Playing Games with Scenario- and Resource-Aware SDF Graphs Through Policy Iteration

Yang Yang[1], Marc Geilen[1], Twan Basten[1,2], Sander Stuijk[1], Henk Corporaal[1]
[1]Department of Electrical Engineering, Eindhoven University of Technology, Netherlands
[2]Embedded Systems Institute, Eindhoven, Netherlands
{y.yang, m.c.w.geilen, a.a.basten, s.stuijk, h.corporaal}@tue.nl

***Abstract*—The two-player mean-payoff game is a well-known game theoretic model that is widely used, for instance in economics and control theory. For controller synthesis, a controller is modeled as a player while the environment, or plant, is modeled as the opponent player (adversary). Synthesizing an optimal controller that satisfies a given criterion corresponds to finding a winning strategy for the controller player. Emerging streaming applications (audio, video, communication, etc.) for embedded systems exhibit both input sensitive and controller sensitive runtime behavior, where the controller's role is runtime management or scheduling. Embedded controllers need to be optimized for dynamic inputs, while guaranteeing throughput constraints. In this paper, we consider this design task for scenario- and resource-aware dataflow graphs that model streaming applications. Scenarios in these models capture classes of dynamic environment behavior. We demonstrate how to model and solve the controller synthesis problem by constructing a winning strategy in a two-player mean payoff throughput game.**

***Index Terms*—Synchronous DataFlow, Maxplus Algebra, Game Theory, Policy Iteration**

I. Introduction

Emerging streaming applications have to adapt to environmental changes for implementation efficiency. Environmental awareness enables systems to achieve higher performance, lower resource usage as compared to implementations without such environmental awareness. Examples include traffic modeling [1] and rate control [2] for MPEG applications, bandwidth sharing [3] in wireless sensor networks, cognitive radio [4] for next generation communication networks, etc. Many streaming applications show data-dependent behavior, i.e., their execution times and resource usage are highly dependent on the properties of the input. At the same time, advances in computer engineering allow both software and hardware to adapt their own behaviors at runtime in response to changes in the environment. For example, software can change its scheduling policy and grant resources to tasks in different orders while hardware can adapt its processors' voltages and frequencies, i.e., so-called Dynamic Voltage and Frequency Scaling (DVFS) [5], [6]. Progress in programmable hardware allows hardware implementation changes at runtime with some overhead. For example, by downloading different bit streams at runtime, reconfigurable computing platforms can dynamically configure their FPGAs with different functionalities optimized for their corresponding input data [7].

To summarize, an adaptive system has the following two features: detecting changes in the environment (**cognitive ability**) and adapting its software/hardware accordingly in time for efficiency (**reconfigurability**). As a result, the design of an adaptive system raises the following two questions: "How to **model** the environment changes?" and "How to reconfigure software/hardware to **guarantee** design objectives in response to the given environment change?"

To answer the first question, we can use the fact that the properties of input data can be known beforehand and that these properties are in many cases embedded into the input data as metadata. Hence, we can capture the environment changes with Markov Chains [8] or finite state machines (FSM) [9]. For example, in the MPEG-2 standard, the input image frames of a decoder can be identified as I frames, P frames and B frames respectively. For raw input without embedded information that helps identification, we can use techniques such as machine learning to classify them. Classification and detection of different input data is studied extensively in literature. In this paper, we mainly focus on the second question and set a throughput target as our design objective.

This paper makes the following contributions:

- We introduce SARA-SDF: an extension to the synchronous dataflow (SDF) [10] model to capture streaming applications with resource sharing in dynamic environments.
- We model the interaction between the streaming application system and its environment as a formal, mean-payoff game in which a good controller corresponds to a player with a winning strategy.
- Based on this model, we solve both the controller synthesis problem under resource and throughput constraints as well as the worst-case environment identification.

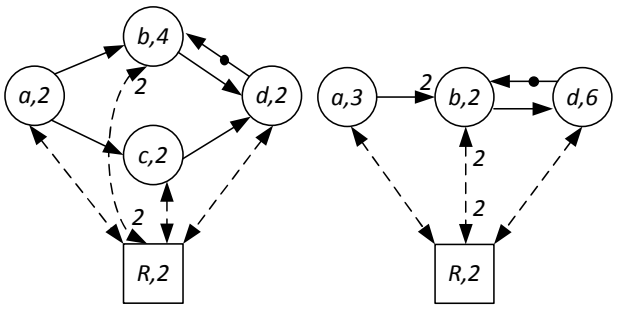
To give some intuition for the question that we are trying to answer, it is useful to walk through an example. In Sec. II, we use a running example to state the problem. The theoretical background is given in Sec. III. In Sec. IV, the game analogy and analysis are illustrated. In Sec. V experiments are reported. In Sec. VI related work is discussed and Sec. VII concludes.

II. Problem Statement

Assume an application with four actors a, b, c, d and data dependencies among those actors. The parameters of actors,

This work has been carried out as part of the Octopus project with Océ Technologies B.V. under the responsibility of the Embedded Systems Institute. This project was partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

978-3-9810801-8-6/DATE12/©2012 EDAA



(a) Input A (b) Input B

Figure 1. RA-SDF models of an application for input types A and B

i.e., execution times and data/resource rates, are depend on the type of application input and are represented as scenarios: A, B. The four actors use the same resource R with b using two units at a time. Fig. 1 shows the RA-SDF [11], [12], [13] models for two possible types of inputs, while the execution time of the actors, the amount of resources and the data/resource rates are annotated with the models (rates 1 are omitted for clarity).

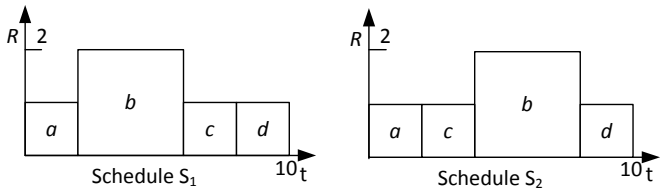


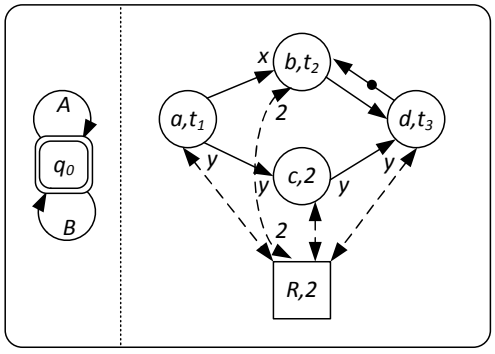
Figure 2. Different schedules of the example application under input A

It is not very hard to see that, in Fig. 1(a), actor b and c cannot be executed in parallel since the resource constraint of R does not allow this to happen. So the schedule of the RA-SDF model under input A (i.e., the execution order of actors b and c) can be reconfigured based on the context (the two options are shown in Fig. 2). In this example, reconfiguration means the manipulation of the execution order of conflicting actors, i.e., the scheduling. In general, a system can reconfigure many properties to adapt its system behavior. Besides adapting its schedule to the environment, it may for example change the frequency and voltage, or resource allocation. We assume that the changes of behaviors after reconfiguration can be modeled as execution time and data/resource rate changes in RA-SDF. We call the entity that chooses the behavior the ***controller***. For simplicity, we model the behavior of RA-SDF under different types together as a Scenario- and Resource-Aware dataflow (SARA-SDF) graph with parameterized execution times and data/resource rates. Fig. 3 shows the SARA-SDF of the example application.

For the SARA-SDF example, the question that we have to answer contains the following three sub questions.

- *What is the highest throughput that we can obtain no matter what input sequence is encountered?*
- *What is the best strategy for the controller to reconfigure the system to obtain the highest throughput?*
- *What is the worst input sequence of the environment that we can have no matter what policy we use to configure the system?*

In this paper, we approach the answers to these questions from a **game theoretic** viewpoint. Fig. 4 illustrates this



	x	y	t_1	t_2	t_3	actor c
A	1	1	2	4	2	enabled
B	2	0	3	2	6	disabled

SCENARIO PARAMETERS TABLE

Figure 3. SARA-SDF of the example application

approach. The problem is viewed as a game played by two players: environment and controller. The environment player decides on the input sequence that feeds into the system while the controller player decides the schedule for every input. The worst-case situation refers to when the input sequence leads to the lowest throughput no matter how the controller reacts to it. The goal of controller player is to maximize the throughput while the 'goal' of the environment player is to minimize it (for worst-case analysis). In the physical view, a designer has to design an embedded system that processes a sequence of different types of input data. The performance requirement gives a throughput constraint that must be satisfied. Since the concrete sequence of input data is not predictable at design time, a designer has to design a controller that, for reasons of efficiency, can control or reconfigure the system parameters such as scheduling or resource allocations based on the type of input data encountered. In the model view, the system

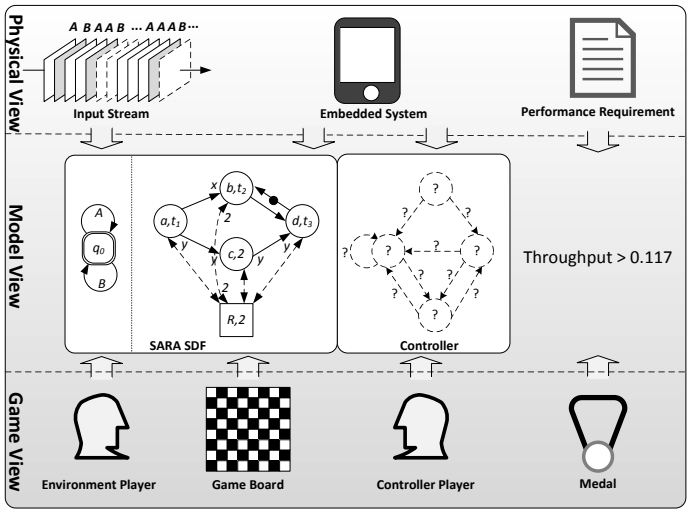


Figure 4. Game theoretic view interpretation of embedded system design

itself is specified as an operational model (a parameterized RA-SDF) and the types of data (scenarios) that it supports, i.e., as a SARA-SDF. The goal of the designer is to synthesize a controller that reconfigures the system based on the

encountered input, while satisfying the throughput constraint. The possible input sequences are captured by the scenario finite state machine in the SARA-SDF. The controller is also modeled as an, initially unknown, to be determined, FSM that gives schedules or configurations based on the current input and history of inputs. Intuitively, we can interpret the interaction between two FSMs, the scenario FSM and the controller FSM as a game played between the environment player and the controller player. We call this the **game view**, and use knowledge of game theory to quantitatively analyze the SARA-SDF and synthesize a controller that is guaranteed to meet the throughput constraint.

To summarize, the problem is to find a winning strategy for the controller player (a controller FSM) to satisfy the throughput constraint of the embedded system no matter what its environment player (a scenario FSM) does, i.e., no matter what sequence of input data types are encountered.

III. Scenario- and Resource-Aware SDF

We combine the *input*, *application*, *architecture* and *mapping* aspects into a single system specification model that we refer to as *Scenario- and Resource-Aware SDF* (SARA-SDF). A SARA-SDF is a tuple $(\mathcal{G}_{RA}, FSM_S)$ that can be viewed as a *specification* of an embedded system with *two* parts:

- a parameterized *Resource-Aware SDF*
- a *scenario finite state machine*

The parameters of the RA-SDF, i.e., execution times, data/resource rates, active status of each actor, are dependent on the *scenario*s of application. For an input with type T, its corresponding scenario is denoted by S_T. For each scenario, there is a corresponding RA-SDF. Finite state machines are used to specify the possible sequences of scenarios. Fig. 3 shows the example of a SARA-SDF. Non-active actors of each scenario are marked as disabled in the table. Like its relatives (SDF [10], SADF [9], RA-SDF [12]), a SARA-SDF requires *consistency* and has *repetition vectors*. A SARA-SDF is *consistent* if and only if, for every scenario S_T there exists a non-trivial repetition vector r_T; this r_T assigns a non-zero number of firings to every *active* actor in the scenario S_T, such that, after any sequence of actor firings conforming to r_T, an *iteration* labeled with S_T, both the number of data tokens in the channels and the number of resource tokens are equal to their initial numbers. The repetition vectors of the example of Fig. 3 are $r_A = [1,1,1,1]$, $r_B = [2,1,0,1]$ for scenarios S_A, S_B respectively. The vectors are in alphabetical order of actors. Fig. 5 shows an execution for scenario sequence $S_B S_A S_A S_B \cdots$, in which x axis is for time, y axis is for resource tokens, T_i denotes the completion time of the ith iteration and L_i denotes the *latency* between two consecutive iteration completion times.

We use a *state-based* execution model to capture the execution of a SARA-SDFG. Every channel or resource token has a time-stamp which represents the time instant it was produced or released. The time-stamps of all tokens at the end of an *iteration* are collectively captured by a time-stamp vector representing the state. The time-stamp vector for each iteration in the example execution in Fig. 5 is denoted by circles annotated with their iteration numbers. For example, the circles annotated with number 1 denote the time-stamp vector after the first iteration, i.e., $\gamma_1 = [5, 11]$. This gives us time-stamp vectors γ (similar to dater functions in timed Petri-nets and the time-stamp vectors of SADF [9], [14].) Together with the scenario FSM state q, they are used as the iteration state of the SARA-SDF. An iteration state is thus denoted by a pair $\{\gamma, q\}$. The time stamps of all tokens in the initial state are zero.

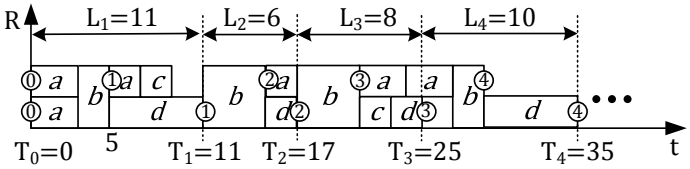


Figure 5. Iteration-based execution

Before every new iteration of a SARA-SDF, one of the transition edges in the scenario FSM is selected and its corresponding scenario parameters are used to instantiate the RA-SDF model. We know the repetition vector of the given scenario. The scheduling of actor firings in the iteration follows the rules given in the RA-SDF. For a given execution according to a specific scenario sequence, we use $\gamma_i \in \mathbb{R}^n$ to denote the time-stamp vector after the ith iteration. We use $\mathcal{M}_i : \mathbb{R}^n \mapsto \mathbb{R}^n$ to denote the schedule applied by the controller during the ith iteration by means of the effect it has on the state, in the form of an operator $\mathcal{M}_i$ such that $\gamma_{i+1} = \mathcal{M}_i(\gamma_i)$.

From initial state $\{\gamma_0, q_0\}$ of a SARA-SDF, we have to anticipate every scenario that q_0 accepts (for which it has a transition) and explore different schedules inside one iteration for each scenario. This generates different new iteration states. By this process we can construct the *iteration state space* of a given SARA-SDF.

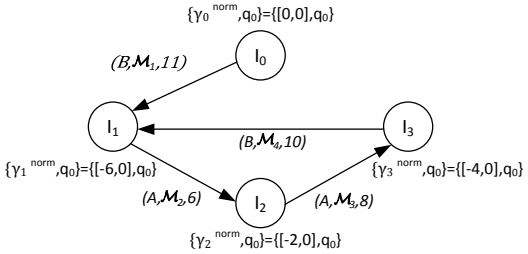


Figure 6. Part of the state space for the example execution

Fig. 6 shows part of the iteration state space corresponding to the example execution in Fig. 5. $\mathcal{M}_i$ is the schedule of the ith iteration as decided by the controller reacting to the ith input observed from the environment. We use the following notation and definitions. $\max(\gamma_i)$ denotes the maximal time stamp in γ_i, which captures the completion time of the schedule for the ith input. The *latency* of the ith iteration is the completion time difference between γ_{i-1} and γ_i, i.e., $L_i = \max(\gamma_i) - \max(\gamma_{i-1})$. We use $\gamma_i^{norm} = \gamma_i - max(\gamma_i)$ to denote the Maxplus-normalized vector [9]. In this execution, the normalized vector of γ_1 is the same as the normalized vector of γ_4, i.e. $\gamma_1^{norm} = \gamma_4^{norm} = [-6, 0]$ and they have the same input state q_0. Therefore the schedule $\mathcal{M}_4$ results in a back edge from $\{\gamma_3^{norm}, q_0\}$ to $\{\gamma_1^{norm}, q_0\}$. The state space records the normalized time stamp vectors, since

only the relative differences between the individual time-stamp affect the future behavior, not their absolute offset. The cycle in the state space allows for a periodic execution $\sigma_{per} = (\{\gamma_1, q_0\} \cdot \{\gamma_2, q_0\} \cdot \{\gamma_3, q_0\})^\omega$. As we will see, best and worst case performance is found on such cycles in the state space.

Given a SARA-SDFG $\mathcal{G}_{SARA}$ and some execution $\sigma = \{\gamma_0, q_0\} \cdot \{\gamma_1, q_1\} \cdots \{\gamma_n, q_n\} \cdots$, the throughput of σ is, as usual, defined as the infimum limit of the number of iterations completed divided by their completion time, or equivalently, as the reciprocal of the average completion time, i.e.,

$$Th(\sigma) = \liminf_{n\to\infty} \frac{n}{max(\gamma_n)} = \liminf_{n\to\infty} \frac{n}{\sum_{i=1}^{n} L_i}$$

The throughput of the example execution with periodic input scenario sequence $S_B(S_A S_A S_B)^\omega$ in Fig. 5 is $\frac{3}{L_2+L_3+L_4} = \frac{3}{24} = \frac{1}{8}$.

The throughput of a system depends on the choices of the controller. Let $\mathbb{C}$ denote all possible controllers of a given SARA-SDFG, and let $C \in \mathbb{C}$ be a particular controller. The throughput of controller C is defined as the infimum (worst case) of all executions that are generated from arbitrary input scenario sequences, denoted by Σ_C.

$$Th(C) = \inf_{\sigma\in\Sigma_C} Th(\sigma) \tag{1}$$

Then, in turn, we can define the throughput of a SARA-SDFG $\mathcal{G}_{SARA}$ as the best possible throughput any controller can achieve, the supremum of the throughput of all possible controllers of $\mathcal{G}_{SARA}$.

$$Th(\mathcal{G}_{SARA}) \quad = \quad \sup_{C\in\mathbb{C}} Th(C) \tag{2}$$

Note that, although the use of supremum suggests that a controller achieving the throughput need not exist, we will see in the following part that it always does.

Input scenarios and controller decisions thus form opposing forces that in their interaction determine throughput. In the following we explicitly model this in order to synthesize an optimal-throughput controller that responds to different scenarios and to check whether it can satisfy the given performance constraints using the shared resources.

IV. A Mean Payoff Game

Use of games as models for analysis and synthesis problems first occurs in [15], in which a specification is translated into a deterministic automaton and the circuit synthesis problem to the computation of a strategy on a finite game graph. Here, we use a very similar approach, i.e., we translate the controller synthesis with throughput constraint problem to finding a winning strategy of a well-known game, a so-called **mean payoff** game on a **bipartite game graph**. We use the latency of an iteration, i.e., L_i as the payoff of the environment move (while $-L_i$ is the payoff of the controller move), then the average latency of an execution (the reciprocal of the throughput) is the mean of the payoffs of all iterations in a play of the game.

Fig. 7 shows the bipartite game graph constructed from the iteration state space of the running example (we explain below how it is constructed). The operation of a system controller on a given (infinite) input sequence, can be viewed as a play of a game with infinite duration between two players on the graph: one iteration in an execution is one round of the game that includes an environment turn and a controller turn. The environment player (circular nodes) provides types (scenarios) of input (outgoing edges annotated with scenarios) and the controller player (square nodes) configures the system with different settings, schedules or other parameters (edges annotated with the corresponding schedules and with the resulting latencies).

During the state space exploration, we create a new environment node for every new iteration state that we encounter and label the new node with the iteration state (normalized time stamp vector and state of the scenario FSM). For every environment node, and each possible next input from that state, a new controller node is created with an edge between the environment node and the new controller node, annotated with the selected input type. Next, we perform an intra-iteration exploration in which we explore different scheduling possibilities using the techniques from [13]. For every new schedule found, we compute the normalized time stamp vector at the end of the iteration and check recurrence with existing environment nodes. An edge is created between the controller node and a newly created or revisited node. The edge is labeled with its schedule and the latency of the iteration. Note that it is not necessarily optimal to simply select the schedule with the smallest latency as the resulting end-state may have a negative effect on future behavior.

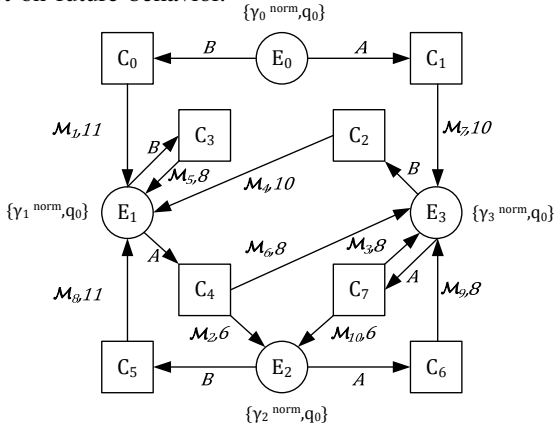


Figure 7. Bipartite game graph

Once the bipartite game graph is built, we can analyze it. For a mean payoff game, the adversarial environment player wants a strategy to maximize the average payoff per move, i.e., maximize average latency, *no matter how the controller player reacts*. (Recall that this players 'desire' to win captures worst case environment behavior. We do not assume that the environment providing input data has any real intention to lower throughput.) At the same time, the controller player wants to minimize the average loss per move, i.e., minimize average latency, *no matter how the environment player reacts*. For controller synthesis, we prefer to find a strategy that only depends on the current state and not on the history of previous states, i.e., a **positional** strategy on the bipartite game graph. It is shown in [16] that there exist optimal positional strategies for both environment and controller to obtain a value v as a Nash equilibrium point [17]. And [18] shows

the time complexity of the algorithm. So the optimal strategy of the controller to obtain maximal throughput $1/v$ is found by solving the mean payoff (i.e., mean latency) game. We use the algorithm from [19] to synthesize such an optimal positional strategy. The obtained controller is given in Fig 8, in which the red edges are the worst-case moves of the environment player while the green edges are the optimal schedules of the controller player (black edges are non-preferable actions of players). The guaranteed throughput of this controller is $1/8.5$ under the worst-case input sequence $B(AB)^{\omega}$. Note that the controller strategy also specifies schedules in response to non worst-case input scenarios.

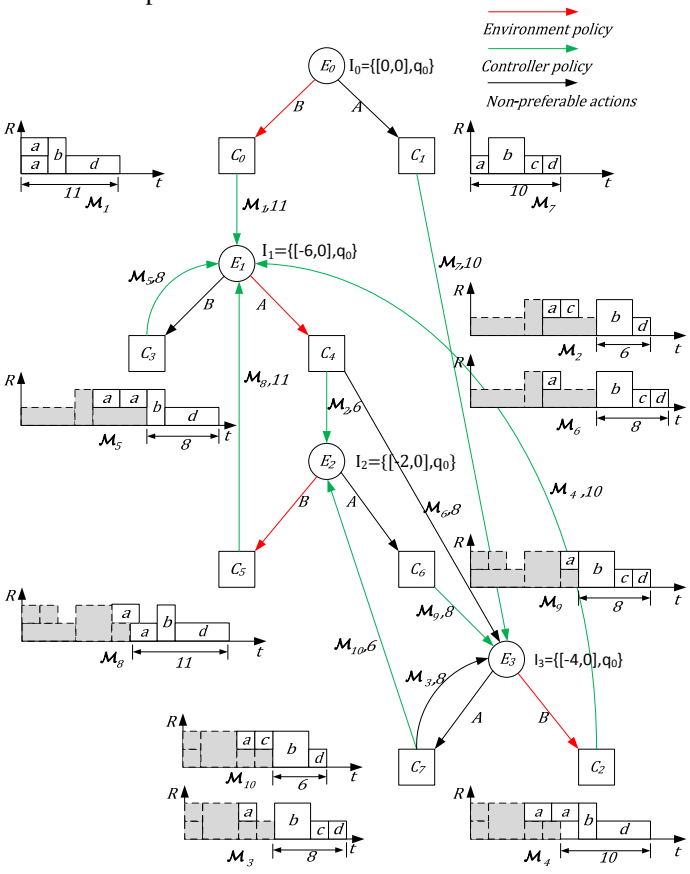


Figure 8. Synthesized controller

V. Experimental Results

We implemented the proposed approach and tested it on four applications that are specified as SARA-SDF graphs and synthesized their controllers. The PC used for the experiments is a 64-bit Linux system with an Intel 2.8Ghz $Core^{TM}$ i7 with 8GB memory. The first test graph is the SARA-SDF example given in Sec. II. The others are a system specification of an MP3 decoder application, a system specification for an MPEG-4 SP decoder [9], and a system specification of a printer image processing pipeline taken from a real industrial case study. Tab. I shows the experimental results of the controller synthesis for the four cases. We use similar techniques to those in [11], [12] to prune the design space. Limits were imposed on the iteration depth and on schedule branches to reduce the size of the iteration state space, and therefore also the size of the synthesized controller. The synthesized controllers can reach the throughputs given in the table. The throughput results are optimal. For MP3 and MPEG-4 SP, the synthesized controllers reach the same throughputs as we know from [9]. MPEG-4 SP has a long exploration time due to its large repetition vectors (many actor firings need to be scheduled in each iteration). The execution time results show that the algorithm spends most of its time on constructing the game graph and that the controller synthesis part is only a small fraction of the total analysis time. Research on a more efficient game graph construction method will be beneficial future work. Note that the refinement of execution times with smaller time units and the involvement of more resources will lead to a larger state space size.

Table I
Controller Synthesis Results

	Example	MP3	MPEG-4 SP	Printer
Iteration depth limit	2	3	3	5
Schedule branching limit	4	4	4	4
State Space Size	4	309	26	609
Game Graph Size	12	1854	260	3654
Exploration Time (ms)	409	126387	224927	74881
Synthesis Time (ms)	1	36	5	89
Throughput	$1.18 \cdot 10^{-1}$	$1.71 \cdot 10^{-7}$	$1.41 \cdot 10^{-3}$	$1.42 \cdot 10^{-3}$

VI. Related Work

We discuss related work from three fields that have been combined in this paper: decision and game theory, Maxplus algebra and dataflow.

Decision and Game Theory. Game theory [20] was originally developed as a mathematical tool to analyze games and economics behavior. The game we investigate in this paper can be classified as a *non-cooperative* game, as introduced by John Nash [17]. Our environment and controller converge to fixed policies that represent a Nash equilibrium point, which means neither can improve their strategy on the current opponent strategy. [16] introduces *mean payoff* games and shows the existence of optimal positional (memory-less) strategies. Our throughput game is modeled as a so-called infinite mean payoff game with perfect information. [18] investigates the complexity of solving *mean payoff* games and shows the complexity is in NP$\bigcap$co-NP by reducing it to a simple stochastic game. Karp's algorithm [21] can be used for playing against a known positional strategy, which amounts to computing the maximal cycle mean (MCM) of the game graph.

The technique of *policy iteration* was invented by Howard to solve stochastic control problems [22], so called Markov Decision Processes (MDP) [23]. Later, it was generalized to stochastic games [24]. However, the generalization requires strictly positive transition probabilities and cannot be directly applied to deterministic games. The applications of policy iteration to deterministic games appeared later in the study of *min-max* functions [25], [26], [27].

A game-theoretic approach is also widely used in controller synthesis for timed automata [28], [29]. Priced timed automata, for instance, are used to model costs and real-time constraints and it is shown that the problem of finding a winning strategy can be modeled as a reachability problem [29].

Maxplus Algebra and Min-max functions. In the study of control theory, Maxplus algebra [14] is widely used in performance analysis of systems with synchronization primitives,

such as timed Petri-nets [30], [31], [32]. Heaps-of-Pieces is a model that combines Maxplus analysis and automata for performance analysis [31], [32]. Spectral analysis techniques [14], [30], [33] are used to analyze the asymptotic timing behavior of such timed Petri-nets. [19], [26], [27] provide practical algorithms for spectral analysis, to compute the cycle-time vector, which are faster than Karp's algorithm in practice.

Dataflow. SDF is a special subclass of Petri-nets and the data and resource consistency can be expressed as transitions and place invariants of Petri-nets. [8] introduces scenarios of dataflow behavior in the SADF model and uses Markov chains to analyze its performance. [34] introduces parameterized SDFG for functional analysis of different scenarios. [9],[35] utilize the iteration concept and use Maxplus algebra and Maxplus automata to analyze the performance of SADF in worst-case scenarios. The explicit resource modeling in SDFGs is introduced in [11], [12] and it investigates the tradeoffs in the design space. Resource sharing can result in multiple execution paths and can be utilized by a controller to optimize resource-performance trade-offs.

VII. Conclusion

This paper introduces a new design approach from a game-theoretic viewpoint to tackle the controller synthesis problem of embedded systems with dynamic input. The approach is examined both theoretically and experimentally. The novelties of our paper are the following: modeling of dynamics in resource-sharing streaming applications, and capturing the environment-controller interaction as a mean-payoff game, yielding a method to synthesize a throughput guaranteeing controller and to identify the worst-case situation of scenarios. The experimental results show synthesis results of good quality; the controller synthesis time is only a small fraction of the construction time of the game graph. Faster construction of the game graph by limiting the scheduling options or iterating known scheduling techniques will be investigated in the future. Analyzing each scenario separately and combining their iteration state spaces together to generate a game graph can be a very good candidate approach. The adversarial environment player can be replaced by a stochastic environment player; the game then becomes a very interesting stochastic game. Results for MDP and Reinforcement Learning can be investigated for this type of game. This paper and the discussion of related work show strong links among studies in game theory, Maxplus algebra and automata models such as SDF. Since the future of embedded system design will be challenged by more and more dynamic environments, a systematic way of applying the results of the three fields to tackle the challenges seems very promising.

References

[1] J. Ni *et al.*, "Source modelling, queueing analysis, and bandwidth allocation for VBR MPEG-2 video traffic in ATM networks," *IEE Proceedings - Communications*, vol. 143, no. 4, p. 197, 1996.

[2] I. Ahmad *et al.*, "On using game theory to optimize the rate control in video coding," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 16, no. 2, pp. 209–219, Feb. 2006.

[3] Z. Fang *et al.*, "Fair bandwidth sharing algorithms based on game theory frameworks for wireless ad-hoc networks," *IEEE Infocom 2004*, vol. 2, no. C, pp. 1284–1295, 2004.

[4] J. Lotze *et al.*, "A Model-Based Approach to Cognitive Radio Design," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 2, pp. 455–468, 2011.

[5] K. Dantu *et al.*, "Frame-based dynamic voltage and frequency scaling for a MPEG decoder," in *ICCAD '02*. IEEE, 2002, pp. 732–737.

[6] P. Pillai *et al.*, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, p. 89, Dec. 2001.

[7] K. Compton *et al.*, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[8] B. Theelen *et al.*, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *MEMOCODE '06*, pp. 185 –194.

[9] M. Geilen *et al.*, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES/ISSS '10*, 2010, pp. 125–134.

[10] E. A. Lee *et al.*, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, pp. 24–35, 1987.

[11] Y. Yang *et al.*, "Automated bottleneck-driven design-space exploration of media processing systems," in *DATE '10*, 2010, pp. 1041–1046.

[12] Y. Yang *et al.*, "Exploring trade-offs between performance and resource requirements for synchronous dataflow graphs," in *ESTIMedia 2009.*, pp. 96 –105.

[13] Y. Yang *et al.*, "Iteration-based trade-off analysis of resource-aware sdf," in *DSD '11*, 2011, pp. 567–574.

[14] G. Cohen *et al.*, *Synchronization and Linearity, An Algebra for Discrete Event Systems*, 1992.

[15] J. R. Buchi *et al.*, "Solving Sequential Conditions by Finite-State Strategies," *Transactions of the American Mathematical Society*, vol. 138, p. 295, Apr. 1969.

[16] A. Ehrenfeucht *et al.*, "Positional strategies for mean payoff games," *International Journal of Game Theory*, vol. 8, pp. 109–113, 1979.

[17] J. Nash, "Non-Cooperative Games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, Sep. 1951.

[18] U. Zwick *et al.*, "The complexity of mean payoff games," in *Computing and Combinatorics*, ser. LNCS. Springer, 1995, vol. 959, pp. 1–10.

[19] V. Dhingra *et al.*, "How to solve large scale deterministic games with mean payoff by policy iteration," in *Valuetools '06*, 2006, p. 12.

[20] J. von Neumann *et al.*, *Theory of Games and Economic Behavior*. Princeton University Press, 2007.

[21] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, no. 3, pp. 309–311, Sep. 1978.

[22] R. A. Howard, *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[23] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, 1957.

[24] A. J. Hoffman *et al.*, "On nonterminating stochastic games," *Management Science*, vol. 12, no. 5, pp. pp. 359–370, 1966.

[25] J. Gunawardena, "From max-plus algebra to nonexpansive mappings: a nonlinear theory for discrete event systems," *Theor. Comput. Sci.*, vol. 293, pp. 141–167, February 2003.

[26] S. Gaubert *et al.*, "The duality theorem for min-max functions," *Comptes Rendus de l'Acad?ie des Sciences - Series I - Mathematics*, vol. 326, no. 1, pp. 43 – 48, 1998.

[27] J. Cochet-Terrasson *et al.*, "A constructive fixed point theorem for min-max functions," *Dynamics and Stability of Systems*, vol. 14, no. 4, pp. 407–433, Dec. 1999.

[28] G. Behrmann *et al.*, "Resource-optimal scheduling using priced timed automata," in *SIGMETRICS Perform. Eval. Rev*. Springer, 2004, pp. 220–235.

[29] P. Bouyer *et al.*, "Optimal strategies in priced timed game automata," in *In FSTTCS 04, LNCS 3328*. Springer, 2004, pp. 148–160.

[30] S. Gaubert, "Performance evaluation of (max,+) automata," *IEEE Trans. on Automatic Control*, vol. 40, pp. 2014–2025, 1993.

[31] S. Gaubert *et al.*, "Asymptotic analysis of heaps of pieces and application to timed petri nets," in *PNPM'99*, 1999, pp. 158–170.

[32] S. Gaubert *et al.*, "Modeling and analysis of timed petri nets using heaps of pieces," *IEEE Trans. on Automatic Control*, vol. 44, pp. 683–697, 1999.

[33] J. Cochet-Terrasson *et al.*, "Numerical computation of spectral elements in max-plus algebra," in *Proc. IFAC Conf. on Syst. Structure and Control*, 1998.

[34] B. Bhattacharya *et al.*, "Parameterized dataflow modeling for dsp systems," *IEEE Trans. on Signal Processing*, vol. 49, no. 10, pp. 2408 –2421, Oct. 2001.

[35] M. Geilen, "Synchronous dataflow scenarios," *ACM Trans. Embed. Comput. Syst.*, vol. 10, pp. 16:1–16:31, January 2011.