RPN [45]	within iterations	within iterations	Ω	•

2) RPN: Reactive Process Networks (RPNs, [45]) is an extension of KPNs where the active configuration (i.e., the set of active processes and channels) may change at run-time. An RPN presents a static interface to the outside world that receives events and data tokens.

KPNs and RPNs have dynamic rates, dynamic topology, and a rage rate of Ω. Our semantic of Ω is any data, e.g., integer, boolean, pointer, etc. KPNs and RPNs do not have any previously studied features, and in their most general form, they are not statically analyzable.

I. Meta-models dataflow MoCCs

1) PSDF: Parameterized Synchronous Dataflow (PSDF, [46]) is a parametric meta-model applied to SDF. An actor is either primitive or hierarchical. A primitive actor is a PSDF subsystem composed of three graphs:

- The init graph handles the parameters' update that affects the body graph's rates (i.e., the dataflow interface) and handles parameters' modification of the subinit graph.
- The subinit graph modifies the body graph parameters that leave the dataflow interface unchanged.
- The body graph models the actor's behavior.

The reconfiguration capability of the subinit graph is more restricted than the init graph but occurs more often.

An actor is hierarchical if its body graph is itself a PSDF subsystem. The init graph performs modifications at the boundaries of an iteration of the PSDF subsystem to which it belongs. The subinit graph updates parameters within an iteration of its PSDF subsystem. Hence, a PSDF subsystem embedded in a hierarchical actor can change some rates of its parent subsystem within an iteration of that parent subsystem.

2) PCSDF: The authors of [46] apply their method to CSDF and yield the Parameterized Cyclo-Static Dataflow (PCSDF). The parameterization of PCSDF is less expressive than VPDF: phases' ratios and sequence fire length are parameterized, while in VPDF, an additional parameter to each phase permits to repeat it a parametric number of times.

3) HPDF: Homogeneous Parameterized Dataflow (HPDF, [47]) is a DF MoCC that refines a top-level actor of the DFG using any dataflow semantic with a well-defined notion of iteration (e.g., SDF, CSDF).

4) PIMM: Parameterized and Interfaced Meta-Model (PIMM, [48]) extends the semantics of any deterministic DF MoCC. To that end, PIMM uses an interface-based hierarchy and a set of parameters. The application of PIMM to SDF yields the Parameterized and Interfaced Synchronous Dataflow (PISDF), which can be seen as an extension of IBSDF.

TABLE XI
FEATURES AND STATIC ANALYZABILITY OF THE META-MODEL DATAFLOW MOCCS.

MoCC	Rate	Topology	Range rate	Features	Static Analyzability
PSDF [46]	within iterations	fix	$\mathbb{N}^*$	*Hi, MM, Pa*	*QSc (for a subclass of PSDF)*
PCSDF [46]	within iterations	fix	$\mathbb{N}$	*Hi, MM, Pa, Ph*	*QSc (for a subclass of PCSDF)*
HPDF [47]	between iterations	fix	$\mathbb{N}^*$	*MM, Pa*	*QSc*
PIMM [48] (PISDF)	within iterations	within iterations	$\mathbb{N}$	*Hi, MM, Pa*	*QSc (for a subclass of PISDF)*
RDF [49]	between iterations	between iterations	$\mathbb{N}^*$	*MM*	*Co, Li*
SAD [50]	fix	fix	$\mathbb{N}^*$	*IniDisIT, MM*	*Co, Li, Sc*

5) RDF: Reconfigurable Dataflow (RDF, [49]) is a DFG with a controller that specifies how and when the DFG may be reconfigured. Graph rewrite rules are applied if specific conditions are fulfilled. RDF verifies liveness and consistency for the initial DFG configuration and all possible transformations.

6) SAD: State-Aware Dataflow (SAD, [50]) tackles the memory persistence of initial tokens across the DFG's iterations. SAD extends the semantics of the initial tokens with an explicit initialization/discard at the start/end of each iteration.

V. EXPRESSIVENESS HIERARCHY FOR DATAFLOW MOCCS

A. Protocol to create an expressiveness hierarchy

The expressiveness hierarchy we propose can be seen as an extension of the DF MoCCs comparison in [34]. The protocol to create an expressiveness hierarchy is the following:

1) Characterize each DF MoCC according to the features described in table II.
2) Assign a score for each feature, then normalize it, i.e., divide it by the maximum possible score. The normalization allows features with a maximum score above 1 to be comparable with others.
3) Classify the features by order of interest in categories and assign a coefficient to each category (a category may be reduced to a single feature). The coefficient assignation allows a designer to increase or reduce the importance of features according to its need.
4) Compute the expressiveness score for each DF MoCC by summing the normalized features' score with the correct weighting, then sort DF MoCCs with respect to their expressiveness score.

B. Features evaluation of dataflow MoCCs

We evaluate the range rate feature as follows. Singletons $\{1\}$ have a score of 0. The score is incremented by 1 – representing an increase in expressiveness– in the following the order: $\{1\}, \mathbb{N}^*, \mathbb{N}, \mathbb{N}^* \times ... \times \mathbb{N}^*, \mathbb{Q}^*, \Omega$.

We assume a DF MoCC with both dynamic rates and dynamic topology can model more systems than the union of the systems modeled by a DF MoCC with only dynamic topology and those modeled by a DF MoCC with only dynamic rate. This latter assumption is represented in table XII. For instance, we evaluate a DF MoCC with topology and rate updates between iterations with 3, not 2. All other feature is single-choice properties evaluated as 1 if present and 0 otherwise.

TABLE XII
EVALUATION OF RATES AND TOPOLOGY UPDATES.

Rate & topology updates across DFG iterations		Topology		
		Fix	Between	Within
Rate	Fix	0	1	2
	Between	1	3	4
	Within	2	4	5

TABLE XIII
FEATURES CLASSIFICATION TO CHOOSE THE MOST SUITABLE MOCC TO DEVELOP A RUNTIME ADAPTATION TOOL FOR TIMED-CONSTRAINED CPS.

Coefficient	2	1	0
Features	TiCons, MEM, Range rate, Rate & topology, IT	Ph, IniSteS, Pa, Hi, AC, CT	MM, MDF, GS, IniDisIT, SWi

C. Features classification and expressiveness score

In future work, we want to develop a tool that provides a run-time fault mitigation capability for time-constrained CPS modeled by a DF MoCC (a typical use case is the ADAS, as shown in Fig. 2). For that purpose, we classify the features in table XIII to choose the most suitable DF MoCC. We choose coefficient 2 for crucial features, coefficient 1 for interesting features yet not mandatory, and the null coefficient for unneeded features. Fig. 3 shows the expressiveness score for the DF MoCCs studied in our survey using those coefficients.

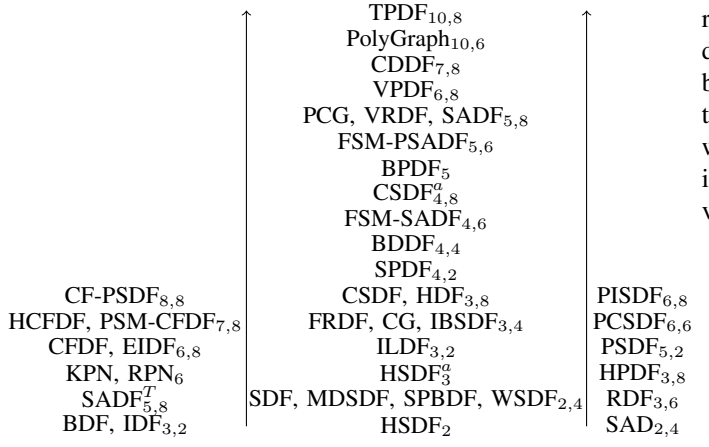

Fig. 3. The expressiveness hierarchy with the expressiveness score (computed with the coefficients of table XIII) shows as a subscript. The Turing complete dataflow MoCCs are on the left, the non-Turing complete ones are in the middle, and the meta-models MoCCs are on the right.

D. Discussion about the expressiveness hierarchy

As shown in Fig. 3, TPDF and PolyGraph are the most suitable DF MoCCs for our needs in modeling CPS: deterministic, statically schedulable, and sizable MoCCs.

The hierarchy methodology we propose has limits. Some features may overlap, e.g., parameters and multiple execution modes, since both may update rates. However, we focus on what features a MoCC has and not how it is modeled or implemented. The hierarchy presents the state of the art knowledge and straightforward claims. We assume that some MoCCs can be enhanced and thus be more expressive. However, it requires significant work and is out of this survey's scope.

Also, we consider auto-concurrency disabled by default since, without a dedicated mechanism, e.g., static token indices, we cannot ensure the non-overlapping of tokens because of different actors' execution times.

The expressiveness hierarchy is extensible. On the one hand, to introduce a new feature, the designer must define how to evaluate it and apply the preferred coefficient. On the other hand, a DF MoCC is added to the hierarchy with an evaluation according to the feature list.

VI. CONCLUSION AND FUTURE WORKS

This work surveys the main dataflow MoCCs, focusing on a highly interesting subclass –deterministic dataflow MoCCs– since they can derive a system's static analyses and safety properties at compile-time. We classify dataflow MoCCs into nine categories with a description of the elements that make each dataflow MoCC unique, as well as their analyzability.

We propose a flexible and extensible expressiveness hierarchy to help system designers to choose the most suitable MoCC for their needs. The hierarchy gives a reliable intuition rather than a strict claim of expressiveness because each dataflow MoCC has unique features.

The validity of the properties derived from static analysis incorrectly assumes that there is never a run-time fault in the system. In future work, we want to develop a tool that performs runtime verification to ensure that static analyses made with dataflow MoCCs are valid. Thus, we could ensure consistency between dataflow application models and their implementation running on MC-SoCs. The runtime verification process would improve the system's safety and security and could be incorporated into the Quality Assurance process during CPS validations.

REFERENCES

[1] J. Dennis, "First version of a data flow procedure language," in *Programming Symposium*, vol. 19, 1974, pp. 362–376.

[2] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, pp. 1235–1245, 1987.

[3] ——, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Trans. Comput.*, vol. 36, pp. 24–35, 1987.

[4] P. Murthy and S. Battacharyya, "Shared memory implementations of synchronous dataflow specifications," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, 2000, pp. 404–410.

[5] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *Proceedings of the 43rd Design Automation Conference*, 2006, pp. 899–904.

[6] M. Adé, R. Lauwereins, and J. Peperstraete, "Data Memory Minimisation for Synchronous Data Flow Graphs Emulated on DSP-FPGA Targets," in *Proceedings - Design Automation Conference*, 1997, pp. 64–69.

[7] M. Geilen, T. Basten, and S. Stuijk, "Minimising Buffer Requirements of Synchronous Dataflow Graphs with Model Checking," in *Proceedings of the 42nd Annual Conference on Design Automation - DAC '05*, 2005, pp. 819–824.

[8] R. Govindarajan, G. Gao, and P. Desai, "Minimizing Buffer Requirements under Rate-Optimal Schedule in Regular Dataflow Networks," *Journal of VLSI Signal Processing*, vol. 31, pp. 207–229, 2002.

[9] A. Ghamarian, S. Stuijk, T. Basten, M. Geilen, and B. Theelen, "Latency Minimization for Synchronous Data Flow Graphs," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*. IEEE, 2007, pp. 189–196.

[10] S. Ritz, M. Pankert, and H. Meyr, "High level software synthesis for signal processing systems," in *Proceedings of the International Conference on Application Specific Array Processors*, 1992, pp. 679–693.

[11] G. Kuiper and M. Bekooij, "Latency analysis of homogeneous synchronous dataflow graphs using timed automata," in *Design, Automation & Test in Europe Conference & Exhibition, 2017*, 2017, pp. 902–905.

[12] M. Pankert, O. Mauss, S. Ritz, and H. Meyr, "Dynamic data flow and control flow in high level DSP code synthesis," in *Proceedings of ICASSP '94. IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. ii, 1994, pp. II/449–II/452.

[13] R. Karp and R. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," *SIAM J. Appl. Math.*, vol. 14, pp. 1390–1411, 1966.

[14] P. Fradet, A. Girault, and P. Poplavko, "SPDF: A schedulable parametric data-flow MoC," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 769–774.

[15] P. Murthy and E. Lee, "Multidimensional synchronous dataflow," *IEEE Transactions on Signal Processing*, vol. 50, pp. 2064–2079, 2002.

[16] J. Keinert, C. Haubelt, and J. Teich, "Modeling and Analysis of Windowed Synchronous Algorithms," in *2006 IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*, vol. 3, 2006, pp. III–892–III–895.

[17] J. Piat, S. Bhattacharyya, and M. Raulet, "Interface-based hierarchy for synchronous data-flow graphs," in *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, 2009, pp. 145–150.

[18] J. Teich and S. S. Bhattacharyya, "Analysis of Dataflow Programs with Interval-limited Data-rates," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 43, pp. 247–258, 2006.

[19] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, pp. 397–408, 1996.

[20] S. Stuijk, M. Geilen, and T. Basten, "Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs," *IEEE Trans. Comput.*, vol. 57, pp. 1331–1345, 2008.

[21] P. Koek, S. Geuns, J. Hausmans, H. Corporaal, and M. Bekooij, "CSDFa: A Model for Exploiting the Trade-Off between Data and Pipeline Parallelism," in *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*, 2016, pp. 30–39.

[22] P. Wauters, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-dynamic dataflow," in *Proceedings of 4th Euromicro Workshop on Parallel and Distributed Processing*, 1996, pp. 319–326.

[23] W. Thies, J. Lin, and S. Amarasinghe, "Phased Computation Graphs in the Polyhedral Model," MIT Laboratory for Computer Science, Technical Report, 2002.

[24] B. Bodin, Y. Lesparre, J.-M. Delosme, and A. Munier-Kordon, "Fast and efficient dataflow graph generation," in *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*, 2014, pp. 40–49.

[25] H. Oh and S. Ha, "Fractional Rate Dataflow Model for Efficient Code Synthesis," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 37, pp. 41–51, 2004.

[26] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Buffer capacity computation for throughput-constrained modal task graphs," *ACM Transactions on Embedded Computing Systems*, vol. 10, pp. 1–59, 2010.

[27] X. Khanh Do, S. Louise, and A. Cohen, "Transaction Parameterized Dataflow: A Model for Context-Dependent Streaming Applications," in *Design, Automation & Test in Europe Conference & Exhibition*, 2016, pp. 960–965.

[28] P. Dubrulle, C. Gaston, N. Kosmatov, A. Lapitre, and S. Louise, "A Data Flow Model with Frequency Arithmetic," in *Fundamental Approaches to Software Engineering*, vol. 11424, 2019, pp. 369–385.

[29] P. Dubrulle, C. Gaston, N. Kosmatov, and A. Lapitre, "Dynamic Reconfigurations in Frequency Constrained Data Flow," in *Integrated Formal Methods*, vol. 11918, 2019, pp. 175–193.

[30] J. Buck and E. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," in *IEEE International Conference on Acoustics Speech and Signal Processing*, 1993, pp. 429–432 vol.1.

[31] J. Buck, "Static scheduling and code generation from dynamic dataflow graphs with integer-valued control streams," in *Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers*, vol. 1, 1994, pp. 508–513.

[32] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur, "BPDF: A statically analyzable dataflow model with integer and boolean parameters," in *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2013, pp. 1–10.

[33] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *IEEE International Conference on Formal and Models for Co-Design*, 2006, pp. 185–194.

[34] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2011, pp. 404–411.

[35] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth, "Parameterized Dataflow Scenarios," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, pp. 669–682, 2017.

[36] W. Plishker, N. Sane, M. Kiemb, and S. Bhattacharyya, "Heterogeneous Design in Functional DIF," in *International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation*, vol. 5114, 2008, pp. 157–166.

[37] S. Lin, L.-H. Wang, A. Vosoughi, J. R. Cavallaro, M. Juntti, J. Boutellier, O. Silvén, M. Valkama, and S. S. Bhattacharyya, "Parameterized Sets of Dataflow Modes And Their Application to Implementation of Cognitive Radio Systems," *Journal of Signal Processing Systems*, vol. 80, pp. 3–18, 2015.

[38] L.-H. Wang, C.-C. Shen, and S. S. Bhattacharyya, "Parameterized core functional dataflow graphs and their application to design and implementation of wireless communication systems," in *SiPS 2013 Proceedings*, 2013, pp. 1–6.

[39] K. Sudusinghe, S. Won, M. van der Schaar, and S. Bhattacharyya, "A novel framework for design and implementation of adaptive stream mining systems," in *2013 IEEE International Conference on Multimedia and Expo (ICME)*, 2013, pp. 1–6.

[40] Chanik Park, Jaewoong Chung, and Soonhoi Ha, "Extended synchronous dataflow for efficient DSP system prototyping," in *Proceedings Tenth IEEE International Workshop on Rapid System Prototyping*, 1999, pp. 196–201.

[41] A. Girault, Bilung Lee, and E. Lee, "Hierarchical finite state machines with multiple concurrency models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 742–760, 1999.

[42] M. H. Wiggers, M. J. Bekooij, and G. J. Smit, "Buffer Capacity Computation for Throughput Constrained Streaming Applications with Data-Dependent Inter-Task Communication," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008, pp. 183–194.

[43] P. O. Meredith, D. Jin, D. Griffith, F. Chen, and G. Rosu, "An overview of the MOP runtime verification framework," *International Journal on Software Tools for Technology Transfer*, vol. 14, pp. 249–289, 2012.

[44] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Information Processing, Proceedings of the 6th IFIP Congress 1974*. North-Holland, 1974, pp. 471–475.

[45] M. Geilen and T. Basten, "Reactive process networks," in *Proceedings of the Fourth ACM International Conference on Embedded Software - EMSOFT '04*, 2004, p. 137.

[46] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, pp. 2408–2421, 2001.

[47] Mainak Sen, S. Bhattacharyya, T. Lv, and W. Wolf, "Modeling Image Processing Systems with Homogeneous Parameterized Dataflow Graphs," in *International Conference on Acoustics, Speech, and Signal Processing*, 2005, pp. 133–136.

[48] K. Desnos, M. Pelcat, J.-F. Nezan, S. Bhattacharyya, and S. Aridhi, "PiMM: Parameterized and Interfaced dataflow Meta-Model for MP-SoCs runtime reconfiguration," in *Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2013, pp. 41–48.

[49] P. Fradet, A. Girault, R. Krishnaswamy, X. Nicollin, and A. Shafiei, "RDF: Reconfigurable Dataflow," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1709–1714.

[50] F. Arrestier, K. Desnos, M. Pelcat, J. Heulot, E. Juarez, and D. Menard, "Delays and states in dataflow models of computation," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2018, pp. 47–54.