SPI - An Internal Representation for Heterogeneously Specified Embedded Systems

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

L. Thiele
ETH Zürich

J. Teich
Uni-GH Paderborn

Abstract

The specification of embedded systems often contains several different specification or modeling languages with fundamental differences in their underlying models of computation. In this paper, we present the SPI model, a common internal representation that integrates the aspects of several models of computation and is targeted to scheduling and allocation as well as performance validation. The model and its applications are explained using an example combining a periodic process model as used in real-time operating systems (RTOS) with the synchronous data flow model (SDF).

1 Introduction

Many applications of embedded systems include functions from different domains. Each of these functions is best modeled with a specification language that matches the application domain of the function, i. e. transformative functions are usually modeled using data flow languages whereas reactive control functions are modeled with event driven models of computation. Therefore, specifications of embedded systems often are heterogeneous with regard to the mixture of different modeling and specification languages.

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

Since the functions in an embedded system are rarely completely independent, system synthesis must regard the combination of such different models of computation. For a coherent design space exploration in the context of scheduling and allocation, a common representation of the heterogeneously modeled system is needed. This representation may also serve as a starting point for the joint validation of timing properties which is needed for safe integration of the differently modeled system parts.

In this paper, we present such an internal design representation, the SPI (**S**ystem **P**roperty **I**ntervals) model. It consists of communicating processes that are not characterized by their detailed behavior but by an abstracted behavior that is described using parameter intervals. These parameters represent only aspects that are relevant for the target applications (scheduling, allocation and performance validation) and include communication behavior, activation rules and latency times for all possible resources. The values of the parameters need not to be constant but are constrained by intervals. This allows abstraction of conditional behavior and enables estimation for system parts in early design stages. Using the concept of process modes, conditional process behavior depending on internal states or consumed data can also be modeled explicitly. The environment can be modeled coherently using virtual model elements that have the same properties as the model elements representing the embedded system.

A representative for ongoing research on modeling of heterogeneously specified systems is the *chart approach [4] that is part of the PTOLEMY project [2]. Contrary to our approach, they combine existing models of computation like data flow models [5] and state machines on different hierarchical levels, i. e. state transitions can activate data flow graphs on a lower hierarchical level and actors in data flow graphs can be refined by state machines. However, the restriction that state transitions in data flow actors can occur only after a complete iteration of the whole data flow graph violates the principle of modularity which is very important for scheduling and allocation.

Another approach is the generalized task graph model GTG [3] that combines and borrows concepts from several existing task models. Similar to SPI, also non-deterministic communication behavior and timing intervals are possible. However, since its main application is rate derivation and timing analysis there are some application-specific restrictions and the concepts are defined not generally but from an application-specific point of view.

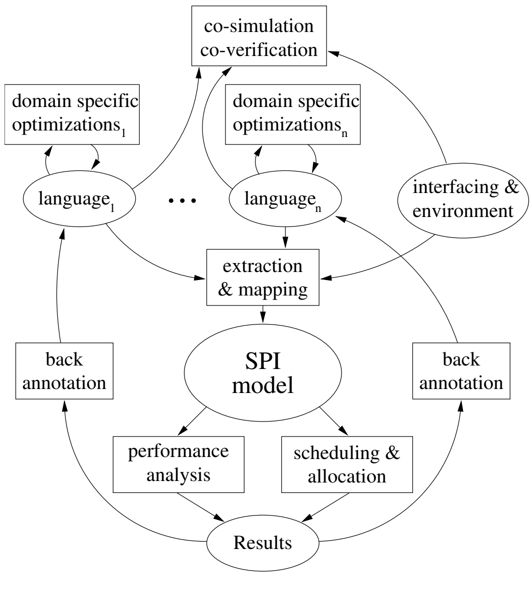

Figure 1: Intended Application

A flow diagram representing the intended application of the SPI model is depicted in Fig. 1. The different system parts may be modeled and optimized independently thereby taking advantage of already existing design environments and domain specific optimization techniques e. g. transformations used in digital signal processing [9]. Then, the aspects relevant for scheduling, allocation and performance validation are extracted and mapped to the SPI model. The different system parts are connected according to the given interfacing information yielding a coherent modelling of the embedded system and its environment. The results of the applied methods are annotated back to the input level to support interactive design optimization. As can be seen from Fig. 1, co-simulation and functional co-verification are considered as independent tasks since they require a semantic interfacing between the different system parts which is not part of SPI due to its abstraction.

The remainder of the paper is structured as follows. In Section 2 the basic concepts of the SPI model are introduced and discussed. This is followed by two application examples in Section 3. The paper is concluded with an outlook on future work.

2 The SPI Model

In this section, the concepts of the SPI model (**S**ystem **P**roperty **I**ntervals) [10, 11] are introduced. An example system will be used throughout the rest of the paper to demonstrate the features of the SPI model and the advantages of using a common internal representation.

This example system is a remote motor controller which is specified as depicted in Fig. 2. 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 process P_1 is specified in a state-based language, process

P_2 in a C derivative and process P_3 in a static data flow language, is depicted in Fig. 2. The interaction of the processes and the environment is loosely defined. P_1 and P_2 are both periodic processes in the sense of the process model underlying rate monotonic scheduling and analysis [6] while P_3 is driven by a periodic input. There is a timing constraint that limits the maximum response time to an erroneously received message part.

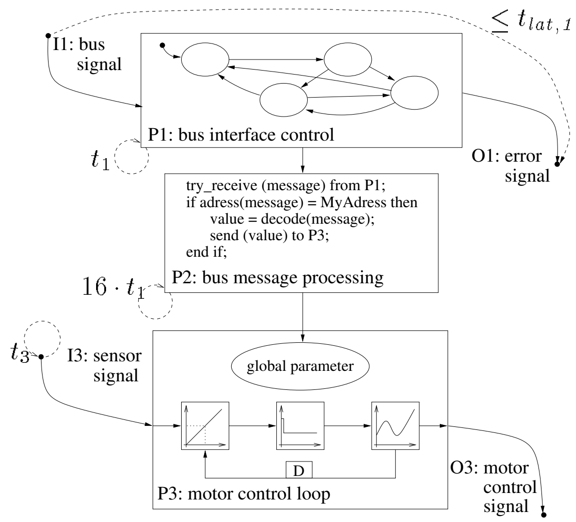

Figure 2: Remote Motor Controller

2.1 Basic Model

The SPI model consists of processes communicating through unidirectional, point-to-point channels. The channels are of two types, either FIFO-ordered queues (destructive read) or registers (destructive write). All communication between processes has to be explicitly modeled in terms of channels. 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,*
- $C = Q \cup R$ *denotes the set of channel nodes with Q and R being the sets of queues and registers, respectively,*
- $E \subseteq (P \times C) \cup (C \times P)$ *denotes the set of edges, and*
- *for each $c \in C$, $indeg(c) = outdeg(c) = 1$.*

Processes and channels are defined as nodes to enable refinement by hierarchical extension.

2.2 Execution Model

The execution of a process is based on activation by data availability, i. e. a process is activated if its required input data is present and may start if it is activated. In section 2.5 and in [10], it is shown how other activation principles e. g. periodic activation can be modeled by activation by data availability. The following three points in time give an informal definition of the process execution model:

- *activation time t_{act}*
  required input data is present; process is 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. destroyed if $C_{in} \in Q$); 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

For scheduling, allocation and performance analysis, knowledge about the detailed functionality of a process is not needed. It is sufficient to know for each process the resource requirements and the timed interaction with its environment. In this context, key properties are the data rates and the latency times that are introduced below.

The properties of the processes and channels are defined by parameters annotated to the corresponding graph elements. This allows an easy adaption of the model to include all required information for a certain optimization goal or task in the design flow.

The parameters need not be fixed but can be specified using uncertainty intervals, i. e. they are constrained by an upper and lower bound. The sources for this non-determinism can be data dependent functionality of a process (e. g. due to `if-then-else` structures) or an incomplete specification. The ability to specify intervals also enables estimation of system parts in early design stages and the integration of legacy code in the limitations of its formal analyzability.

For communication scheduling and the derivation of activation rules for processes, the communicated amount of data has to be known. Therefore, *data rates* denoting the number of data tokens communicated over certain channels at each process execution are specified. Together with the data size of a token, the absolute amount of communicated 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 kth 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 $S_c = [s_{c,min}, s_{c,max}]$.

Note that the interval representation allows processes to have non-constant data rates. Contrary to existing data flow models (e.g., synchronous data flow [5]) where data rates are fixed and time-invariant, our representation allows for the modeling of non-determinism to abstractly capture conditional behavior.

Using process P_2 in Fig. 3, the meaning of data rate intervals can be easily explained. At each execution, P_2 consumes 16 tokens containing a message from channel R_2. If the message is for this controller, P_2 decodes the message and produces one token containing a control word on channel R_4. Otherwise, no control word and, thus, no token is written to R_4. Therefore, P_2's data rate interval for the production on channel R_4 is $[0, 1]$.

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 [1]*

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

[1] Note that $d_{c,av} \leq d_c$ since $d_{c,av}$ is the already communicated part of the total amount of data d_c. If no latency is associated with the channel then $d_{c,av} = d_c$.

Besides the communication behavior captured by the above constructs, the resource requirements are of central importance. In the SPI model, they are captured by latency times denoting the time a resource is taken for a process execution.

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 bounds 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.

Another annotation for processes and channels is the *virtuality flag* denoting the fact whether a model element is part of the system to be implemented or has been introduced for modeling purposes. Virtual model elements do not have to be implemented but are used to visualize information like rate constraints (see Section 2.5) or relative execution rates. As will be seen from the following examples, virtual model elements are denoted by dotted lines.

2.4 Process Modes

With the above presented annotations, a process in the SPI model has a single behavior that is obtained by abstraction of possibly several different execution paths in the original process and represented using non-determinism bounded by uncertainty intervals. In some cases, abstraction may lead to too imprecise modelings with not enough detail for efficient scheduling or successful validation. Therefore, the concept of process modes is introduced to be able to explicitly model different process behaviors.

Definition 5 ***(Process Modes)***
A mode m_p of a process $p \in P$ is a tuple of a latency time interval $Lat_p = [lat_{p,min}, lat_{p,max}]$, an input data rate interval $R_c = [r_{c,min}, r_{c,max}]$ for each of its input channels $c \in Inputs(p)$ and an output data rate interval $S_c = [s_{c,min}, s_{c,max}]$ for each of its output channels $c \in Outputs(p)$.

Associated with each process $p \in P$, there is a non-empty, finite set of process modes $M_p = \{m_{p,1}, \ldots, m_{p,n_p}\}$ where $n_p \in \mathbb{N}$ is the number of process modes $m_{p,i}$.

A process mode describes a subset of the possible process behaviors, i. e. a mode can be obtained for a subset of execution paths just like the single process behavior is obtained for the set of all possible execution paths of a process.

An example for a process with different possible execution paths is the bus interface control process of the example system in Fig. 2. Assuming this process is to be modeled using two modes, the different execution paths can be grouped as follows. The process either reads an erroneous message part from the bus and outputs an error message (mode m_1) or reads a correctly received message part and in case of that part being the last of a telegram writes the whole telegram to the decoder (mode m_2). Then,

$$m_1 = ([0.6, 0.9]t_1, 1, 1, 0, 1, 1)$$

$$m_2 = ([0.7, 1]t_1, 1, 1, [0, 16], 0, 1)$$

are the process modes describing process P_1[2] in the SPI model of Fig. 3. Throughout the example, the process latencies are expressed as

[2]The data rates are in rising order with respect to their corresponding channel indices (within the groups of input and output channels).

fractures or multiples of t_1, the period of the incoming bus signals.

Usually (and in the above example), a process adapts its behavior, i. e. selects its mode, based on the content of its input data. Since the data content is not yet exposed by the previous set of annotations, *mode tags* modeling content information may be associated with the data tokens.

The production of mode tags is modeled by *mode tag production rules* that are associated with the processes producing the data containing the information represented by the mode tags.

Definition 6 ***(Production of Mode Tags)***
Associated with each process $p \in P$, there is a finite set of mode tag production rules *TP_p. Each tag production rule is a mapping $tp_p : I_p \mapsto O_p$ where I_p is the set of input tag patterns and O_p the set of output tag production patterns.*

An input tag pattern is a tuple with one entry for each incoming channel where each entry is a boolean expression about the association of certain mode tags with the first token on that channel. Each input tag pattern $i \in I_p$ maps to an output tag production pattern $o \in O_p$ that denotes the mode tags produced on each outgoing channel if i is matched. The entire tag production for the current execution is the union of all output tags produced by the individual matched production rules. The produced tags are associated with all tokens produced on the corresponding channels at the current execution.

Note that the production of mode tags like the production of data tokens may be mode-dependent. Thus, the definition of process modes m_p has to be extended to include mode tag production rules. The following example demonstrates the use of tag production rules.

Process P_{bus} in Fig. 3 models the environmental production of bus telegrams that may be correct or faulty. This fact is denoted by a mode tag $'e'$ that is attached to all erroneous bus telegrams. Then, process P_{bus} has the following two modes representing the cases of producing a correct (m_1) or a faulty telegram (m_2).

$$m_1 = (0, 1, 1, 1)$$
$$m_2 = (0, 1, 1, 1, < * > \mapsto < e, * >)$$

Since P_{bus} is a source process the input tag pattern is empty and thus true. Therefore, if P_{bus} executes in mode m_2 a token with the tag $'e'$ attached is produced on R_1. Nothing can be said about the selection of the mode since there is no incoming data for P_{bus}. For the modeling of a certain error rate, a probability for each mode could be annotated but this is currently not part of the SPI model.

The use of mode tags for process mode selection is part of the activation function that is described in the next section.

2.5 Activation Function

An *activation function* determines when a process is ready for execution i. e. is activated and in which modes a process can be executed.

Definition 7 ***(Activation function)***
Associated with each process $p \in P$, there is an activation function *that may be formulated as a finite set of rules σ where each rule is a mapping of a predicate to a finite set of modes $M_\sigma \in M_p$.*

The predicate of a rule σ is a function on the numbers of available tokens $c.num$ and on the tag sets $c.tag$ of the first available token[3] on some input channels $c \in Inputs(p)$. The value of the predicate is either 'true' or 'false'. The union of all sets M_σ with the predicate of rule σ being 'true' is the set of modes the process can execute in. If and only if this set is not the empty set, the process is activated.

[3]The restriction to consider only the first token on each channel is legitimate due to the nature of mode tags. The consumption of two contradicting tags from the same channel during the same instantiation can be compared to the irrelevant case of reading a packet with two contradicting headers.

Note that the predicate of a rule σ has to be 'false' if in some input queue c there is a smaller number of available tokens than possibly consumed by the process in a mode $m \in M_\sigma$. Otherwise, the execution of this process may lead to an attempted consumption of not available data. Usually, the set of possible modes for the current execution has just one element, otherwise the mode the process is executed in is chosen non-deterministically from this set.

Evidently, this activation function is capable of representing data flow or event driven models of computation. In the following, it will be shown how other activation principles can be modeled using this activation function.

Periodic activation (e. g. of process P_{time} neglecting the input channels Q_8, Q_10 and Q_13) can be modeled by a virtual channel Q_{11} 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 Q_{11} and, thus, an activation function consisting of a single rule $Q_{11}.num \geq 1 \mapsto m_1$ mapping to the only mode of P_{time}. With one initial data token on the channel ($\hat{d}_{Q_{11}} := 1$) 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 2.6).

Another possible activation principle is the *activation by relative execution rates* (e. g. RTOS semantics: no exact periodicity but constrained mobility intervals [6]). An example for this is process P_1 in Fig. 3 which has to be executed once during every period t_1. To model this fact, the virtual process P_{time} with an exact period of t_1 and two virtual channels Q_7 and Q_8 with a preassigned token are introduced. With the modes of P_1 (as defined in Section 2.4), the activation function is as follows:

$$(Q_7.num \geq 1) \wedge ('e' \in R_1.tag) \mapsto m_1$$

$$(Q_7.num \geq 1) \wedge ('e' \notin R_1.tag) \mapsto m_2$$

Then, with its first execution P_{time} enables one activation of P_1 whose execution in turn leads to another activation of P_{time} etc. The chosen mode depends on the presence of the tag $'e'$ denoting a transmission error on channel R_1.

2.6 Environmental Modeling

Evidently, the modeling of an embedded system has to include the environment and its constraints imposed on the system. Using virtual model components, system and environment can be modeled in a coherent way. Examples for virtual processes modeling the environment are the source processes P_{bus} and P_{sensor} as well as the sink processes P_{error} and P_{motor}. Note that the channels between the environment and the system are not virtual, since this communication has to be accounted for during implementation.

Another important part of the environment are *timing constraints*. In the SPI model, latency constraints can be specified for paths of channels and restrict the time tokens may spend on these paths.

Definition 8 ***(Latency Path Constraints)*** *A path constraint is a labeled path in the model graph. A path is of the form* $(P_1 \longrightarrow C_1 \longrightarrow P_2 ... \longrightarrow C_n \longrightarrow P_{n+1})$ *while involving* $n+1$ *processes,* n *channel nodes and* $2n$ *edges. As each channel uniquely identifies its sender and its receiver process, we describe a path more shortly by an expression of the form*

$$path = (c_1, \cdots, c_n)$$

where $c_1, \cdots, c_n \in C$*. A latency path constraint*

$$LC_{path} = [t_{lat,min}, t_{lat,max}]$$

denotes the time interval in which tokens are allowed to travel along the specified path from C_1 *to* C_n*. The path constraint must be satisfied for any token during any possible execution of the system.*

For a constructive method to check these latency path constraints see [11]. Using virtual model elements, other types of timing constraints like periodicity constraints can be modeled using latency constraints (e. g. process P_{time} in Figure 3).

3 Application Examples

In this section, two standard methods, static LCM-scheduling (e. g. [1]) and GRMS analysis [8], are applied to a SPI modeling of the example remote motor controller to show the validity of our approach.

The extraction of relevant aspects from the input descriptions and the mapping to the SPI model is dependent on user information regarding the level of abstraction in the extraction step and the granularity in the mapping step. Using the concept of process modes, the degree of abstraction used in parameter extraction can be controlled. The extreme cases are specifying a mode for each possible execution path of a process (high accuracy of modeling but exponential growth of paths with number of branches) or specifying a single behavior using uncertainty intervals (low accuracy but problem size reduction). Thus, during parameter extraction, a trade off between problem size and accuracy of modeling which is directly related to the accuracy of the results is possible.

Examples for this trade off can be seen in the different mappings in Fig. 3 and Fig. 5. They differ in the mapping of the SDF (P_3) to either a single process (Fig. 5) or three processes each representing an actor (Fig. 3). A decision that was made for both mappings is the representation of process P_1 with a single behavior including both modes. The reason for this decision is that due to the insufficient information about the environment (missing error probability on the bus) the mode information can not be used for off-line scheduling and analysis, anyway. The response time constraint is mapped to the latency path constraint $LC_{(R_1,Q_3)} = [0, 0.8\,t_1]$.

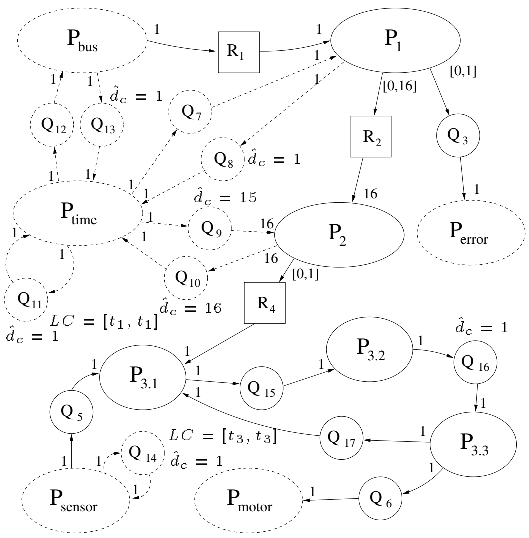

Figure 3: Remote Motor Controller (SPI Mapping I)

3.1 Static LCM-Scheduling

In this section, a static non-preemptive scheduling method (e. g. [1]) is applied to the SPI model of the remote motor controller as depicted in Fig. 3. All process instances in the macro period which is the least common multiple (LCM) of all process periods are statically scheduled. Then, the execution of the system consists of sequential executions of this macro period schedule that is shown in Fig. 4.

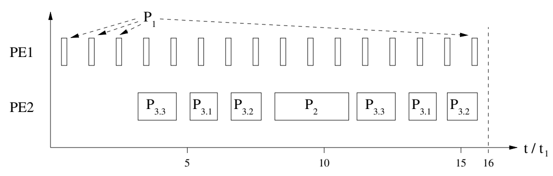

Figure 4: Gantt Diagram of Remote Motor Controller

Note that due to the response time constraint for P_1 it is not possible to schedule P_1 on the same processing element as the other processes using a non-preemptive scheduling method. Therefore, two processing elements are chosen that may have a lower performance than the processing element that is used in the single processor solution that is shown in the next section.

3.2 Performance Validation

In this section, the generalized rate monotonic scheduling (GRMS) analysis method [8] is applied to the SPI model of the remote motor controller as depicted in Fig. 5. Assume that all processes are mapped to the same processing element. For this case, the latency times of the processes can be seen in the first column of Table 1.

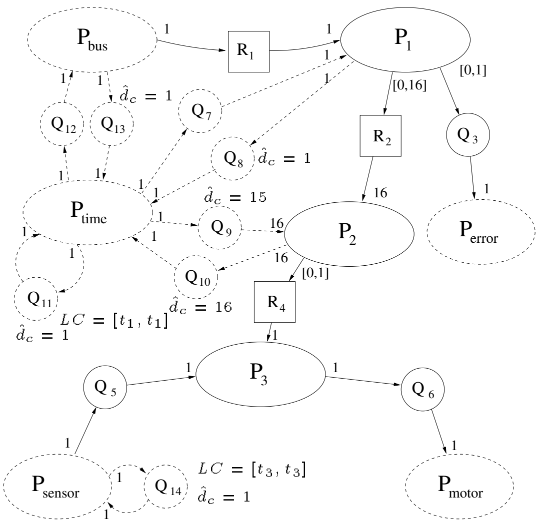

Figure 5: Remote Motor Controller (SPI Mapping II)

Process	Latency	C_i	T_i	D_i
P_1	$[0.2, 0.3]\, t_1$	$0.3\, t_1$	t_1	$0.8\, t_1$
P_2	$[0.6, 1]\, t_1$	t_1	$16\, t_1$	$16\, t_1$
P_3	$[1.1, 1.2]\, t_1$	$1.2\, t_1$	$8\, t_1$	$8\, t_1$

Table 1: Latency Time Intervals and GRMS Parameters

A prerequisite for the GRMS analysis is that the processes are independent. Although the processes in the example communicate with each other, they are independent since the communication is non-blocking. The latency path constraint over process P_1 can be accounted for as deadline D_1 while for the other processes the deadlines equal their periods ($D_i = T_i$). The GRMS method only uses the worst case execution times C_i which can be found with the other parameters ($t_3 = 8\, t_1$ assumed) in Table 1. With

$$\forall i, 1 \leq i \leq n, \frac{C_1}{T_1} + \cdots + \frac{(C_i + E_i)}{T_i} \leq i(2^{1/i} - 1)$$

$$\textit{where} \quad E_i = T_i - D_i \quad \textit{and} \quad D_i \leq D_{i+1}$$

and

$$\frac{(C_1 + T_1 - D_1)}{T_1} = 0.3 + 1 - 0.8 = 0.5 \leq 1$$

$$\frac{C_1}{T_1} + \frac{C_3}{T_3} = 0.3 + 1.2/8 = 0.45 \leq 0.83$$

$$\frac{C_1}{T_1} + \frac{C_3}{T_3} + \frac{C_2}{T_2} = 0.3 + 1.2/8 + 1/16 = 0.51 \leq 0.78$$

it is shown that the timing constraints can be fulfilled.

4 Conclusion

We proposed a model for the coherent representation of heterogeneously specified embedded systems with respect to scheduling, allocation and performance validation. With this model called SPI, properties like timing, communication behavior, modes and other properties that are important for the above mentioned problems can be represented using intervals to allow incomplete specification and non-deterministic behavior. This also enables the integration of system parts with different underlying models of computation.

The approach includes the capability to vary the degree of modelling accuracy using the concepts of parameter intervals and process modes. An extension of the SPI model to allow the representation of different types of function variants and reconfigurable systems can be found in [7].

We are currently planning to develop a publicly available SPI workbench which will include an open extensible data structure representing the SPI model, extractors for several modeling or specification languages, methods for scheduling, allocation, timing analysis and verification and a user interface that will allow easy integration of model extensions or new methods. The user will be able to perform experiments by embedding his own methods into the infrastructure provided by the workbench.

References

[1] Th. Benner and R. Ernst. An approach to mixed systems co-synthesis. In *Proceedings Fifth International Workshop on Hardware/Software Co-Design (Codes/CASHE '97)*, pages 9–14, 1997.

[2] W.-T. Chang, A. Kalavade, and E. A. Lee. Effective heterogeneous design and co-simulation. In G. de Micheli and M. Sami, editors, *Proceedings of NATO/ASI Workshop on Hardware/Software Co-design*, pages 187–212, Tremezzo, Italy, 1995. Kluwer Academic Publishers.

[3] A. Dasdan, D. Ramanathan, and R. K. Gupta. A timing-driven design and validation methodology for embedded real-time systems. *ACM Transactions on Design Automation of Electronic Systems*, 3(4), October 1998.

[4] A. Girault, B. Lee, and E. A. Lee. A preliminary study of hierarchical finite state machines with multiple concurrency models. Technical Report UCB/ERL M97/57, Electronics Research Laboratory, College of Engineering, Univ. of Cal. at Berkeley, 1997.

[5] 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.

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

[7] K. Richter, D. Ziegenbein, R. Ernst, L. Thiele, and J. Teich. Representation of function variants for embedded system optimization and synthesis. In *submitted to Proceedings 36th Design Automation Conference (DAC '99)*, New Orleans, USA, June 1999.

[8] 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.

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

[10] D. Ziegenbein, R. Ernst, K. Richter, J. Teich, and L. Thiele. Combining multiple models of computation for scheduling and allocation. In *Proceedings Sixth International Workshop on Hardware/Software Co-Design (Codes/CASHE '98)*, pages 9–13, Seattle, USA, 1998.

[11] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele. Representation of process mode correlation for scheduling. In *Proceedings International Conference on Computer-Aided Design (ICCAD '98)*, San Jose, USA, November 1998.