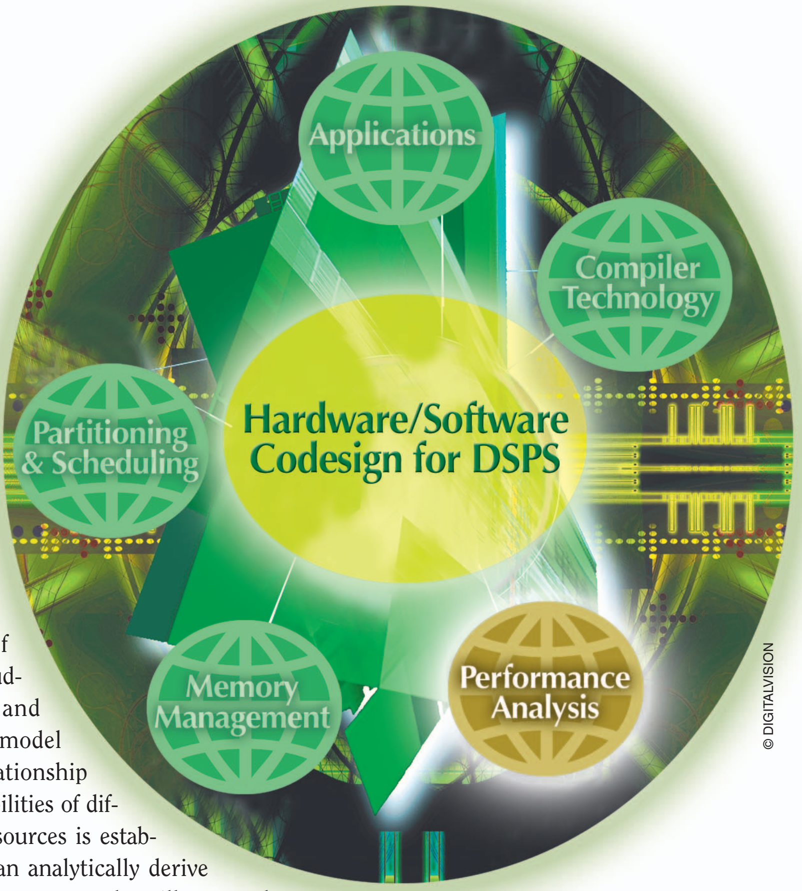
[Lothar Thiele, Ernesto Wandeler, and Samarjit Chakraborty]

Performance Analysis of Multiprocessor DSPs

[A stream-oriented component model]



This article introduces a composable component model that can be used for performance analysis of complex digital signal processing (DSP) systems consisting of several computation resources communicating via some shared media. We review different methods of analyzing the performance of such communication-centric systems, including simulation, trace-based simulation, and symbolic methods. Finally, a component model that can be used to formally state the relationship between the usage and the processing capabilities of different computation and communication resources is established. Using this component model, we can analytically derive performance measures of complete DSP systems. We then illustrate the applicability of this component model with an example.

MOTIVATION

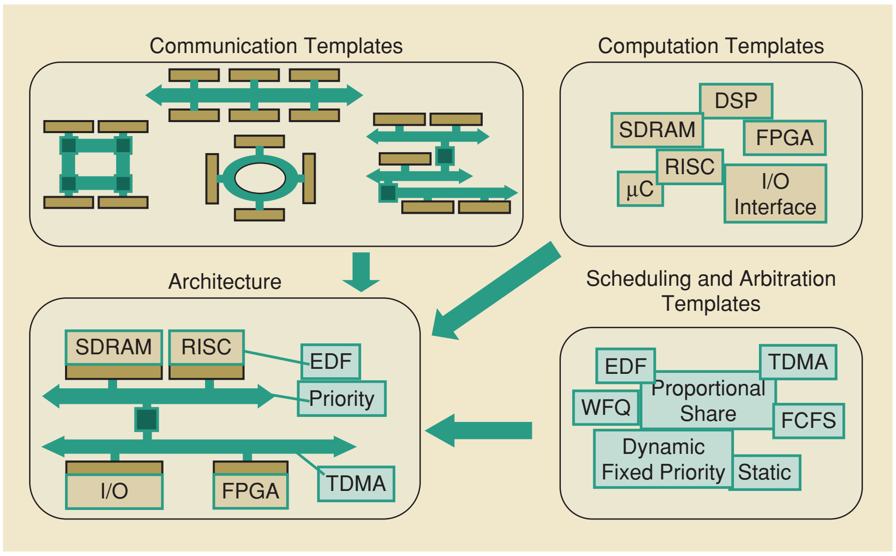
Complex DSP systems are today comprised of a heterogeneous combination of different hardware and software components such as CPU cores, dedicated hardware blocks, different kinds of memory modules and caches, various interconnections and I/O interfaces, run-time environments, and drivers. They are integrated on a single chip, or coupled via some communication network, and run specialized software to perform a dedicated function. The process of determining the optimal hardware and software architecture for such processors includes issues such as resource allocation, partitioning, and design space exploration.

Typically, the questions faced by a designer during a system-level design process include (as shown in Figure 1): Which functions should be implemented in hardware and which in software (partitioning)? Which hardware components should be chosen (allocation)? How should the different functions be mapped onto the chosen hardware (binding)? Do the system-level timing properties meet the design requirements? What are the different bus utilizations,

1053-5888/05/$20.00©2005IEEE

and which bus or processor acts as a bottleneck? There are also questions related to the on-chip memory requirements, off-chip memory bandwidth, and the expected power consumption of the system.

One of the major challenges in the design process is to evaluate essential characteristics of the final implementation early in the design stage. This can help in making important design decisions before investing too much time in detailed implementations. Three main problems arise in the case of complex DSP systems. First, the architecture of such systems, as already mentioned, is highly heterogeneous; the different architectural components are designed assuming different input event models and use different arbitration and resource-sharing strategies. This makes any kind of *compositional analysis* difficult. Second, stream-based DSP applications very often rely on a high degree of concurrency. Therefore, there are multiple control threads, which additionally complicate timing analysis. And third, we cannot assume that a DSP system only needs to process periodic events where a fixed number of bytes is associated with each event. If, for example, the event stream represents a sampled voice signal, then after several coding, processing, and communication steps, the amount of data per event and the timing may have changed substantially. In addition, the data streams to be processed often arrive over a network. In this case, packet processing needs to be integrated with media processing. Therefore, we not only have to deal with constant data rate assumptions and properties but also with event streams that are sporadic or bursty. Examples include the ability to react to external events or to deal with best-effort traffic for coding, transcription, or encryption.



[FIG1] Representation of template-based design involving communication, computation, and scheduling.

There is a large body of work devoted to system-level performance analysis of embedded system architectures (see [1] and the references therein). Currently, the analysis of such heterogeneous systems is mainly based on simulation (using SystemC or tools like VCC [2]) or trace-based simulation, such as in [3]. The main advantage of using simulation as a means for performance analysis is that many dynamic and complex interactions in an architecture can be taken into account; such interactions are otherwise difficult to model analytically. However, in terms of *guaranteeing correctness*, simulation-based approaches do not reveal worst-case bounds on essential properties like delay of events in an event stream, throughput, and memory. In addition, they often suffer from long running times (depending on the accuracy aimed for) and from high set-up effort for each new architecture and mapping to be analyzed.

To guarantee certain timing properties within reasonable analysis times, formal analysis based on abstract system models is necessary. Analytical performance models for DSP systems and embedded processors were proposed in [4] and [5]. Here, the computation, communication, and memory resources of a processor are all described using simple algebraic equations that do not take into account the dynamics of the application (variations in resource loads and shared resources). These "back-of-the-envelope" analytic approaches are often not accurate enough. Therefore, the estimation results show large deviations from the properties of the final implementation.

Other approaches restrict the model of computation; for example, the model can be restricted to synchronous dataflow or extensions thereof (see [6] and [7] for examples). In this case, accurate analysis results can be obtained, but with the price of a limited application domain. Although there has been some work [8] on analyzing special classes of heterogeneous and distributed embedded systems, these methods do not work in a general setup.

Another approach is to define internal models of computation as the basis for performance analysis and design space exploration, as in [9]. These models are based on the notation of intervals to describe important system properties. Yet, so far, a comprehensive method for performance analysis, which is a necessary requisite for scheduling and mapping, is missing. In [10], the authors propose a method for early performance analysis that is based on classical scheduling results from hard real-time system design. In particular, they use different well-known abstractions of the timing behavior of event streams and provide additional interface between them. In [11] and [12], a unifying approach to performance analysis based on real-time calculus was proposed. Real-time calculus itself is based on network calculus [13]. It is particularly suited for modeling and reasoning about event streams and performance guarantees, and it generalizes a number of well-known event models from the real-time systems domain. The unifying approach proposed in [11] and [12] is based on a generalized event model that allows the modeling of hierarchical scheduling and arbitration. It can be used to model both computation and communication resources. This approach provides deterministic worst-case guarantees, but other methods based on stochastic models of resources and event streams, such as Petri nets also exist [14], [15].

Despite these advances, no methodology for performance analysis exists that integrates seamlessly with the current component-based approach to system synthesis. Whereas it is possible to combine components of a functional system specification (such as using heterogeneous models of computation such as in Ptolemy [16], System C, or even UML component models), this is not possible with analytical models for performance analysis. As a result, performance analysis today is largely detached from a hierarchical design flow.

ONE OF THE MAJOR CHALLENGES IN THE DESIGN PROCESS IS TO ESTIMATE ESSENTIAL CHARACTERISTICS OF THE FINAL IMPLEMENTATION EARLY IN THE DESIGN STAGE.

This article serves to unify some of the above mentioned approaches by providing a framework into which different analysis methods can be embedded. There are several composition properties of this new framework:

- *Method composition*: Different parts of the system can be modeled using different performance analysis methods.
- *Process composition*: If a stream needs to be processed by several consecutive application processes, the associated performance components follow the same structural composition.
- *Scheduling composition*: Within one implementation, different scheduling methods can be combined, even within one computing resource (hierarchal scheduling). The same property holds for the scheduling and arbitration of communication resources.
- *Resource composition*: One implementation can consist of different heterogeneous computing and communication resources. They can be composed in much the same way as processes and scheduling methods.
- *Building components*: Combinations of processes, associated scheduling methods, and architecture elements can be combined into performance components. This way, one can associate a performance component to a combined hardware/OS/software module of the implementation, which exposes the performance requirements but hides internal implementation details.

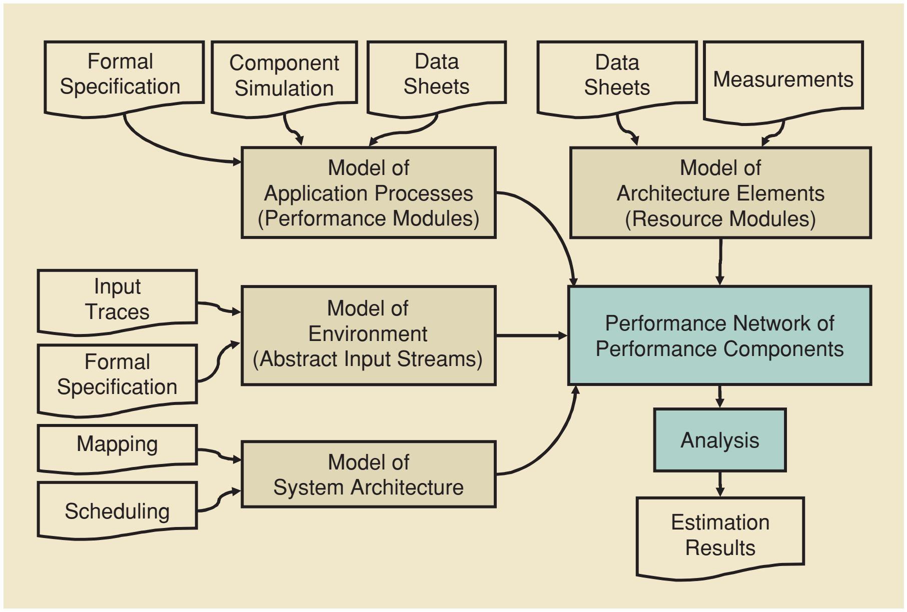
FRAMEWORK FOR MODULAR PERFORMANCE ANALYSIS

We will describe the framework by starting with well-known functional specifications of an application. This framework allows the inclusion of several principally different analysis methods (based on stochastic models, real-time calculus, or classical real-time models). To be concrete, we will use real-time calculus as a running example (e.g., [17] and related stream-processing applications in [11] and [18]). Figure 2 shows an overview of the whole approach.

To understand the basic concept behind this approach, a system architecture may be viewed as a network of heterogeneous architectural elements (representing different kinds of processors, communication resources). A stream of data to be processed by this architecture flows from one architecture element to the next and gets processed by the application procedures mapped onto these elements. As the stream propagates through this system architecture, it gets more and more complex (for example, its "burstiness" increases) because of the different resource-sharing strategies used on the different architecture elements, as well as because of the possibly variable execution times of each element. As a result, very soon the timing properties of the stream get too complex to be analyzed using standard models from the real-time systems area, like periodic or sporadic event models.

The framework proposed in this article is based on the above stream-oriented model. It is comprised of four steps:

1) *Abstraction*: In this step, the different aspects of the system under consideration must be converted into corresponding abstractions. Event streams will be converted into abstract event streams that carry their impor-


Formal Specification
Component Simulation
Data Sheets
Data Sheets
Measurements
Model of Application Processes (Performance Modules)
Model of Architecture Elements (Resource Modules)
Input Traces
Formal Specification
Model of Environment (Abstract Input Streams)
Performance Network of Performance Components
Mapping
Scheduling
Model of System Architecture
Analysis
Estimation Results

[FIG2] **Elements of analytic performance analysis.**

tant timing properties such as burstiness, jitter, and rate information. In a similar way, we will convert the properties of the architecture elements into an abstract representation denoted as resource modules. Finally, we need to determine performance modules that abstract important properties from the given application processes. While application processes work on event streams, the new performance modules will transform the abstract event streams.

2) *Performance components*: The performance modules described above will be combined into larger components. This way, we can take into account resource sharing on common architecture elements. Here, arbitration and scheduling schemes are taken into account.

3) *Performance network*: The different performance components can now be combined into a performance network that represents the performance aspects of the whole system, including the abstract representation of the input streams as provided by the system environment.

4) *Analysis*: Finally, an analysis method must be developed that provides end-to-end performance properties based on the performance network.

At this point, we will introduce an example that will be used throughout the article.

EXAMPLE 1

Let us suppose that a set of input streams must be processed. In this case, the functional specification of the application consists of application processes that operate on event streams. Figure 3 represents a simple example. We have two streams *a* and *b* that enter through input processes *a*0 and *b*0 and leave through output ports *a*8 and *b*4. As in process networks, the event streams consist of ordered sequences of timed events, where the events may have data objects associated with them. The processes are connected via directed links. These links contain unbounded first-in, first-out (FIFO) queues for storing events until they are consumed by the receiving component. At this point, we will be not specific about the operation of the components, such as whether they have blocking or nonblocking behavior or how many events they consume or emit at each firing.

We also need a specification of the architecture. In the lower part of Figure 3, we see as architecture elements a bus, a microprocessor μP, a DSP, and an I/O element. Moreover, the edges between the application processes and the architecture elements denote the mapping. Because this simple example has only one bus, all communication is mapped onto it.

THE ROLE OF ABSTRACTION

In the case of the conventional functional specification of an application, the application processes are usually described by their functional behavior. This specification can then be used to either implement the functionality in hardware/software or perform a simulation. This type of representation is common in the design of complex DSP systems, and it is supported by many tools and standards [for example, unified modeling language (UML)]. The whole functionality is partitioned into a set of components that are communicating via explicit ports and links.

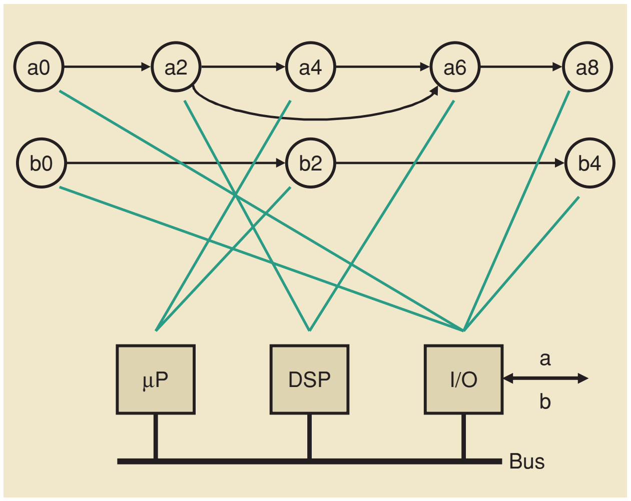
TO GUARANTEE CERTAIN TIMING PROPERTIES WITHIN REASONABLE ANALYSIS TIMES, FORMAL ANALYSIS BASED ON ABSTRACT SYSTEM MODELS IS NECESSARY.

But since we want to analyze the performance of the application on a specific architecture, we need to 1) consider the timing behavior instead of the functional behavior, 2) abstract from the concrete time domain, and 3) consider the use of common resources (architecture elements). This way, we go from event streams to abstract event streams, from application processes to performance modules, and from architecture elements to resource modules and streams.

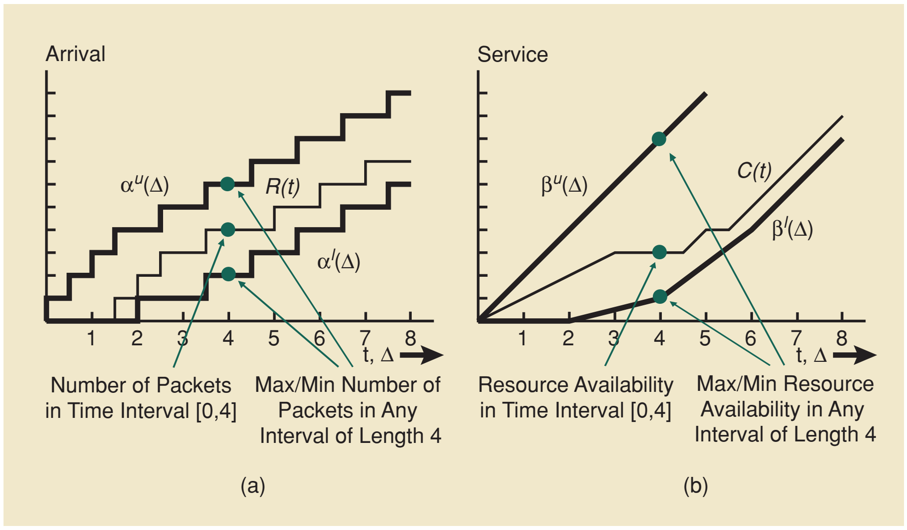
There are several possibilities for this kind of abstraction such as conventional methods for hard real-time systems, as shown in [10], statistical models (Petri nets and Markov processes as in [14] and [15]), or real-time calculus as in [11] and [12]. This way, we move away from simulation and are able to talk about performance analysis independent of concrete input traces.

EXAMPLE 2

As an example of an abstraction of event streams that can be used for performance analysis, let us use the notation of arrival curves, as in [17]. In contrast to event streams as in the upper part of Figure 3, we use a representation that concentrates on the timing in an interval domain. (Note that one can generalize the notation towards events that carry distinguishable objects, see, e.g., [19].) Here, $R(t)$ denotes the number of events that



[FIG3] The upper part of the figure shows a functional specification of the application. The application processes, which are denoted as nodes connected by edges, communicate via event streams. The lower part of the figure shows an architecture specification with its architecture elements.



[FIG4] Abstract interpretation of event streams as (a) arrival curves and architecture elements; in the case of real-time calculus, as (b) service curves.

arrive in the time interval $[0, t]$. The upper and lower arrival curves denote the abstract event stream and are obtained by using a sliding window of size Δ and counting the maximum and minimum number of events for the whole event sequence. In other words, we have

$$\alpha^l(\Delta) \leq R(t+\Delta) - R(t) \leq \alpha^u(\Delta) \quad \forall t \geq 0, \Delta \geq 0.$$

There are several ways of obtaining arrival curves, such as from a set of concrete traces, from analytic properties of event streams like periods, bursts and jitter, or from formal representations like timed automata.

The major step towards a component-based analysis methodology is to make the flow and usage of resources a first-class citizen of the model. As a result, we will be able to compose resources and schedule methods in the same way as we compose application processes. Again, we will exemplify this approach by using real-time calculus, but hierarchical scheduling as in [20] can be unified in a similar way.

EXAMPLE 3

We will present a possible approach for representing the availability of computation and communication resources in architecture elements. It is adapted to the notion of arrival curves, but similar representations can also be found in the case of statistical performance analysis methods. The function $C(t)$ represents the available resource units in the time interval $[0, t]$, such as the number of processing cycles in case of a computing resource or the number of bytes that can be transmitted by a communication link. We define the service curve of an architecture element as the upper and lower bound on the available number of resource units in any time interval of a certain length (sliding window), as shown in Figure 4:

$$\beta^l(\Delta) \leq C(t+\Delta) - C(t) \leq \beta^u(\Delta) \quad \forall t \geq 0, \Delta \geq 0.$$

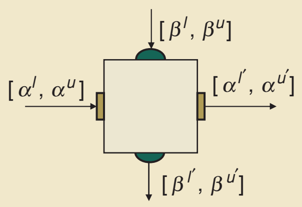
In particular, for each architecture element, an abstract representation in the form of a service curve can be associated. It can be determined using various approaches like investigation of data sheets or transformation from formal specifications.

Finally, we need to abstract application processes to performance modules. They represent the performance aspects of a process that is executed on a resource. At the same time, the use of the resource is also modeled. In other words, they transform the abstract event streams and the abstract resource streams. Through this approach, we provide the necessary means for a composable performance analysis. In contrast to conventional approaches, we are able to represent that an application process changes the timing properties of event streams (as in adding delay, jitter, or bursts) and that it changes the resource that is available to other communication or computation processes. In addition, as we will see later on, it is possible

$$v(\Delta) \wedge w(\Delta) = \min\{v(\Delta), w(\Delta)\}$$

$$v(\Delta) \underline{\oplus}\, w(\Delta) = \inf_{0\leq\lambda\leq\Delta}\{v(\lambda) + w(\Delta-\lambda)\} \qquad v(\Delta) \underline{\oplus}\, w(\Delta) = \inf_{0\leq\lambda}\{v(\Delta+\lambda) - w(\lambda)\}$$

$$v(\Delta) \overline{\oplus}\, w(\Delta) = \sup_{0\leq\lambda\leq\Delta}\{v(\lambda) + w(\Delta-\lambda)\} \qquad v(\Delta) \overline{\oplus}\, w(\Delta) = \sup_{0\leq\lambda}\{v(\Delta+\lambda) - w(\lambda)\}$$



$$\alpha^{u'} = ((\alpha^u \underline{\oplus}\, \beta^u) \overline{\otimes}\, \beta^l) \wedge \beta^u$$

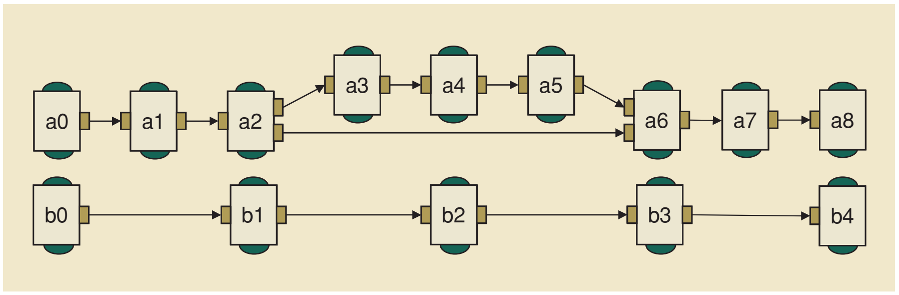
$$\alpha^{l'} = ((\alpha^l \overline{\otimes}\, \beta^u) \underline{\oplus}\, \beta^l) \wedge \beta^l$$

$$\beta^{u'} = (\beta^u - \alpha^l) \underline{\otimes}\, 0$$

$$\beta^{l'} = (\beta^l - \alpha^u) \overline{\oplus}\, 0$$

[FIG5] A set of equations that describe the processing of abstract event and resource streams through a simple application process that uses an input buffer to store waiting requests and processes events in a greedy manner. In the lower left corner, the graphical representation of a performance module is shown.

to take into account scheduling and arbitration policies by combining performance *modules* with performance *components*. To understand this in more detail, we provide an example in terms of real-time calculus.



[FIG6] **Performance modules related to the functional specification of an application according to Figure 3.**

EXAMPLE 4

In the case of real-time calculus, the path from an application process to equations that transform abstract event and resource streams is well known (see, for example, [11] and [12]). For the sake of giving an example, find the relations that describe the processing of a single analysis component enclosed in Figure 5. In this very simple model, it is assumed that the application process executes for a certain amount of time for each event and, as a result, a new event is emitted from the application process. There are more complicated models possible that take into account the fact that events have types and that the application process has an internal state and behaves differently for each event type (see, for example, [21]).

The lower left panel of Figure 5 shows a graphical representation of a performance module.

PERFORMANCE COMPONENTS

So far, the mapping of the application to an architecture and the consideration of scheduling or arbitration of finite resources is still missing. Conventionally, this step is conducted in a nontransparent manner.

To this end, we will combine the performance modules described above with performance components. They have the same type of interface as performance modules in that they process abstract event streams and abstract resource streams. In general, performance components may contain modules and other components. This concept provides hierarchy and composability.

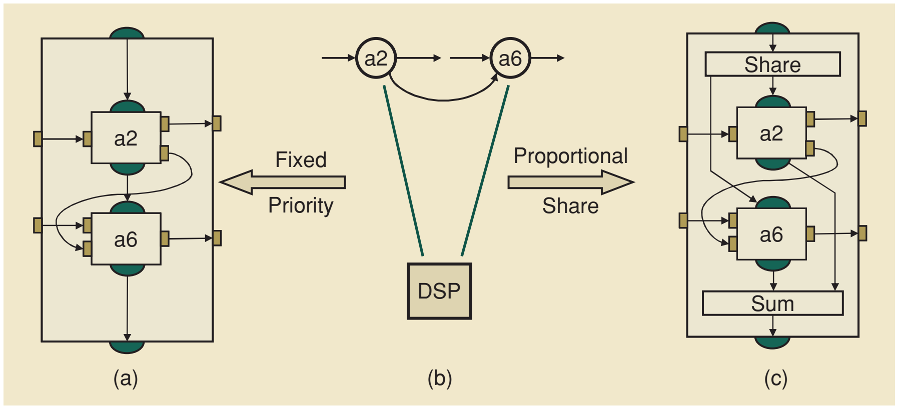
EXAMPLE 5

From the functional specification given in Example 1 and Figure 3, we derive the network of performance modules shown in Figure 6. In this performance network, we did not only model the computation components of Figure 3 (in our case components $a0$, $a2$, $a4$, $a6$, $a8$, $b0$, $b2$, and $b4$), but the communication links that connect these computation components are also modeled (components $a1$, $a3$, $a5$, $a7$, $b1$, and $b3$). In this way, we can seamlessly consider both computation and communication aspects in a performance analysis.

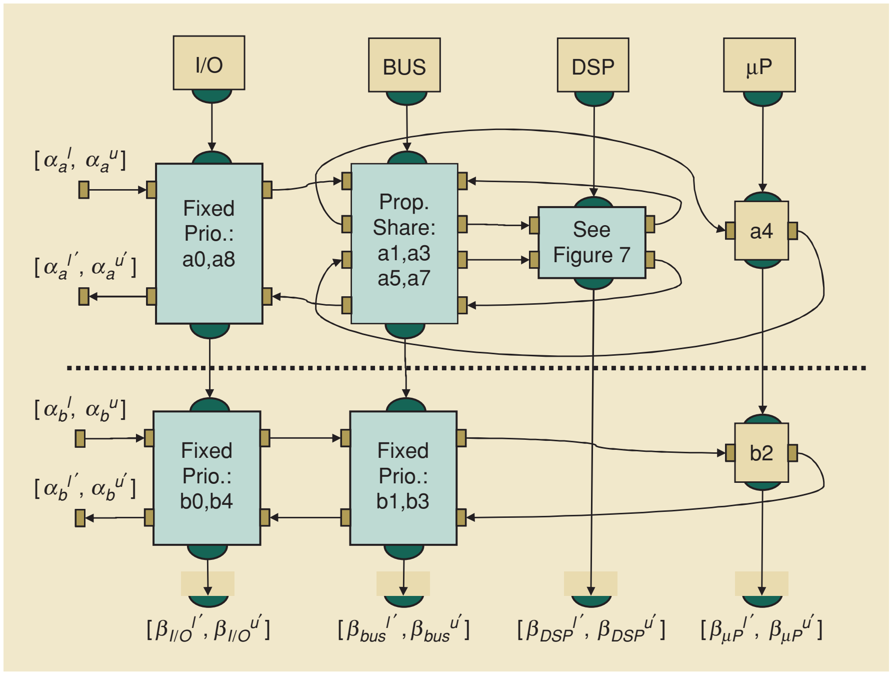
Now, let us first suppose that the processes $a2$ and $a6$ from Figure 3 are mapped to the architecture element DSP using a preemptive fixed priority scheme where $a2$ has a higher priority than $a6$. From the manner in which this scheduling algorithm works, one can easily deduct the composition of the corresponding performance modules to a performance component, as shown in Figure 7(a). In Figure 7(b), the component structure in case of a proportional share (GPS) algorithm is shown where each application process receives a fixed portion of the available capabilities. Note that the initial abstract resource stream enters the performance component from the top, whereas the remaining resource stream leaves from the bottom. It can be used by other components, such as for processing the abstract event stream b. Further details and other scheduling policies are available in [11], [12], and [19].

The performance components can be used in various respects:

- They can be combined to perform an end-to-end performance analysis of the system level (see the section titled "Performance Network").
- A provider of a hardware/software component that will be integrated by a system house into a complete system can provide such an analysis component model. This way, the system house can carry out a performance analysis of the overall system without having access to the details of the hardware/software components.
- In the above scenario, the system house can even add functionality and check for the performance of the combined



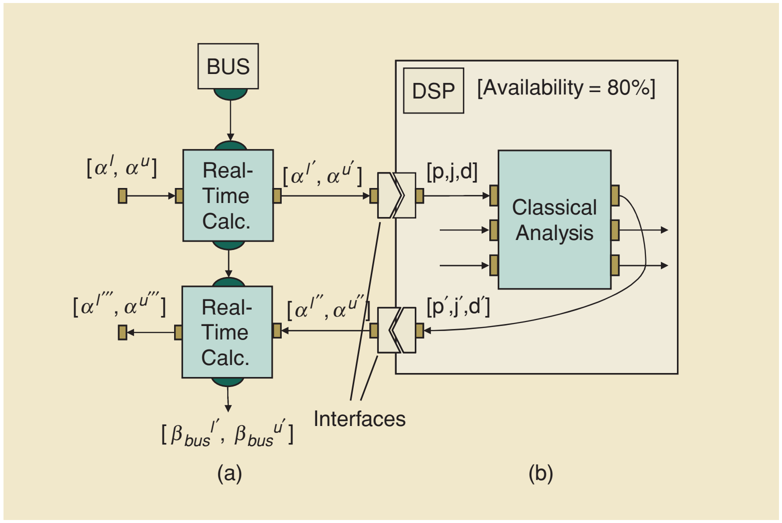
[FIG7] **(a)–(c) Combination of basic performance analysis components with a larger analysis component that reflects resource sharing. (a) Represents a subnetwork for fixed priority sharing where *a2* has a higher priority than *a6*. (c) Shows a generic component that implements the proportional share policy.**


I/O
BUS
DSP
µP
$[\alpha_a^l, \alpha_a^u]$
$[\alpha_a^{l'}, \alpha_a^{u'}]$
Fixed Prio.: a0,a8
Prop. Share: a1,a3 a5,a7
See Figure 7
a4
$[\alpha_b^l, \alpha_b^u]$
$[\alpha_b^{l'}, \alpha_b^{u'}]$
Fixed Prio.: b0,b4
Fixed Prio.: b1,b3
b2
$[\beta_{I/O}^{l'}, \beta_{I/O}^{u'}]$
$[\beta_{bus}^{l'}, \beta_{bus}^{u'}]$
$[\beta_{DSP}^{l'}, \beta_{DSP}^{u'}]$
$[\beta_{\mu P}^{l'}, \beta_{\mu P}^{u'}]$

[FIG8] An example of a performance analysis network. The shaded boxes represent components as, for example, those shown in Figure7.

system. Both of these scenarios can be found frequently in industry; a prominent example of such a business relationship is the automotive industry.

The performance components defined above can now be combined into a performance network that represents the whole process structure, the mapping of the available computation and communication resources, and the scheduling policies that have been applied. The next section will also show how important performance measures can be determined through formal analysis of the network.



[FIG9] Performance network with performance components of two different analysis methods, namely (a) real-time calculus performance components and (b) a classical methods performance component. Interfaces that transform abstract event models are depicted between the performance components.

PERFORMANCE NETWORK

So far, we have been building performance components using information about the application structure, the mapping of the elements of the architecture, and the scheduling policy used. We will now show how these components can be combined into a network to conduct a performance analysis of the whole system.

EXAMPLE 6

We will use the same setup as in Example 1 and Example 5. We suppose that application processes *a2* and *a6* are mapped onto the DSP module (proportional share), *a4* is mapped onto the μP module, *a0* and *a8* are mapped to the I/O processor (fixed priority), and *a1, a3, a5,* and *a7* are mapped onto the bus using proportional share arbitration [e.g., time division multiple access (TDMA)]. The corresponding performance network is depicted in the upper half of Figure 8. The shaded boxes represent performance components that consist of several modules combined according to a scheduling or arbitration method. Then, we suppose that the processing of stream *b* is conducted using the remaining processing and communication capabilities. This scenario may reflect a situation where a hardware/software module is provided that processes some stream *a* and, later on, the processing of *b* is added with a lower priority to not influence the existing application.

Finally, we need to analyze the network shown in Figure 8. To this end, we need to provide the following:

1) specification of the abstract input streams, i.e., the arrival curves *a* and *b* (from the left of Figure 8)
2) specification of the initial resource capabilities, i.e., the service curves associated with the hardware modules (from the top of Figure 8)
3) a set of relations that describe how the abstract event and resource streams are processed by a component (from Example 4 and Figure 5)
4) a way to compute important properties from the abstract event and resource streams, such as delay, jitter and memory demand.

With this list, we have provided means to determine steps 1, 2, and 3. By solving the equations for the whole network (involving fixed-point calculations), the characterization of all abstract streams is obtained. From this information, delay, jitter, and memory requirements can easily be extracted.

OTHER ANALYSIS METHODS

Throughout this article, real-time calculus has been used as a running example of an analysis method in the presented framework of modular performance analysis. Looking at real-time calculus, there are methods available to handle blocking and nonblocking communication schemes, as well as a variety of scheduling and abstraction methods in addition to the ones presented here (see [11] and [12] for examples). There are also many possibilities to obtain the necessary models and parameters from a given specification, such as by using data sheets, simulations of basic components, or formal models like timed automata. However, the presented framework is not restricted to the application and architecture models given so far or to real-time calculus as the analysis method. Rather, it is possible to use other models and analysis methods within this framework, which can be tailored towards specific application areas. And as mentioned earlier, it is even possible to use several analysis methods in a single performance network, such that different parts of a system can be modeled and analyzed using different methods.

THE MAJOR STEP TOWARDS A COMPONENT-BASED ANALYSIS METHODOLOGY IS TO MAKE THE FLOW AND USAGE OF RESOURCES A FIRST-CLASS CITIZEN OF THE MODEL.

To be suitable for the presented framework, a new analysis method must first define appropriate abstractions for the different elements used in the framework; that is, it must define abstract event streams, abstract resource modules, and abstract performance modules. These models should be closely linked to the underlying model of computation of the application area that the analysis method is being tailored towards. Examples include the adaptation of event models like *periodic with jitter* for use in context with classical methods developed in the real-time systems community or stochastic models in context with stochastic analysis methods. Further, the analysis method must define how performance modules are combined into performance components and, finally, how these components can be analyzed.

These new performance components can then be combined into a performance network, either together with performance components of the same analysis method or with performance components of other analysis methods. In the latter case, additional interfaces must be defined that transform the abstract element models of one analysis method into the abstract element models of another analysis method. Depending on the level of detail included in the different abstract models, these transformations may be lossy.

EXAMPLE 7

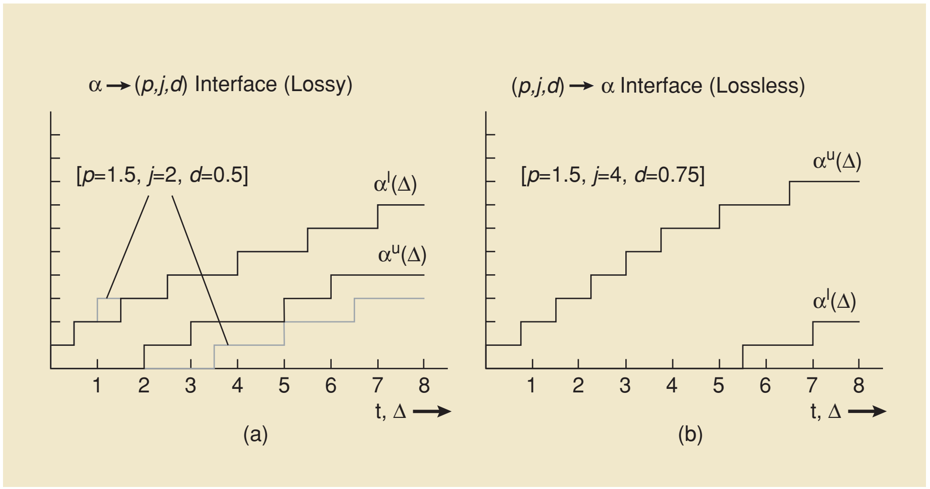
Let us assume a performance network with performance components that use real-time calculus as the analysis method. Suppose we then want to add a performance component that models a DSP and uses classical methods for performance analysis. As the event model for this classical methods performance component, periodic with jitter is used, where an event stream is modeled by the three parameters period (p), jitter (j), and minimum event-inter-arrival-time (d). For classical methods, no composable resource models exist; instead, a set of performance modules is typically assured a certain constant share of the overall resource.

The resulting performance network that combines both real-time calculus performance components and the classical methods performance component is shown in Figure 9. Also depicted in this figure are the interfaces needed to transform the abstract event models between the two incorporated analysis methods.

Figure 10 shows two examples of the interfaces used in Figure 9. In Figure 10(a), a transformation from arrival curves into the periodic with jitter event model is shown. Since the periodic with jitter event model is more coarsely grained, it cannot include all details of the arrival curve event model. Hence, this transformation is lossy. On the other side, the transformation from the periodic with jitter event model into arrival curves is lossless, as can be seen in Figure 10(b).

CONCLUSION

In comparison to known approaches, the framework described here leads to a component-based approach to the analysis of system



[FIG10] Interfaces that transform arrival curves into the periodic with (a) jitter event model and (b) vice versa. The transformation from arrival curves to the periodic with jitter event model is lossy (the shown (p,j,d)-parameters model the gray arrival curves instead of the actual black arrival curves), while the inverse transformation is lossless.

properties, thereby increasing our understanding of the behavior of complex DSP systems and computing in general.

To perform a system-level timing analysis of a heterogeneous, stream-based processing architecture, where different parts of the application are mapped to different kinds of architecture elements, we envisioned a novel analytical framework. It consists of abstract models for the system architecture and the architecture elements to model the underlying hardware (the processing and the communication architecture), the scheduling or arbitration schemes, and the event streams. In particular, the individual system element modules are combined to a performance network that describes essential system properties such as delay and memory. The framework can be used to analyze complex and heterogeneous architectures and to answer questions related to timing and other system properties (like the load on various processors and buses) in a single, unified manner.

Since the approach is based on abstract models of applications/algorithms and architectures, a given scenario can be analyzed within a short time. This is in sharp contrast to simulation-based approaches, which usually achieve better accuracy but often require hours of simulation time.

The analytic approach is intended to complement existing estimation methods, not to replace them. Any automated or semi-automated exploration of different design alternatives (design space exploration) could be separated into multiple stages, each having a different level of abstraction. It would then be appropriate to use an analytical performance evaluation framework, such as the one described in this article, during the initial stages and resort to simulation only when a relatively small set of potential architectures is identified.

AUTHORS

Lothar Thiele is a full professor of computer engineering at the Swiss Federal Institute of Technology, Zurich. His research interests include models, methods, and software tools for the design of embedded systems, embedded software, and bioinspired optimization techniques. He received the Dissertation Award from the Technical University of Munich in 1986, the Outstanding Young Author Award from the IEEE Circuits and Systems Society in 1987, the Browder J. Thompson Memorial Award of the IEEE in 1988, and the IBM Faculty Partnership Award in 2000–2001. In 2004, he joined the German Academy of Natural Scientists Leopoldina.

Ernesto Wandeler is a Ph.D. student at the Computer Engineering and Networks Laboratory of the Swiss Federal Institute of Technology, Zurich. His research interests include models, methods, and tools for system-level performance analysis of real-time embedded systems. He received a Dipl. El.-Ing. degree from ETH Zurich in 2003. In the same year, he was awarded the Willi Studer Price for his diploma and the ETH Medal for his diploma thesis, both from the Swiss Federal Institute of Technology, Zurich.

Samarjit Chakraborty is an assistant professor of computer science at the National University of Singapore. He obtained his Ph.D. in electrical engineering from the Swiss Federal Institute of Technology, Zurich, in April 2003. His research interests are in system-level design and analysis of real-time and embedded systems, with an emphasis on architectures for streaming applications. He was awarded the ETH Medal for his doctoral dissertation and the European Design and Automation Association's (EDAA) Outstanding Doctoral Dissertation Award in 2004, in the category "New Directions in Embedded System Design Automation."

REFERENCES

[1] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Englewood Cliffs, NJ: Prentice Hall, 1994.

[2] The Cadence virtual component co-design (VCC) [Online]. Available: http://www.cadence.com/ products/vcc.html.

[3] K. Lahiri, A. Raghunathan, and S. Dey, "Evaluation of the traffic performance characteristics of system-on-chip architectures," in *Proc. Int. Conf. VLSI Design*, Jan. 2001, pp. 29–35.

[4] A. Agarwal, "Performance tradeoffs in multithreaded processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 5, pp. 525–539, Sept. 1992.

[5] M. Franklin and T. Wolf, "A network processor performance and design model with benchmark parameterization," in *Network Processor Design: Issues and Practices*, vol. 1, P. Crowley, M. Franklin, H. Hadimioglu, and P. Onufryk, Eds. San Mateo, CA: Morgan Kaufmann, 2003, ch. 6, pp. 117–140.

[6] E.A. Lee and T.M. Parks, "Dataflow process networks," *Proc. IEEE*, vol. 83, no. 5, pp. 773–799, 1995.

[7] S. Sriram and S.S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization.* New York: Marcel Dekker, 2000.

[8] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event triggered distributed embedded systems," in *Proc. 10th ACM Int. Symp. Hardware/Software Codesign*, 2002, pp. 187–192.

[9] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich, "SPI—A system model for heterogeneously specified embedded systems," *IEEE Trans. VLSI Syst.*, vol. 10, no. 4, pp. 379–389, 2002.

[10] K. Richter, M. Jersak, and R. Ernst, "A formal approach to MpSoC performance verification," *IEEE Computer*, vol. 36, no. 4, pp. 60–67, Apr. 2003.

[11] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert, "Embedded software in network processors—Models and algorithms," in *Proc. 1st Workshop Embedded Software (EMSOFT)* (Lecture Notes in Computer Science 2211). Lake Tahoe, CA: Springer-Verlag, 2001, pp. 416–434.

[12] L. Thiele, S. Chakraborty, M. Gries, and S. Kunzli, "A framework for evaluating design tradeoffs in packet processing architectures," in *Proc. 39th Design Automation Conference (DAC).* New Orleans, LA, June 2002, pp. 880–885.

[13] J.-Y.L. Boudec and P. Thiran, *Network Calculus—A Theory of Deterministic Queuing Systems for the Internet (*Lecture Notes in Computer Science 2050). New York: Springer-Verlag, 2001.

[14] C. Lindemann, *Performance Modelling with Deterministic and Stochastic Petri Nets.* New York: Wiley, 1998.

[15] P. Fortier and H. Michel, *Computer Systems Performance Evaluation and Prediction.* Bedford, MA: Digital Press, Aug. 2003.

[16] E.A. Lee, "Overview of the ptolemy project," Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M01/11, 2001.

[17] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Proc. 6th Design, Automation and Test in Europe (DATE)*, Munich, Germany, Mar. 2003, pp. 10,190–10,195.

[18] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister, "Performance evaluation of network processor architectures: Combining simulation with analytical estimation," *Comput. Netw.*, vol. 41, no. 5, pp. 641–665, Apr. 2003.

[19] E. Wandeler, A. Maxiaguine, and L. Thiele, "Quantitative characterization of event streams in analysis of hard real-time applications," in *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2004, pp. 450–459.

[20] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proc. 24th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2003, pp. 2–13.

[21] E. Wandeler and L. Thiele, "Abstracting functionality for modular performance analysis of hard real-time systems," *in Asia South Pacific Design Automation Conf. (ASP-DAC)*, Jan. 2005, pp. 697–702.

SP