Combining Multiple Models of Computation for Scheduling and Allocation

D. Ziegenbein, R. Ernst, K. Richter*
TU Braunschweig

J. Teich, L. Thiele
ETH Zürich

Abstract

Many applications include a variety of functions from different domains. Therefore, they are best modeled with a combination of different modeling languages. For a sound design process and improved design space utilization, these different input models should be mapped to a common representation. In this paper, we present a common internal representation that integrates the aspects of several models of computation and is targeted to scheduling and allocation. The representation is explained using an example combining a classical process model as used in real-time operating systems (RTOS) with the synchronous data flow model (SDF).

1 Introduction

There are numerous system specification and modeling languages with fundamental differences in their underlying models of computation, such as event driven computation or data flow. Many complex designs use more than one modeling language to describe system functions of different characteristics. Since these functions are rarely completely independent, system simulation, verification and implementation must regard the combination of such different models of computation.

In this work, we present an internal representation which shall enable scheduling and allocation (or hardware/software partitioning, resp.[1]) of systems described with more than one model of computation. Fig. 1 shows the intended application of this representation. The different system parts may be modeled and optimized independently. Input level optimization uses domain specific techniques, such as e. g. transformations used in digital signal processing [6]. Then, the information useful for scheduling and allocation shall be extracted from the models and mapped to the common internal representation. After scheduling and allocation, the results shall be annotated back to the input level to support interactive design optimization. Fig. 1 shows that verification and simulation are treated as independent tasks with their own representation which takes some burden from the internal representation for scheduling and allocation.

This sketchy overview helps to derive the requirements for this internal representation. For illustration, we will use two examples of input languages. The first example is the process model underlying rate monotonic and deadline

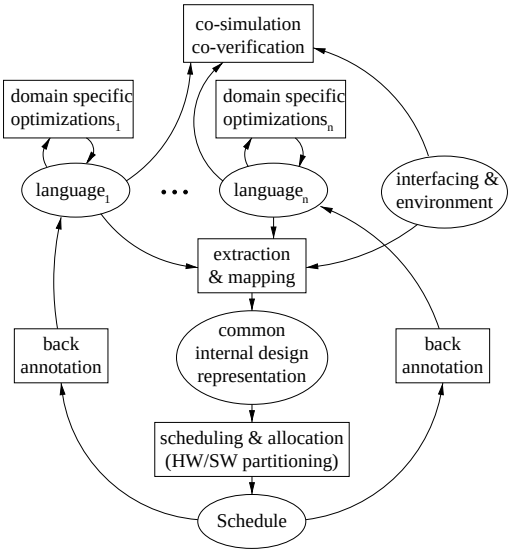

Figure 1: Intended Application

monotonic analysis and scheduling [4] (RMS/DMS) which is widely used in real-time system design. Processes are executed periodically with fixed rate constraints, with identical deadlines in each period. Latency times are limited by the process periods chosen by the designer. While the original model regards independent processes and single processors only, generalized RMS (GRMS) considers synchronization aspects which includes access to shared variables and multiprocessors (an overview of the numerous approaches is given in [5]). More recent work adds explicit communication between processes [9] to include communication scheduling and reduce performance requirements, or derives rate constraints automatically [1]. We want to include the original model, the model with communication and, finally a model with upper and lower bounds on communication latencies to account for *conditional communication* (communication depending on the result of evaluating an if-then-else construct) which is necessary to react to sporadic events and to implement data dependent behavior (examples: packet transfer depending on packet header, error message if broken sensor etc.). Alluding to its application to real-time operating systems we will call it the RTOS model.

The second example considers synchronous data flow graphs (SDF) which are used in digital signal processing [3]. In SDF, the number of data tokens produced and consumed per execution of a process is constant and fixed at compile-time (unconditional communication). The SDF design representation efficiently supports pipelining, retiming and buffer optimization but is restricted to model only static data flow.

*This part of the work was supported by the German DFG.

[1] Depending on the design context, hardware/software partitioning and allocation are related tasks which will not further be distinguished

It should be noted that SDF in its original form was a model of computation to represent concurrency and was later extended to support scheduling. The RTOS model, on the other hand, has originally only been defined as a basis for scheduling of independent tasks and was extended to support modeling of communicating processes. The two models can be used to illustrate the design information required for scheduling and allocation:

- *Execution time or I/O timing of processes*: This information is target architecture dependent and is typically obtained by system analysis and estimation. In general, execution times are given as time intervals due to data dependent process execution times. In both examples, I/O timing has clear semantics with all input data read at the beginning of a process execution and all output data written in the end.
- *Information on ready times and deadlines*: This information allows to determine the cost of resource sharing and the computing requirements. Here, we see major differences between both models. In SDF, a process is ready (can be executed) when all input buffers contain a sufficient number of data tokens. There is no explicit deadline, but deadlines can be derived from throughput and memory requirements using mobility or "urgency" [2] criteria which both depend on the target architecture rather than on the input description. In the RTOS model, a process is ready at the beginning of a period, and the deadline is explicitly defined. The two examples demonstrate that the input models provide the required information in different ways which must be unified in a common internal representation.
- *Information on the amount of communicated data*: There are again major differences. In SDF, communication is determinate and data independent. Buffering of communicated data is permitted to increase throughput while in the RTOS model, communication can be conditional and data buffering is typically not considered or even not permitted to control latency times.

The approach to a common internal representation is based on simple basic constructs which are enhanced by annotations which capture the details of the input model of computation. Each input language and its underlying model is mapped to a specific set of annotations. The main issue is the consistency of these annotations in order to allow scheduling and allocation across input language semantics. This paper will focus on the basic constructs and annotations for systems with a static set of processes, i. e. a set of processes that does not change at run time. Both example languages are of this kind. Systems with a dynamically changing set of processes require the concept of system states and system state transitions and will be the focus of a later paper as well as additional annotations for increasing scheduling efficency and capturing incomplete specifications using nondeterminism [7].

There are many scheduling techniques, i. e. preemptive or non preemptive scheduling, scheduling with dynamic or static priorities, event driven or periodic process execution. The scheduling technique is part of the design space. Rather than covering all these techniques, we have selected one to demonstrate the completeness of the internal representation with respect to the two input models.

2 The Internal Representation

This section describes the concepts of our common internal design representation. To demonstrate some concepts of the representation we use an example of a remote motor controller. The system collects message parts from a bus and tests them for an error (P_1), decodes the collected message (P_2) and sends a control word to the motor control loop (P_3). A description of the system, in which processes P_1 and P_2 are specified as periodic communicating processes (RTOS model) and process P_3 as an SDF process, is depicted in Fig. 2. Note that there is a maximum latency constraint $t_{lat,1}$ that constrains the time between the reception of a message part and the production of an error signal, i. e. the completion of process P_1.

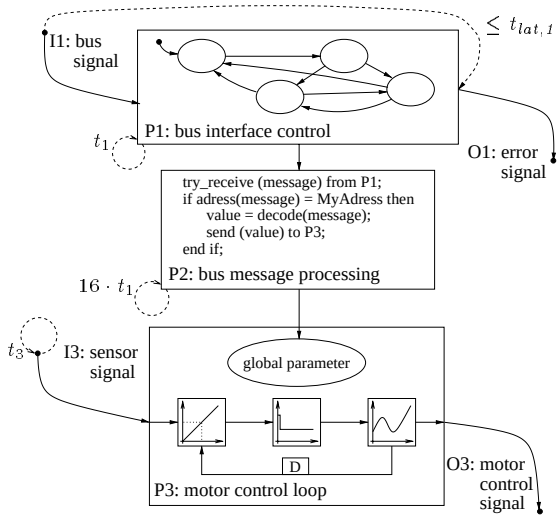

Figure 2: Remote Motor Controller (RTOS, SDF)

2.1 Basic Model

The basic model consists of processes that may have local data enabling them to have internal states. A side effect of allowing local data is that shared static data objects can be modeled by processes as well. The processes communicate with each other through unidirectional buffered channels that show a FIFO-like behavior. The basic model can be represented by a *model graph*.

Definition 1 ***(Model Graph)***
The model graph is a directed bipartite graph $G = (P, C, E)$ *where*

- P *denotes the set of process nodes,*
- C *denotes the set of channel nodes, and*
- $E \subseteq (P \times C) \cup (C \times P)$ *denotes the set of edges.*

Processes as well as channels are represented by nodes to enable refinement through hierarchical extension.

2.2 Execution Model

After defining the structure of the internal representation, we now introduce the underlying *process execution model* that

is based on activation by data availability, i. e. a process is activated if its required input data is present. The event-driven computation as well as the data flow model of computation are both based on activation by data availability since events can be considered as a special kind of data. But as we will show in section 2.4 other activation principles e. g. activation by periodicity constraints can be transformed into activation by data availability, too.

There are three points of time during the execution of one process instance that need to be distinguished:

- *activation time* t_{act}
  required input data is present; process gets activated
- *starting time* t_{start}
  resource is taken; input data is read; process starts execution
- *completion time* t_{comp}
  input data is consumed (i. e. destructed); output data is written; resource is released; process execution is completed

Note that the process execution's effect becomes visible on the channels as one atomic action at the end of execution.

Since communication may also consume time, the time a token is put on a channel (t_{comp} of writing process) need not equal the time this token is available for being read and, thus, for activating the succeeding process. Therefore, this time is defined as the *output availability time* t_{av}.

2.3 Annotations

The information required for scheduling and allocation is annotated to the corresponding graph elements. In our model, this information may be uncertain due to the following reasons:

- *Data dependent functionality*: A process may perform a different function and, thus, communicate a different amount of data (*conditional communication*) at each invocation depending on how it was activated, e. g. computing mode, error handling, etc.
- *Incomplete specification*: Apart from uncertainties caused by the environment, it may be desirable to specify non-determinism in certain cases on purpose.

Due to these facts, the annotated information need not to be constant but can be constrained by an upper and lower bound. Therefore, the annotations are modeled by (uncertainty) intervals. Stochastic processes $x(k)$ are introduced to capture the uncertain behavior. These stochastic processes are discrete regarding the execution index k and its domains are the (uncertainty) intervals of its information X. This indexing helps defining activation rules and allows easy transition to (partially) deterministic behavior in later design steps.

2.3.1 Communication

For communication scheduling and for the derivation of activation rules of processes (e. g. synchronous data flow [3]), the amount of data to be communicated between two processes has to be known. Therefore, a *data rate* denoting the number of data tokens communicated at a process execution is specified. Together with the data size of a token, the absolute communicated amount of data can be easily calculated.

Definition 2 ***(Data Rates)***
Let $Inputs(p) = \{c \in C \mid e = (c, p) \in E\}$ *denote the set of input channels of process* $p \in P$ *and* $Outputs(p) = \{c \in C \mid e = (p, c) \in E\}$ *denote the set of output channels of* p.

Associated with each process node $p \in P$ *and each input channel* $c \in Inputs(p)$*, there is an input data rate* $r_c(k)$ *that denotes the number of data tokens the process* p *consumes from the channel* c *at its* k*th execution. This rate* $r_c(k)$ *is constrained by an interval* $R_c = [r_{c,min}, r_{c,max}]$*, such that* $\forall k : r_c(k) \in R_c$*. Analogously for each output channel* $c \in Outputs(p)$*, there is an output data rate* $s_c(k)$ *and a constraining interval* $S_c = [s_{c,min}, s_{c,max}]$.

Furthermore, we need to define some constructs to keep track of the tokens and to model the availability of data.

Definition 3 ***(Data on Channels)***
Associated with each channel $c \in C$*, there are the numbers* $\hat{d}_c$*,* d_c*, and* $d_{c,av}$ *where*

- $\hat{d}_c$ *denotes the initial number of data tokens,*
- d_c *denotes the total number of data tokens at a given point of time,*
- $d_{c,av}$ *denotes the number of data tokens available for activation of the succeeding process at a given point of time*

on channel c*. Note that all three numbers may be uncertain.*

2.3.2 Timing

Scheduling and allocation require the modeling of latency times of processes and communication channels. Usually, they are gathered by a timing analysis tool (e. g. [8]) or estimated.

Definition 4 ***(Latency Times)***
Associated with each process $p \in P$*, there is a latency time* $lat_p(k) \in Lat_p = [lat_{p,min}, lat_{p,max}]$ *where* $lat_{p,min}$ *[*$lat_{p,max}$*] denotes the lower [upper] bound on the execution time (*$t_{comp,p}(k) - t_{start,p}(k)$*) of instance* k *of process* p.

Analogously, associated with each channel $c \in C$*, there is a latency time* $lat_c(k) \in Lat_c = [lat_{c,min}, lat_{c,max}]$ *that limits the communication time (*$t_{av,p}(k) - t_{comp,p}(k)$ *where* p *writes on channel* c*) for a token on channel* c.

Note that latency times are resource dependent. Therefore, uncertain latencies denote upper and lower timing bounds for any (remaining) feasible mapping of processes and channels to possible resources. During scheduling and allocation, these uncertainties are (gradually) reduced by mapping decisions.

Since we do not restrict the communication to a single behavior, the latency time for channels depends also on the chosen method of communication (e. g. burst or packet transmission) and on the amount of data to be communicated. A detailed communication modeling is supported by enabling a hierarchical refinement of a channel.

2.3.3 Virtual Components

For modeling purposes, we introduce the concept of *virtuality* for processes and channels. These do not have to be implemented. Their importance will be understood from later examples.

Definition 5 ***(Virtuality)***
Associated with each process $p \in P$ and each channel $c \in C$, there is a virtuality flag $v \in \{true, false\}$ which denotes the fact whether the process or channel is part of the system to be implemented ($v := false$) or has been introduced for modeling purposes only ($v := true$).

As can be seen in Fig. 3, virtual graph elements are visualized by dotted lines. Note that virtual processes and channels are mapped to dummy resources.

2.4 Activation

An activation rule determines when each process is ready for execution and can be scheduled. As mentioned earlier, our model is based on activation by data availability.

Definition 6 ***(Activation Rule)***
Associated with each process $p \in P$ and each activation k, there is an activation rule

$$A_p(k) = \bigvee_{j=1 \ldots j_{max}(p)} a_j(p,k)$$

that enables the kth activation of p if and only if one or more of $j_{max}(p)$ activation patterns

$$a_j(p,k) = \bigwedge_{c \in Inputs(p)} (d_{c,av} \geq v_j(c,k))$$

is true. Each activation pattern secures that there are enough available data tokens $d_{c,av}$ on each input channel $c \in Inputs(p)$ for a set of possible combinations of data consumptions $r_c(k) \in [r_{c,min}, v_j(c,k)]$ with $v_j(c,k) \in R_c$.

Evidently, the chosen activation rule is valid for data or event driven models of computation. In the following, we will show how other activation principles like periodic activation can be mapped onto the chosen activation rule using virtual processes and channels.

Periodic activation can be modeled by a virtual channel c_v starting and ending at the process to be activated. The process has a static consumption and production rate of one data token per execution for channel c_v. With one initial data token on the channel ($\hat{d}_{c_v} := 1$) supporting the first activation, each execution now enables its following activation. The time between two consecutive executions can be constrained by latency constraints (to be introduced in section 2.7).

Another possible activation principle is the *activation by relative execution rates* (e. g. RTOS semantics: no exact periodicity but constrained mobility intervals [4]). An example for this is process P_2 in Fig. 3 which has to be executed once during every period $16 \cdot t_1$ and, thus, once during 16 executions of P_{time}. This is modeled by two virtual channels (C_9 and C_{10}) with preassigned tokens between both processes. Thus, with its first execution P_{time} enables one activation of P_2 whose execution enables another 16 activations of P_{time} that lead to another activation of P_2 etc.

2.5 Update Rules

Based on the chosen activation rule, *update rules* can be specified to formally define the semantics of our representation.

Definition 7 ***(Update Rules)***
Initially, $d_c := \hat{d}_c \forall c \in C$ and $N_p := 0 \forall p \in P$ holds. The value of N_p at a certain time denotes that process $p \in P$ has completed already N_p activations.

1. *A process $p \in P$ becomes activated (for the $(N_p + 1)$st time) at a certain time, if $A_p(N_p + 1)$ becomes true at that time.*
2. *A process $p \in P$ may start execution, if it is activated.*
3. *If process $p \in P$ starts execution at time τ, then p completes at $\tau + lat_p(N_p + 1)$ and the following update rules are executed:*
   - $N_p := N_p + 1$;
   - $t_{comp,p}(N_p) := \tau + lat_p(N_p)$ *where $t_{comp,p}(N_p)$ is the completion time of the N_pth activation of p.*
   - *One of the activation patterns with $a_j(p, N_p) = true$ is selected. If there are several true activation patterns and thus several possible data consumptions, one is chosen randomly. Let the corresponding index be denoted $\hat{j}$. Then we have $r_c(N_p) \in [r_{c,min}, v_{\hat{j}}(c,k)]$.*
   - $\forall c \in Inputs(p) : d_c := d_c - r_c(N_p)$
   - $\forall c \in Outputs(p) : d_c := d_c + s_c(N_p)$

   These rules are executed sequentially, but the whole update rule is considered as an atomic action.
4. *At time $t_{comp,p}(N_p) + lat_c(N_p)$ the communication of the written data on channel $c \in Outputs(p)$ completes and $d_{c,av} := d_{c,av} + s_c(N_p)$.*

2.6 Environmental Modeling

The modeling of embedded systems always has to include modeling of the environment and its data sources and sinks in particular. The sources as well as the sinks can be modeled by virtual processes. The communication between those environmental processes and system processes has to be implemented. Therefore, the channels representing this communication are not virtual. Examples for environmental processes are the source P_{sensor} or the sink P_{motor} in Fig. 3.

2.7 Constraints

So far, the notation and the semantics of the internal representation have been introduced. It remains to explain how to model *constraints* that have to be fulfilled by all legal schedules. In this paper, we focus on timing constraints as the most important type of constraints for scheduling. The extension of our model to feature other constraints is simple but may require the addition of corresponding information to all processes (e. g. power consumption).

Conditional communication and activation based on communicated data leads to conditional time constraints if constraining the time difference between two process executions.

Therefore, we associate timing constraints with production and consumption times of data tokens. Since due to the atomic process execution the time of consumption equals the time of process completion, this association is valid.

Since our representation features hierarchy, we need timing constraints over process chains, but for the sake of simplicity in this paper, we restrict ourselves to timing constraints for tokens that are produced and consumed by adjacent process nodes.

Definition 8 ***(Latency Constraint)***
Associated with each channel $c \in C$, there may be an interval $LC = [t_{lat,min}, t_{lat,max}]$ that denotes a latency constraint that limits the difference between the production time t_{prod} and the consumption time t_{cons} for all tokens on channel c:

$$\forall b \in \{data\ tokens\ on\ channel\ c\}:$$
$$t_{lat,min} \leq (t_{cons,b} - t_{prod,b}) \leq t_{lat,max}.$$

An example how a latency constraint limits the lifetime of tokens on a channel can be seen at channel C_{11} in Fig. 3). Other timing constraints like *correlation and rate constraints* or even more complicated timing behavior like sporadic bursts of external events can be modeled using latency constraints and virtual channels and processes. For example, rate constraints can be modeled by a virtual channel from the process to be constrained to itself with $\hat{d}_c = 1$, data consumption and production rates of 1 and a latency constraint $[t_{rate}, t_{rate}]$ (e. g. P_{sensor} in Fig. 3).

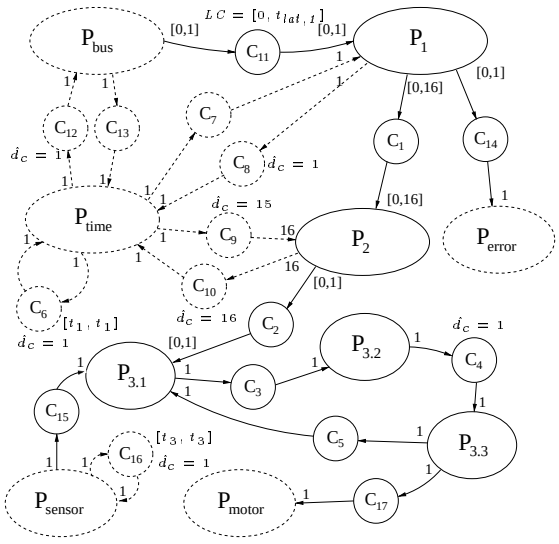

Figure 3: Remote Motor Controller (our representation)

3 Example

One of several possible mappings of the remote motor controller to our internal representation is depicted in Fig. 3. Note that the SDF in process P_3 is mapped to three processes ($P_{3.1}$, $P_{3.2}$ and $P_{3.3}$) while the state machine in P_1 is mapped to a single process (designer's decision).

A Gantt diagram for a static schedule and $t_3 = 8 \cdot t_1$ is shown in Fig. 4. Only the resources for processes are shown while channel and dummy resources are omitted.

4 Conclusion

We presented a common internal design representation that integrates different models of computation regarding scheduling and allocation. The example shows the capabilities of our approach.

Only the basic concepts of our model have been introduced. Other constructs to deal with system states, incomplete specification and memory allocation will be presented later or can be previewed in [7].

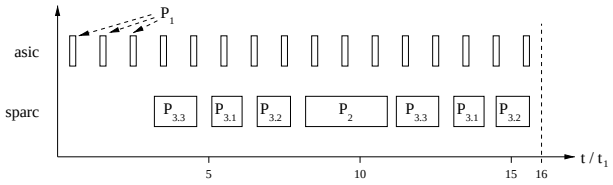

Figure 4: Gantt Diagram of Remote Motor Controller

References

[1] A. Dasdan, A. Mathur, and R. K. Gupta. RATAN: A tool for rate analysis and rate constraint debugging for embedded systems. In *Proceedings ED&TC '97*, pages 2–6, February 1997.

[2] A. Kalavade and E. A. Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *Proceedings Codes/CASHE '94*, pages 42–49, 1994.

[3] E. A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), January 1987.

[4] C. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard-real-time environment. *Journal of the ACM*, pages 46–61, 1973.

[5] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, January 1994.

[6] S. D. Stearns. *Digital Signal Analysis*. Hayden Book Company, New Jersey, 1975.

[7] L. Thiele, J. Teich, and D. Ziegenbein. Funstate - functions driven by state machines. Technical Report 33, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zürich, January 1998.

[8] W. Ye and R. Ernst. Embedded program timing analysis based on path clustering and architecture classification. In *Proceedings ICCAD '97*, San Jose, USA, 1997.

[9] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *Proceedings ICCD '95*, pages 64–69, October 1995.