UNIFIED SPECIFICATION OF CONTROL AND DATA FLOW

Thorsten Grötker, Rainer Schoenen, Heinrich Meyr

Integrated Systems for Signal Processing
Aachen University of Technology
D-52056 Aachen, Templergraben 55, Germany
e-mail: {groetker,schoenen,meyr}@ert.rwth-aachen.de

ABSTRACT

Many signal processing systems use event driven mechanisms - typically based on finite state machines (FSMs) - to control the operation of computationally intensive (data flow) parts. The state machines in turn are often fueled by external inputs as well as by feedback from the signal processing portions of the system. Packet-based transmission systems are a good example for such a close interaction between data and control flow. For an efficient design flow it is of crucial importance to be able to model and analyze the complete functionality of the system within one single design environment. Therefore, we developed a computational model that integrates the specification of control and data flow by combining the notion of data flow graphs with event driven process activation.

1. INTRODUCTION

Signal processing systems typically consist of both control and data flow parts. The control flow often consists of *reactive control* that implements higher protocol levels and *flow control* that governs the operation of the data flow parts. The data flow parts are used to model the computationally intensive parts that are most efficiently described as operations on infinite streams of data samples.

Data flow systems [1] are described as networks of processes performing the signal processing and signals connecting the ports of those processes. *Static data flow* requires the processes to consume and produce fixed numbers of samples (called *rate*) at their ports upon each activation. *Dynamic data flow* denotes the case when data rates can be computed dynamically during runtime.

In a variety of cases control and data flow parts can neither be specified nor analyzed separately since they are interacting in a tight loop. We refer to these as *mixed control/data flow systems*. Packet-based transmission systems are an example for this class of applications. There, performance analysis (e.g. to measure the packet error rate) requires to simulate a sufficiently large number of packets - resulting in an even longer sequence of different processing states. Thus, one can neither replace the data flow part with a simplified (statistical) model nor can one assume the state of the control flow portions to be static.

Functional aspects to be modeled are

- the existence of multiple (processing-) states
- event driven state transition
- generation of events by control and data-flow subsystems
- while in state `X`: perform `F(X)`
- upon state transition `X` $\rightarrow$ `Y`: execute the procedure `P(X,Y)`
- generate data in state `X`, consume it in state `Y`

For a most efficient design process, the modeling technique should be sufficiently abstract to model the functionality independent of a specific implementation and the context of its use ($\rightarrow$ design reuse). In particular, it should neither prejudice hardware nor software implementations. Furthermore, it is necessary that the specifications result in fully determinate behaviour to be able to analyze functional models independent of a special implementation.

In the following section we will first review existing approaches to modeling and simulation of heterogeneous systems and describe their limitations. Next, we will introduce a new computational model named *process coordination calculus* (PCC). Then, we will present an example of the expressiveness of PCC specifications.

2. EXISTING APPROACHES

Basically, existing modeling techniques fall into three categories: those that are dedicated to handling data flow dominated systems, those that are best suited to model (reactive) control systems, and techniques that are based on low level implementation models. The latter do not fulfill our requirements with respect to modularity and abstraction.

The first class consists of blockdiagram based systems like SPW [3], COSSAP [3] or DSPStation [4], and functional or applicative data flow languages like Silage [5] or Lustre [6]. The bandwidth of these systems reaches from cycle based models to such capable of handling dynamic data flow.

In principle, all these systems can be used to implement control in the form of clocked finite state machines. The problem is that they do not support the notion of events. These have to be emulated by the use of data flow signals. Yet, the data flow signals have to be *consistent* [1], i.e. the data rates[1] of all signals connected to the FSM have to be adjusted by (controlled) decimation or interpolation so that exactly one value per FSM-activation is produced and consumed. This makes modeling a tedious and inefficient task and leads to poor implementations [7]. Furthermore, control oriented portions of a design become quite dependent on the context of their use so that they are sensitive to modifications of their environment - changing a single (dynamic) data rate can require major parts to be re-designed.

The lack of a notion of events leads to another problem. Consider the case depicted in figure 1. Two data flow processes (`A` and `C`) are connected to a FSM. The FSM is supposed to be activated whenever one of the data flow processes generates a data sample ('event') on α or γ. In data flow semantics the FSM can

- either be activated when `A` and `C` have generated a fixed and a priori known number of data samples or

[1] data rate = number of samples consumed (produced) per activation

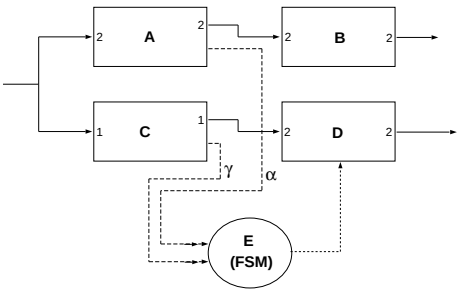

Figure 1. scheduling problem

- check the number of data samples available on α and γ to decide which data sample to consume.

The first case either disables an immediate activation of the FSM, i.e. E cannot be activated whenever only one sample is available, or ignores one event while waiting for the other. The second case - which is a variant of the so-called *nondeterministic merge* - is known to result in nondeterministic behaviour [8].

Even the Ptolemy system [9], that has been designed to deal with a heterogeneous mix of computational models for means of co-simulation, has to face this problem. In the Ptolemy environment different models of computation (*domains*[2]) can communicate with each other through so-called `Wormholes`. These embed another computational model appearing at the outside as processes (`Stars`) of the domain they are instantiated in. Executing an event driven simulation in the context of an existing data flow domain does not offer a solution since the embedded event driven simulation will have to appear as a data flow process to its environment. Thus, we still have to face the same problems as described above.

Textual [10] and graphical [11] languages based on finite automata exist for the specification, simulation and implementation of a more control-oriented type of application. Typical features include flowchart editors to specify procedures to be carried out upon a state transition and block-diagram editors that can be used to instantiate and interconnect cycle based models (including non-FSM type components). Specifying signal processing functionality with a cycle based technique proved to be quite inefficient when it comes to modeling multi rate dynamic data flow systems [12]. Furthermore, cycle based models strongly favour hardware implementations since they are not well suited for software code generation.

Codesign Finite State Machines (CFSMs) [13] offer a model that prejudices neither hardware nor software implementation. They target at optimized implementations of small reactive systems. The basic communication mechanism - event broadcast with non-blocking write onto one-place buffers - does not support the notion of data flow and makes the behaviour of a CFSM network nondeterministic.

Since none of the existing models of computation fulfills the requirements stated in section 1 we developed a computational model that allows to integrate the specification of control and data flow within one design environment. It combines the notion of data flow graphs with event driven process activation.

3. PROCESS COORDINATION CALCULUS

Computational models can often be described as hybrid specifications consisting of a *host language* and a *coordination language* [2]. The coordination language is used to describe a system as a network of functional units and defines the semantics of communication and process activation. The behaviour of the processes is described in the host language - typically some kind of imperative procedural language.

In this sense, the *process coordination calculus* (PCC) defines a coordination language. It does not make assumptions on how a specific functionality has been implemented - control-oriented processes may be directly implemented in a high level language or by the use of a protocol compiler or a state machine tool.

3.1. Primitive Elements

In contrast to existing modeling techniques that typically offer only one process type a PCC model consists of two types of processes: *data-driven* and *event-driven* processes. Both differ in their semantics of activation. Basically, an event-driven process must be activated immediately after an event connected to one of its inputs has occured. In contrast, a data-driven process can be activated when the signals connected to its input ports are filled with enough data samples to fulfill its (possibly data-dependent) activation condition.

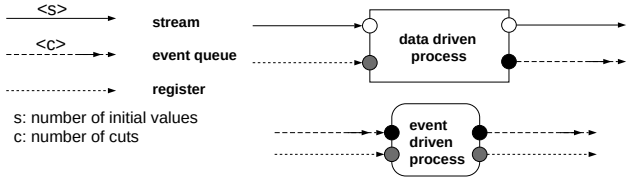

Figure 2. PCC primitives

The primitive elements of a PCC specification are depicted in figure 2. PCC supports three signal types. Two operations - *Read* and *Write* - are defined for any signal.

A *stream* is a FIFO buffer of limited but a priori unknown capacity. The number of initial values of a stream is indicated by an integer number (cp. fig. 2).

Event queues are used to transmit *events*. An *event* is a 2-tupel $(Value, Token)$ with $Token \in \{0, 1\}$. *Read*(*event*) returns $Value$ and resets $Token$ ($Token = 0$). *Write*(*event*) is valid if and only if $Token = 0$, i.e. an event must be processed before a new event can be created on the same signal line. *Write*(*event*) assigns to $Value$ and sets $Token = 1$.

A signal of type *register* is a one-place buffer; assignment operations overwrite the current content. To guarantee a determinate behaviour in the case of concurrent *Read* and *Write* operations two values are maintained for a register: an *actual value* and a *projected value*. *Read*(*register*) returns the actual value while *Write*(*register*) assigns to the projected value. The actual value is assigned the projected value before the next *Read* operation can take place.

Major goals in the design of PCC were to avoid overspecifications, to overcome the need to anticipate implementation level characteristics at the system level, and to keep functional models free from information on the context of their use. In order to achieve this, we pursue a special approach to introducing determinism:

$$\begin{aligned}\text{PCC specification} &= \text{process network } \mathcal{N} \\ &+ \text{scheduling constraints } \mathcal{R}\end{aligned}$$

3.2. Data Flow Model

To be able to define scheduling constraints we restrict the capabilities of data flow processes so that the existence of a bounded-length cyclic schedule (*complete cycle*), that returns a data flow subgraph into its original state (distribution of data samples), can be checked. Complete cycles of shortest length are called *minimum cycles*. An element $q_S(i)$ of the repetition vector $\vec{q}_S$ gives the maximum number of activations of the i-th data flow process within one minimum cycle. Static data flow [1] can be analyzed for the

[2] In the Ptolemy system the implementation of a domain is called `Universe`.

existence of bounded-length cyclic schedules. Yet, it imposes obstacles when modeling complete signal processing systems. PCC uses a more general data flow model called *bounded-rate multi-phase data flow* (BMD) that contains elements of models published by Buck [14], Bilsen [15], and Zepter [16]. For the sake of simplicity we restrict ourselves to single phase models for the remaining sections.

Using this model the stream-type ports of a data-driven process may either be *static* or *controlled*. While a static port has a constant data rate, the data rate of a controlled port depends on the values of the tokens of a *controlling port*. The controlling port has to be a static port (rate $r_c \geq 1$, fixed) of an integer data type. The controlled port is associated with a 3-tupel $<B, c, M>$, where B denotes an upper bound of the ports rate ($r \leq B$), c specifies the controlling port, and M is a *multiplicity* such that $r = \sum_{i=1}^{r_c} M * |c|_i = B * p_{\phi(c)}$ where $p_{\phi(c)}$ is a symbolical variable ($p_{\phi(c)} \in [0;1]$) and $\phi(c)$ specifies the signal connected to port c. $|c|_i$ denotes the i-th value at port c during one activation. The maximum value $|c|_i$ can assume is $|c|_{max} = \frac{B}{r_c * M}$. Given that Π_c denotes the set of all controlling ports in a PCC network, it is required that

$$\forall c_i, c_j \in \Pi_c : \phi(c_i) = \phi(c_j) = s \Leftrightarrow |c_i|_{max} = |c_j|_{max} := \hat{s}$$

Data flow analysis is necessary to compute the *consistency* [1] and liveness of the graph. For each stream signal s connecting two data driven processes ν_1 and ν_2 ($\nu_1 \xrightarrow{s} \nu_2$) we can write a *balance equation*

$$q(\nu_1) * r(\nu_1, s) = q(\nu_2) * r(\nu_2, s) \qquad (= n(s))$$

with $q(\nu)$ being the number of activations of ν per complete cycle ($n(s) = \frac{number\ of\ tokens}{complete\ cycle}$). In the case of controlled ports we write $r(\nu, s) = B_{\nu,s} * p_{\phi(c)}$. The graph is said to be *strongly consistent* if there exists a non-zero solution $\vec{q}(\vec{p})$ for this set of balance equations regardless of the value of the symbolical variables $\vec{p}$. Strong consistency implies that the graph will not have unbounded memory requirements due to unbalanced production and consumption of data. However, liveness and the existence of a bounded-length schedule cannot be inferred. Those can be guaranteed if the corresponding static graph (i.e. a graph were the rates of the controlled ports are set to $r(\nu, s) = B_{\nu,s}$) is free of deadlocks (cp. [1]) and an additional set of conditions is fullfilled. For all data flow processes ν where $q(\nu)$ is a function of symbolic variables $q(\nu) = f(p_{s_{\nu,1}}, \ldots, p_{s_{\nu,N}})$ the following has to be true.

$$\forall i \in \{1, \ldots, N\} : q(\nu) = K * \hat{s}_{\nu,i} * n(s_{\nu,i}),\ K \in I\!N$$

We refer to these as *control consistency conditions*. An example is presented in section 4.

In the context of PCC, the operation of a (BMD) data flow graph consists of repetitive executions of minimum cycles. Each data flow process ν_i is scheduled $q_S(i)$ times ($q_S(i) = q(\nu_i)|_{\vec{p}=\vec{1}}$) per minimum cycle; whether it is actually executed depends on the values of the controlling signals.

3.3. Scheduling Constraints

A central point in our approach to modeling control and data flow is the use of scheduling constraints to achieve determinate behaviour. These constraints can be represented by a binary relation $\mathcal{R}$ that defines a (partial) ordering of the $q_S(i)$ activations[3] of those data flow processes ν_i that interfere with event driven processes. This ordering decomposes the minimum cycle into N_a phases with $N_a = \frac{\#activations(event\ driven\ subgraph)}{minimum\ cycle}$. We call $\mathcal{R}$ *well formed* if the resulting PCC specification shows determinate behaviour.

Let N_D (N_E) be the set of all data flow processes ν_i (all event driven processes η_j) of the process network $\mathcal{N}$. $D_{i,j}$ ($j = 1...q_S(i)$) denotes the j-th activation of data flow process ν_i. The *activation set* $A_{\nu_i} = \{D_{i,1}, ..., D_{i,q_S(i)}\}$ comprises all activations of ν_i within one minimum cycle. $S_{D \leftrightarrow E}$ denotes the set of all data flow processes interfacing event driven processes. The corresponding activation set is $A_{D \leftrightarrow E} = \bigcup_{\forall \nu_i \in S_{D \leftrightarrow E}} A_{\nu_i}$.

Given these terms $\mathcal{R}$ can be defined as a binary relation $\mathcal{R} \subseteq A_{D \leftrightarrow E} \times A_{D \leftrightarrow E}$.

We write $x \preceq y$ ($x, y \in A_{D \leftrightarrow E}$) when $(x, y) \in \mathcal{R}$ and $x \npreceq y$ when $(x, y) \notin \mathcal{R}$.

A set of necessary conditions $\mathcal{R}$ must fulfill is that it has to be *reflexive* ($\forall a \in A_{D \leftrightarrow E} : a \preceq a$), *antisymmetric* ($\forall a, b \in A_{D \leftrightarrow E} : (a \preceq b \wedge b \preceq a \Rightarrow a = b)$), and *transitive* ($\forall a, b, c \in A_{D \leftrightarrow E} : (a \preceq b \wedge b \preceq c \Rightarrow a \preceq c)$).

Given that $\mathcal{R}$ was *linear* ($\forall a, b, c \in A_{D \leftrightarrow E} : (a \preceq b \vee b \preceq a)$) it would define a total ordering of the activations of the data flow processes interfering with event driven subgraphs. To make the behaviour of a PCC process network deterministic, we can apply weaker constraints presented in [17]

Before we look at an example, let us first consider the execution of event-driven processes. An event queue may be annotated an integer number of *cuts* (cp. fig. 2) with each cut delaying an event for the duration of one phase. Event driven subgraphs that do not have at least one cut in every directed cycle are rejected as ill-formed.

For execution, an event driven graph is made acyclic by cutting at the designated cuts. Then, the processes are sorted topologically in a *scheduling list*. Upon activation this list is used to check each process whether events have occured at its inputs. If so, the process will be activated. Events emitted in the same phase appear as *synchronous* at the inputs of an event driven process.

Consider a PCC specification of the example depicted in figure 1. The data rates (specified by the numbers at the data flow ports) yield a repetition vector $\vec{q}_S = (q_A, q_B, q_C, q_D)^T = (1, 1, 2, 1)^T$. Without any scheduling constraints the behavior is not determinate. Determinism can be introduced by simply stating *"execute A_1 prior to C_1"* with X_i denoting the i-th activation of process X. This leads to $N_a = 4$ and a schedule of the following shape:

$$\{..., \underbrace{A_1, B_1?, C_1, B_1?, C_2, B_1?, D_1, B_1?}_{minimum\ cycle}, ...\}$$

Thus, the scheduling constraints do not specify the phase in which B_1 is executed ("B_1?" indicates all possible positions within the minimum cycle).

A subset of $\mathcal{R}$ can be derived from the precedence constraints of the process network $\mathcal{N}$. In the example shown in figure 1 constraints like $C_1 \preceq D_1$ or $C_1 \preceq D_2$ result from those precedences. The remaining constraints that are needed to make $\mathcal{R}$ well constructed can be specified in various ways.

Tool support[4] for the specification of $\mathcal{R}$ will offer a graphical interface where the constraints take the form of edges that connect nodes reflecting the activations $D_{j,i} \in A_{D \leftrightarrow E}$. A template graph will be provided indicating the relationships to be established with undirected arcs. Additionally, one can select a specific scheduling policy to create necessary constraints in a way that throughput or memory requirements are optimized. Another option is the use of a

[3] within a minimal cycle (non-overlapped schedule)

[4] Currently, a code generator named *MOLIERE* that uses *C++* based PCC specifications is under development at our laboratory.

textual interface offering language primitives like `asap(`X`)`, `alap(`X`)`, and X_i `<=` Y_j that are evaluated sequentially.

The use of scheduling constraints results in a very efficient design style. Compared to modeling techniques that maintain a global notion of time - like cycle-based approaches or classical discrete event systems with a real-valued model of time - it is not required to anticipate details of an implementation that do not influence its behaviour. Instead, after specifying the flow of information using the intuitive notation of a process network one has to define at most the total ordering of those data flow processes that interfere with the event driven portions of the design (often even less [17]) to obtain determinate behaviour. Since the number of interfaces between control and data flow is typically small compared to the total number of processes in a complete system only very few constraints have to be specified.

Apart from resulting in a more efficient modeling technique, one gains access to a broader design space by making less assumptions on the total ordering of process activations. Furthermore, the notation of scheduling constraints proves generally helpful in the context of code generation. The duality of process network $\mathcal{N}$ and scheduling constraints $\mathcal{R}$ keeps the processes itself free from maintaining context dependend information (scheduling, relative processing speed). Thus, optimum reusability can be achieved.

4. EXAMPLE

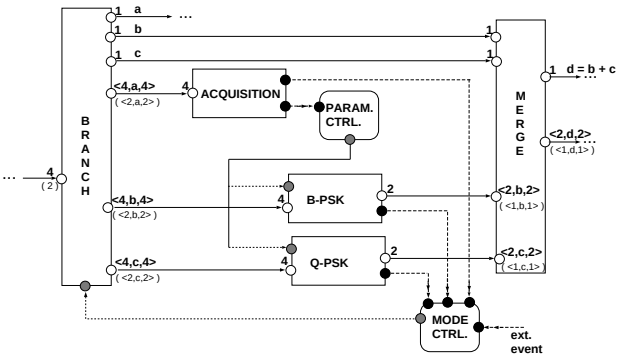

Figure 3. typical example

Figure 3 shows a structure often found in digital receivers. This scenario contains all functional elements described in section 1. The signal of type register written by the event-driven process `MODE_CTRL` is used by the data-driven process `BRANCH` to detect the state of the receiver. In an initial phase the data samples are directed to the `ACQUISITION` unit. The corresponding stream is controlled by the signal connected to the (static) output port a. When the acquisition phase is finished, `AQUISITION` emits two events. One is used to update parameters (e.g. an estimated frequency offset) that will be read in a following state. Depending on the value of the second event a state machine inside `MODE_CTRL` will switch to either `B-PSK` or `Q-PSK` mode. The corresponding process will then consume data samples received from `BRANCH` until either an event generated by this process or an external event will initiate a state transition.

In this example, data rates are indicated by numbers or 3-tupels at the respective ports. One might think of using the set of data rates specified in parentheses. Then, the data-flow graph would still be strongly consistent but the control consistency condition described in section 3.2 is violated. In this case, `BRANCH` could switch from `B-PSK` to `Q-PSK` after writing only two samples. While `MERGE` would still be waiting to read one sample from `B-PSK` the tokens produced by `Q-PSK` could not be consumed and would require unbounded memory to be queued.

5. CONCLUSION

The modeling technique presented in this paper combines the expressive power of data flow systems with the notion of event-driven control mechanisms. Since PCC is a *coordination language* it does not replace existing FSM or protocol based methods and tools. Instead, it offers the possibility to "plug in" reactive processes.

PCC supports an intuitive and modular design style. Since the processes do not need to maintain a notion of time, the reuse of functional models within another context and the exploration of HW/SW tradeoffs is greatly simplified. Determinism is introduced by means of scheduling constraints.

REFERENCES

[1] E. A. Lee, "Consistency in dataflow graphs," *IEEE Trans. on Parallel and Distr. Systems*, Apr. 1991.

[2] W.-T. Chang, A. Kalavade, and E. Lee, "Effective Heterogeneous Design and Co-Simulation," in *Proc. of NATO Advanced Study Institute on Hardware/Software Co-Design*, Kluwer Academic Publishers, 1995.

[3] Cadence Design Systems, 919 E. Hillsdale Blvd., Foster City, CA 94404, USA, *SPW User's Manual*.

[4] Mentor Graphics, 1001 Ridder Park Drive, San Jose, CA 95131, USA, *DSP Station User's Manual*.

[5] D. Genin, P. Hilfinger, J. Rabaey, C. Scheers, and H. De Man, "DSP specification using the Silage language," in *Proc. ICASSP'90*.

[6] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language Lustre," *Proceedings of the IEEE*, Sep. 1991.

[7] O. Mauss, M. Pankert, and H. Meyr, "Rapid prototyping of a wireless modem for low rate data communications," in *ITG Fachbericht 127, Mikroelektronik für die Informationstechnik*, Mar. 1994.

[8] E. Lee and T. Parks, "Dataflow process networks," *Proceedings of the IEEE*, 1995.

[9] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A platform for heterogenous simulation and prototyping," in *Proc. 1991 European Simulation Conf.*.

[10] F. Boussinot and R. de Simone, "The Esterel language," *Proceedings of the IEEE*, Sep. 1991.

[11] M. von der Beeck, "A comparison of statecharts variants," in *Proc. of Formal Techniques in Real Time and Fault Tolreant Systems*, 1994.

[12] H. Meyr, T. Grötker, and O. Mauss, "Concurrent HW/SW Design for Telecommunication Systems: A Product Development Perspective," in *Proc. of NATO Advanced Study Institute on Hardware/Software Co-Design*, Kluwer Academic Publishers, 1995.

[13] M. Chiodo, P. Giusto, A. Jurecska, H. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, "Hardware-software codesign of embedded systems," *IEEE Micro*, 1994

[14] J. Buck, "Static scheduling and code generation from dynamic dataflow graphs with integer-valued control streams," in *Proc. of 28th Asilomar Conference on Signals, Systems, and Computers*, 1994.

[15] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Static scheduling of multi-rate and cyclo-static DSP-applications," in *Proc. of IEEE Workshop on VLSI Signal Processing*, Oct. 1994.

[16] P. Zepter, T. Grötker, and H. Meyr, "Digital Receiver Design using VHDL Generation from Data Flow Graphs," in *Proc. 32nd Design Automation Conf.*, June 1995.

[17] T. Grötker, R. Schoenen, and H. Meyr, "PCC: A Modeling Technique for Mixed Control/Data FLow Systems," *Proceedings of the European Design and Test Conference*, 1997.