Transformation of SDL Specifications for System-Level Timing Analysis

Marek Jersak, Kai Richter, Rafik Henia, Rolf Ernst
Institute of Computer Engineering (IDA)
Technical University of Braunschweig
D-38106 Braunschweig, Germany
jersak,richter,henia,ernst@ida.ing.tu-bs.de

Frank Slomka
Computer Engineering Laboratory (DATE)
University of Paderborn
D-33098 Paderborn, Germany
slomka@date.uni-paderborn.de

ABSTRACT

Complex embedded systems are typically specified using multiple domain-specific languages. After code-generation, the implementation is simulated and tested. Validation of non-functional properties, in particular timing, remains a problem because full test coverage cannot be achieved for realistic designs. The alternative, formal timing analysis, requires a system representation based on key application and architecture properties. These properties must first be extracted from a system specification to enable analysis. In this paper we present a suitable transformation of SDL specifications for system-level timing analysis. We show ways to vary modeling accuracy in order to apply available formal techniques. A practical approach utilizing a recently developed system model is presented.

1. INTRODUCTION

Todays complex embedded systems are typically specified using multiple domain-specific languages and tools. Apart from functional simulation and test, analysis of non-functional properties, in particular timing, is required in order to verify that the chosen target architecture, binding and scheduling decisions satisfy all constraints.

Existing system-level design methodologies mainly address specification, simulation, synthesis and functional test. For multi-language designs, integration focuses on language- and tool-interfaces, while the semantics of the underlying computational models are largely ignored.

This shallow integration is a less suitable solution for system-level timing analysis, since critical corner case are hard to find. Complete simulation of a system is not an alternative due to the huge number of cases that would have to be exercised for realistic systems. Instead, system-level timing analysis requires deep integration [3], i. e. a holistic view of the system which takes into account key properties of the specification independent of specification languages to facilitate multi-language design. Functional details that do not influence resource demand can be abstracted.

Formal analysis approaches exist in the area of real-time schedulability tests. Task activation is represented by event models, and resource demand is considered from the perspective of a scheduler. This information is not directly available from standard specification languages or design tools. Therefore, an application specification first has to be transformed into a suitable representation for analysis. For state-based languages, this transformation is difficult because activation models can be complex due to their control dominated semantics, and because processes can have many different behaviors.

In this paper we focus on the transformation of the language SDL (Specification and Description Language) into a representation suitable for system-level timing analysis. SDL is widely used for the specification of communication protocols. We present transformations at different levels of abstraction. The key is to capture the properties of the specification in sufficient detail to allow reasonably accurate analysis using available formal techniques, but to remain sufficiently abstract such that the parameters that those techniques require are available. A practical approach utilizing the recently developed SPI model is presented.

The goal of the SPI (**S**ystem **P**roperty **I**ntervals) project is to provide a framework for system-level timing analysis [23]. The idea is to transform the (potentially multi-language) input specification into a language-independent intermediate representation which is well suited for system-level analysis, and to apply existing or novel analysis techniques. Input language transformation has been demonstrated for Matlab/Simulink [11]. However, Matlab/Simulink is a language with regular, periodic process activation and thus considerably easier to transform and analyze than SDL.

After an overview of related work, we summarize the properties needed for system-level timing analysis. We then focus on how to obtain the implementation-independent part of this information from SDL specifications. Transformations at different levels of abstraction are considered using a wireless-communication protocol as an illustrating example. In the final section, an example is presented that shows how our approach bridges the gap between SDL specifications and the application of different timing-analysis techniques.

CODES '02 Estes Park, Colorado USA

2. RELATED WORK

Several specification languages for reactive systems, e. g. SDL [9] or StateCharts [7], and specification languages for transformative systems, e. g. synchronous data-flow [15] or Matlab/Simulink [17] exist. Each of these languages is best suited for and thus established in certain application domains, and supported by domain-specific tools. Specialized languages have obvious benefits, in particular modeling and simulation efficiency and ease of use of tools, and therefore a single super-language is not likely to emerge [14, 5].

SDL is widely used for the specification of the protocol automata of communication systems. The language composes asynchronous *extended finite state machines* (EFSMs). Communication is abstracted into unbounded FIFOs or shared variables, and timing into precedence relations.

Available SDL tools focus mostly on specification, simulation and synthesis. The standard specification and simulation tool for SDL is Telelogic Tau [20]. Tau can also generate C code from SDL for different targets. Other code-generators exist that generate hardware descriptions from SDL, e. g. [2, 4], or support HW/SW co-synthesis from SDL [19]. The MASCOT system [1] supports co-simulation of Tau and Matlab/Simulink [17]. The COSMOS system [22, 3] supports multi-language co-simulation of SDL and Matlab/Simulink, and HW/SW co-synthesis from SDL. For this purpose, the SDL specification is translated into an intermediate representation called SOLAR [10]. SOLAR consists of interacting, hierarchical EFSMs communicating via refinable channels. Only shallow integration is provided with Matlab/Simulink, which is not translated into SOLAR and has to be synthesized separately.

Ptolemy [14] is a framework that allows to combine different models of computation for simulation and potentially code-generation using a well structured set of Java classes and interface taxonomy. Specifications written in Ptolemy could be abstracted into a SPI representation in analogy to commercially available specification tools.

None of the available tools supports formal system-level timing analysis. This is because fully executable specifications present too much detail that is not interesting for timing-analysis, and hide those properties which are required.

System-level scheduling analysis is mostly based on a schedulers view of the system. The classic paper by Liu and Layland [16] provides schedulability tests for several priority-based strategies for periodic processes. This work has since been extended in many directions. Fidge [6] presents a good overview about existing analysis approaches. In all these approaches, key parameters (priorities, time slots, execution and communication delays, etc.) are inserted into a mathematical framework to determine system-level properties such as load, or task and bus response times.

3. REPRESENTATION OF SDL SPECIFICATIONS FOR SYSTEM-LEVEL TIMING ANALYSIS

In this section, we show how to transform SDL into a representation suitable for system-level timing analysis. The SPI model [23] is used as a basis for this representation.

3.1 Requirements

System-level timing analysis is concerned with temporal properties of the interaction of SW- and HW-components in a system, in order to verify timing constraints. This requires certain information about:

- *application*: An application is modeled as communicating processes. Process activation conditions (e. g. periodic or event-driven) and the communicated data per activation have to be known. Detailed process function is *not* required.
- *architecture*: The architecture describes the resources (processors and busses) and available resource-sharing strategies, which result from the selected operating systems and bus arbitration protocols.
- *mapping and scheduling*: Mapping and scheduling (assignment of priorities, slot times etc.) of processes and communication on resources have to be provided. Core execution times (the time process execution or communication would take if the resource was exclusive) also have to be known. They result from mapping decisions.

3.2 SPI Model Basics

The SPI (**S**ystem **P**roperty **I**ntervals) model was developed to facilitate system-level analysis of multi-language designs, in particular system-level timing analysis [23]. In this section, only the basic concepts of SPI, which are necessary to understand this paper, are introduced informally.

In the SPI model, a system is represented as a set of concurrent processes which communicate tokens via unidirectional channels that are either FIFO queues (destructive read) or registers (destructive write). Processes as well as channels are not characterized by their exact internal functionality but by their abstract external behavior. This includes process activation conditions and the amount of communicated data per execution. For example, process P_1 in Fig. 1 consumes 1 data token and produces 2 data tokens per execution. By default P_1 is activated, i. e. ready for execution, if there are enough tokens available for one execution (in the example at least one) on its incoming channel. A core execution time of $1ms$ is annotated for simplicity, but in fact results from a mapping decision.

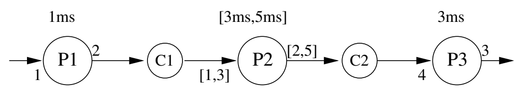

Figure 1: SPI Example

A key concept in SPI is the possibility to specify all properties as intervals. For example, process P_2 consumes at least 1 and at most 3 tokens from channel C_1 and produces at least 2 and at most 5 tokens on channel C_2 per execution. P_2 is activated if at least 3 tokens are available at its input. The use of intervals allows to conservatively bound a set of possible behaviors, and to apply system-level analysis techniques which operate with conservative bounds and thus cover the whole design space. This avoids testing the huge number of situations that have to be considered in simulation-based validation approaches. Application specifications in different languages or models of computation can be captured with the same few SPI elements. Therefore, analysis of multi-language designs becomes feasible.

Additional concepts in SPI include process modes to model distinct process behaviors; mode tags to model process mode correlations; virtual processes and channels to model the system environment, timing constraints and otherwise hidden scheduling dependencies. We will explain these concepts in more detail in Sec. 3.4.

3.3 Introduction to SDL

SDL [9] was originally designed for the specification of protocol automata in communication systems and is now the standard language for this purpose. SDL automata can perform data operations which leads to the *extended finite state machine* (EFSM) model of computation. An SDL system is structured into blocks and sub-blocks, and EFSMs are described using processes and services. Each process or service contains an independent EFSM, which communicate through asynchronous signals. Each process has its own FIFO input queue that collects all incoming signals. Different services of one process share all process variables and the process input-queue.

Each transition of an EFSM is triggered either by the receipt of an asynchronous signal, the change of a local variable or a combination of both. Transitions and communication are atomic and basically timeless, except that precedence relationships of signals are maintained. A local variable can change due to the expiration of a timer, which allows periodic activation of transitions.

3.4 Basic Representation of SDL in SPI

Our basic representation of an SDL process in SPI is a single SPI process P_{Fsm} with an input queue Q_{In}, an output queue Q_{Out} and a virtual self-queue Q_{State}, which is initialized with one token. P_{Fsm} is activated when one token on Q_{In} and one token on Q_{State} are available. Upon execution, the tokens are consumed, a new token is produced on Q_{State} and zero or more tokens can be produced on Q_{Out}. Additionally, we associate a *mode tag* [23] with the token on Q_{State} to capture information about the state of the SDL EFSM. A mode tag is a name-value pair, which allows to capture additional information about the data that a token represents. For each mode tag, a domain D of possible values is defined. To account for uncertainty, the value of a mode tag can be any subset of D. In our mapping from SDL to SPI, we define D to be the set of all states of the EFSM. The mapping is shown in Fig. 2 for a simple SDL process. Q_{State} is virtual (indicated by a dashed line) [23], since it does not model external communication of process P_{Fsm}, but instead represents information about P_{Fsm} which otherwise would be hidden inside the process implementation.

The SPI model allows to distinguish between different process behaviors using the concept of *process modes* [23]. Modes are selected bases on the number of tokens and their tags at the process input ports. In our SPI representation of an SDL process, N_{States} modes can be distinguished based on the tag on Q_{State}. If tokens on Q_{In} are additionally tagged with a set of possible signals that the tokens can represent, then the set of possible modes is $N_{Modes(Fsm)} = N_{States} \times N_{InputSignals}$. The resulting modes for our example are listed in Fig. 3. Mode-dependent data rates and tag-production can also be annotated directly at process ports in a SPI graph, as shown in Fig. 5.

For SDL processes with multiple input and output channels, two

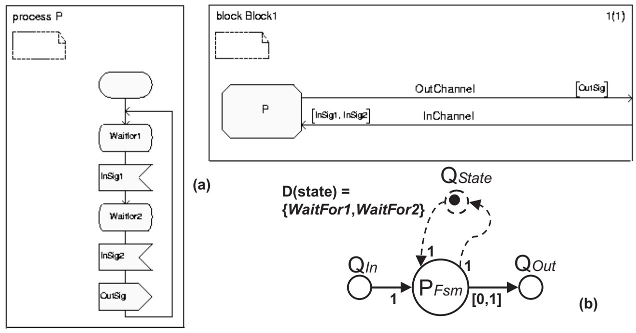

Figure 2: SDL process (a) and mapping to SPI (b)

Mode	QState-tag	QIn-tag	Read (QIn,QState)	Write (QOut(tag), QState(tag))
m1	WaitFor1	InSig1	(1,1)	(0 (-) ,1(WaitFor2))
m2	WaitFor1	InSig2	(1,1)	(0 (-) ,1(WaitFor1))
m3	WaitFor2	InSig1	(1,1)	(0 (-) ,1(WaitFor2))
m4	WaitFor2	InSig2	(1,1)	(1(OutSig),1(WaitFor1))

Figure 3: One possible set of modes and tag productions for the SPI process in Fig. 2

SPI processes, P_{Merge} and P_{Fsm}, are generated. This decouples the collection of signals from different sources and the activation of modes of P_{Fsm} based on signal tags, reducing the maximum number of modes from

$$N_{Modes(Fsm)} \quad = \quad N_{States} \times N_{InputSignals} \times N_{InputChannels}$$

to

$$N_{Modes(Fsm)} + N_{Modes(Merge)} = N_{States} \times N_{InputSignals} + N_{InputChannels}$$

An SDL process with 2 input and 2 output channels and the corresponding SPI representation with decoupled processes P_{Merge} and P_{Fsm} are shown in Figs. 4 and 5.

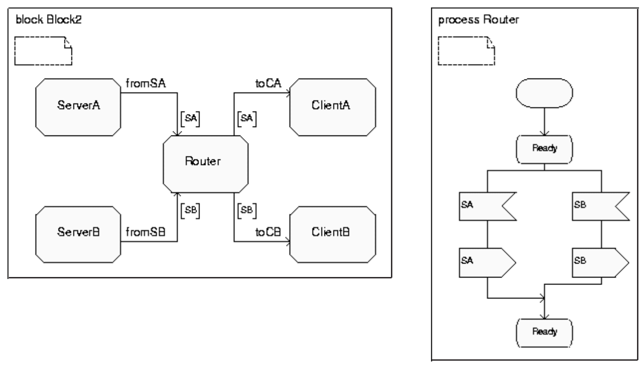

Figure 4: SDL process with two inputs and outputs

SDL signals exchanged between processes inside the same block are consumed in the order in which they were produced. On the other hand in SPI, a process with sufficient tokens at its inputs is activated, but does not have to be executed immediately. Therefore, if two tokens arrive at *different* inputs of process P_{Merge} without

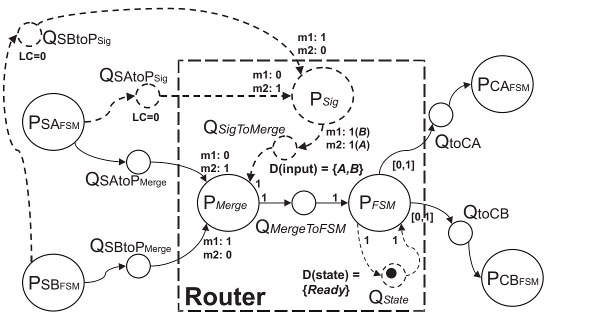

Figure 5: SPI representation of SDL process *Router* from Fig. 4

P_{Merge} having been executed in between, it is impossible to determine the arrival order of the two tokens. We solve this problem in our SPI representation of SDL by capturing the order, in which the tokens were produced, into virtual queue $Q_{SigToMerge}$. Virtual process P_{Sig} is executed in one of its modes immediately after P_{SAFsm} or P_{SBFsm} due to latency constraints of zero on its input channels ($LC = 0$), and produces a properly tagged token at its output. The tag is evaluated to determine from which of its inputs P_{Merge} reads the next token. Again, virtual elements are used to represent information about the system behavior which otherwise would be hidden. If communicating processes reside in different blocks, those virtual elements are omitted.

We have also modeled SDL services, dynamic process instantiation and termination, the 'store' mechanism and timers in SPI. Due to the limited space of this paper, the reader is referred to [8].

4. HIGHER LEVELS OF ABSTRACTION

The detailed modeling presented in the previous section on one hand shows the expressiveness of SPI, but on the other hand can lead to a system complexity for realistic applications which cannot be analyzed with current techniques. In this context, we can benefit from using SPI, which supports uncertainty, and choose the level of abstraction appropriately for different system elements. We present three approaches towards abstraction using parts of the MAC (medium access control) protocol of a DECT wireless base station as an example.

4.1 DECT Example

Our DECT example has been kindly provided by the authors of [19]. The system-level SDL view in Fig. 6 shows six different SDL processes. The process Lower Layer Management Entity (LLME) manages the complete MAC layer. Handover and broadcast management are performed by the process Multi Bearer Controller (MBC) and the process Broadcast Message Controller (BMC). The control of the radio links is performed by the cell site function processes (CSF). The process A_CSF performs protocol data transmission, while the other two CSF processes perform speech data transmission.

4.2 Combination of Behaviors

The use of intervals in SPI allows conservative combination of distinct process behaviors into a single abstraction. This greatly reduces the number of cases that have to be considered during analysis, and also accounts for the fact that most current analysis techniques do not distinguish different process behaviors.

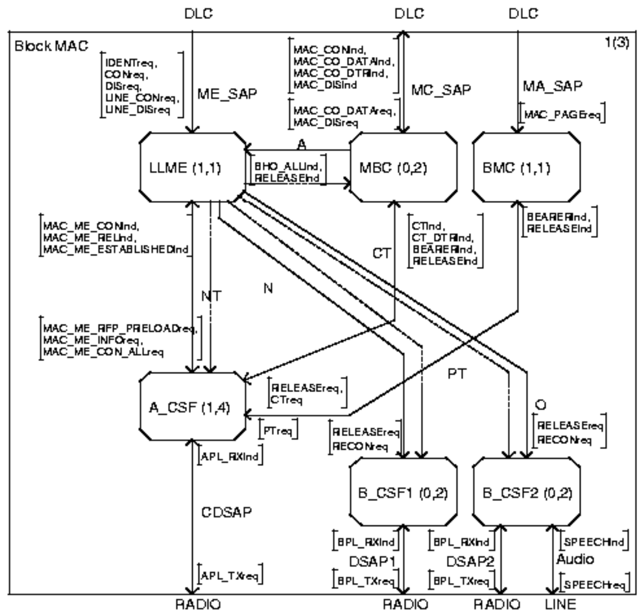

Figure 6: SDL Specification of DECT MAC protocol

Consider SDL process *A_CSF* in Fig. 6 and its mapping to SPI in Fig. 7. Different from Fig. 2, the distinction between different states of the corresponding EFSM has been abstracted (making the self-FIFO of P_{A_CSF} unnecessary). Process P_{A_CSF} is simply activated by a token arriving on channel $Q_{MergeToFSM}$. Process $P_{A\_CSFmerge}$ is activated by a token arriving on channel Q_{A_CSFSig} and by the corresponding token arriving on one of $Q_{fromLLME}$, $Q_{fromBMC}$ or $Q_{fromMBC}$.

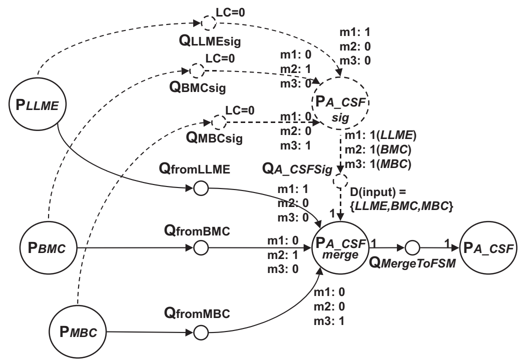

Figure 7: SPI representation of SDL process P_{A_CSF}

Because of the signal-flow through the system, the activation of process P_{A_CSF} is the superposition of the activations of processes P_{LLME}, P_{BMC} and P_{MBC}, with additional jitter introduced by the mapping- and scheduling-dependent delays of $Q_{fromLLME}$, $Q_{fromBMC}$, $Q_{fromMBC}$, $P_{A\_CSFmerge}$ and $Q_{MergeToFSM}$. Analysis details of such activation propagation and superposition can be found in [18]. Obviously, execution time and output data rate intervals of P_{A_CSF} are wide due to abstraction of its transitions, which leads to more conservative analysis results. Therefore, the next two sections show transformations that increase modeling accuracy without forfeiting analyzability.

4.3 Separation of Activation Principles

So far, process activation by asynchronous events has been discussed. Additionally, periodic activation, which may be specified using SDL timers, SDL clock extensions such as [19] or periodic timing constraints, e. g. [12] is possible. If different transitions in the same SDL process are activated asynchronously and periodically, then for analysis the asynchronous and periodic parts of the SDL process can be viewed as mutually exclusive SPI processes. This separation overcomes restrictions of existing analysis techniques which usually can only deal with a single activation principle per process. Communication between the two processes is via SPI register channels, which do not contribute to process activation. We have applied this technique to process P_{A_CSF}, which apart from the event-activated transitions shown in Fig. 7 also has periodic transitions (Fig. 8). Periodic activation is expressed using a periodic rate constraint in consistence with [12]. $P_{A\_CSF periodic}$ cycles between send and receive modes based on the value of tag *state*. Such cyclo-static behavior can be exploited for analysis (e. g. [13]).

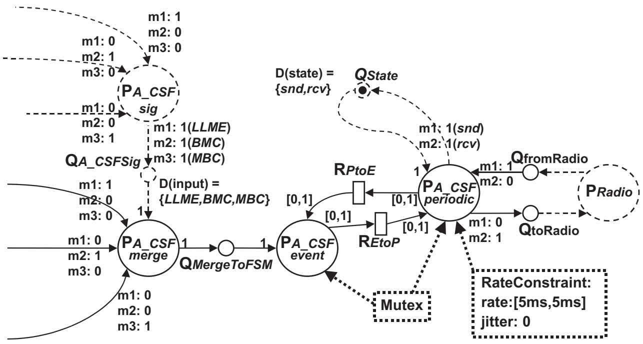

Figure 8: Separation of SDL FSM into periodically and asynchronously activated processes

4.4 Separation of Scenarios

Typically, in an SDL specification only a subset of signals are exchanged and only a subset of transitions are taken during a certain mode of operation of the system. This is often called a scenario. For example, 'idle', 'connect', 'active' or 'disconnect' scenarios exist in a simple telephony protocol.

Scenarios are mutually exclusive, and thus can be analyzed separately, which reduces analysis complexity without sacrificing accuracy. For timing analysis only process-transitions which can be executed during a particular scenario have to be considered, leading to narrower execution-time intervals. Furthermore, only timing-constraints applicable to the scenario have to be verified.

In Fig. 9, we use message sequence charts (MSCs) to shows the SDL signals which are exchanged during the *radio-to-line* scenario in the DECT MAC protocol. Processes $P_{A\_CSF periodic}$ and $P_{B\_CSF2 periodic}$ are periodically (every $5ms$) sending or receiving data to/from the radio. $P_{B\_CSF2 periodic}$ is also sending data to the line every $10ms$. Data from the line is arriving periodically every $1/40 \times 10ms$ and activates process $P_{B\_CSF2 event}$, which has to collect the data and prepare the next B-Frame. $P_{A\_CSF event}$ is receiving sporadic bursts of data from P_{DLC} (data link control, the next higher protocol layer) via P_{MBC}, P_{BMC} and $P_{A\_CSF merge}$. SDL Processes P_{LLME} and P_{B_CSF1} (Fig. 6) are not active.

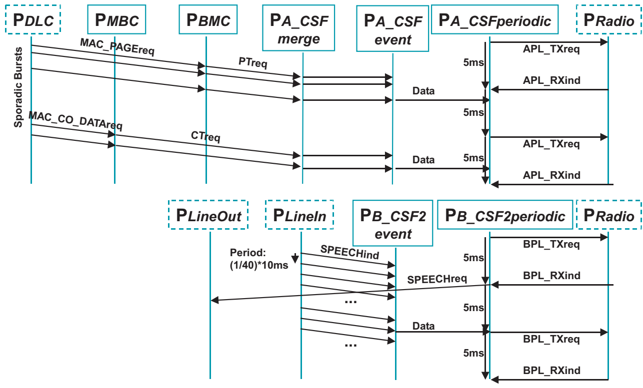

Figure 9: MSC for radio-line scenario

5. EXAMPLE

Our example shows how the transformations presented in this paper can be used to bridge the gap between an application specification in SDL and existing system-level timing analysis techniques. The example is based on the *radio-to-line* scenario introduced in Sec. 4.4. We focus on the processes necessary for the A-frame (Figs. 7 and 8) and assume a mapping of function to resources shown in Fig. 10. Other processes and channels are mapped to the resources but not shown in the figure.

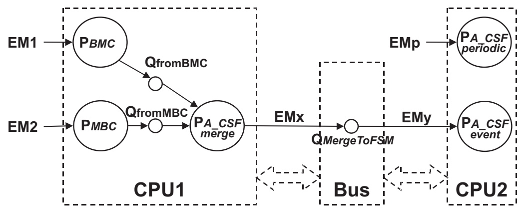

Figure 10: Analysis graph for part of the radio-line scenario

In the step shown, we applied our knowledge from Fig. 9 that P_{LLME} is inactive in this scenario. We then translated the communication between processes into event models [18] (EM_1 etc.), since these are needed for scheduling analysis. We chose to implement all communication between SDL processes P_{BMC}, P_{MBC}, and P_{A_CSF} on a single logical channel of the bus. Process $P_{A\_CSF merge}$ is implemented as the bus driver and takes care of transmitting signals to P_{A_CSF} in the right order. Recall from Sec. 4.3 that $P_{A\_CSF periodic}$ and $P_{A\_CSF event}$ have been introduced for analysis purposes and are mutually exclusive. We can achieve this in case of a fixed-priority scheduler on CPU_2 by assuming that they have the same priority.

From the DECT-MAC specification, we know that EM_p is purely periodic (Fig. 9). Additional designer knowledge tells us that event models EM_1 and EM_2 are sporadic bursts. We can analyze the timing of P_{BMC} and P_{MBC} using the approach from Tindell [21] which supports burst. In addition to process response times, we also obtain the output event models, i. e. the abstract timing behavior of the output signals. Since communication is merged, we have to merge the event models in order to analyze bus performance. This is done by superimposing the output event models; we obtain EM_x which is also a sporadic burst. Now, we can analyze the bus performance to derive event model EM_y. Finally, we can analyze the performance of CPU_2. For the analysis of the bus and CPU_2, we can chose from the vast amount of existing methods for a single resource (for an overview see [6]). Result correctness is guaranteed due to the use of event models and appropriate event model operations [18].

6. CONCLUSION AND OUTLOOK

In this paper we presented transformations from SDL specifications into an intermediate representation based on the SPI model. This representation is well suited for system-level timing analysis using existing and novel analysis techniques. We thus closed the gap between a system specification in a standard language, SDL, and system-level timing analysis, which assumes a different viewpoint, namely that of an operating-system. We explained the key benefits of each of the proposed transformations. The separation of activation principles into separate processes allows to employ simpler and at the same time less conservative activation models. It also reduce the number of different behaviors per process. Predictability and therefore analyzability increase. The separation of scenarios helps us to focus on one scenario at a time, reducing the overall problem in two ways. Firstly, we omit all processes which are inactive in this scenario. Secondly, we further reduce the number of behaviors which have to be considered at a time. Finally, we combine all remaining behaviors of one process into a single abstraction to apply existing analysis techniques.

Our next steps will be to apply the indicated analysis techniques for our DECT example using real performance numbers. Furthermore, we want to demonstrate that a unified SPI representation of a heterogeneously-specified system enables analysis across language boundaries, by analyzing a system specified part in SDL and part in Simulink.

7. REFERENCES

[1] P. Bjureus and A. Jantsch. Heterogeneous system-level cosimulation with SDL and Matlab. In *Proc. 2nd Forum on Design Languages, FDL*, Sept. 1999.

[2] O. Bringmann, A. Muth, F. Slomka, W. Rosenstiel, G. Färber, and R. Hofmann. Mixed abstraction level hardware synthesis from SDL for rapid prototyping. In *Proc. 10th IEEE International Workshop on Rapid System Prototyping (RSP'1999)*, Clearwater, USA, 1999.

[3] P. Coste, F. Hessel, P. L. Marrec, et al. Multilanguage design of heterogeneous systems. In *Proc. 7th Int. Workshop on Hardware/Software Co-Design CODES/CASHE99*, Rome, Italy, May 1999.

[4] J. Daveau, G. Marchioro, et al. VHDL generation from SDL specifications. In *Proc. 13th IFIP Conference on Computer Hardware Description Languages (CHDL'97)*, Toledo, Spain, 1997.

[5] R. Ernst and A. A. Jerraya. Embedded system design with multiple languages. In *Proc. 5th Asia South Pacific Design Automation Conference (ASP-DAC'00)*, Jan. 2000.

[6] C. J. Fidge. Real-time schedulability tests for preemptive multitasking. *Real-Time Systems*, 14:61–93, 1998.

[7] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[8] R. Henia. Entwurf eines prototypen zur Übersetzung von SDL nach SPI. Technical report, IDA, TU Braunschweig, 2002.

[9] ITU. *ITU-T. Recommendation Z.100. Specification and Description Language*, 1993.

[10] A. Jerraya and K. O'Brien. *Computer Aided Software/Hardware Engineering*, chapter SOLAR: An Intermediate Format for System-Level Modeling and Synthesis. IEEE Press, 1994.

[11] M. Jersak, Y. Cai, D. Ziegenbein, and R. Ernst. A transformational approach to constraint relaxation of a time-driven simulation model. In *Proceedings 13th International Symposium on System Synthesis*, Madrid, Spain, Sept. 2000.

[12] M. Jersak, D. Ziegenbein, and R. Ernst. A general approach to modeling system-level timing constraints. In *Proceedings 4th Forum on Design Languages*, Lyon, France, 2001.

[13] M. Jersak, D. Ziegenbein, F. Wolf, K. Richter, R. Ernst, F. Cieslok, J. Teich, K. Strehl, and L. Thiele. Embedded system design using the SPI workbench. In *Proceedings 3rd Forum on Design Languages*, Tübingen, Germany, Sept. 2000.

[14] E. A. Lee et al. Overview of the ptolemy project. Technical report, University of California at Berkeley, Mar. 2001.

[15] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[16] C. L. Liu and J. W. Layland. Scheduling algorithm for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20, 1973.

[17] The MathWorks. *Simulink*. http://www.mathworks.com.

[18] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proc. of Design, Automation and Test in Europe Conference (DATE'02)*, Paris, France, Mar. 2002.

[19] F. Slomka, M. Dörfel, and R. Münzenberger. Generating mixed hardware/software systems from SDL specifications. In *Proc. 9th International Symposium on Hardware/Software Co-Design (CODES 2001)*, Copenhagen, Denmark, Apr. 2001.

[20] Telelogic. *Tau*. http://www.telelogic.com/products/tau/.

[21] K. W. Tindell. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, Mar 1994.

[22] 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.

[23] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich. SPI – A system model for heterogeneously specified embedded systems. *to appear in IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2002.