

Context-Aware Process Networks

Hylke W. van Dijk and Henk J. Sips

Abstract—In industry, embedded systems for stream-based processing are often modelled and verified by using process networks, such as Kahn process networks. An advantage of Kahn networks is that they allow asynchronous operation of process components in a network. A problem in these networks, however, is that asynchronously interfering events cannot be handled properly because they are intrinsically indeterminate and therefore destroy the compositional properties of the network.

In this paper, we propose to extend the Kahn model of computations with a simple indeterminate construct. We call the resulting network a context-aware process network (CAPN). We show that these networks are capable of handling certain classes of events and can still be reduced to a class of parametrised Kahn networks.

Index Terms—asynchronous communication, asynchronous coordination, discrete event processing, stream-based processing, KPN, CAPN, QoS

EMBEDDED systems for stream-based processing (video, audio) are often found in consumer electronics applications. They need to operate flawlessly and hence much effort is spent in the design and verification of these applications. A popular model of computation for these applications is the Kahn process network model (KPN). The Kahn model is compositional and allows asynchronous coupling of the processing nodes in a network. An important advantage of Kahn networks is that verification of the concurrency properties of a network can be done without knowing the internal processing details of the nodes, provided that the port scans of a node are done in the prescribed way.

Apart from handling streams, embedded systems need to be able to handle events. A simple example of an event is a user pushing a button. Incorporating events in Kahn networks is awkward; often events are turned into a stream, e.g., a coherent stream of “on” and “off” events.

However, modern embedded systems require more complex forms of handling events. Events, typically, represent changes in the *context* of the embedded system. The button example represents a discrete change in the mode of the operation of the embedded system, but changes of context can also be more gradual. An example of a gradual change is an increase or decrease in transmission rate of a system due to external causes (e.g. a wireless link).

Proper system integration in embedded systems relies on the availability of *determinate* – and thus compositional – components in order to optimise functionality, performance, and resource utilisation. Unless one effectively can capture (isolate) their context dependent parts, any composition of indeterminate components yields a non-deterministic system. Although examples of working non-deterministic systems are ubiquitous, the current practice is not satisfactory.

Having a proper Model of Computation (MoC) greatly improves the effectiveness of the system integration phase in developing context-aware embedded systems. Basically there exist two approaches. One either develops a MoC based on discrete-event processing or a MoC based on stream-based processing. One could recast a state-based MoC such as finite state machines or Petri nets, which allows for model-checking and verification. But due to possible state-space explosions these MoCs are considered less suitable for an analytical evaluation of embedded stream-based processing applications.

In this paper we propose an approach that starts from a well known MoC for stream-based processing applications: Kahn process networks. We recast this model to support context awareness, i.e. we add *asynchronous coordination* functionality.

Kahn process networks (KPN) [1]–[3] have the distinct property of being determinate and therefore being compositional. The KPN model of computation facilitates asynchronous *communication* among components in a system. To facilitate asynchronous *coordination*, we extend the KPN MoC and form a MoC called **context-aware process networks** (CAPN). Although our extension necessarily sacrifices the determinate property of KPN, we show that the indeterminacy in a network can be concentrated into a control stream. This so-called *oraclisation* [4] makes CAPN compositional again.

This paper is organised as follows. We conclude this introduction with a motivating example and the practical solution of our approach. In Section I we provide some background on the loss of the compositional property when incorporating indeterminate constructs. In Section II we recapitulate KPN semantics and formalise CAPN semantics. Subsequently in Section III we model an indeterminate CAPN system as a parameterised KPN system. In Section IV we analyse CAPN for specific cases.

A. Motivating example

The type of stream-based applications we consider can be abstracted by a common producer-consumer system. Figure 1 shows a producer separated from a consumer by a transport system. The transport system of Figure 1 is left unspecified; it might suffer from imperfections such as latency (buffering), error injection, or out-of-order delivery.

In an ideal environment, interference free and with access to infinite resources, applications can effectively be modelled with a producer, a perfect transport system, and a consumer. The only necessary communication links are the ones that transfer the data: Tx and Rx in Figure 1. The context inputs Pctx, Tctx, and Cctx are void in this case. A model of computation for such a system should implement asynchronous communication, the Kahn process network MoC of Section II has that property.

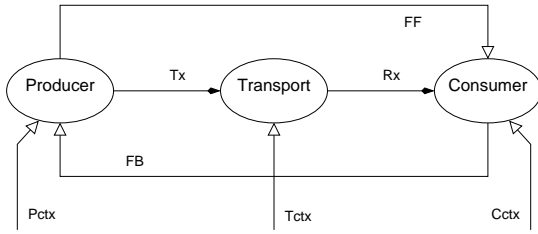


Fig. 1. Context-aware stream-based process network with asynchronous event-driven control; feedback (FB), feed forward (FF), and context sensing (Xctx).

Unfortunately, the environment of an embedded system is far from ideal. Implementations of a producer, transport, or consumer system usually share resources with other systems. Context awareness allows system components to adapt to variations of their context. In Figure 1 the respective components sense their context through the inputs Pctx, Tctx, and Cctx.

Sensing contextual information is typically an asynchronous process. The updates of context information and the effectuation of this information need not be synchronised with data handling. A video decoder, e.g., uses a clock to maintain real-time, isochronous, operation. Every once in a while the decoder has the option to change its mode of operation, e.g., adjust the so-called frame-skip factor. Only at those instances the decoder will obey the external timer information that arrives at its own pace. The coordination of the timer process and the decoder process is asynchronous; time stamps may be overruled before being applied or being used multiple times.

Similar asynchronous coordination is observed between a producer and a remote consumer, which is especially the case when these system components need to be flexible. Firstly because the optimal rate of exchanging information most likely differs with the implementation, and secondly because the cooperation may be on an ad-hoc basis with various parties, which obstructs the clear definition of a common denominator for the information exchange rate that serves all.

B. Modelling asynchronous cooperation

Given the above considerations, a modelling construct that captures asynchronous coordination is fairly straightforward. The transfer of stream-oriented data is modelled by unidirectional FIFO links and the transfer of context-oriented information is implemented by unidirectional register (REG) links. When unbounded FIFOs are applied, asynchronous communication between components is guaranteed. When non-blocking REGs are applied, asynchronous coordination between components is guaranteed.

It is reasonably simple to implement the REG construct in a simulation environment. Experiments, done by ourselves and others (Section IV-D), indicate that a MoC that includes FIFO and REG links indeed is a powerful modelling paradigm. In the remainder of this paper we analyse the non-trivial consequences of this paradigm.

I. INDETERMINATE PRIMITIVES

The introduction of REG links in Kahn semantics adds an indeterminate construct to an otherwise determinate MoC.

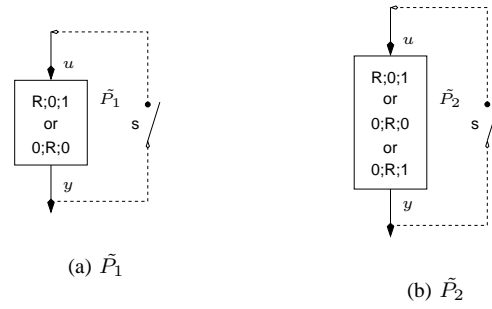


Fig. 2. Indeterminate processes with explicit context.

In this section we provide some background on appearance, handling, and consequences of the application of indeterminate primitives.

The most prominent and well studied indeterminate modelling construct is the *merge* primitive [5], [6]. A merge primitive, or any other indeterminate primitive, has *context* dependent input-output behaviour and therefore lacks a compositional property, a phenomenon known as the Brock and Ackerman anomaly [5]. As an example, due to [4], consider an indeterminate process \tilde{P}_1 with a single input u and a single output y (Fig. 2(a)). \tilde{P}_1 transforms the input sequence u into the output sequence y . Internally, \tilde{P}_1 has two deterministic modes of operation indicated by a control parameter α : [$\alpha = 0$] means a read from the input port, followed by a write of a 0, followed by a write of a 1 to the output port, denoted as $R;0;1$. [$\alpha = 1$] means a write of a 0 to the output port, followed by a read from the input port, followed by a write of a 0 to the output port, denoted as $0;R;0$.

The input-output relation of the network \tilde{P}_1 depends on the indeterminate selection mechanism (α), the *context* (s open or closed), and the (in)availability of events on u of the network. Table I(a) gives the input-output relations (with s open) of the networks of Figure 2 (\perp denotes no output).

One can devise a network \tilde{P}_2 with equivalent input-output relations to those of \tilde{P}_1 . \tilde{P}_2 has the same modes of operation as \tilde{P}_1 and an additional one, namely: [$\alpha = 2$] means a write of a 0 to the output port, followed by a read from the input port, followed by write of a 1 to the output port, denoted as $0;R;1$. Obviously \tilde{P}_1 and \tilde{P}_2 have equivalent input-output relations, Tables I(a) and I(b) respectively. Both processes generate either $\{\perp, 0\}$ or $\{0;1, 0;0\}$.

However, when operating in a different context (closing s) the two network components may behave differently, meaning that they lose their compositional property; compare Tables I and II.

The internal control parameter α of process \tilde{P}_1 has been left unspecified so far. One option is to generate the control sequence $c = \langle \alpha_0, \alpha_1, \dots \rangle$ by a merge process that merges two streams of control events. One stream with events $\alpha = 0$ and one with events $\alpha = 1$. The behaviour of the network depends on the *fairness* of the merge process. It turns out there exists a hierarchy of merge primitives [6] with provable inequivalent expressive power. Common merge primitives are ranked according to their expressibility from strong to weak

TABLE I

INPUT-OUTPUT RELATIONS OF FIG. 2, S OPEN; u HAS AND $\neg u = \{\perp\}$ HAS NO INPUT EVENTS.

(a) \tilde{P}_1 .			(b) \tilde{P}_2 .		
c	$\neg u$	u	c	$\neg u$	u
0	\perp	0;1	0	\perp	0;1
1	0	0;0	1	0	0;0
			2	0	0;1

TABLE II

INPUT-OUTPUT RELATIONS OF FIG. 2; S CLOSED.

(a) \tilde{P}_1 .		(b) \tilde{P}_2 .	
c		c	
0	\perp	0	\perp
1	0;0	1	0;0
		2	0;1

as: *fair* merge, *angelic* merge, *infinity-fair* merge, and *unfair* merge. Fair merge is truly fair in all circumstances, angelic merge is fair only when both inputs are finite, whereas infinity-fair merge is only fair when both input streams are infinite. Unfair merge, the ordinary “or” operation, does not give any guarantees at all.

In their original paper [5], Brock and Ackerman prove the non-compositional property of input-output relations when using merge process components. Their proof explicitly relies on the fairness of the applied merge primitive, namely fair merge. Russell proved the non-compositional property for unbounded choice (or unfair merge) [7] and thus demonstrates the non-compositional property for any merge primitive.

The hierarchy of expressiveness of the merge primitives is reflected in the fact that weaker merge primitives can be composed out of stronger merge primitives; the angelic merge can be composed of fair merge primitives, infinity-fair merge can be composed of angelic merge primitives. The reverse, however, is not possible.

Let u_0, u_1 be the input sequences of a merge primitive and let y be its output sequence. Let y_0 and y_1 be two streams constructed from the output stream y as follows: events of y that originate from u_0 go to y_0 , events of y that originate from u_1 go to y_1 . The order of events of y is preserved on y_0 and y_1 . The *fair* merge primitive guarantees that $y_0 = u_0$ and $y_1 = u_1$ under all circumstances. *Angelic* merge transmits all events of stream u_0 if the stream u_1 is *finite* and transmits all events of stream u_1 if the stream u_0 is finite. Angelic merge thus behaves like fair merge when both u_0 and u_1 are finite. *Infinity-fair* merge is the dual of angelic merge. Infinity-fair merge transmits all events of u_0 provided u_1 is *infinite* and it transmits all events of u_1 provided u_0 is infinite. Thus infinity-fair merge behaves like fair merge when both u_0 and u_1 are infinite.

Indeterminacy thus comes in grades, which has consequences for the possible isolation of indeterminate behaviour. An effective isolation of indeterminate behaviour may retain

the compositional property, especially when the context of the indeterminate entity can be characterised adequately.

The mere name of the infinity-fair merge is due to the composition of a deterministic component and an indeterminate random number generator [8]. The random number generator acts as the oracle and generates a sequence of positive integers. The determinate process has two input ports for streams u_0 and u_1 , an output port y , and a control port c to which the generated random number stream is connected. The process starts with reading an integer from the control port that specifies the quantity of tokens to be read alternately from u_0 and u_1 . See Figure 3 for a diagram.

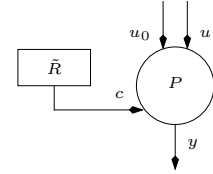


Fig. 3. Infinity-fair merge oraclised.

Infinity-fair merge can be oraclised since it exposes a (weak) form of monotonicity [9]. Monotonicity defines causality for models of communication that operate on (ordered) sequences rather than implement global time. Angelic merge also has a (different) form of weak monotonicity, whereas fair merge has not and consequently can not be oraclised.

The implementation of a non-monotonic function requires preemption of the computations, for the output stream is affected by non-causal events on the input stream. Preemption is commonly implemented by constructs like timeout, interrupt, or polling. Since angelic merge and infinity-fair merge behave in a sense monotonically, a preemption-free implementation is possible. Fair merge, being non-monotonic, cannot be implemented preemption free [10].

Strict monotonicity of the associated (monotone) functions of the processes in the system is important because of the existence of the least fixed point of such as system which makes the system determinate and compositional. For a subset of the dataflow models of computations analytical procedures to find the least fixed point are known, however in general verification tools are used, such as SPIN [11].

II. ASYNCHRONOUS MoC

Kahn process networks (KPN) (Section II) are determinate; given an input, the output follows deterministically, independent from the implementation. Hence KPNs are compositional, a prerequisite for complex system development. KPNs implement asynchronous communication but synchronous coordination. In order to facilitate asynchronous coordination we introduce an, indeterminate, register link. We call the resulting MoC Context-Aware Processing Networks (CAPN). We develop CAPN in view of KPN so as to isolate (*oraclise* [4]) its indeterminate properties.

A. Kahn Process Networks

Kahn process networks [1] are characterised by lacking global memory and a global schedule. Kahn process networks

connect nodes. An autonomous process is associated with each node. Nodes have ports and each port is linked to at most one other port through a unidirectional unbounded FIFO queue. Processes are allowed to read, destructively only, from an input port and are allowed to write to an output port. The read operation ($\text{get}()$) is blocking, the write operation ($\text{put}()$) is non-blocking. Kahn semantics guarantee a causal relation between input events and output events; incoming events do never influence previously generated output events (monotonicity). As a direct consequence, KPNs are deterministic and thus compositional; their input-output relations are independent from the implementation (the execution order) and the context.

Kahn process networks have proved their usefulness in methodologies for developing embedded systems [2], [3], [12], [13]. A particularly well explored set of networks is the set dataflow networks [14], which is related, but conceptually different, to KPNs. A dataflow network has a global schedule, whereas KPN has not. In view of KPN, the nodes of a dataflow network have cyclic behaviour and receive, implicitly or explicitly, control events. A cycle starts with the read of a control event that specifies a mode of operation for the current cycle. In particular, the order and the number of $\text{get}()$ and $\text{put}()$ invocations is predefined by this mode of operation.

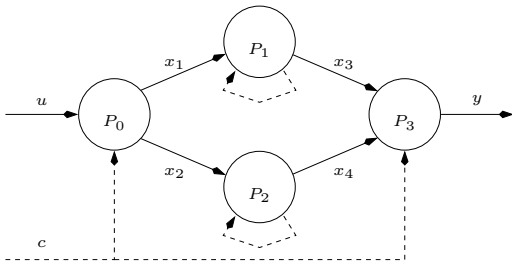


Fig. 4. Dataflow process network.

The diagram of Figure 4 illustrates a dataflow network, a typical switch that allows a stream to be processed through either P_1 or P_2 . The nodes P_1 and P_2 have implicit (static) control; every control event (a so-called firing) triggers one $\text{get}()$ and one $\text{put}()$ operation, in that order. The nodes P_0 and P_3 receive their control events coherently from the stream c . A control event effectively activates the upper path (through P_1) or the lower path (through P_2). Incoherently supplied control events may deadlock the network.

B. Context-Aware Process Networks

Implementing feedback coordination in the Kahn process network yields synchronous coordination. Consider Figure 5. Here the producer and consumer are connected through unidirectional unbounded FIFOs. The link labelled xfer carries the data, the link labelled cont feeds back context (coordination) information. Using Kahn semantics, any feasible schedule for the producer and consumer results in determinate (fixed) behaviour of the network, because the course of events on each link does not change.

To turn KPN semantics into CAPN semantics we introduce a unidirectional non-blocking register (REG) link, which has

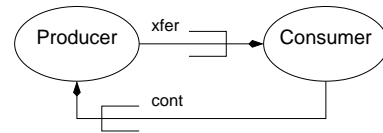


Fig. 5. Synchronous coordination (KPN).

destructive and replicative behaviour. We define the behaviour of a REG link as follows. Writing ($\text{put}()$) to a full register overwrites a previously written value. Reading ($\text{get}()$) from a register returns the last received value; a value can be returned multiple times.

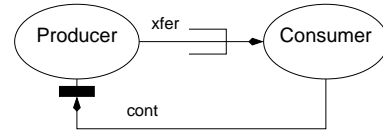


Fig. 6. Asynchronous coordination (CAPN).

As a direct consequence, the behaviour of a CAPN network depends on the applied schedule (context). To demonstrate this, regard the networks of Figures 5 and 6 as dataflow networks with implicit control. Every invocation (firing) of the producer triggers a $\text{get}()$ from link cont followed with a $\text{put}()$ on link xfer . The actual content of the $\text{put}()$ depends on the observed context information. Similarly, each invocation of the consumer triggers a $\text{get}()$ from xfer and a $\text{put}()$ on cont . In case of KPN the coordination is synchronous; all events sent by the consumer arrive at the producer. In case of CAPN the coordination is asynchronous. In this case, events sent by the consumer may be replicated or discarded depending the schedule. To show this consider the following:

Let λ_0 be the default value (after reset) of the register and $\langle \lambda_1, \lambda_2, \dots \rangle$ be the sequence of coordination events emitted by the consumer on successive invocations. If the schedule alternately invokes the producer and consumer, the producer observes the same sequence of events: $\langle \lambda_0, \lambda_1, \lambda_2, \dots \rangle$. A different schedule that alternately invokes the producer k times followed by k successive invocations of the consumer lets the producer observe $\langle (\lambda_0)^k, (\lambda_k)^k, (\lambda_{2k})^k, \dots \rangle$. That is, k replications of λ_0 followed by k replications of λ_k ; all intermediate coordination events are discarded.

III. KPN MODELLING OF CAPN

In this section we analyse the behaviour and semantics of the REG link. We start of with commenting on its applicability followed by a model to isolate (oraclise) the indeterminate parts of the REG link. Our analysis yields a KPN model of the REG link with explicit context dependence: a control stream. A generating process of this control stream is readily achieved using, not surprisingly, a merge construct.

A. Select

Given the REG link we can easily construct a select mechanism that passes events from whichever input port has events

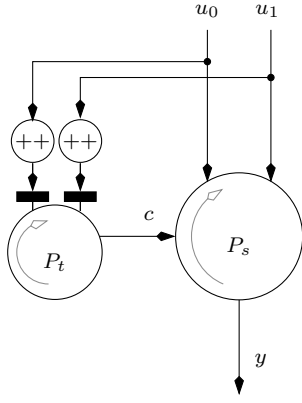


Fig. 7. Select composition

available. In Figure 7 each input stream is forked to a counting node, which outputs a successive number for every received event. These counter values are communicated to a process P_t over a REG link. The process P_t , therefore, never blocks but may read the same counter value multiple times. By maintaining state information P_t can detect changes of the counter value, which indicates the arrival of an event. The arrival of an event triggers a send of the respective channel number (u_0 or u_1) to the process P_s . Note that executable implementations usually use a trigger function to indicate the update of any counter. Kahn's own `warn()` operator is an example of such a trigger [1].

B. Oraclised CAPN

The schedule dependent behaviour of context-aware process networks can be modelled using KPN. In this section we derive a static model. The basic behaviour of CAPN is that a REG link may replicate tokens or discard tokens, which can be modelled through an Interpolator and Decimator respectively.



Fig. 8. Basic CAPN behaviour.

Figure 8 shows two KPNs with parameterised dataflow processes: an Interpolator (L) and a Decimator (M). Their behaviour is defined as follows. Let $\langle \lambda_0, \lambda_1, \lambda_2, \dots \rangle$ be the input sequence of events to an Interpolator or Decimator, then the corresponding output sequences are given as:

$$\text{Interpolator}(L) : \langle (\lambda_0)^L, (\lambda_1)^L, \dots, (\lambda_k)^L, \dots \rangle \quad (1)$$

$$\text{Decimator}(M) : \langle \lambda_{M-1}, \lambda_{2M-1}, \dots, \lambda_{kM-1}, \dots \rangle \quad (2)$$

An Interpolator replicates each incoming token L times, whereas a Decimator discards $M-1$ tokens out of every block of M incoming tokens.

A simple composition of an Interpolator and a Decimator yield the networks of Figure 9. The network of Figure 9(a) has

a Decimator followed with an Interpolator and the network Figure 9(b) has an Interpolator followed with a Decimator. Their input-output relations are respectively denoted as $y = \text{DecInt}(M, L; u)$ and $y = \text{IntDec}(L, M; u)$, where L and M are the static parameters, u the input stream and y the output stream.

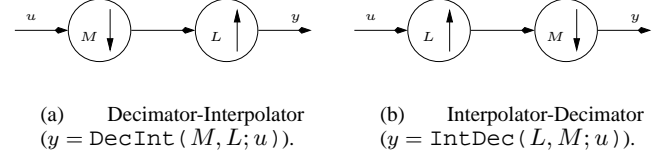


Fig. 9. Simple composition.

The simple composition networks of Figure 9 replicate and discard events, which follows from their input-output relations. Let $u = \langle \lambda_0, \lambda_1, \dots \rangle$ be the input sequence of events to either one of the networks. The resulting output sequence y for the Decimator-Interpolator network of Figure 9(a) is readily derived as:

$$\text{DecInt}(M, L; u) : y = \langle (\lambda_{M-1})^L, (\lambda_{2M-1})^L, \dots \rangle \quad (3)$$

Deriving the input-output relations of the Interpolator-Decimator network of Figure 9(b) requires some more evaluation due to the interdependency of the parameters M and L . Here the output sequence y has replicated consecutive events as follows:

$$\text{IntDec}(L, M; u) : y = \langle (\lambda_0)^{\varphi[0]}, (\lambda_1)^{\varphi[1]}, \dots \rangle \quad (4)$$

The replication factor $\varphi[j]$ is periodically and recursively defined through the floor operation as $\varphi[j] = \lfloor \frac{\theta[j]}{M} \rfloor$ (which can be zero). The period of $\theta[j]$ equals $M/\text{gcd}(L, M)$, in which $\text{gcd}(L, M)$ is the greatest common divisor of L and M ; $\theta[j]$ is defined as:

$$\begin{aligned} \theta[0] &= L \\ \theta[j+1] &= L + \theta[j] \bmod M \end{aligned} \quad (5)$$

C. Sample-rate converter

In the previous section we developed a parameterised model for a simple composition of the basic behaviour of CAPN. More complex behaviour can be accomplished by an arbitrary, yet parameterised, composition of Decimators and Interpolators. The actual sequence of Decimators and Interpolator as well as their individual parameters (L_i and M_j) depend on the context (schedule) of the network. To make this dependency explicit we devise a dynamic model of the REG link, which concentrates the dependency in a control stream, $c = \langle \alpha_0, \alpha_1, \dots \rangle$. The sample-rate converter (SRC), node ($S_{\uparrow,1}$) in Figure 10, does precisely this. The SRC is a dataflow process node, with state q , that operates as follows: At the i -th invocation, the sample-rate converter reads the i -th control event α_i , $\alpha_i \in \{-1, +1\}$. A negative control value yields reading from the data input port and storing the read value ($q \leftarrow \text{get}()$). A positive control value yields writing the current state to the data output ($\text{put}(q)$). Let $q = \lambda_{-1}$ and $u = \langle \lambda_0, \lambda_1, \lambda_2, \dots \rangle$

be the current state of the input FIFO, then the behaviour of the Interpolator and Decimator of Equations (1) and (2) are mimicked by the respective control streams c_I and c_D as follows:

$$\text{Interpolator}(L) : c_I = \langle -1, (1)^L, -1, (1)^L, -1, \dots \rangle \quad (6)$$

$$\text{Decimator}(M) : c_D = \langle (-1)^M, 1, (-1)^M, 1, \dots \rangle \quad (7)$$

As an example, let $c = \langle -1, (1)^k, (-1)^k, (1)^k, (-1)^k, \dots \rangle$ and u and q be defined as above, then the output sequence y equals $\langle (\lambda_0)^k, (\lambda_k)^k, (\lambda_{2k})^k, \dots \rangle$, which corresponds to the result derived in Section II. The first entry in c ensures proper initialisation.

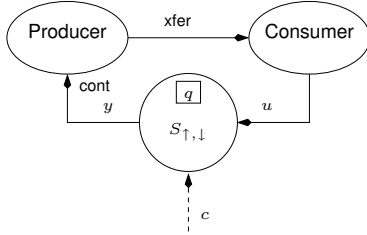


Fig. 10. Oraclised REG link.

The SRC substitutes the REG link behaviour, provided an appropriate control stream can be generated. Moreover, the substitution turns a CAPN into a KPN with manifest indeterminate behaviour. Figure 10 is the oraclised form of Figure 6, in which the schedule related context information is concentrated in the control stream c . In the following subsections we consider two different sources of this control stream: we will discuss a non-deterministic configuration with a free running schedule and a deterministic configuration in which we generate a cyclic control stream from a predefined pattern.

1) *Non-deterministic control stream*: The control stream of the SRC of Figure 10 abstracts the context of the network. One possible way of generating such a stream is through the use of a merge primitive, emphasising the indeterminate aspect of the REG link. Figure 11 is an emulation of the REG link using a SRC and angelic merge. A free running schedule of this process network effectively generates the control stream. The emulation requires a minor modification of the producer and consumer process when accessing the REG channel. A `get()` of the producer from the register port is preceded with a `put($\alpha = 1$)` to the merge construct. Similarly, a `put()` of the consumer to the register is preceded with a `put($\alpha = -1$)` to the merge construct. The merge node passes the incoming events to the SRC; either $\alpha = -1$ or $\alpha = 1$. The SRC behaves like specified. As a result, the network has the same schedule dependent behaviour as in Section II. Angelic merge is the weakest possible form of the merge primitive that guarantees a deadlock free implementation. Suppose the consumer applies a finite number of updates, then all consecutive producer requests should be acknowledged. Or vice versa, if the producer poses a finite number of requests, then the consecutive send operations of the consumer should be handled by the SRC. Angelic merge guarantees precisely this.

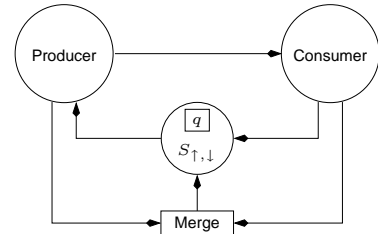


Fig. 11. REG link emulation.

2) *Cyclo-static control stream*: In case of a deterministic context of the network, the control stream of the SRC of Figure 10 is also deterministic. We propose to generate such a stream through the use of a parameterised sample rate converter that builds upon the parameterised Interpolator and Decimator processes of Section III-B. The Interpolator, Decimator, and their compositions have a regular cyclic behaviour. We capture the cyclic behaviour by using a converter process with an associated characteristic set of coefficients (h) (Figure 12). This set h describes a sample rate conversion pattern, which is comparable to the kernel of a convolution process. The canonical form of h uniquely characterises the converter process, moreover h is compositional.

We denote the converter process as $y = \text{SRC}(h; u)$. The coefficients of h define the replication factor ($\in \mathbb{N}$) of an input data token in the output data stream y ; h is applied cyclically to the input stream u . Let $u = \langle \lambda_0, \lambda_1, \lambda_2, \dots \rangle$ be the input data stream and $h = \{h_0, h_1, \dots, h_{N-1}\}$ be the characteristic (ordered) set of coefficients with period N , then the output sequence y of $\text{SRC}(h; u)$ is given as:

$$y = \langle (\lambda_0)^{h_0}, (\lambda_1)^{h_1}, \dots, (\lambda_{N-1})^{h_{N-1}}, \dots, (\lambda_i)^{h_{i \bmod N}}, (\lambda_{i+1})^{h_{(i+1) \bmod N}}, \dots \rangle. \quad (8)$$

As an example, the output sequences of the Interpolator and Decimator, Equations (1) and (2) respectively, are generated by applying their respective coefficients h_I and h_D to an input sequence u ; h_I and h_D are parametrically given as:

$$\text{Interpolator}(L) : h_I = \{L\} \quad (9)$$

$$\text{Decimator}(M) : h_D = \{(0)^{M-1}, 1\} \quad (10)$$

The length (dimension) of the characteristic set of coefficients, denoted $|h|_0$, determines the size of the block of input tokens the sample rate converter takes per cycle. The usual ℓ_1 vector norm of the characteristic set of coefficients, denoted $|h|_1$, determines the number of tokens in the output for every input block; $|h|_p = \sum_{i=0}^{N-1} (h_i)^p$ for $p \in \{0, 1\}$. With respect to Equations (9) and (10) we have: $|h_I|_0 = 1$, $|h_I|_1 = L$, $|h_D|_0 = M$ and, $|h_D|_1 = 1$.

The **composition** of two finite characteristic sets of coefficients f and g yields a finite set $h = g \circ f$. In order to form their composition f and g must be *balanced*: f and g are balanced iff $|f|_1 = |g|_0$, i.e., the number of outputs per cycle from $\text{SRC}(f; u)$ matches the number of inputs per cycle of $\text{SRC}(g; u)$ for any u . The balance condition can always be met

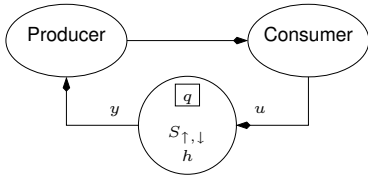


Fig. 12. Cyclo-static REG link emulation ($y = \text{SRC}(h; u)$).

because the cyclic extension of any finite h by a factor k yields equivalent behaviour of the converter process, $\text{SRC}(h; u) = \text{SRC}((h)^k; u)$, and f and g are finite.

Given two balanced finite characteristic sets of coefficients f and g their composition is found through the following procedure. Partition the set g according to the value of the subsequent entries of f . Let f_i be the i -th entry in f and g'_i be the corresponding i -th sub partition of g then due to the partitioning scheme we have $|g'_i|_0 = f_i$. In case $f_i = 0$, the corresponding g'_i equals a set with empty elements only, $\{\perp\}$, which has expected properties: $|\{\perp\}|_0 = |\{\perp\}|_1 = 0$. Consequently we define h through its entries as: $h_i = |g'_i|_1$. The partitioning of g is guaranteed to fit since $|f|_1 = |g|_0$. Observe that the resulting characteristic set of coefficients inherits the properties of its constituents: $|f|_0 = |h|_0$ and $|g|_1 = |h|_1$. Moreover h is finite by construction.

As an example consider the composition $h = g \circ f$. Let $f = \{5\}$ and $g = \{0, 0, 1\}$. Thus h is the characteristic set of coefficients of an $\text{IntDec}(L = 5, M = 3)$. We first balance f and g ; let $r = \text{gcd}(|f|_1, |g|_0)$, we extend f and g cyclically as follows

$$\begin{aligned} \{(f) \frac{|g|_0}{r}\} &= \{5, 5, 5\}, \\ \{(g) \frac{|f|_1}{r}\} &= \{0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1\}. \end{aligned}$$

Subsequently we partition the cyclically extended g according to extended f and form

$$g' = \{\{0, 0, 1, 0, 0\}, \{1, 0, 0, 1, 0\}, \{0, 1, 0, 0, 1\}\}.$$

Taking the ℓ_1 -norm of the subsets yields

$$h = \{1, 2, 2\},$$

which is equivalent with the result of Equation (4). The period of h , $|h|_0$ is equal to the cyclic extension factor of f , $\frac{|g|_0}{r} = \frac{M}{\text{gcd}(L, M)}$. Further the number of outputs of h , $|h|_1$, is equal to the cyclic extension factor of g , $\frac{|f|_1}{r} = \frac{L}{\text{gcd}(L, M)}$. Hence, the composition $h = g \circ f$ of an Interpolator f and a Decimator g only depends on the ratio L/M ; the norms $|h|_0$ and $|h|_1$ do not change if we multiply L and M with a factor $k \in \mathbb{N}^+$.

It is straightforward to derive the characteristic set of coefficients for a generic $\text{DecInt}(M, L; u)$ composition. Let $f = \{(0)^{M-1}, 1\}$ and $g = \{L\}$ be the characteristic set of coefficients for an Decimator and Interpolator, respectively. In this case f and g are balanced and g' is readily formed

$$g' = \{(\{\perp\})^{M-1}, \{L\}\},$$

which yields

$$h = \{(0)^{M-1}, L\}.$$

Finally, we give the relation between a characteristic set of coefficients h and the cyclically extended control stream c of Figure 10.

$$\begin{aligned} \text{if } h_i = 0, \text{ then } \alpha_i &= \langle -1 \rangle \\ \text{if } h_i > 0 \text{ then } \alpha_i &= \langle -1, (1)^{h_i} \rangle \end{aligned}$$

The cyclo-static sample rate converter we introduced in this section is a proper reasoning framework for further analysis. We showed in particular that the composition of two cyclo-static sample rate converter again yields a cyclo-static sample rate converter, which is uniquely characterised by a characteristic set of coefficients. In the next section we apply our reasoning framework.

IV. ANALYSIS

We have introduced CAPN in order to facilitate asynchronous coordination. However indeterminate, for a specific class of dataflow networks we can parameterise the schedule dependent behaviour of CAPN. In this section we analyse CAPN in a cyclo-static dataflow context. We derive a dataflow network with parameterised, and thus asynchronous, coordination. The overall behaviour of the network depends on the actual setting of the parameters. We show in an example that in the case of cyclo-static dataflow it is possible to derive a closed expression under which proper behaviour of the network is guaranteed. A second more generic example is discussed to support the case for using CAPN as a useful MoC for context-aware applications.

A. Asynchronous coordination

Consider the cooperative producer-consumer network of Figure 13. The diagram presumes the synchronous dataflow (SDF) MoC, which is the most restricted and best analysable member of a family of dataflow models of computation [14], [15]. The cyclo-static dataflow (CSDF) yields a useful extension to (SDF) for which practical analysis techniques are available [16]. A node in an SDF network specifies the number of tokens it reads or writes on every invocation (firing) of the node. A node in an CSDF network specifies a *sequence* of the number of tokens it reads or writes on every invocation (firing) of the node. The sequence is repeated cyclically.

The coordination between the consumer and the producer of Figure 13 is strictly synchronous, which we will demonstrate by means of the topology or incidence matrix, Γ , of the network [17]. The rows of an incidence matrix Γ correspond to a channel, the columns correspond to a node. In SDF the non-zero entry $\gamma_{i,j}$ specifies the production or consumption of tokens ($\gamma_{i,j} > 0$ respectively $\gamma_{i,j} < 0$) on channel i when node j is invoked. Let q_ℓ be the state of the network (the number of tokens per channel) and ρ the schedule vector of which the entries ρ_j specify the number of invocations of the respective nodes. The right-hand side multiplication of Γ with ρ yields the state evolution: $q_{\ell+1} = q_\ell + \Gamma \rho$. Thus, the right-hand side null space ρ_o of Γ ($\Gamma \rho_o = 0$) equals the least-fixed-point cumulative schedule. That is, the entries ρ_j of ρ_o specify the cumulative number of invocation for each node j such that the network returns to its initial state. The right-hand side null space is a necessary condition for the consistency of the

network, however insufficient for guaranteeing the existence of a valid static schedule [16].

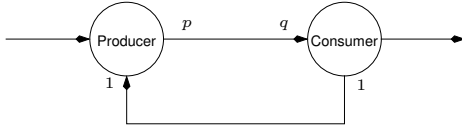


Fig. 13. SDF synchronous cooperative network.

The incidence matrix of the network of Figure 13 equals:

$$\begin{bmatrix} p & -q \\ -1 & 1 \end{bmatrix} \quad (11)$$

A non-trivial (one-dimensional) right-hand side null space exists only if Γ is singular, in fact it is known that $\text{rank}(\Gamma)$ equals the number of links in the network minus one. The network is consistent if the determinant of Γ is zero, i.e., $p = q$. Thus we have

$$\begin{bmatrix} p & -p \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \mathbf{0},$$

which specifies the cumulative schedule but a constant multiplication. This scheme implies synchronous coordination as p and q are tightly coupled.

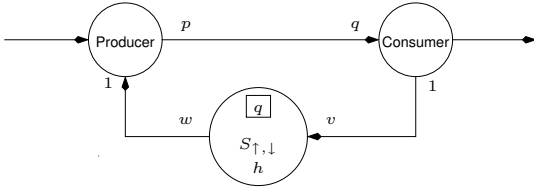


Fig. 14. SDF static asynchronous cooperative network.

In order to facilitate asynchronous coordination we statically decouple the producer and consumer with a sample rate converter scheme as depicted in Figure 14. The characteristic set of coefficients h defines the cyclo-static behaviour of $S_{\uparrow, \downarrow}$, turning the network into a CSDF network. The cyclo-static description of $S_{\uparrow, \downarrow}$ specifies at its input port a sequence of $|h|_0 \times 1$ and sequence h at its output port; the period of $S_{\uparrow, \downarrow}$ equals $|h|_0$. The producer and consumer have period equal to 1. The corresponding incidence matrix is constructed using the total number of samples per period [16], which corresponds to $v = |h|_0$ at the input port and $w = |h|_1$ at the output port. The incidence matrix is then as follows:

$$\begin{bmatrix} p & -q & \\ -1 & 1 & -v \\ & & w \end{bmatrix} \quad (12)$$

The right-hand-side null space of Γ is non-trivial if $pw = qv$. Let $r = \text{gcd}(p, q)$ and $w = k\frac{q}{r}$, $v = k\frac{p}{r}$ then we have

$$\begin{bmatrix} p & -q & \\ -1 & 1 & -k\frac{p}{r} \\ & & k\frac{q}{r} \end{bmatrix} \begin{bmatrix} k\frac{q}{r} \\ k\frac{p}{r} \\ 1 \end{bmatrix} = \mathbf{0},$$

which effectively decouples the coordination by a (parameterised) factor of $\frac{w}{v}$.

B. Explicit context dependence

To study the consequences of the context on the behaviour of a network we have to some extent interpret the execution details of the nodes of that network. In this section we consider the network of Figure 14 in some more detail, adding enough execution details for circumventing lengthy simulations for all kind of characteristic sets of coefficients.

We safely assume that the consumer supplies a control token to the producer to request a particular mode of operation of the producer. The consumer is obviously best served when the producer follows these requests seamlessly. However this is generally not the case, because of the differences in their relative rate at which the consumer supplies (q) and the producer acknowledges (p) mode requests and the characteristics of the sample rate converter in the feedback loop. With this interpretation we focus on the control. Consequently, the producer-consumer network equals an Interpolator-Decimator network with corresponding characteristic sets of coefficients: $g = \{p\}$ for the producer and $f = \{(0)^{q-1}, 1\}$ for the consumer. The entire network can be analysed through the composition $s = h \circ g \circ f$.

From Section III-C.2 we already know that the composition $g \circ f$ only depends on the fraction $\frac{q}{p}$. Without loss of generalisation we presume p and q to be co-prime: $\text{gcd}(p, q) = 1$. Since we consider a consistent network $\frac{q}{p} = \frac{w}{v}$ we also have a consistent closed network: $|s|_0 = |s|_1$. In the sequel we consider characteristic sets of coefficients for the sample rate converter where $|h|_0 = v = kp$ and $|h|_1 = w = kq$, for some $k \in \mathbb{N}^+$.

Suppose the feedback system implements a Decimator followed with an Interpolator: $h = \text{DecInt}(kp, kq)$. In this case there is an closed form of $s = h \circ g \circ f$:

$$\begin{aligned} s &= \{(0)^{kp-1}, 1\} \circ \{kq\} \circ \{p\} \circ \{(0)^{q-1}, 1\} \\ &= \{(0)^{kp-1}, 1\} \circ \{kpq\} \circ \{(0)^{q-1}, 1\} \\ &= \{(0)^{kp-1}, kpq\} \circ \{(0)^{q-1}, 1\} \\ &= \{(0)^{kp-1}, kpq\} \circ \{(0)^{q-1}, 1\}^{kp} \\ &= \{(0)^{kp-1}, kp\}, \end{aligned}$$

which depends on p and k only. For the control part this is the worst possible behaviour. During the entire period there is a delay of kp control tokens, but the last token.

Alternatively, suppose the feedback system implements an Interpolator followed with a Decimator: $h = \text{IntDec}(kq, kp)$. Here the k factor can be safely ignored since the characteristic set of coefficient of an Interpolator-Decimator network does not change with k . The closed form of s , in this case, involves two recursive sets. Since one $\text{IntDec}(p, q; u)$ distributes the number of output tokens evenly over the number of input tokens of the period of h , a composition of two such systems does something similar: The p available tokens are distributed in q parts over the p entries in s . From the perspective of the control this behaviour is preferred as delays are evenly distributed over the period.

Mathematically speaking we would define s optimum for $s = \{\alpha_0 \pm \delta_0, \dots, \alpha_{k-1} \pm \delta_{k-1}\}$, where $\alpha \rightarrow 1$ and $\delta \rightarrow 0$,

under the constraints that s meets the consistency requirement $|s|_0 = |s|_1 = kp$. The measure for the control lag (ϵ_i) for each token over the period of s , namely, equals

$$\epsilon_i = (i + 1) - \sum_{j=0}^i |\alpha_j \pm \delta_j|; \quad \text{for } 0 \leq i < kp$$

The average lag is minimised when the energy of s is uniformly distributed over the entire vector.

As a more practically oriented example consider an application in which the transport from producer to consumer uses a fixed sized FIFO buffer. Overloading of the buffer breaks the communication. A simple scheme to prevent overloading is the following: The producer labels its output data tokens with increasing numbers, the label for the next token is maintained in a counter x . The consumer acknowledges the proper receipt of data tokens, meaning that once every p tokens it returns the label of the most recently received data token. The producer maintains a “window” of W tokens. Before sending the next data token the coordination information is read (label λ). The producer subsequently will stop to emit data if: $x - \lambda \geq W$, i.e., when the control lags more than W samples, $\max \epsilon_i > W$.

For the static case of Figure 14 we can derive the conditions when the producer will stop operating. Consider the case that has an Interpolator-Decimator feedback network, thus $s = \{(0)^{kp-1}, kp\}$, hence $\max \epsilon_i = kp - 1$. In this case the system will not overload the FIFO if $kp < W$. Alternatively consider the case that has an Decimator-Interpolator feedback network. Here we have $\max \epsilon_i = \lceil \frac{p}{q} \rceil$.

C. TCP/IP example

The TCP/IP protocol is a practical example of the foregoing. The diagram of Figure 15 is a simplified CAPN model of the protocol with asynchronous communication and asynchronous coordination. A CAPN model of a complete TCP/IP transceiver is given in Figure 16. In TCP/IP, the consumer communicates a so-called “window” size. The producer obeys this context information to reduce packet loss due to a transport system overload. In case of TCP/IP, the context information relates to the degree of filling of the receive buffer.

Whether or not a devised protocol will operate appropriately depends on *when* the window-size context information is available and *when* this information is applied at the producer side. The exact ordering of these events depends on the relative schedule of consumer and producer. When oracles REG links (Figure 10) are applied, the synchronisation of the coordination is entirely determined by the properties of the respective control streams.

Properties of the control stream define the behaviour of the network, and vice versa. A specification of the control stream constrains the implementation. For instance we can simply extract the maximum number of discarded or replicated packets. Both values give rise to a control *delay*, which as we know from control theory may yield an instable system. On the other hand, given the producer-consumer processes, the maximum value of the delay can be calculated. Any proper implementation must guarantee a control stream that is within the bounds of the maximum delay, c.f. the factor kp above.

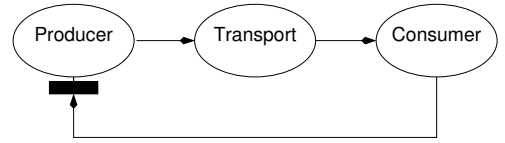


Fig. 15. IP CAPN model.

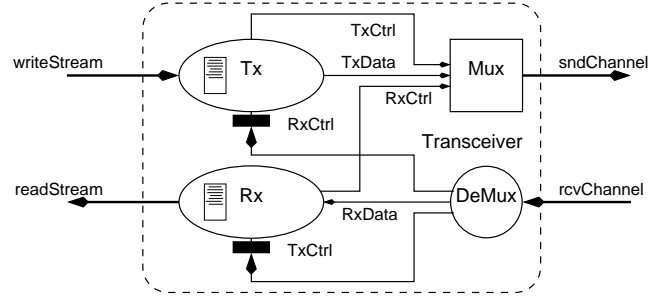


Fig. 16. TCP/IP CAPN model.

D. Related work

Indeterminate primitives such as the merge (or select) operator have been applied by other researchers. It is indeed a powerful modelling paradigm in modern design patterns for embedded systems. In [18], a select operation is proposed to model infrequent events, like a user pressing a button. The select construct is combined with the design space exploration methods of [12]. Lieverse et al. use an executable model in which there is a strict separation between the application domain (KPN) and the implementation domain (discrete events). When the methodology is extended with a select construct in the application domain, this strict separation can no longer be maintained.

The SPI workbench [19] evaluates possible schedules for heterogeneous systems. SPI implements a non-executable (non-functional) model. The modelling language, much like Kahn semantics, is a coordination language among components that are allowed to apply different MoCs. Components communicate with each other through FIFO buffers or plain registers. SPI establishes the bounds on the behaviour of the entire system. To do so, SPI requires an extensive abstraction of the communication, execution pattern, and resource utilisation of each component. SPI uses an explicit communication model, i.e. the availability of data is a predicate to enable process execution. Data dependent control is modelled stochastically with uncertainty intervals whereas CAPN hides data dependent control in the respective process nodes.

FunState [20] is an internal data representation developed in parallel with SPI. *FunState* distinctly separates control and data, whereas SPI, like CAPN, does not enforce this separation. *FunState* enables methods for formal verification (deadlock free, no overload, etc.) and methods for design and evaluation of schedules. Like SPI, *FunState* has limited support for asynchronous coordination.

In [21] the synchronous piggybacked dataflow model is introduced, which targets synchronous state update requests. Unlike our asynchronous coordination, the piggybacked dataflow model associates a mode change requests with a specific data

token. It extends the static dataflow model by having each token carry a pointer to a limited-access global state; per global state only one process may update the state but infinitely many processes are allowed to observe the state.

In [22] the parameterised dataflow model is introduced, which might yield an opportunity for conducting more dynamic analysis of our models. We considered static parameterised models whereas the parameterised dataflow model allows for dynamic parameter definition and parameter value reconfiguration.

V. CONCLUSION

In this paper, we have introduced context-aware process networks (CAPNS) as a modelling technique to analyse stream-based embedded systems that are subject to context changes. We have shown that these indeterminate CAPNS can be transformed to parameterised Kahn process networks (KPNS) by concentrating the context aware parts of the network in control streams. This transformation makes the otherwise indeterminate CAPNS compositional and traditional analysis techniques, such as fixed-point analysis, can subsequently be applied to the resulting parameterised network.

In the general case, analysis of CAPNS might require a numerical evaluation for all values of the context parameters in the network. However, for a specific example we have shown that a closed form solution can be obtained. Closed form analysis requires information about the functional behaviour of (part of) the network components. The example suggests that monotonicity of the functional behaviour is a strong requirement for obtaining closed solutions in CAPNS. Further research is needed to formalise this requirement.

Acknowledgement

The research reported here has been conducted in the Ubicom project www.ubicom.tudelft.nl, a multidisciplinary research project sponsored by Delft University of Technology. We thank Prof. Ed Deprettere of Leiden University for the many discussion that helped to improve and shape our work.

REFERENCES

- [1] G. Kahn, "The semantics of a simple language for parallel programming," in *Proceedings of the IFIP Congress 74*. North-Holland Publishing Co., 1974.
- [2] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation, and synthesis," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 366–390, Mar. 1997.
- [3] T. Stefanov, B. Kienhuis, and E. Deprettere, "Algorithmic transformation techniques for efficient exploration of alternative application instances," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES 2000)*, 2002, pp. 7–12.
- [4] J. R. Russell, "On oraclizable networks and Kahn's principle," in *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, 1990, pp. 320–8.
- [5] J. D. Brock and W. B. Ackermann, "Scenarios: A model of nondeterminate computation," in *Formalization of programming concepts*, 107th ed., ser. Lecture Notes in Computer Science, J. Diaz and I. Ramos, Eds. Springer, 1981, pp. 225–259.
- [6] P. Panangaden and E. Stark, "Computations, residuals and the power of indeterminacy," in *Proceedings of ICALP 1988*, ser. Lecture Notes in Computer Science 317, T. Lepisto and A. Salomaa, Eds., 1988, pp. 439–454.

- [7] J. R. Russell, "Full abstraction for nondeterministic dataflow networks," in *30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 170–5.
- [8] P. Panangaden and V. Shanbhogue, "The expressive power of indeterminate dataflow primitives," *Information and Computation*, vol. 98, no. 1, pp. 99–131, 1992.
- [9] P. Panangaden, "The expressive power of indeterminate primitives in asynchronous computation," in *Foundations of Software Technology and Theoretical Computer Science*, 1995, pp. 124–150.
- [10] P. Panangaden and V. Shanbhogue, "McCarthy's amb cannot implement fair merge," in *Proceeding of the 8th conference on foundations of software technology and theoretical computer science*, ser. Lecture Notes in Computer Science 338, K. Nori, Ed., 1988, pp. 348–363.
- [11] G. J. Holzmann, "The model checker SPIN," *Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.
- [12] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, "A methodology for architecture exploration of heterogeneous signal processing systems," *Journal of VLSI Signal Processing for Signal, Image and Video Technology*, vol. 29, no. 3, pp. 197–207, 2001.
- [13] A. Kienhuis, "Design space exploration of stream-based dataflow architecture Methods and Tools," Ph.D. dissertation, Delft University of Technology, Jan. 1999, ISBN 90-5326-029-3.
- [14] W. A. Najjar, E. A. Lee, and G. R. Gao, "Advances in the dataflow computational model," *Parallel Computing*, vol. 25, pp. 1907–1929, 1999.
- [15] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI Signal Processing Systems*, vol. 21, no. 2, pp. 151–166, June 1999.
- [16] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, February 1996.
- [17] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, sep 1987.
- [18] A. de Kock, G. Essink, W. Smits, P. van der Wolf, J.-Y. Brunel, W. Kruijtzter, P. Lieverse, and K. Vissers, "Yapi: Application modeling for signal processing systems," in *Proc. 37th Design Automation Conference (DAC'00)*, 2000, pp. 402–405.
- [19] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich, "SPI – a system model for heterogeneously specified embedded systems," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 4, pp. 379–389, 2002.
- [20] K. Strehl, L. Thiele, M. Gries, D. Ziegenbein, R. Ernst, and J. Teich, "Funstate—an internal design representation for codesign," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 524–44, 2001.
- [21] C. Park, J. Jung, and S. Ha, "Extended synchronous dataflow for efficient DSP system prototyping," *Design Automation for Embedded Systems*, vol. 6, pp. 295–322, 2002.
- [22] B. Bhattacharyya and S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2408–21, 2001.



Hylke W. van Dijk received an M.Sc. degree in 1993 (EE of Delft University of Technology (TU Delft)). After some years in industry he rejoined TU Delft in 1998 and became a system architect in the Ubicom research programme. Currently Hylke is a Post Doc in the Moose project. His interests concentrate on systematic system development, in particular on the coordination among system components and system processes in a multidisciplinary environment.



Henk J. Sips (A'78) was born in Amsterdam, The Netherlands, on October 14, 1950. He received the M.Sc. degree in 1976 in electrical engineering and the Ph.D. degree in 1984 from Delft University of Technology, Delft, The Netherlands. Currently he is an Professor in Computer Science at the Delft University of Technology. His research interests include parallel systems, distributed systems, mobile systems, and low power systems.