Hardware/Software Codesign of Embedded Systems - The SPI Workbench

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

L. Thiele
ETH Zurich

J. Teich
Uni-GH Paderborn

Abstract

Embedded systems typically include reactive and transformative functions, often described in different languages and semantics which are well introduced in the various application fields. A large part of the system functionality and components is reused from previous designs including legacy code. There is little hope that a single language will replace this heterogeneous set of languages. A hardware/software codesign process must be able to bridge the semantic differences for verification and synthesis and should accept limited knowledge of system properties. This paper presents the SPI model which is based on intervals of system properties and is specifically targeted to cosynthesis. This model is the basis of a workbench which is currently under construction in an international cooperation.

1 Introduction

Many embedded systems applications consist of a combination of reactive and transformative functions. Often, several languages with different underlying models of computation are used in the design of an individual system. The languages are selected because of their particular suitability for certain applications and optimizations, or because they have become generally accepted as a standard within an application field. The lack of coherency of the different languages, methods and tools is a substantial obstacle on the way to higher design productivity and design quality. A similar problem occurs when reused components shall be integrated, possibly described in another language and incompletely documented. Examples would be components of other companies or "legacy code".

In many years of cooperation with a team at ETH Zurich (Switzerland), a representation called SPI (**S**ystem **P**roperty **I**ntervals) [21, 22] has been developed which permits the safe integration of different system parts and enables system optimization across language boundaries. SPI is based on the model of communicating processes, whose behaviors are described by a set of annotations. These annotations enable the adaptation to different input models of computation. A major step towards higher semantic flexibility of the model is the use of behavioral intervals, e.g. time intervals and communication intervals, i.e. intervals of produced and consumed data capturing data-dependent communication. Typically, only a subset of the annotations is necessary for each input language. The consistency of these annotations is a prerequisite for global system optimization across the boundaries of different input languages.

Moreover, behavioral intervals allow the specification of upper and lower bounds in the input language. This enables the integration of processes whose internal functional details are only partially known, particularly "legacy code". If the execution rates of such processes are low, the wide interval boundaries are often of little importance, only.

Due to the use of intervals for the communication behavior the causal coupling of process activations is lost and optimization possibilities are limited. This problem becomes obvious when comparing the synthesis of synchronous and dynamic data flow constructs. Therefore, virtual processes and channels were introduced in the SPI model. These virtual components can be used to specify relations between processes concerning timing and causality. In combination with behavioral intervals, these virtual components allow the modeling of the system environment as well as the representation and combination of a variety of models of computation in a consistent form. Since they are not part of the system functionality, they do not need to be implemented, rather do they represent additional information for synthesis.

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 the above mentioned virtual model elements that have the same properties as the model elements representing the embedded system.

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

Based on this work, an open workbench for the analysis and synthesis of mixed reactive/transformative systems described in different languages will be developed.

The remainder of the paper is structured as follows. After an overview of related work the concept of the SPI workbench is presented in Section 3. In Section 4 the basic concepts of the SPI model are introduced and discussed. This is followed by an application examples in Section 5. The paper is concluded with an outlook on future work.

2 Related Work

An outline of the state of the practice and the state of the art in the area of hardware/software codesign is given in [6]. There, the insufficient coherency of the different languages, methods and tools is identified as a substantial obstacle on the way to a higher design productivity and to a reliable design process. An important aspect is the combination of domain-specific languages with their specific optimization procedures. Traditionally, flow-oriented models of computation are used in the area of signal processing or control engineering. In the last two decades, these were enhanced by models which are more suitable for the analysis of the design space and thus for synthesis, from synchronous data flow graphs (SDF) to boolean data flow graphs to dynamic data flow. A very instructive comparison of such representations can be found in [10]. These models are already accepted in industrial design driven by an extensive set of design tools which also support application optimization, as for instance MatrixX or MATLAB/SIMULINK. A similar development can be observed for event-oriented models of computation, which are of special importance for reactive systems, e. g. STATEMATE in automotive engineering or SDL in the area of telecommunications. There are many other examples, mainly from academia, like OCCAM [2], ESTEREL [1] and LOTOS [4], each of which represents a different model of computation.

In hardware/software codesign, programming languages like VHDL, C, C++ or Java are usually used as a basis for the description of more abstract models. The commercial codesign system CoWare uses a client-server mechanism for process communication, which is implemented in C-functions. Cx [7] and SpecC [19] use communicating processes with message passing via abstract channels, whose behavior is described by C functions. In the COSMOS codesign system [17], synchronous communicating finite state machines described in the SOLAR model are mapped to C or VHDL processes which, again, communicate via abstract channels. Communication between processes and channels is realized using an RPC mechanism. The RPC mechanism is also used in Matisse [12] where C++ methods enable process communication. CHINOOK implements message passing with JAVA methods [5] while in [18] C++ methods are used for the same purpose.

Languages like VHDL are obviously able to implement different models of computation enabling the common simulation of these models. For system synthesis, process communication has to be identified from such a detailed input description and abstracted, otherwise process scheduling as well as memory sizing are not possible. Exceptions are simple but less efficient scheduling techniques which neglect communication like round robin scheduling or the traditional Rate Monotonic Scheduling (RMS) [13]. Thus for synthesis, common modeling at this high level of detail has no advantages.

Approaches to common modeling of substantially different semantics for synthesis are rare. Lees recent work [11] defines a common description, which not only captures the system behavior in a uniform fashion but uses the same level of detail as in the (abstract) input description. Thus, even common simulation or verification become possible at a high level of abstraction. The resulting model, however, is comprehensive and does not consider behavioral intervals. In our previous work, we found that behavioral intervals are a key property to efficiently capture incompletely known or heavily data dependent behavior. Therefore, [11] appears less suitable for synthesis, while the SPI model is of limited expressiveness for system verification.

Other approaches can be found which extend the existing models or provide rules for the transition between different models, as for instance the PCC model proposed in [8]. The PCC differentiates event- and data-controlled processes with different activation mechanisms and introduces a partial order at the transitions between the different sub-graphs in order to exclude possible non-deterministic behavior. This approach seems too restrictive for a common representation of a large number of different languages and models of computation.

In principle, it seems possible to define a class of Petri Nets suitable for common modeling of different semantics. In fact, past work of our partners at the ETH Zurich is based on a Petri Net class. However and not unlike VHDL or C, Petri Nets can be used to model a large variety of models of computation, but the characteristics needed for synthesis would have to be determined by a Petri Net analysis. This additional effort would not be compatible with the goal of a highly efficient representation.

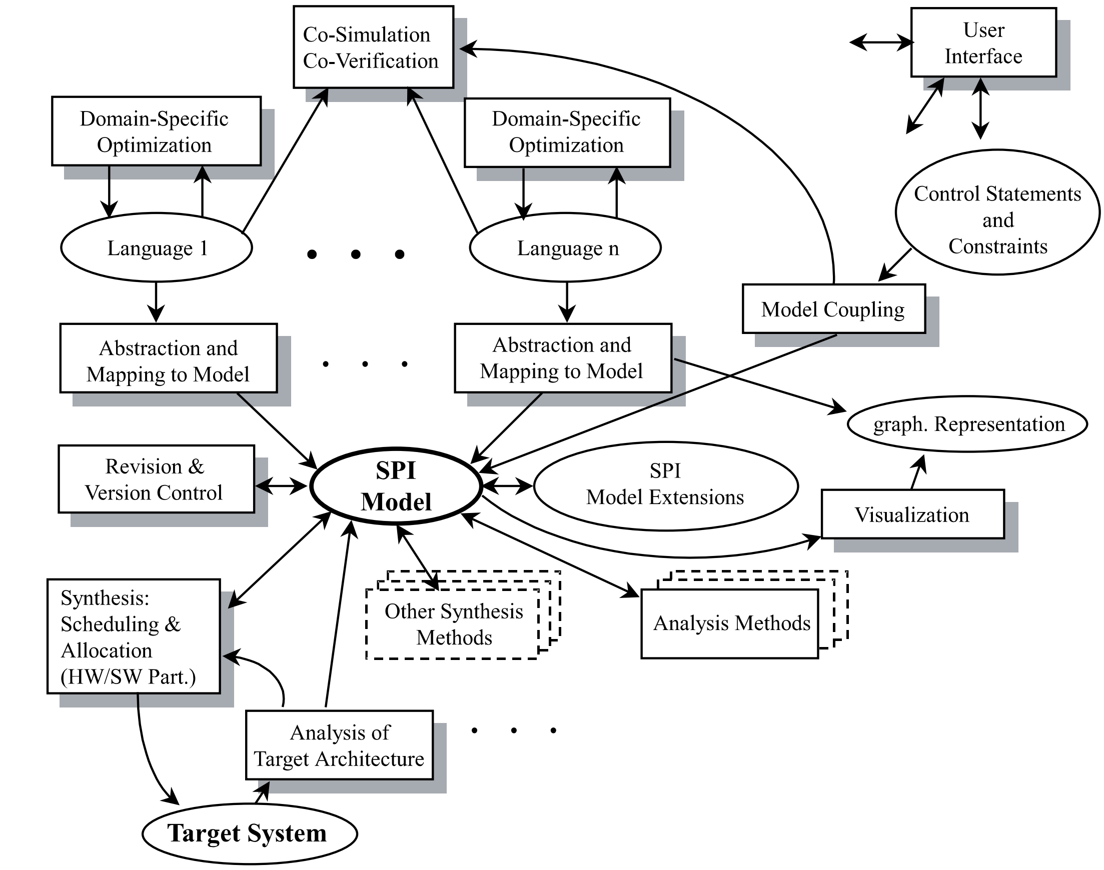

Figure 1: Structure of the SPI Workbench

3 The SPI Workbench

In this section, the concept and the intended structure of the SPI (**S**ystem **P**roperty **I**ntervals) workbench is presented. This workbench shall enable the analysis and synthesis of mixed reactive/transformative systems described in several languages with possible differences in their underlying models of computation.

A main incentive for the creation of the workbench is the missing infrastructure in this field. Research in system design automation requires a high effort to build the necessary environment to obtain results for relevant examples. This seems to discourage young faculty and smaller research groups to work in the field or motivates research based on over-simplifying assumptions. An open research platform with various opportunities for original contributions, exchange of algorithms and access to demonstrators seems to be a reasonable approach to this problem.

Fig. 1 shows the intended workbench structure. Input is a system with its system function captured and optimized in application specific languages. The advantage of such a multi-language representation compared to using a uniform system specification language is the possibility to include the traditional design environment of the application developer with domain-specific optimization techniques and tools.

The SPI model shall serve as an internal abstract representation of the mixed system function specifically targeted to synthesis. For this purpose, all information relevant to synthesis is abstracted from the input languages and transformed into the semantics of the SPI model. A crucial point for the applicability of the SPI workbench is the availability of transformations from widely used standard languages to the SPI model. The principle of this translation and transformation has already been shown for different standard languages and models like SDF, periodic processes, VHDL processes, or StateCharts [14]. Language transformations e.g. for Matlab/SIMULINK will be implemented in parallel to the workbench thereby providing the necessary infrastructure for other research groups to use the workbench.

Hence, the SPI model is not a universal specification language but models the behavior of the processes only in so far as it is relevant for synthesis, i.e. the resource utilization, the communication and timing behavior. On the SPI level, the subsystems can be merged to a uniform graph representing the whole system. Examples can be found in sections 4 and 5 or in [22].

Then, this uniform SPI graph shall serve as the input description for global analysis and synthesis of the system. This homogeneous system description enables combined design optimizations of the subsystems regardless of language boundaries. It has been shown that standard scheduling techniques can be applied to SPI graphs. This includes static methods like periodic LCM-scheduling [21] as well as preemptive methods as rate monotonic scheduling [20]. A first approach to scheduling of an extension of the SPI model that combines static and dynamic scheduling can be found in [16].

The control of the workbench will be based on a graphical user interface. This user interface shall be able to visualize a SPI graph and its relation to the components of the input description. Visualization appears mandatory for SPI graph manipulation and synthesis control. It offers additional possibilities for debugging of the workbench function itself as well as of the implemented synthesis techniques.

This very comprehensive workbench is to be created and used in international cooperation. Current cooperation partners are teams at the TU Braunschweig, the ETH Zurich, and the Uni-GH Paderborn. Also involved are teams at Princeton University and Indian Institute of Technology Delhi. But as already mentioned, the workbench shall not be limited to these partners, but shall be publicly available and extendable.

4 The SPI Model

In this section, the concepts of the SPI model [21, 22] 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 [13] 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.

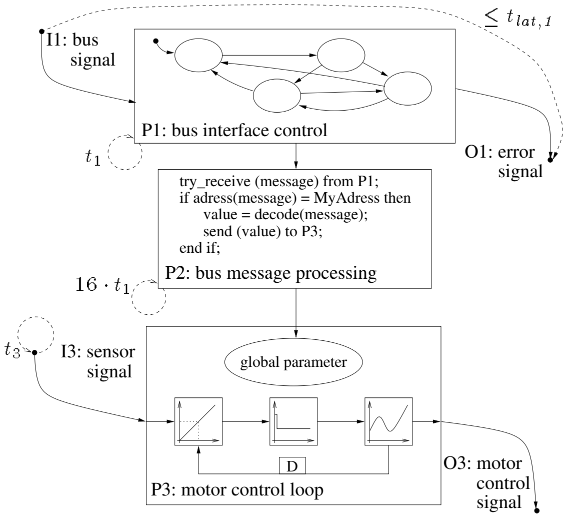

Figure 2: Remote Motor Controller

4.1 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.

4.2 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 4.5 and in [21], 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}.

4.3 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 [9]) 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]

[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 $d_{c,av} = d_c$.

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

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 4.5) or relative execution rates. As will be seen from the following examples, virtual model elements are denoted by dotted lines.

4.4 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 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

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

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.

4.5 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.

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 4.6).

Another possible activation principle is the *activation by relative execution rates* (e. g. RTOS semantics: no exact periodicity but constrained mobility intervals [13]). 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 4.4), 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.

4.6 Modeling of the Environment

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

[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.

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 [22]. 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).

5 Application Example

In this section, a off-line non-preemptive scheduling method (e. g. [3]) is applied to the SPI model of the remote motor controller as depicted in Fig. 3. Note that the SDF in process P_3 is 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. These mapping decisions are left to the designer as part of a trade off between problem size and accuracy of modeling. Another mapping decision 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 $LC_{(R_1, Q_3)} = [0, 0.8\,t_1]$.

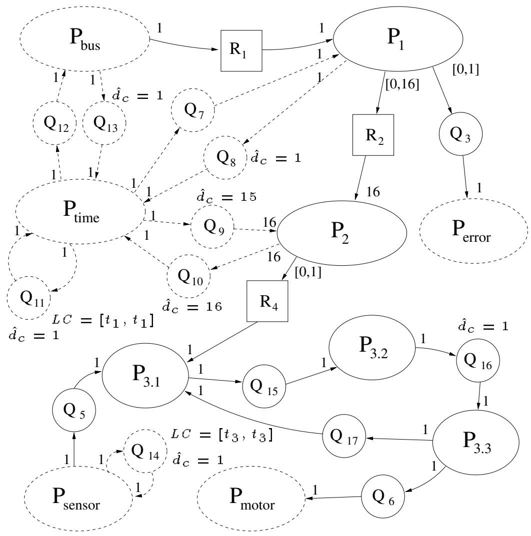

Figure 3: Remote Motor Controller (SPI Mapping)

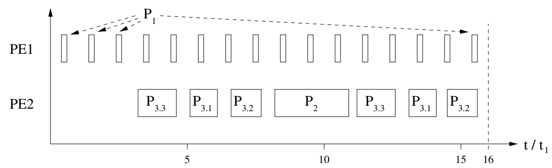

Figure 4: Gantt Diagram of Remote Motor Controller

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 ($t_3 = 8t_1$ assumed).

6 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 nondeterministic 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 [15].

Based on the SPI model, a publicly available SPI workbench is to be developed. This workbench 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] F. Balarin, P. Giusto, A. Jurecska, et al. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, May 1997.

[2] A. Balboni, W. Fornaciari, and D. Sciuto. Cosynthesis and cosimulation of control-dominated embedded systems. *Journal on Design Automation of Embedded Systems*, 1(3):257–288, June 1996.

[3] 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.

[4] C. Carreras, J.C. Lopez, M.L. Lopez, C. Delgado-Kloos, N. Martinez, and L. Sanchez. A co-design methodology based on formal specification and high-level estimation. In *Proceedings Fourth International Workshop on Hardware/Software Co-Design (Codes/CASHE '96)*, Pittsburgh, USA, 1996.

[5] P. Chou, R. B. Ortega, and G. Borriello. The chinook hardware/software co-synthesis system. In *Proceedings 8th International Symposium on System Synthesis (ISSS '95)*, pages 22–27, Cannes, France, September 1995.

[6] R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design & Test of Computers*, pages 45–54, April 1998.

[7] R. Ernst and Th. Benner. Communication, constraints and user directives in COSYMA. Technical Report CY-94-2, Institut für DV-Anlagen, Technische Universität Braunschweig, 1994.

[8] T. Grötker, R. Schoenen, and H. Meyr. PCC: A modeling technique for mixed control/data flow systems. In *Proceedings European Design & Test Conference (ED&TC '97)*, pages 482–486, Paris, France, March 1997.

[9] 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.

[10] E. A. Lee and Th. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–799, May 1995.

[11] E.A. Lee. Modeling concurrent real-time processes using discrete events. *Annals of Software Engineering*, 1998.

[12] B. Lin. A system design methodology for software/hardware co-development of telecommunication network applications. In *Proceedings 33rd Design Automation Conference (DAC '96)*, Las Vegas, USA, June 1996.

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

[14] K. Richter. Developing a general model for scheduling of mixed transformative/reactive systems. Master's thesis, Institut für DV-Anlagen, TU Braunschweig, January 1998.

[15] 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.

[16] K. Strehl, L. Thiele, D. Ziegenbein, and R. Ernst. Scheduling hardware/software systems using symbolic techniques. In *Proceedings Sixth International Workshop on Hardware/Software Co-Design (Codes/CASHE '99)*, Rome, Italy, May 1999.

[17] C. A. Valderama, M. Romdhani, J. M. Daveau, et al. *Hardware/Software Co-Design: Prinicples and Practice*, chapter COSMOS: A Transformational Co-design Tool for Multiprocessor Architectures. Kluwer Academic Publishers, 1997.

[18] C. Weiler, U. Kebschull, and W. Rosenstiel. C++ base classes for specification, simulation and partitioning of a hardware/software system. In *Proceedings of VLSI*, Tokyo, Japan, 1995.

[19] J. Zhu, R. Dömer, and D. D. Gajski. Syntax and semantics of the SpecC language. In *Synthesis and System Integration of Mixed Technologies*, Osaka, Japan, December 1997.

[20] D. Ziegenbein and R. Ernst. A framework for high-level performance validation of embedded hw/sw systems. In *Proceedings High-Level Design Validation and Test Workshop (HLDVT '98)*, San Diego, USA, November 1998.

[21] 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.

[22] 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.