A Reactive and Adaptive Data Flow Model For Network-of-System Specification

Sabine Francis and Andreas Gerstlauer
The University of Texas at Austin, Austin, TX, USA
francisabine@utexas.edu, gerstl@ece.utexas.edu

***Abstract*—With embedded systems being increasingly networked, appropriate models of computation and communication are needed for specification of such networks-of-systems. Traditional dataflow models have shown their usefulness in analyzing isolated systems. However, these models cannot express the inherent requirements of connected applications, such as dynamic behavior associated with network losses and reactivity to external events. This paper proposes a Reactive and Adaptive Data Flow (RADF) model that introduces a notion of empty tokens to expose network losses and provide adaptivity at the application level while maintaining overall determinism. Empty tokens, combined with expanded actor semantics, are also used to model reactivity to sporadic external events. We formally define RADF semantics and show efficient methods for analyzing RADF graphs in terms of their worst-case throughput and latency.**

***Index Terms*—Adaptivity, connected embedded systems, data flow modeling, network losses, reactivity.**

I. Introduction

Embedded and cyber-physical systems are increasingly networked and driven by data-dominated, computationally intensive applications. Wireless sensor networks (WSNs) and the Internet of Things (IoT) performing signal and data processing tasks are examples of Networks-of-Systems (NoS) characterized by their connected and streaming behavior.

In traditional embedded system design processes, Models of Computation (MoCs) are used for formal specification of isolated system behavior. Such MoCs focus on expressing computational aspects, including tradeoffs with analyzability related to properties such as concurrency, determinism and deadlocks. However, they typically lack support for richer communication semantics required to model NoS. Traditional embedded MoCs overconstrain the implementation of the system by assuming and guaranteeing a lossless communication between processes. In reality, network communication is inherently lossy. System implementations should be exposed and allowed to adapt to losses in order to explore tradeoffs between, among others, Quality of Service (QoS) and latency. Existing communication models, such as queuing theories, support analyses of network effects. However, they in turn do not account for expressing system computation. Novel, unified Models of Computation and Communication (MoCCs) are thus needed to simultaneously capture both aspects.

Streaming application behavior in NoS is naturally captured by dataflow MoCs and supported by effective analysis and synthesis techniques. However, applications are also becoming more dynamic as sporadic external events interact with streaming components. Typical reactive MoCs, such as Synchronous Reactive (SR) models, allow capturing event-driven behavior in control-oriented form, but do not provide an appropriate abstraction for data-driven streaming computations. By contrast, basic dataflow models can not easily express reactive behavior as they rely on a predefined ordering of events to maintain determinism. A unified MoCC should thus support both reactive and streaming behavior.

This paper proposes a new MoCC called Reactive and Adaptive Data Flow (RADF). RADF is based on an extension of dataflow MoCs. We demonstrate our extensions on top of a Synchronous Data Flow (SDF) basis; they can, however, be equally applied to other dataflow variants. RADF captures several dynamic aspects, by introducing notions of adaptivity and reactivity. Adaptivity refers to the ability of an application or its implementation to alter its behavior depending on exposed network losses. Reactivity refers to the ability of a system to adjust its behavior in reaction to external sporadic events. RADF supports a network-like communication model; traditional lossless queues between actors are replaced by lossy ones. In addition, RADF incorporates a concept of empty tokens, which allows modeling lost data, while preserving determinism and analyzability. This permits for dynamic adaptivity to be modeled, without changing the semantics of the underlying model. Empty tokens also allow modeling unavailability of data. When combined with extended actor semantics, reactivity to sporadic external events can be expressed.

The rest of the paper is organized as follows. Section II discusses the related work. Section III presents the formal definition of the RADF model. RADF performance analysis is discussed in Section IV. Finally, Section V concludes the paper with a summary and outlook on future work.

II. Related Work

The main differences between RADF and other MoCs are the modeling of dynamic behavior through a concept of empty tokens and support for event-driven executions.

Generally, streaming specifications are modeled by process network or dataflow MoCs. Kahn Process Networks (KPNs) support reactivity at the expense of undecidable model properties. Boolean Data Flow (BDF) [1] and the CAL Actor Language (CAL) [2] are also Turing complete with similar restrictions in analyzability. By contrast, SDF allows to statically analyze the graph but, due to its fixed consumption and production rates, does not support any form of dynamic behavior. The Cyclo-Static Data Flow (CSDF) model [3] allows for cyclical rate changes, but is still static in its expressiveness.

Many other generalizations of SDF support dynamic behavior, by using varying rates and switching between scenarios or modes of operation. Parameterized and modal dataflow

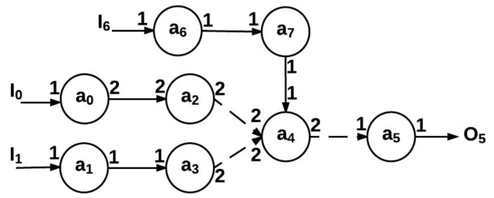

Fig. 1: RADF graph of wireless sensing example.

models, such as Heterochronous Dataflow (HDF) [4], Parameterized Synchronous Data Flow (PSDF) [5], Scenario-Aware Data Flow (SADF) [6] and their variants [7], [8], combine dataflow models with state machines to dynamically transition between different configurations. Similarly, Reactive Process Networks (RPN) extend KPNs to switch between streaming modes at the occurrence of a particular event [9]. Nevertheless, to ensure analyzability, all the previously listed models allow rates to change only between two different iterations of the graph. This puts a limitation on the dynamic behavior, in terms of adapting execution to unpredictable changes, such as token losses in the middle of an iteration. Furthermore, as we will show later, modeling token losses as mode switches results in state explosion as the number of combinations of losses is exponential. Given the semantics of empty tokens, these restrictions are relaxed in RADF. Boolean Parametric Data Flow (BPDF) [10], Variable-Rate Data Flow (VRDF) [11] and Schedulable Parametric Data-Flow (SPDF) [12] allow changes in boolean parameters or actor rates within an iteration. However, BPDF and SPDF are limited to boolean or non-zero choices in rate adaptivity, respectively, while VRDF and SPDF both require a separate analysis to guarantee global consistency. By contrast, RADF supports more expressive adaptivity patterns, where empty tokens separate adaptivity from rate consistency of the underlying dataflow model.

The StreamIt programming language is designed to facilitate the coding of streaming applications [13]. It expands on the SDF model by supporting dynamic rates and synchronized controlled messages, also known as teleport messages. The latter are similar to empty tokens as they are not always received. However, they are out-of-band messages, which are not part of the in-band streams. In addition, they are a property of the actors. In our model, empty tokens are a property of the regular, in-band communication channels.

III. Formal Definition

In this section, we formally define the RADF model. We briefly summarize the semantics of the SDF model used in this paper as the basis for RADF. In the following, we adopt the original dataflow firing rule notation by Dennis [14].

An SDF graph is a network of actors communicating over unbounded FIFO channels. The system is thus defined by a tuple $\langle \mathcal{A}, \mathcal{C} \rangle$, where $\mathcal{A}$ and $\mathcal{C}$ represent, respectively, the sets of actors and channels. In addition, $\mathcal{I}_p$ and $\mathcal{O}_p$ denote the finite sets of input and output data ports of an actor $a_p \in \mathcal{A}$. In SDF, actors consume and produce a fixed number of tokens per firing and have a single firing rule, $\mathbf{R_p}$, that specifies when a_p fires. $\mathbf{R_p} = \{R_{p,1}, ..., R_{p,N}\}$ is formed of N finite token patterns, one for each of the N input ports of a_p. Each pattern $R_{p,n}$ must only contain $*$ representing token wildcards. a_p fires when each $R_{p,n}$ forms a prefix of the sequence of unconsumed tokens at its input port $i_{p,n} \in \mathcal{I}_p$. For example, assume an actor a_j with two input ports, $i_{j,0}$ and $i_{j,1}$. At each of its firings, a_j consumes two tokens from $i_{j,0}$ and one token from $i_{j,1}$. The firing rule of a_j is thus $\mathbf{R_j} = \{[*, *], [*]\}$. a_j executes when at least two tokens are available at $i_{j,0}$ and one token is available at $i_{j,1}$. We denote by $r_{p,m}$ the firing or production rate of a_p associated to its output port $o_{p,m} \in \mathcal{O}_p$. For example, at each firing of a_j, a_j produces two tokens on its output port $o_{j,0}$; $r_{j,0}$ is thus 2. Channels connect input and output ports. A channel $c_{p,q} \in \mathcal{C}$ is characterized by the tuple $\langle o_{p,m}, i_{q,n}, t_{p,q} \rangle$, where $o_{p,m} \in \mathcal{O}_p$ and $i_{q,n} \in \mathcal{I}_q$ correspond to an output port and input port, respectively, and $t_{p,q}$ to the number of initial tokens that may exist on the channel. The state of an SDF graph is the number of tokens on each channel. Its initial state is thus determined by the number of initial tokens on each of its channels. An iteration is defined as the minimal sequence of actor firings that brings the graph back to its initial state [12].

An RADF graph is based on the same topology of actors, channels and tokens. What changes are the semantics of the actors and channels. Fig.1 shows an example of a wireless sensing application that will be used to illustrate our extensions. I_0 and I_1 are sensor inputs preprocessed by actors a_0 and a_1, which in turn send their outputs to feature extraction actors a_2 and a_3. Actors a_6 and a_7 process an input event that is triggered, e.g., once a button is pressed. Results are agglomerated in a gateway a_4 and sent to a cloud server a_5.

A. Adaptivity

Adaptivity is attained by exposing network losses to the application level. The communication channels, connecting the actors in RADF, have the same underlying semantics as the queues of SDF models. However, and to model network losses, they are by default allowed to be lossy (represented graphically by dashed arrows). Optionally, channels can be specified to be lossless (represented by solid arrows). Channels expose losses to the application by replacing lost data by empty tokens.

Definition 1 (**Empty Tokens**). *Empty tokens, represented in token patterns by the symbol $\oslash$, are tokens that do not carry any useful data.*

Definition 2 (**Lossless Channels**). *A channel $c_{p,q}$ is lossless if it maps the output token pattern $[*, *, ..., *]$ on $o_{p,m}$ to an identical input token pattern on $i_{q,n}$.*

Definition 3 (**Lossy Channels**). *$c_{p,q}$ is lossy if it maps the output token pattern $[*, *, ..., *]$ on $o_{p,m}$, to an input token pattern on $i_{q,n}$, that can contain one or more empty tokens.*

It further follows that users may specify different execution versions for each actor, depending on the combination of empty and nonempty tokens present on their input ports. A default version executes the actor when no other version is defined for a given input token pattern. In addition, a special idle version allows an actor to not execute when all its input tokens are empty.

Definition 4 (**RADF Actor**). *RADF actors are described using multiple variants. Let $a_p \in \mathcal{A}$ be modeled by K*

distinct versions, such that a_p^k corresponds to the k^{th} version associated with a specific firing rule $\mathbf{R_p^k}$. Given SDF as the basis, consumption and production rates are the same across patterns. Tokens belonging to prefix patterns take any of the values $\{\oslash, \mathrm{x}, *\}$, where $\oslash$ represents an empty token, x a non-empty token and $*$ a token wildcard that covers any empty or non-empty token. Actors can have the following variants:

- *k^{th} version, a_p^k: a_p^k is executed iff the token patterns on its inputs match the associated firing rule $\mathbf{R_p^k}$.*
- *Default version, a_p^0: a_p^0 is mandatory and executed for possible input patterns not matched by any other a_p^k.*

Actors executing the aforementioned versions cannot produce any empty tokens. In addition, an actor can optionally have a special variant:

- *Idle version, $a_p^{\oslash}$: $a_p^{\oslash}$ is triggered when the input sequences on all inputs are formed of only empty tokens. However, this variant is restricted to not execute any code. Furthermore, its outputs are only formed of an all-empty token pattern.*

The concept of empty tokens coupled with fixed rates for actors guarantees determinism while allowing for dynamic behavior of RADF graphs. By contrast, and as was stated earlier, existing modal models, like SADF, put limitations on the dynamic behavior. Taking Fig.1 as an example, we assume that a_5 implements the $a_5^{\oslash}$ variant. If $c_{4,5}$ drops the token in the second firing of a_5, it follows from Definition 4 that a_5 does not execute any code. In SADF, this corresponds to a scenario where a_5 is omitted from the graph. However, SADF restricts this scenario change to not occur during the same iteration of the graph. SADF would therefore require token losses and hence the scenario to execute to be known a priori, at the start of the iteration. In contrast, in RADF, various variants can be triggered whenever an actor fires, possibly with different variants within the same iteration.

B. Reactivity

Reactivity in RADF is attained through support for presence of empty or non-empty tokens in external inputs and subsequent execution of idle or non-idle actor variants. Empty tokens model the absence of sporadic events in input patterns, allowing an RADF graph to vary its behavior depending on the occurrence of such events. To illustrate reactive execution, we assume that a_4, a_6 and a_7 of Fig.1 implement a variant a_4^1 having a firing rule $\mathbf{R_4^1} = \{[\mathrm{x},\mathrm{x}], [\mathrm{x},\mathrm{x}], [\oslash]\}$, and idle versions $a_6^{\oslash}$ and $a_7^{\oslash}$, respectively. During a given iteration of the graph, the button is not pressed: a_6 does not receive any event, i.e. receives an empty token. Following Definition 4, a_6 fires its idle version, which does not execute any code, and empty tokens are output on $c_{6,7}$. Subsequently, a_7 fires its idle variant. When a_4 receives non-empty tokens on its input ports connected to $c_{2,4}$ and $c_{3,4}$, it fires its non-idle variant a_4^1 as empty tokens are injected on $c_{7,4}$. a_6 and a_7 are said to form a reactive island. The system implementation can be optimized to not execute idle firings of such reactive chains.

Definition 5 (**Reactive Island**). *A reactive island is a largest possible region in the graph formed of connected actors, where firing idle variants of the source actors of the island triggers all subsequent actors in the island to fire their idle variants.*

C. Reactive and Adaptive Data Flow

Definition 6 defines the complete RADF model.

Definition 6 (**RADF**). *An RADF model is formed by the tuple $\langle \mathcal{A}, \mathcal{C}, \mathcal{C}_L \rangle$, where $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{C}_L$ represent, respectively, the sets of actors, lossless channels and lossy channels.*

1) *The execution semantics of the actors in $\mathcal{A}$ are defined in Definition 4.*
2) *$\mathcal{C}$ and $\mathcal{C}_L$ are defined, respectively, in Definition 2 and Definition 3. Initial tokens, which might be empty, may be present on the channels to avoid deadlocks.*

IV. Performance Analysis

The performance of RADF graphs is studied in terms of their throughput and latency metrics.

A. Throughput

For analysis purposes, we can assume that at the beginning of each iteration, token losses are known. Under this assumption, the different scenarios can be explicitly enumerated, and the RADF graph can be converted into an equivalent SADF model with a fully connected FSM. SADF throughput analysis techniques introduced in [15] can then be applied. These techniques are at least linear in the number of scenarios. However, the number of SADF scenarios resulting from the RADF to SADF conversion is exponential. An RADF-SADF conversion is possible by traversing the RADF graph in a breadth-first fashion and iteratively constructing SDF graphs forming the different SADF scenarios. For every RADF node visited, existing partial SDF subgraphs are replicated and each of them extended with one of the different RADF actor variants. We use Fig.1 to illustrate the conversion, where we assume that a_4 implements 32 variants, corresponding to its 32 possible input token patterns, and a_5 implements two variants. In addition, all other actors possess a single variant as their input ports are connected to lossless channels. Thus, converting the RADF graph of Fig.1 into an SADF model results in 64 distinct scenarios. This shows how modal models in general grow exponentially in the number of modes as the number of variants per lossy link increases. Thus, even under the assumption of token losses being known a priori, RADF expresses behavior that would require an exponential number of configurations. In the worst case, we assume that all channels of an RADF graph are lossy. Every actor a_p, with $r_{p,m}$ as its firing rate, has $2^{\sum_{m=0}^{M} r_{p,m}}$ possible patterns. The total number of modes or scenarios thus equals $\prod_{p=0}^{P} 2^{\sum_{m=0}^{M} r_{p,m}}$.

Instead, RADF supports an alternative approach for computing the throughput that applies the simpler SDF analysis techniques introduced in [16] directly on the graph formed of the RADF actor variants taking the worst-case execution time (WCET). For SADF, this approach over- or underestimates results [15]. Obtaining SADF throughput from the SDF graph formed of the WCET actors across all scenarios can be too pessimistic because the resulting SDF graph might not represent a feasible scenario or a feasible scenario sequence. The former also implies that simply analyzing the worst-case scenario in an SADF graph might be too optimistic. This can occur in pipelined executions where iterations can overlap in such a way that accumulating individual actor WCET is instead triggered by a sequence of different scenarios. For worst-case analysis of RADF, scenarios with infeasible or independent non-WCET actor variants can be excluded. This significantly reduces complexity compared to the full expansion above. However, lossless channels can create dependencies between actor variants that are not captured by the single WCET combination. We can assume that in the worst case all channels are lossy. Under this assumption, the SDF graph composed of WCET variants always represents a possible scenario and a possible execution sequence of the RADF graph. This approach is simpler, but does not always provide tight bounds.

Definition 7 (Feasible Variant). *We denote by feasible variant an actor variant that can execute in a given graph instance.*

Definition 8 (Actor WCET). *We denote by the WCET of an actor its WCET among all its feasible variants.*

Proposition 1. *Assuming lossy channels, the combination of WCET actor variants results in a possible execution sequence of an RADF graph.*

Proof: Let $c_{p,q}$ connect the output and input ports of a_p to a_q, respectively. Even when part of a reactive island, actors a_p and a_q will have at least one feasible non-idle variant triggered by non-empty tokens received from primary inputs or non-idle actors outside the island. Since feasible idle variants do not execute any code, it further follows from Definition 4 that the WCET variant of a_p will be a non-idle version that always outputs non-empty tokens. Assuming that $c_{p,q}$ is lossy, a_q can execute any of its feasible versions, including its WCET variant. Given that the input patterns characterizing these variants can only be realized through the loss of zero or more tokens on $c_{p,q}$, the combination of variants of a_p and a_q is a feasible execution sequence. ■

Proposition 2. *A worst-case bound on throughput of an RADF graph is calculated as the throughput of the SDF graph formed by taking the WCET of each actor.*

Proof: From Proposition 1, it follows that the SDF graph composed of the actors' WCET variants is a possible execution sequence of the RADF graph assuming lossy channels. Furthermore, lossless channels can not increase the execution time of actors beyond their WCET. Given that the throughput analysis searches for the longest cycle, analyzing the WCET SDF gives thus a lower bound on the throughput. ■

B. Latency

Latency is defined as the difference in the firing times between source and sink nodes of the dataflow graph. Similarly to the throughput analysis, one approach converts the RADF model into its equivalent SADF model, and finds the latency per the methods proposed in [17]. However, and following the same reasoning as for throughput calculation, a simpler approach finds an upper bound on the latency by analyzing the latency of the WCET SDF. This analysis is based on the $(max, +)$ characterization of the resulting SDF graph [18].

V. Summary and Conclusions

In this paper, we presented a Reactive and Adaptive Data Flow (RADF) MoCC designed to model embedded and cyber-physical networks-of-systems. RADF extends traditional dataflow models by incorporating the notion of empty tokens to represent network losses and the absence of sporadic events, and by extending actor semantics to define different variants. These properties make RADF more expressive than traditional dataflow models and easier to analyze than existing modal models. We showed that analyzing the throughput and the latency of the RADF graph, formed of the actors having the worst-case execution times over all variants, provides RADF performance metrics with low complexity. Future work involves expansion of the semantics of the model and their concrete realizations, which, similar to existing system design methods, will require synthesis approaches, where multiple implementation choices can potentially be supported. We also plan to investigate analysis techniques to support probabilistic performance treatment, including tradeoffs in terms of latency, throughput and quality of service.

VI. Acknowledgements

This research is supported by NSF grant CSR-1421642. We also thank the anonymous reviewers for their valuable feedback in improving this paper.

References

[1] J. Buck and E. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," in *ICASSP*, 1993.
[2] J. Eker and J. Janneck, "CAL language report," University of California at Berkeley, Tech. Rep., Dec 2003.
[3] G. Bilsen, et al., "Static scheduling of multi-rate and cyclo-static DSP-applications," in *VLSI Signal Processing VII*, 1994.
[4] A. Girault, et al., "Hierarchical finite state machines with multiple concurrency models," *IEEE TCAD*, vol. 18, no. 6, Jun 1999.
[5] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE TSP*, vol. 49, no. 10, Oct 2001.
[6] B. Theelen, et al., "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *MEMOCODE*, 2007.
[7] M. Skelin, et al., "Parametrized dataflow scenarios," in *EMSOFT*, 2015.
[8] K. Desnos, et al., "PiMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration," in *SAMOS*, 2013.
[9] M. Geilen and T. Basten, "Reactive process networks," in *EMSOFT*, 2004.
[10] V. Bebelis, et al., "BPDF: A statically analyzable dataflow model with integer and boolean parameters," in *EMSOFT*, 2013.
[11] M. Wiggers, et al., "Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication," in *RTAS*, 2008.
[12] P. Fradet, et al., "SPDF: A schedulable parametric data-flow MoC," in *DATE*, 2012.
[13] W. Thies, et al., "Teleport messaging for distributed stream programs," in *PPoPP*, 2005.
[14] E. Lee and E. Matsikoudis, *The Semantics of Dataflow with Firing*. Cambridge University Press, 2007.
[15] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES+ISSS*, 2010.
[16] A. Ghamarian, et al., "Throughput analysis of synchronous data flow graphs," in *ACSD*, 2006.
[17] F. Siyoum, et al., "End-to-end latency analysis of dataflow scenarios mapped onto shared heterogeneous resources," *IEEE TCAD*, vol. 35, no. 4, April 2016.
[18] M. Skelin, et al., "Worst-case latency analysis of SDF-based parametrized dataflow MoCs," in *DASIP*, 2015.