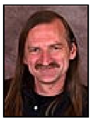
Chip Design
Tools, Technologies & Methodologies

BLOGS

Verification Vertigo



To subdue the enemy without fighting is the supreme excellence

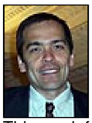
I am excited today to be able to talk about a new product that I had to keep quiet about for some time. First of all some...

EDA Thoughts

Merry Mergers

In 2009 I expect that EDA companies will continue to merge in order to stay financially viable. Here are a few rules and...

Domeika's Dilemma



5 months in multi-core - what has changed?

This week found me in Zurich, Switzerland, delivering a talk to researchers. The purpose of my talk and the other talks at...

Wizards of Microwave

Meet Us IRL (In Real Life) At DesignCon 2009

Despite rumors from some Web 2.0 fanatics that it isn't necessary, you can still actually meet people "IRL" (in real life...

POLL

Where will the device design growth be in ten years?

- Multicore
- Programmable

• ARTICLE

[Printer Friendly]

Published in September / August 2008 issue of Chip Design Magazine

[ESL]Platform-Development Approach Simpli□□?es Mobile-Device Design

Try a service-oriented NoTA method when designing mobile-device platforms.

By Vincent Perrier and Klaus Kronlöf

At the Nokia Research Center (NRC), signi□□?cant research activity is performed in the area of mobile-terminal architectures. The NRC is a separate unit within Nokia (www.nokia.com) and therefore isn't attached to a speci□□?c product-development business unit. For the ITEA model-based approach to real-time embedded systems (MARTES; www.martes-itea.org) European research project, the NRC worked on a mobile-terminal case study. That study focused on communication-centric mobile-terminal architectures, which are designed for the digital-convergence era. In this context, NRC has adopted the MCSE method for architectural modeling. (MCSE is a French acronym for an electronic system co-design methodology, which was developed by Professor Calvez at the University of Nantes, France.) The NRC also used CoFluent Studio as a mobile-device platform-architecture modeling toolset supporting the MCSE method.

Nokia's work in the MARTES project is closely connected to the NRC's own service-oriented architecture concept, which is called Network-on-Terminal Architecture (NoTA). NoTA is an interconnect-centric, modular, service-oriented architecture for current and future mobile-device platforms. It promises to provide superior performance and effective horizontalization via eased integration. The development method associated with NoTA ensures that designs are stepwise veri□□?able against end-user requirements. That method also is flexible and scalable, enabling reuse on different levels.

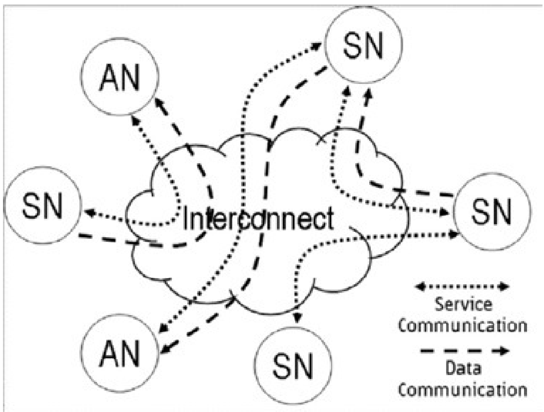


Figure 1: Here is a graphical representation of the NoTA logical architecture.

Speci□□?cally, a NoTA platform consists of loosely connected services running on heterogeneous subsystems. In NoTA-based systems, service and data communication is routed via the network stack. NoTA takes these principles and adapts them for use in a highly embedded system. The NoTA method includes

VISIT THE SYSTEM-LEVEL DESIGN ONLINE COMMUNITY

System-Level Design COMMUNITY

This brand new online community is the destination for embedded system design, verification and debugging of system-on-chip (SoC) designs.

Site includes news, articles, white papers, videos, blogs, polls, ask the expert and other valuable resources provided in an advertising-free environment. VISIT TODAY!

Sponsored by: ARM Mentor Graphics Virage Logic

DESIGN CENTERS

Chips - ASIC and ASSP
Memory
Low Power
Programmable Logic
IP Design, Verification, Integration
Analog/Mixed Signal
Chip-Package-Board
DFM-DFY-Verification
Emerging Technologies
Electronic System Level (ESL)
SOC Interfaces

TECHNICAL PAPERS

ZeBu™: A Unified Verification Approach for Hardware Designers and Embedded Software Developers a platform-development flow, which ensures that services, subsystems, and the interconnect topology are matched to end-user requirements. It also provides formal, reusable speci□□?cations for the platform entities. The NoTA logical architecture consists of three types of foundation elements called application nodes (ANs), service nodes (SNs), and interconnect (see Figure 1).

NoTA de□□?nes two main levels of protocols for the interconnect, H_IN and L_IN. H_IN is a high-level protocol stack that provides communication functionality for platform services and applications. L_IN, the low-level protocol, provides the physical connection between subsystems.

A NoTA subsystem implements a set of services. A subsystem is an architectural concept that doesn't necessarily align with chip boundaries. There may be several subsystems on a chip. In addition, a subsystem may extend outside the boundaries of a chip (see Figure 2).

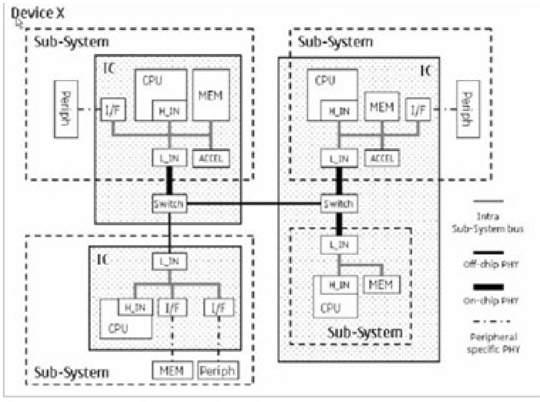


Figure 2: NoTA subsystems are depicted here.

Platform-Architecture Development

The common practice in platform-architecture development is quite informal. It also is heavily reliant upon the system architect's experience. Often, this development is done with spreadsheets that forecast results based on the results that were observed on previous designs. This approach is feasible when changes in successive generations of the architecture are relatively small. Yet such an informal approach becomes problematic when dealing with truly novel architectural concepts, which call for the systematic exploration of widely different alternatives. Furthermore, platform requirements are typically expressed in technical terms that aren't properly connected to end-user needs.

The NoTA platform-architecture development method aims to overcome these pitfalls of informal practices. NoTA-based systems are engineered in a systematic-requirements-driven manner. The NoTA approach is characterized by the following principles:

Separation of concerns: The ability to develop different system aspects independently from each other facilitates reuse. It also improves the ability to manage complexity. In the NoTA method, the following domains are separated:

- End-user requirements
- Platform functionality (i.e., services provided by the platform)
- Platform architecture (i.e., definition of subsystems and communication)
- Infrastructure implementation of subsystems (software and hardware) and interconnect protocols (software and hardware)

Each domain has self-contained models. In the final system, these domains are related to each other. Fixing these relations between domains is postponed until the time at which the system instance is defined. The system instance can be defined as a product or product platform.

Model-based engineering: In the NoTA method, the artifacts developed in different phases of the process are models with well-de□□?ned semantics. This helps to avoid misunderstandings and the consequent errors caused by ambiguousness or hidden meanings of informal documentation. Nokia also requires the availability of analysis, veri□□?cation, transformation, code-generation, and synthesis tools that operate on models.

Reuse of models: Nokia believes that the ability to effectively reuse models in different contexts really improves design productivity compared to conventional methodologies. In the NoTA method, different types of models are stored in repositories. These models can be retrieved and used to compose new system con□□?gurations.

Early validation and veri□□?cation: One motivation behind model-based engineering is the early validation and veri□□?cation of speci□□?cations and designs. In the NoTA method, the validation and veri□□?cation processes start early—at the end-user requirements phase—with executable use-case models. Later, the focus is on the correctness of platform speci□□?cation and performance analysis—both the speci□□?cation and implementation phases. The validation and veri□□?cation in the NoTA method aren't limited to logical correctness. They also cover non-functional aspects, such as real-time performance and energy consumption.

Platform-Architecture Modeling

The NRC has adopted MCSE for architectural modeling in NoTA. According to this method, an architectural model is developed by building the functional architecture or timed-behavioral model (e.g., the functional model of the system with timing information) as well as the platform architecture (executive structure). In addition, the functional blocks are mapped onto the executive structure. The architectural-modeling toolset includes tools that support model creation and mapping according to the MCSE method.

The requirements for a NoTA-based platform come from the end-user requirements, which are expressed as use-case models. The selected collection of use cases is □□?rst studied. All of the services used in the re□□?ned use-case models, which are called primary services, are identi□□?ed. Next, the set of required services is reduced in order to minimize overlap and redundant services. When there are several versions of the same service needed, the version that ful□□?lls all of the requirements is selected. The others are discarded. As a result of this process, the set of required primary services is de□□?ned.

The use-case models—together with the set of required primary services—are used to build the functional architecture model. That model consists of service-node (SN) and application-node (AN) models. An SN model represents an instance of a service. There may be several instances of the same service. For its part, the AN model de□□?nes the way that the application uses the services in a particular use case.

In NoTA, a service is speci□□?ed in a special format called service interface speci□□?cation (SIS). SIS includes the interface signature of the service in question as well as a description of its externally observable behavior. This description is expressed as a □□?nite state machine (FSM). SIS also includes the relevant, non-functional attributes of the service, such as timing and power consumption. In architectural modeling, SN models are derived directly from the SIS. AN models are derived from the execution traces of use-case models.

The platform-architecture model consists of blocks representing the subsystems and routing switches. Mapping the SNs and ANs into the subsystems and de□□?ning the communication-network topology among the subsystems yields the architectural model. The interconnect-node (IN) functionality is integrated into the components of the platform-architecture model. Figure 3 shows how the architectural-modeling method is applied to NoTA using the architectural-modeling toolset.

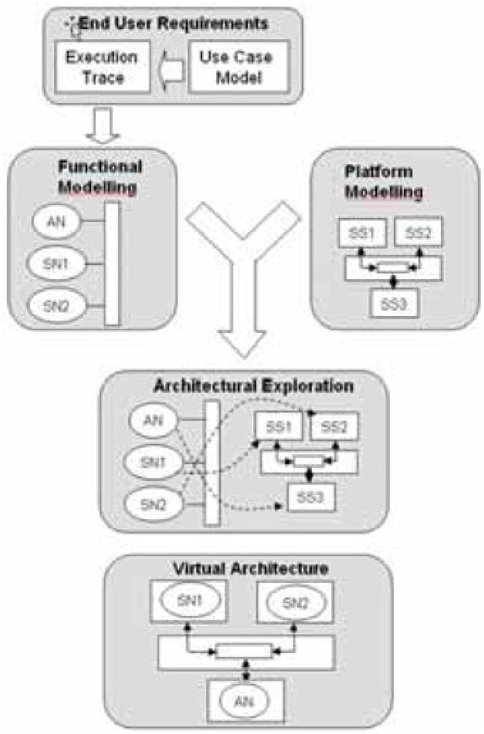


Figure 3: Shown here is the architectural exploration flow.

Functional Architecture
The functional or timed-behavioral model describes the system's logical partitioning and behavior. The functional model in NoTA consists of the ANs and SNs. The SNs include all of the primary services of the use case as well as any additional secondary services that are used by the primary services.

The functional editing tool in the architectural-modeling toolset captures the graphical description of the SNs and ANs. The internal model of each SN is derived directly from the corresponding SIS. The behavior of the AN is de□□?ned by the use case utilizing the generated XML trace. The service requests to the SNs are modeled as messages. All service requests in NoTA are passed over the IN. The functional behavior of a system consisting of the ANs, SNs, and an ideal IN can be simulated independently without a de□□?nite platform architecture.

In NoTA, the behavior of the SNs is modeled as FSMs. They must be represented as either SystemC models or as graphical functional models within the toolset. Functions are lower-hierarchical-level models used as "black boxes" within the model. The behavior of these functions can be further de□□?ned with algorithms written in C or C++. NRC has adopted the latter approach with additional C++ algorithms to read in and interpret the XML □□?les. A parser module extracts the needed information from the XML model and assigns it to the correct placeholders in the SN graphical template.

There are two main types of communication in the NoTA network. Models for the data object communication are added to the SNs in order to get insight about the data amount between different nodes. In practice, the same functional link is used for both the service communication and data traffc. The type of link and the message that's sent into it are modeled as a C++ class. That C++ class contains □□?elds for routing, message type, and size and sub-classes for the content.

The use-case behavior is imported into the graphical model as an XML-trace □□?le. That □□?le consists of a sequence of service requests with possible additional parameters. The □□?le is read in the ANs. Corresponding service requests are sent to the correct services.

Platform Architecture

The platform model is an abstract representation of the physical architecture. The architectural-modeling toolset provides a set of building blocks for platforms. These generic performance models of physical computing, communication, and storage units can be parameterized by the user. The building blocks include processors, shared memories, signals, and connections that use communication nodes. At this design step, the subsystems are outlined as placeholders for the SNs and ANs. One subsystem consists of a processor in which the SNs are run in parallel. The subsystem's actual implementation isn't modeled. The routing switch (RS) is modeled with a routing model that contains built-in, performance-statistics-gathering functionalities. One processor is reserved for each RS. The subsystems are connected to the RS with communication nodes. The resulting network topology represents the accurate interconnect that's needed in the architectural simulations (see Figure 4).

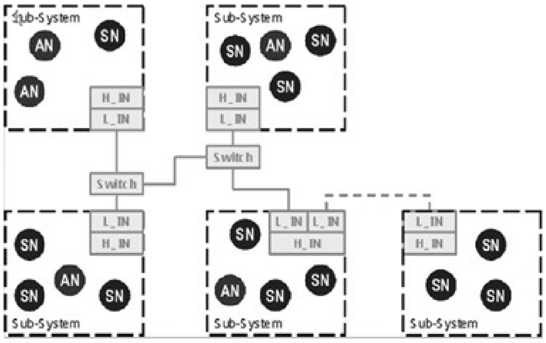


Figure 4: Here is the network topology for architectural simulations.

The SN and AN models contain performance data, such as the time taken to process a service call. The generic hardware-processor model provided within the toolset should therefore be selected as the processor type. That model is capable of running the SN models independently of each other. Later, real software models will be run for certain SNs. At that point, the generic software-processor models—including a generic real-time-operating-system (RTOS) model—become useful. The platform models themselves have tunable parameters that affect the overall system performance. For example, the bandwidths of the RSs can be con□□?gured independently.

Architectural Exploration

The architecture design consists of three main parts: decisions about the number and type of subsystems in the device, the interconnect topology between the subsystems, and the mapping of the SNs and ANs into the subsystems. A subsystem can be de□□?ned as a collection of
SNs that have an IN connection. One subsystem can contain several SNs. For example, a storage subsystem can act as a conventional mass-storage or a streaming-media server. These require completely diffrent services. Because they use the same hardware resources, however, it's bene□□?cial to locate them inside the same subsystem. The SNs can be distributed across the subsystems in several different ways. Accordingly, several architectural con□□?gurations can be evaluated to identify potential bottlenecks and maximize system performance.

Network traffc analysis is a key output in verifying the designed architecture. Each designed architecture needs to be simulated against all of the use cases. There are certain parameters related to the SNs, ANs, INs, and RSs that could be optimized during the architecture design. Local buffer sizes are commonly the most important of such parameters.

The steps within the architectural exploration comprise the following:

1. De□□?ne the number and types of subsystems.
2. De□□?ne the interconnect topology to connect the subsystems. This determines the number and types of routing switches.
3. Map SNs and ANs into the subsystems.
4. Run simulations against the original use cases.
5. Analyze results.
6. Go back to Step 3 to optimize the current architecture.
7. Go back to Step 1 to try different architectures.

8. Choose the most optimal case(s).
9. Select the virtual architecture.

After the exploration of different architectures within the toolset is complete and the optimal case is selected, the architecture solution consists of the aspects listed below. These parts set the requirement speci□□?cations for each subsystem:

- A set of subsystems connected together with a certain interconnect topology
- Mapping of the decomposed use-case-originated services into the above subsystems
- Veri□□?ed and re□□?ned performance parameters for the services
- All of the above veri□□?ed against the original end-user use case

Meta-Model-Based Integration

As part of their tool-enhancement work in MARTES, Telelogic and CoFluent Design implemented a meta-model-based integration of their tools using the Eclipse Modeling Framework (EMF) and its Ecore format. This technology also has been used to implement other MARTES meta-model-based tool integrations in the project.

The tool integration provides an alternative way to transfer use-case behavior between the tools. The idea is to transfer the whole executable use-case model instead of its execution trace. The Telelogic Tau UML modeler is used purely for the application model. Real-time performance attributes can be added to the UML model as tagged values of the UML pro□□?le. Although that pro□□?le is speci□□?c for this purpose, it can be regarded as an adaptation of the MARTES application model. One important bene□□?t of transferring the whole model is that users can now deal with feedback from the platform-architecture model. 佯at feedback will potentially affect use-case behavior.

A video-player case-study example is presented here to explain how the model transfer works (see Figure 5). In this example, services are modeled as state machines with performance attributes. 佯e application node contains a state machine, which is modeling its behavior in a particular use case. Modeling is done in a two-level structure: use-case level and service level. Several use cases can be examined by changing the application node.

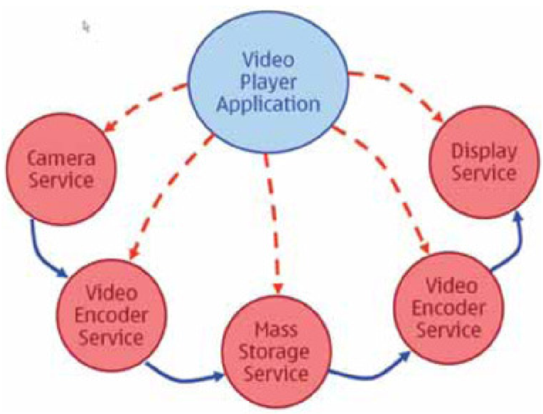


Figure 5: This example shows how the model transfer works.

The application model of the example contains ・・?ve classes: Camera, Display, VideoEncoder, VideoDecoder, and MassStorage. It also houses the signals that are exchanged between them. The top-level structure of the application model is de・・?ned by a composite structure diagram (see Figure 6). The behaviors of the classes are modeled as state machines (see Figure 7). Use-case composition also is achieved in UML (see Figure 8).

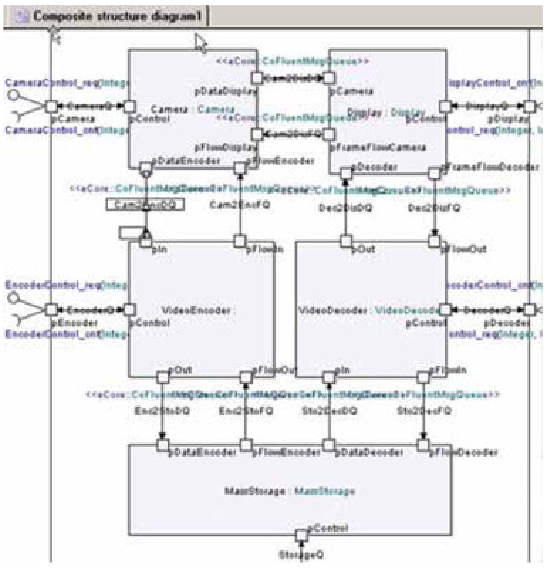


Figure 6: This top-level structure diagram defines the application model's top-level structure.

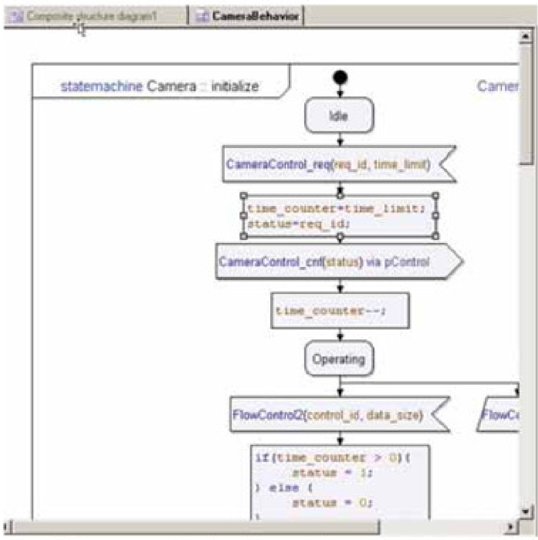


Figure 7: The state-machine-flow diagram depicts the behaviors of the classes.

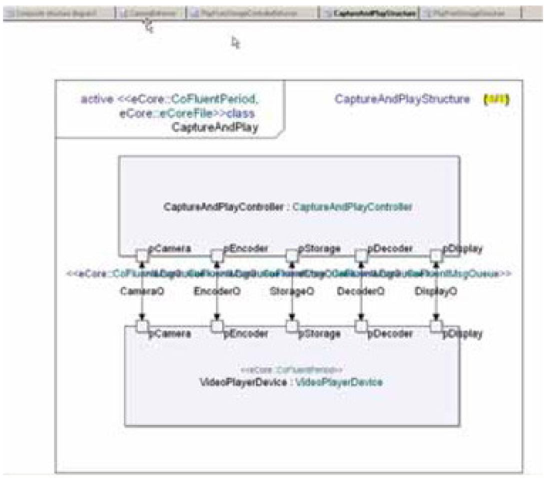


Figure 8: Use-case composition may be achieved in UML.

The stereotypes of the UML pro□□?le that's used in the tool integration contain tagged values. These values enable the setting of performance attributes used by CoFluent Studio. Figure 9 shows an example of how to set those attributes in Tau. The resulting model is executable and can be simulated in Tau (see Figure 10). This simulation is untimed, as the UML simulator doesn't deal with real-time performance properties.

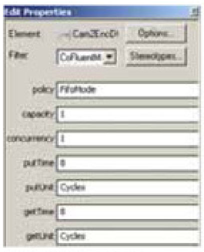


Figure 9: Performance attributes may be set in Tau.

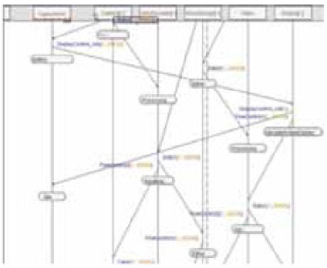


Figure 10: Here, untimed UML simulation is performed in Tau.

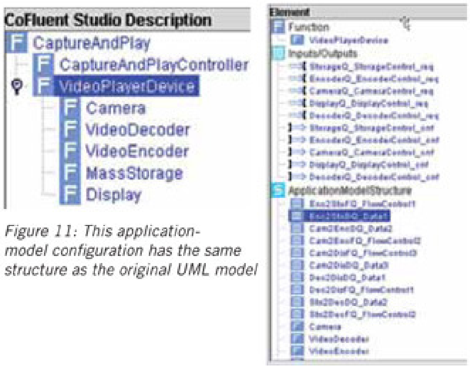


Figure 11: This application-model configuration has the same structure as the original UML model

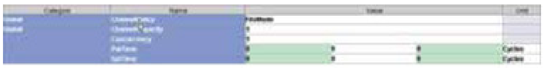


Figure 12: State-machine behavior and performance attributes are preserved.

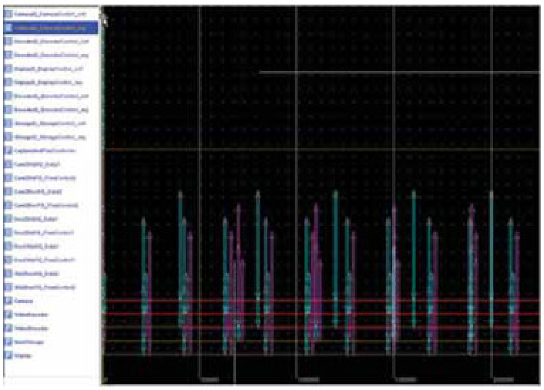


Figure 13: Timed simulation results are obtained for the application model.

The add-in developed in the MARTES project generates an Ecore □□?le from the model according to the de□□?ned meta model. This Ecore □□?le can be imported to the architectural-modeling toolset. The resulting application-model con□□?guration has the same structure as the original UML model (see Figure 11). The model's graphical representation is lost in the transfer. However, the state-machine behavior and performance attributes are preserved. Figure 12 shows the attributes originally set in UML as tagged values. The application model can be simulated in CoFluent Studio (see Figure 13). Although the functional behavior is the same as in Tau, the simulation is now timed.

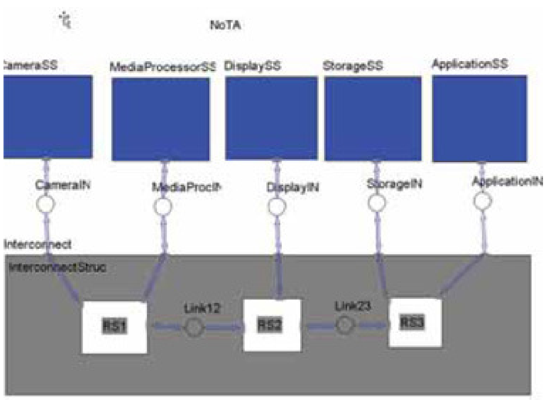


Figure 14: To show a complex example, an execution platform model is used.

The execution-platform and system-architecture models are constructed in the architectural-modeling toolset. The system-architecture model results from the mapping of the application onto the execution platform model. Figure 14 shows an example with a complex platform. That platform contains □□?ve processor units with an interconnect to which the whole application model is allocated. The allocated model is obtained by drag-and-drop mapping. The resulting hierarchy is shown in Figure 15. It can be simulated to study the impact of the platform and mapping. For example, Figure 16 shows the scheduling of the VideoEncoder and VideoDecoder in the single MediaProcessorSS CPU, which prohibits overlap in their execution.

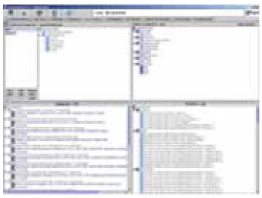


Figure 15: This mapping hierarchy can be simulated to study the impact of both the platform and mapping.

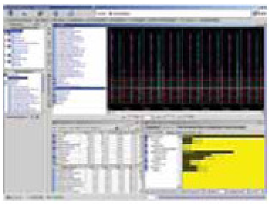
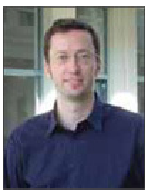


Figure 16: In the simulation results for the mapping alternative, scheduling prohibits any overlap in the execution of the video decoder and encoder.

In conclusion, NRC's report stated, "The choice of CoFluent Studio for architectural exploration was straightforward for many reasons…The link between UML-based requirements modeling and non-UML architecture exploration is realized as execution traces in the □□?rst phase of this case study… The meta-model-based tool integration implemented by Telelogic and CoFluent Design in the MARTES project enabled the use of the use-case model directly instead of execution traces…We experimented with the early version of the tool integration. (…) The principle works as expected and the results are encouraging."

Klaus Kronlöf graduated from Helsinki University of Technology, Department of Technical Physics in 1981 (Master of Science in electrical engineering) and 1984 (Lic. Tech.). He held various research and teaching positions at Helsinki University of Technology in 1981-85. He joined the Nokia Research Center (NRC) in 1985 and has worked in NRC as a project manager during 1987-93. Since 2008, Kronlöf has worked as a Principal Member of Engineering Staff in the Smart Spaces Laboratory of NRC focusing on system architecture.



Vincent Perrier is CoFluent Design's co-founder and director in charge of products and marketing. An embedded-systems expert, Perrier has over 15 years of technical, sales, and marketing experience in the embedded-systems industry and design-automation tools. He holds a computer engineering degree (Master of Science) from the University of Nantes, France.

....

HOME | ABOUT | SUBSCRIBE | ADVERTISE | CONTACT US | PRIVACY STATEMENT