COFLUENT METHODOLOGY FOR UML

UML SysML MARTE Flow for CoFluent Studio

A CoFluent Design White Paper

By Thomas Robert and Vincent Perrier

www.cofluentdesign.com

ACRONYMS AND ABREVIATIONS

BDD	*Block Definition Diagram*
CPU	*Central Processing Unit*
DSL	*Domain-Specific Language*
HW	*Hardware*
IBD	*Internal Block Diagram*
IC	*Integrated Circuit*
IP	*Intellectual Property*
MARTE	*Modeling and Analysis for Real-Time and Embedded systems*
SW	*Software*
SysML	*Systems Modeling Language*
TLM	*Transaction-Level Modeling*
UML	*Unified Modeling Language*

ABSTRACT

SysML is a UML profile that allows the creation of standard descriptions of a system. However, this profile is too generic to address embedded and real-time system design. The MARTE UML profile attempts to fill this gap by providing elements from both embedded software and hardware engineering. Unfortunately, it remains mainly descriptive in nature, since no commercial tools are available to simulate the models and extract performance data.

CoFluent Design offers the CoFluent methodology for UML to provide comprehensive framework and guidelines for joint use of UML, SysML and MARTE. The methodology offers simulation of multicore/multiprocessor hardware/software embedded system and chip models, enabling designers to observe the system behavior and analyze performance properties. The CoFluent methodology delivers modeling rules and method with tool support.

CoFluent Design's tool support includes:

- CoFluent UML profile extending UML 2.2, SysML 1.1 and MARTE 1.0 profiles
- Integration with leading UML modeling environments
- Link to CoFluent Design's CoFluent Studio SystemC 2.2-based simulation environment for model execution and extraction of performance figures

The link to CoFluent Studio is achieved by model transformation from UML to CoFluent's Ecore-based internal model description. The CoFluent methodology for UML complies with CoFluent Studio's embedded system architecting flow and the MARTE profile's intent that separates the application or functional view from the execution platform view. The execution platform view is often called the hardware resource view.

Hardware/software partitioning is described in a mapping or allocation view, and the resulting allocated view represents the actual embedded software threads executing on the various cores and operating systems that constitute the hardware and firmware. The resulting allocated view is a "virtual system", since it encompasses the full hardware/software system.

Existing virtual platform and virtual prototype environments require the assembly of detailed intellectual property block models. CoFluent Design's virtual system modeling and simulation technology overcomes many of the limitations of virtual platforms since it can be executed before detailed hardware intellectual property block models and embedded software are available. Thus, it removes the inherent limitations due to the availability of the models or their important development time and associated cost. The CoFluent technology also goes beyond traditional UML simulation that does not take into consideration architectural and non-functional performance dimensions such as thread priorities and scheduling, time constraints, bus transactions, memory accesses, power consumption, memory footprint, cost, etc. Virtual systems provide fast and accurate evaluation of various use cases and design scenarios by executing UML specifications and predicting the behavior, performance and power consumption. Accurate prediction is critical for multicore and low-power designs.

CoFluent Studio with UML support enables system designers to store and exchange design information internally and between third parties in a standard format. It allows the delivery of executable specifications and SystemC test cases for further system validation with SystemC-based virtual platform environments.

☞ Prerequisite:

The reader is assumed to have a basic knowledge of UML, SysML and MARTE technologies in order to fully understand this white paper.

☞ Acknowledgement:

UML models shown in screenshots were captured with No Magic's MagicDraw tool.

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

This paper presents the CoFluent methodology for UML and associated tool support. The methodology is based on UML with SysML and MARTE profiles. It allows automatic model transformation from this format to the CoFluent DSL.

The methodology can be seen as a set of modeling rules and restrictions applied to SysML/MARTE. These rules enable the simulation of multicore/multiprocessor hardware/software system models for behavioral and performance data prediction, without restricting the expressiveness of SysML and MARTE. The CoFluent methodology for UML does not add new elements to standard profiles. It does, however, offer an optional profile to represent several features related to CoFluent DSL and necessary for efficient embedded system modeling and simulation.

A “Multi-Media Example” model is used to illustrate the design flow. This model is a high-level description of an audio-video decoder running on a generic platform. The generic platform consists of a processor and hardware accelerator.

☞ Throughout this paper, the following typographic rules are applied:

- Specific UML / SysML / MARTE words are displayed in a different font. Example: `block`.
- Names of model elements (`property` names, `block` names, `package` names…) and CoFluent `profile` elements are displayed in italic. Example: *Functions*.

2 MODEL PACKAGING

The model packaging reflects the separation between application and platform models. The application or functional model includes elements of the environment generating stimuli for system simulation under specific use scenarios or use cases. The application model is mapped onto the platform model to add generic physical constraints. The result is a refined virtual system model.

- Functional architecture describes applicative structures with timed behaviors. Functions execute in full parallelism. They synchronize through events and exchange data.

- Physical architecture describes a generic execution platform, including processing units such as a CPU or IC, physical links such as bus or routing network, storage units including memories, and firmware such as schedulers. In this view, the platform is “empty” in the sense that CPUs execute no instructions and ICs are not yet wired with logic.

- Virtual system architecture is the result of the mapping step, which consists in allocating functions to processing units, data to storage units, and routing inter-processor data on physical links. The mapping or allocation can be seen as a model itself. The obtained virtual system model is the functional architecture with execution constraints such as scheduling, bus arbitration and memory size.

Packaging is also important to maximize both reuse and model readability. Figure 1 shows the proposed package hierarchy.

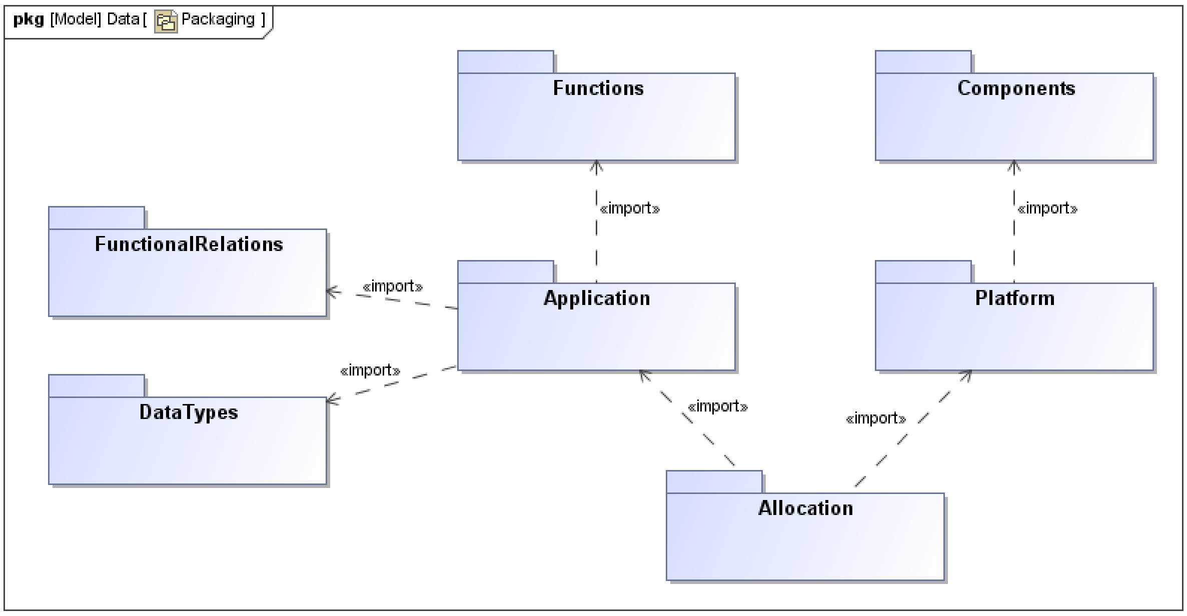


Figure 1: Package diagram of the model

The package named *Functions* contains functions of the application model. These may be reused in other models. *ComponentsLib* contains components of the platform model, which also may be reused in other models. *Application* and *Platform* packages contain models themselves. The *Allocation* package contains the allocation model, using elements from application and platform models.

Two other packages can be defined containing the data types and the functional relations (data communication and synchronization elements between functions). It is merely illustrative packaging, since only *Application*, *Platform* and *Allocation* packages are mandatory.

3 Application Modeling

The application model is a set of functions exchanging data and/or synchronizing through events, shared variables and message queues. Functions can be internal to the system under design or pertain to the environment and thus, contribute to exercising the chosen use case scenario by interacting with the system under a stimuli/responses scheme. There are two different aspects in an application model:

- The structural aspect encapsulates the different functions and the relations, or data paths, between them.
- The behavioral aspect is the control flow used to process data and synchronize functions.

1.1 Structure

Functions encapsulating other functions are called “container functions”. Functions defined at the lowest hierarchical levels are called “leaf functions”.

A SysML `internal block diagram` (`IBD`) is used to represent the applicative structure of the system. A top-level SysML `block` is used as a root container.

Functions of the lower hierarchical level are all `properties` of the top-level `blocks`. `Blocks` appearing in the *Application* `package`, or in an imported `package`, define the `types` of these `properties`. Sub-functions of these `blocks` are also represented as `properties`, and so on.

The `block definition diagram` (`BDD`) in Figure 2 shows the functional hierarchy. This representation is not required in order to correctly represent the CoFluent model: an `IBD` is sufficient. However, a `BDD` aids in understanding functional hierarchy in the application, particularly the distinction between leaf and container functions. Moreover, a `BDD` is compliant to existing UML modeling methodologies and most of the designers feel more comfortable having both a `BDD` and an `IBD`.

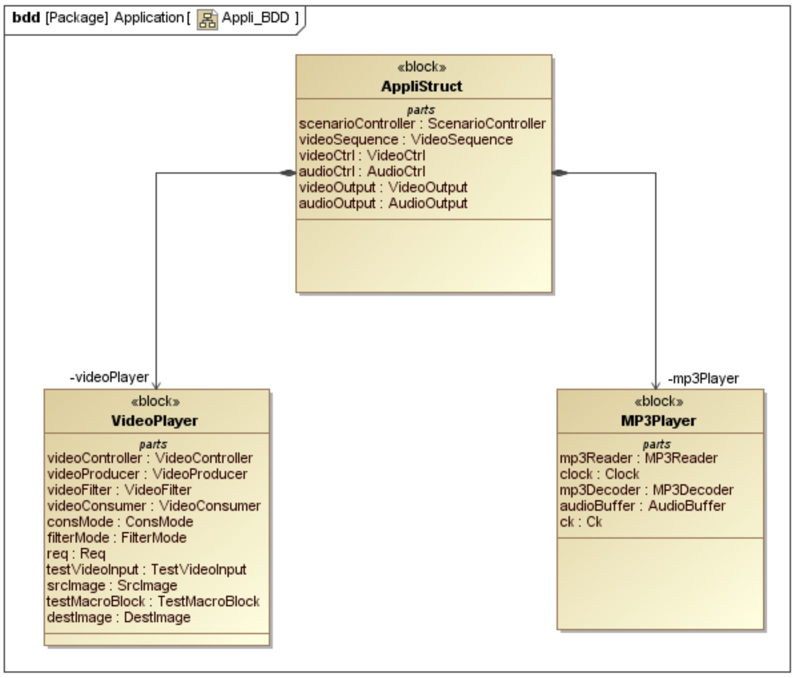

bdd [Package] Application [Appli_BDD]
«block»
AppliStruct
parts
scenarioController : ScenarioController
videoSequence : VideoSequence
videoCtrl : VideoCtrl
audioCtrl : AudioCtrl
videoOutput : VideoOutput
audioOutput : AudioOutput
-videoPlayer
-mp3Player
«block»
VideoPlayer
parts
videoController : VideoController
videoProducer : VideoProducer
videoFilter : VideoFilter
videoConsumer : VideoConsumer
consMode : ConsMode
filterMode : FilterMode
req : Req
testVideoInput : TestVideoInput
srcImage : SrcImage
testMacroBlock : TestMacroBlock
destImage : DestImage
«block»
MP3Player
parts
mp3Reader : MP3Reader
clock : Clock
mp3Decoder : MP3Decoder
audioBuffer : AudioBuffer
ck : Ck

Figure 2: BDD of the application
Leaf functions and relations appear as properties in the lowest-level containers

Figure 3 shows the leaf functions, which are the lowest hierarchical level functions defined in *Functions* `package` of the SysML model.

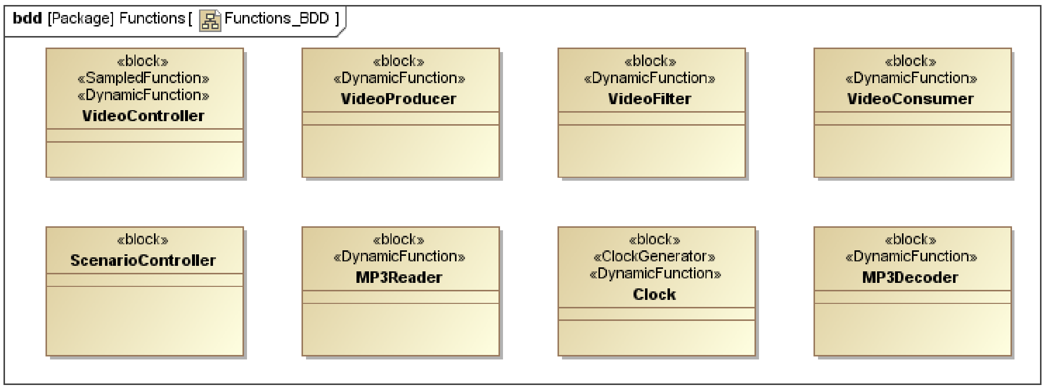

bdd [Package] Functions [Functions_BDD]
«block» «SampledFunction» «DynamicFunction» VideoController
«block» «DynamicFunction» VideoProducer
«block» «DynamicFunction» VideoFilter
«block» «DynamicFunction» VideoConsumer
«block» ScenarioController
«block» «DynamicFunction» MP3Reader
«block» «ClockGenerator» «DynamicFunction» Clock
«block» «DynamicFunction» MP3Decoder

Figure 3: Leaf functions defined in Functions package

Using these blocks as `types` for the container functions `properties` enables reuse, as other `properties` could be typed with these `blocks`. The `blocks` can be reused in the same model or in other application models.

The connections between application elements are represented in the `IBD` of the top-level `block`, named *AppliStruct* for the example as shown in Figure 4.

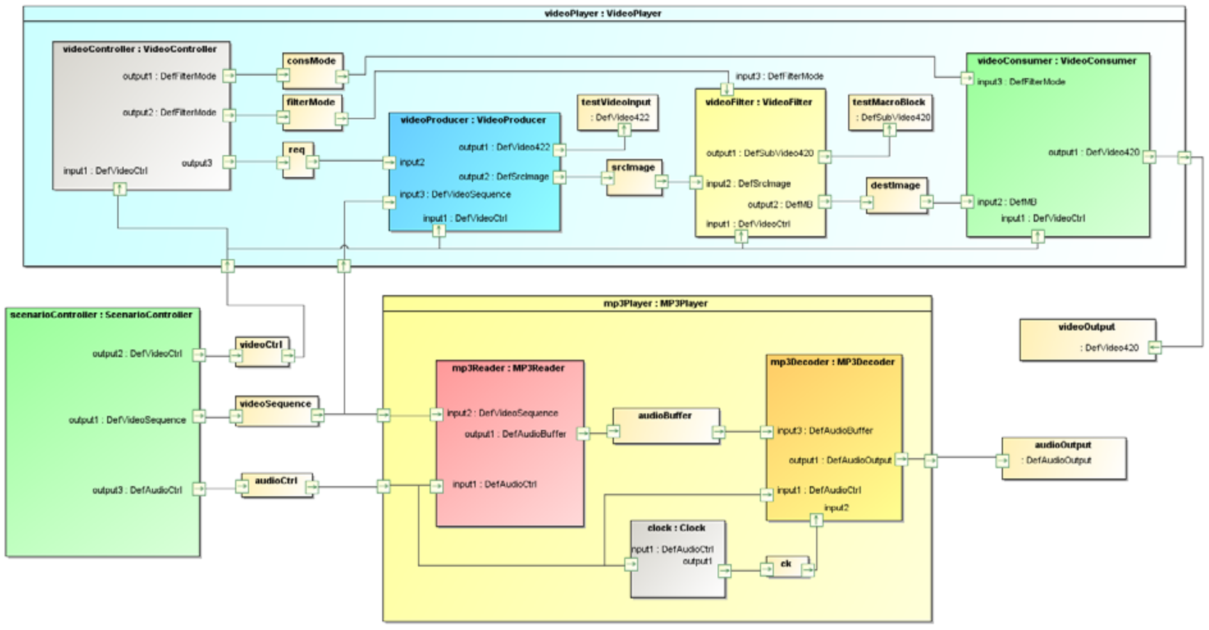

videoPlayer : VideoPlayer
videoController : VideoController
videoProducer : VideoProducer
videoFilter : VideoFilter
videoConsumer : VideoConsumer
scenarioController : ScenarioController
mp3Player : MP3Player
mp3Reader : MP3Reader
mp3Decoder : MP3Decoder
clock : Clock
audioBuffer
videoOutput
audioOutput

Figure 4: IBD of the application

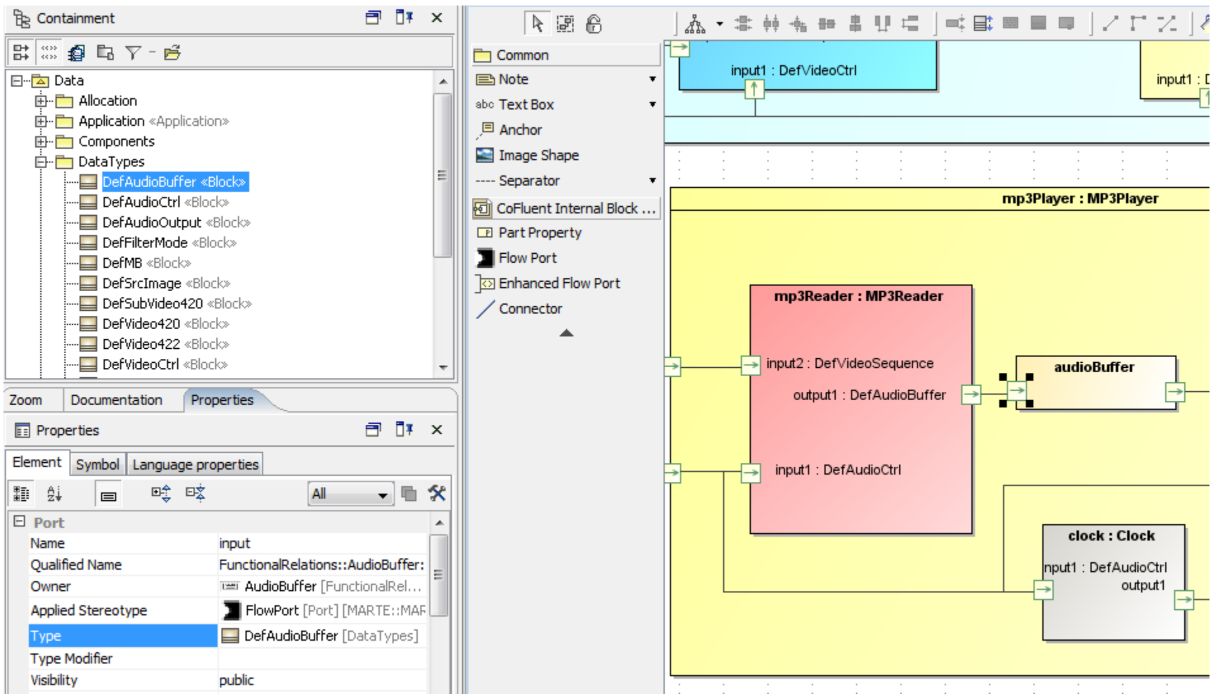
Relational elements are represented by `properties` with two input and output `ports`. The `types` of these `properties` are SysML `blocks` defined in the *FunctionalRelations* `package` with the following `stereotypes` from MARTE profile:

- `NotificationResource`: Event enabling synchronization between functions. Equivalent to CoFluent "event".
- `SwMutualExclusionResource` with `SharedDataComResource`: Shared variable enabling asynchronous data read/write. Equivalent to CoFluent "shared variable".
- `MessageComResource`: Enables synchronous buffered data exchanges. Equivalent to CoFluent "message queue".

There are `assembly connectors` between `ports` of different `properties`. `Delegation connectors` are used for ports forwarding between external and internal functions.

Data types are defined by `blocks` in the *Application* `package` or imported from other `packages`. The data types are carried by the `ports`. `Ports` are MARTE `FlowPorts`, with directions `in`, `out`, `inout`.

Figure 5 shows the `type` of the selected `flow port`. Here, the data type *DefVideoSequence* is a `block` defined in the *DataTypes* `package`.



Port

Name	input
Qualified Name	FunctionalRelations::AudioBuffer:
Owner	AudioBuffer [FunctionalRel...
Applied Stereotype	FlowPort [Port] [MARTE::MAF
Type	DefAudioBuffer [DataTypes]
Type Modifier	
Visibility	public

Figure 5: Port type definition

As shown on the `IBD`, application is composed of three main parts:

- *scenarioController,* shown in the bottom left corner, provides the *videoSequence* and controls execution of video and MP3 players.
- *videoPlayer,* shown at the top, decodes *videoSequence*.
- *mp3Player,* shown at the bottom, decodes the MP3 encoded sound contained in *videoSequence*.

☞ Important note about stereotypes application:

Models described in this document make an extensive use of MARTE `stereotypes`. It is also possible to apply optional CoFluent `stereotypes`.

`Stereotypes` applied to `blocks` may be also applied to `properties` typed by these `blocks`. `Stereotypes` can even be applied only to `properties`. For instance, it is possible to apply a `stereotype` to a `property` of a container function in the application model, independently from the `stereotype` applied to the `block` defined in the function library as `type` of the `property`.

If both `block` and `property` are stereotyped with the same `stereotype`, `tagged values` from the `property` will be used when transforming to the CoFluent model.

1.2 Behavior

Leaf function types are `blocks` that can contain a `behavior`. An `activity diagram` can describe this `behavior`.

Placing the `activity` under the `block` in the hierarchy of the SysML model is sufficient to express that the `activity` is representing the `behavior` of the `block`. However, this link can be explicitly described by setting the `ClassifierBehavior property` of the `block`.

Figure 6 depicts an example of an `activity`, *MP3DecoderAct*.

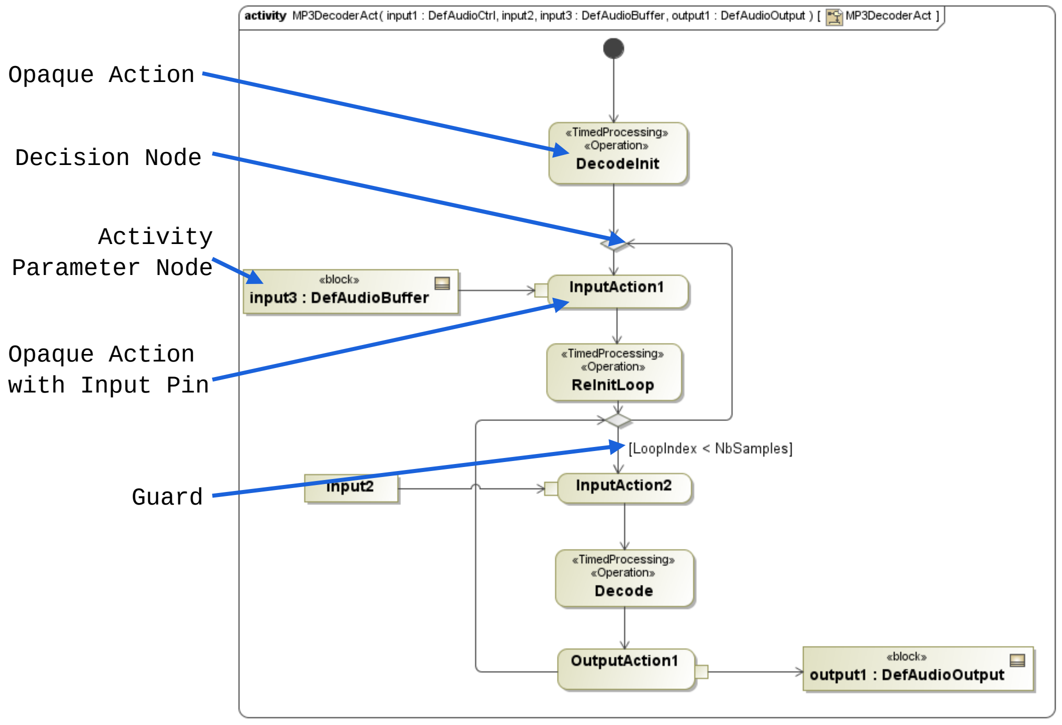

activity MP3DecoderAct(input1 : DefAudioCtrl, input2, input3 : DefAudioBuffer, output1 : DefAudioOutput) [MP3DecoderAct]
Opaque Action
Decision Node
Activity Parameter Node
Opaque Action with Input Pin
Guard
«TimedProcessing» «Operation» DecodeInit
«block» input3 : DefAudioBuffer
InputAction1
«TimedProcessing» «Operation» ReInitLoop
[LoopIndex < NbSamples]
input2
InputAction2
«TimedProcessing» «Operation» Decode
OutputAction1
«block» output1 : DefAudioOutput

Figure 6: SysML activity diagram

Inputs and outputs of the activity are activity parameter nodes, allocated to corresponding ports of the application structure via the MARTE Allocate stereotype.

Input and output pins allow read/write on ports in behaviors. Opaque actions model operations to be performed, for example data read/write and processing. Decision node/fork and join/fork and merge can be used in the control flow description. In Figure 6, the first decision node represents an infinite loop, whereas the second is a loop with condition, where *LoopIndex* < *NbSamples*.

For opaque actions, the MARTE stereotype TimedProcessing can be used in order to set operation duration. C/C++ is used as action language. Code can also be added in opaque action body for SystemC-based simulation within CoFluent Studio after model transformation.

1.3 CoFluent Application Profile

Use of the CoFluent profile is optional. The *Application* package of the CoFluent profile allows accessing the following CoFluent-specific features.

- Specific model element attributes including read and write times for relations and specific CoFluent functions such as sampled function, clock generator, divider, dynamic function, message routing function.

- Specific C/C++ code: Global declarations, pre-processing and post-processing code, local declarations for each function, and type definitions.

- Generic parameters and variables: they are used to change values in the model at simulation time.

  There are four stereotypes for generic parameters: *EnumerationGP* for enumerations, *IntegerGP* for integers, *FloatGP* for floats and *DoubleGP* for doubles.

  In order to add an integer generic parameter to the UML model, the parameter must first be defined with a block to which the right stereotype is applied. Then, attribute values of the parameter can be set. Lastly, a `property` must be added in the `block` to which the parameter is applied. The `type` of this `property` is the generic parameter `block` previously defined. If the generic parameter is global, a `property` typed by the generic parameter `block` must be added to the root `block` of the application.

1.4 Tools: UML Modelers and CoFluent Studio Integration

1.4.1 MagicDraw

The CoFluent *Application* `profile` and associated CoFluent plugin are currently available for No Magic's MagicDraw UML modeling tool.

The SysML/MARTE application model can be captured in MagicDraw and then imported from CoFluent Studio. MagicDraw can also be directly integrated into CoFluent Studio as an Eclipse plugin, with one-click integration for the user, in order to work within a single modeling environment.

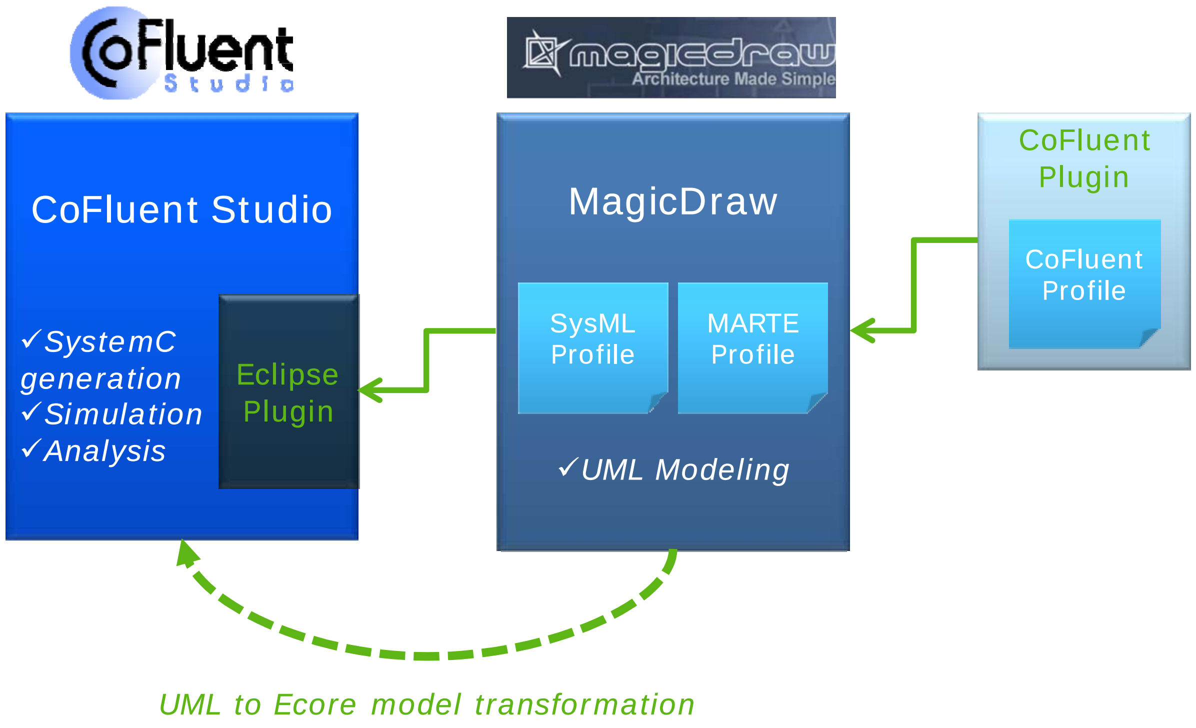


Figure 7: CoFluent Studio with integrated MagicDraw

MagicDraw customization has been developed to ease the modeling guidelines application and the CoFluent profile usage. A MagicDraw validation module for CoFluent is available to ensure that the SysML/MARTE model is compliant to the methodology and can be transformed into a CoFluent model. All these elements are part of the CoFluent plugin for MagicDraw.

☞ Attributes and algorithms capture:

The CoFluent model attributes (calibration data) and algorithms that are not captured in the SysML model can be edited in CoFluent Studio.

1.4.2 Papyrus

The CoFluent `profile`, which includes *Application*, *Platform* and *Allocation* `packages`, is also available for the Papyrus UML modeling environment. However, a Papyrus to CoFluent Studio model transformation is not commercially available.

1.5 Application Model Execution in CoFluent Studio

The SysML/MARTE application model can be automatically transformed into a CoFluent model. The transformation is implemented with Java and the Eclipse Modeling Framework (EMF). The output is a tool-compatible model compliant with CoFluent meta-model. CoFluent meta-model in turn conforms to Ecore, Eclipse's Meta-Object Facility (MOF).

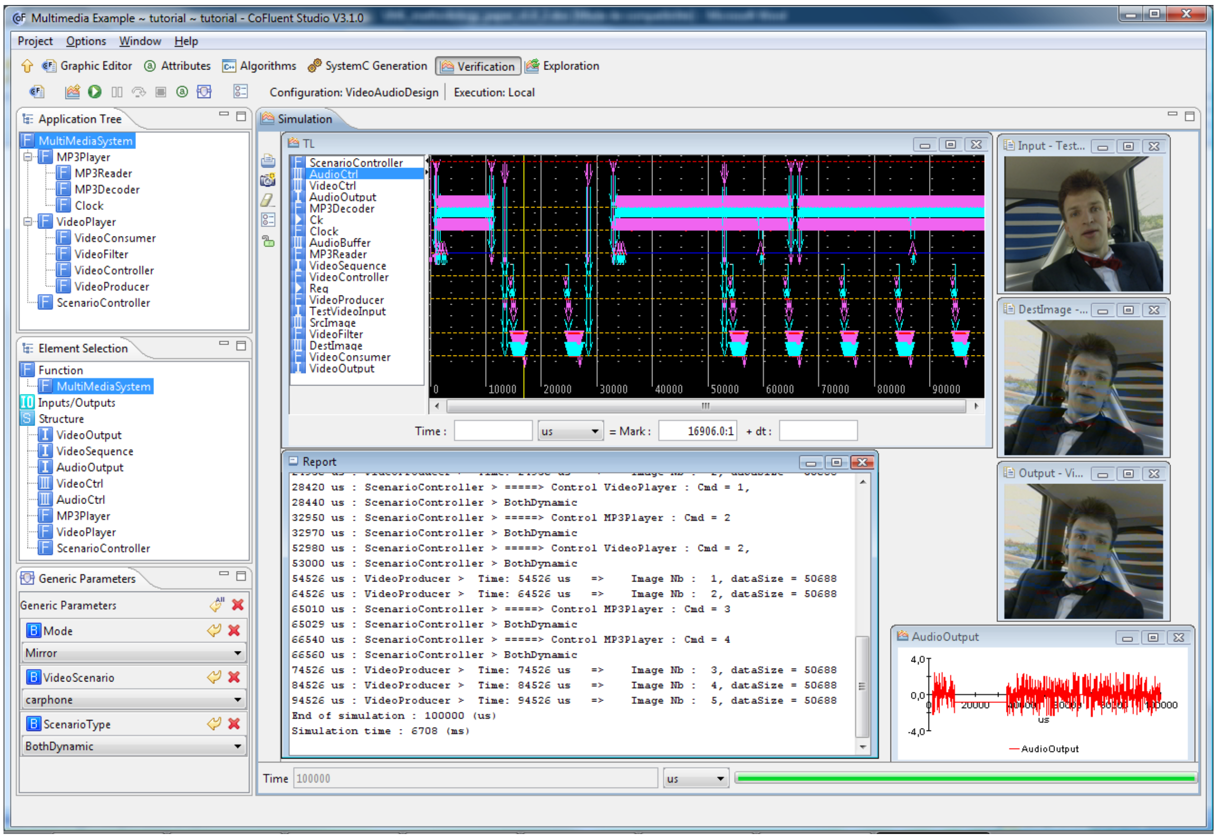
In CoFluent Studio, the obtained Ecore model is used to automatically generate a SystemC TLM model that includes the user-defined C/C++ code. The generated code uses CoFluent SystemC 2.2 and TLM 2.0-based simulation library. SystemC models can be generated for various environments such as OSCI SystemC library, CoWare, Synopsys Innovator, Synopsys System Studio, and Mentor Graphics Questa. This feature enables reuse of CoFluent-generated SystemC models as test cases for further verification and validation within a wide range of virtual platform environments.

Once the SystemC model is generated and compiled, the model can be run to extract behavioral and timing information. Deadlocks or conflicts in data accesses can be identified. Latencies can be measured. It is also possible to change generic parameter values and see how the changes impact model execution.

Figure 8 shows the model simulation results in CoFluent Studio.

☞ Current tool support:

CoFluent Studio v3.2 supports application modeling and simulation with UML. Platform model and mapping are captured using the CoFluent DSL.


Multimedia Example ~ tutorial ~ tutorial - CoFluent Studio V3.1.0
Project Options Window Help
Graphic Editor Attributes Algorithms SystemC Generation Verification Exploration
Application Tree
Simulation
TL
Input - Test...
DestImage -...
Output - Vi...
AudioOutput
Element Selection
Generic Parameters
Report

Configuration: VideoAudioDesign | Execution: Local

Mode: Mirror
VideoScenario: carphone
ScenarioType: BothDynamic

Time : us = Mark : 16906.0:1 + dt :

```
28420 us : ScenarioController > =====> Control VideoPlayer : Cmd = 1,
28440 us : ScenarioController > BothDynamic
32950 us : ScenarioController > =====> Control MP3Player : Cmd = 2
32970 us : ScenarioController > BothDynamic
52980 us : ScenarioController > =====> Control VideoPlayer : Cmd = 2,
53000 us : ScenarioController > BothDynamic
54526 us : VideoProducer >  Time: 54526 us   =>    Image Nb :  1, dataSize = 50688
64526 us : VideoProducer >  Time: 64526 us   =>    Image Nb :  2, dataSize = 50688
65010 us : ScenarioController > =====> Control MP3Player : Cmd = 3
65029 us : ScenarioController > BothDynamic
66540 us : ScenarioController > =====> Control MP3Player : Cmd = 4
66560 us : ScenarioController > BothDynamic
74526 us : VideoProducer >  Time: 74526 us   =>    Image Nb :  3, dataSize = 50688
84526 us : VideoProducer >  Time: 84526 us   =>    Image Nb :  4, dataSize = 50688
94526 us : VideoProducer >  Time: 94526 us   =>    Image Nb :  5, dataSize = 50688
End of simulation : 100000 (us)
Simulation time : 6708 (ms)
```

Time 100000 us

Figure 8: CoFluent Studio behavioral model simulation results

The user monitors the activity of the different functions, as well as the data exchanges between the functions, in the timeline (*TL)* window. With this feature, the user can clearly recognize the messages that are sent to the two players. The different functions and message queues related to the MP3 player are represented in the upper part of the window. The video player is located in the lower part.

The CoFluent Studio timeline is similar to a UML sequence diagram displayed horizontally (time runs from left to right). It offers additional capability as well. The state of each function, running, blocked, and idle, can be observed and analyzed in detail. Computation and data read/write transaction durations can be precisely observed since the model execution is fully timed.

The *Report* window displayed below the timeline in Figure 8 is used to print text information during the simulation. Just as software programmers that can add printf statements in their code for printing information during the program execution, CoFluent users can also monitor and print out information during the simulation execution in this window.

Image processing on the right side of the screen in Figure 8 shows the different steps of image processing through the video channel. The first picture represents the input picture that is processed. Next, the content of the *destImage* message queue is shown. Finally, the output picture that is available in *videoOutput* is displayed. It is also possible to plot variables against the execution time to monitor their values, such as audio output level of the *mp3Player* in the window shown in the lower right section of the screen.

4 PLATFORM MODELING

The platform is a set of computation and storage resources connected by physical links. It is represented using `Hardware Resource Model` from MARTE profile.

The platform structure is described by an `IBD`. A top-level SysML `block` is used as a root container.

Components of lower hierarchical levels are `properties` of the top-level `blocks`. `Types` of these `properties` are defined by MARTE `stereotyped` `blocks`, appearing in the *Platform* `package` or in an imported `package`. Sub-functions of these `blocks` are also represented as `properties`, and so on.

Schedulers can be allocated to MARTE `HwProcessor` using the MARTE `allocate stereotype`. The schedulers are defined as `blocks` stereotyped with MARTE `GaExecHost`.

Figure 9 depicts `blocks` defined in the imported *Components* `package`. The MARTE `stereotype` used for each one is visible. The `package` contains computation resources (*UserComputer, CoProcessor*), communication resources (*Bus, USB, SPI*), memories (*DisplayEnv, Source, Mem*) and a scheduler (*Scheduler*).

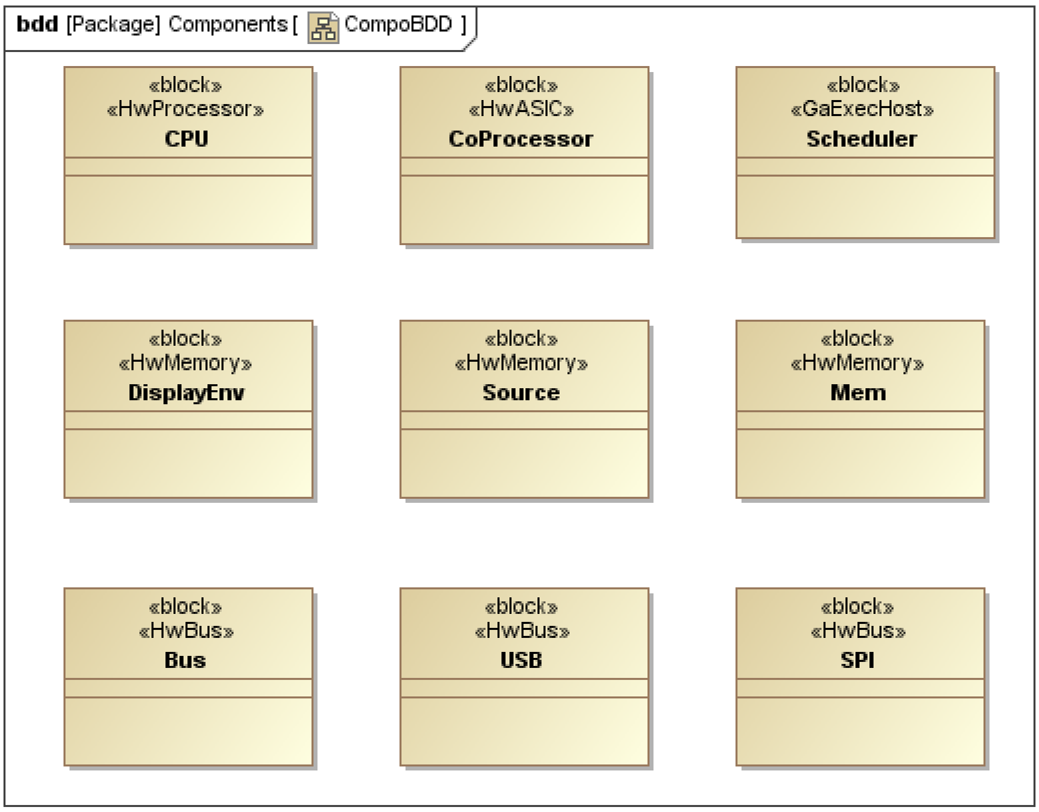

bdd [Package] Components [CompoBDD]
«block» «HwProcessor» CPU
«block» «HwASIC» CoProcessor
«block» «GaExecHost» Scheduler
«block» «HwMemory» DisplayEnv
«block» «HwMemory» Source
«block» «HwMemory» Mem
«block» «HwBus» Bus
«block» «HwBus» USB
«block» «HwBus» SPI

Figure 9: Blocks defined in Components package

Figure 10 depicts the `IBD` of the platform. `Ports` are MARTE `hwEndPoints`. `Assembly connectors` are used to connect `end points` to buses. `Delegation connectors` are used for ports forwarding between different hierarchical levels. Schedulers are allocated to hardware processors using `Assign stereotype` from MARTE.

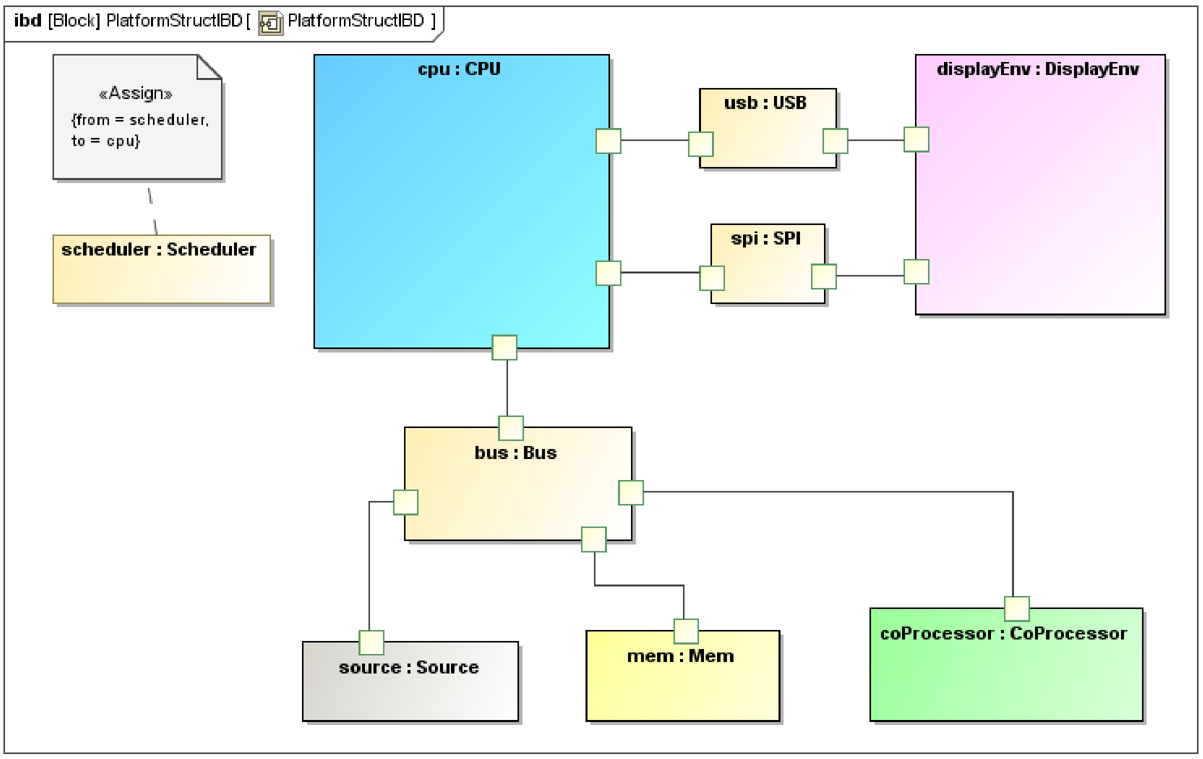


Figure 10: Composite diagram of the platform

5 ALLOCATION

To obtain an allocated virtual system model, functions of the application are mapped onto computation resources of the platform, while shared variables and exchanged data are mapped onto memories and physical links. `Assign` from MARTE profile is used to model this mapping or allocation.

Software mapping is achieved by assigning functions from the application to tasks (`blocks` stereotyped with MARTE `schedulable resource`). The schedulers of the platform schedule tasks.

Hardware mapping is done by directly assigning functions to ICs of the platform.

Link mapping is the allocation of functional relations to physical links or memories of the platform.

Figure 11 shows the different assignments for the example model.

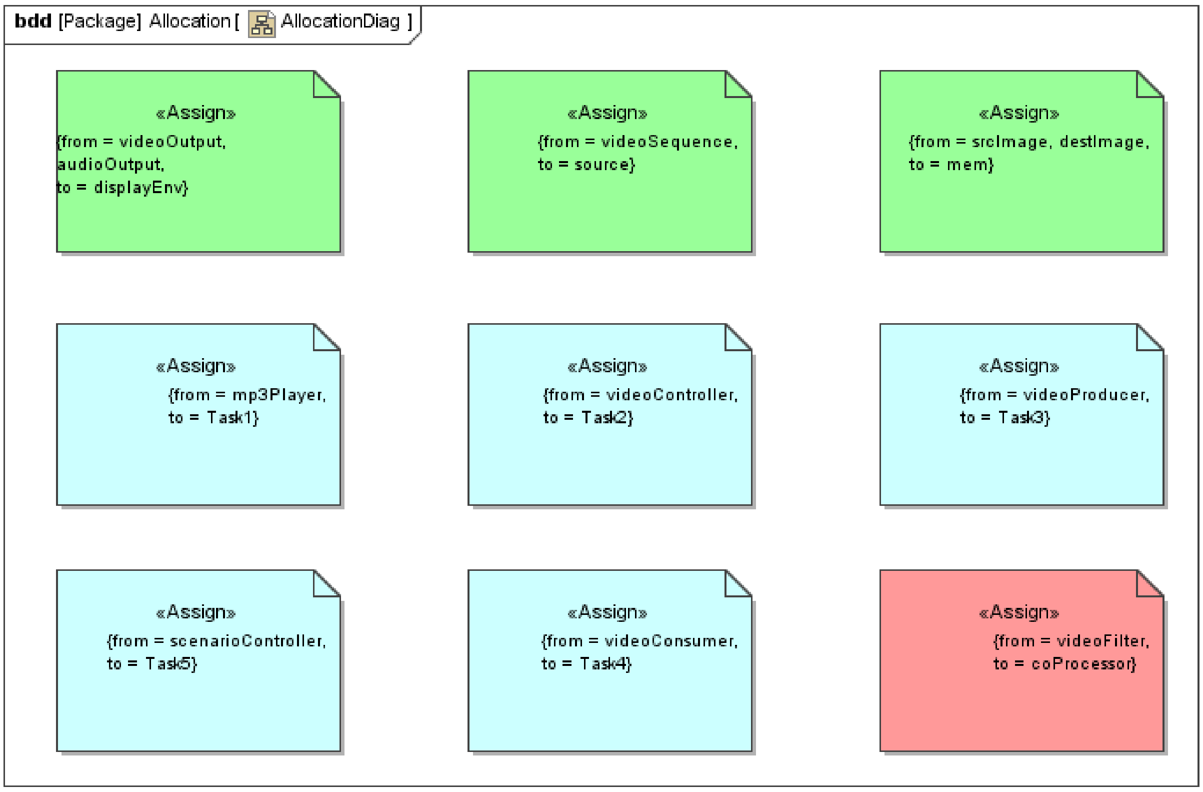

bdd [Package] Allocation [AllocationDiag]
«Assign» {from = videoOutput, audioOutput, to = displayEnv}
«Assign» {from = videoSequence, to = source}
«Assign» {from = srcImage, destImage, to = mem}
«Assign» {from = mp3Player, to = Task1}
«Assign» {from = videoController, to = Task2}
«Assign» {from = videoProducer, to = Task3}
«Assign» {from = scenarioController, to = Task5}
«Assign» {from = videoConsumer, to = Task4}
«Assign» {from = videoFilter, to = coProcessor}

Figure 11: Assignments for application to platform mapping

1.6 Allocated Model Execution in CoFluent Studio

Figure 12 illustrates the obtained timeline for the allocated, or architecture, model simulation in CoFluent Studio. The descriptions were created with CoFluent DSL.

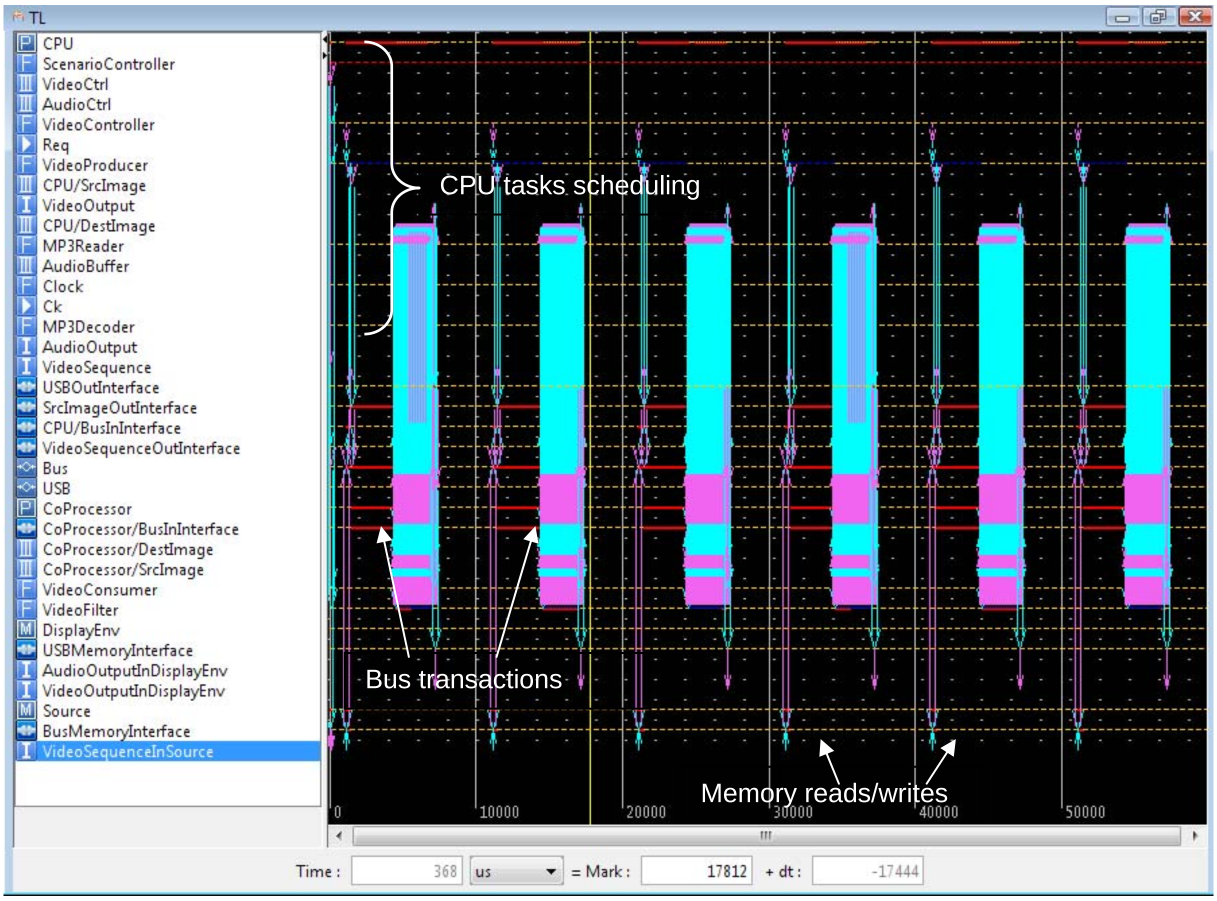
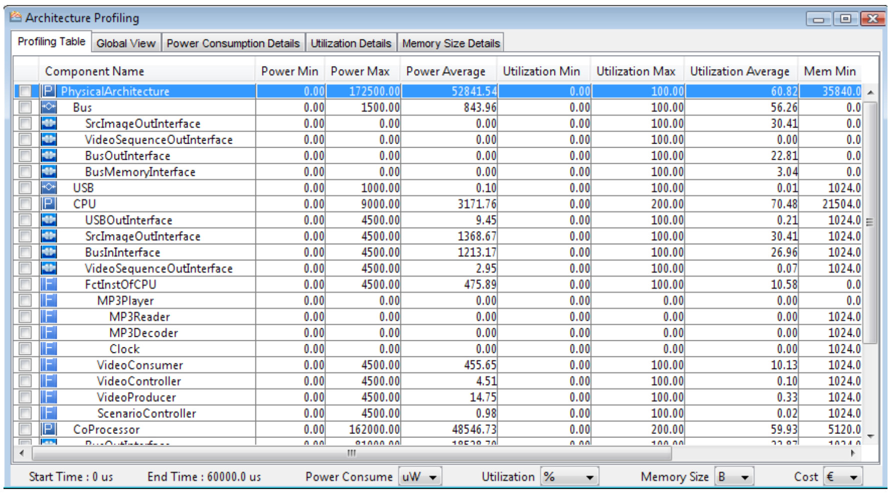


Figure 12: Allocated model simulation timeline

Unlike results available for the behavioral simulation, the model execution results include the activity of the processors, busses and interfaces. This feature is useful to evaluate if resources of the system under study are overloaded. It also will help determine if the software partitioning is adequately done. Additionally, the ability of a software-processing unit to perform several tasks and the impact of a bus to transfer can be monitored.

Figure 13 shows the performance profiling windows obtained at the end of the simulation. The first window displays a global results table, and the second window displays the dynamic load of selected resources.



Component Name	Power Min	Power Max	Power Average	Utilization Min	Utilization Max	Utilization Average	Mem Min
PhysicalArchitecture	0.00	172500.00	52841.54	0.00	100.00	60.82	35840.0
Bus	0.00	1500.00	843.96	0.00	100.00	56.26	0.0
SrcImageOutInterface	0.00	0.00	0.00	0.00	100.00	30.41	0.0
VideoSequenceOutInterface	0.00	0.00	0.00	0.00	100.00	0.00	0.0
BusOutInterface	0.00	0.00	0.00	0.00	100.00	22.81	0.0
BusMemoryInterface	0.00	0.00	0.00	0.00	100.00	3.04	0.0
USB	0.00	1000.00	0.10	0.00	100.00	0.01	1024.0
CPU	0.00	9000.00	3171.76	0.00	200.00	70.48	21504.0
USBOutInterface	0.00	4500.00	9.45	0.00	100.00	0.21	1024.0
SrcImageOutInterface	0.00	4500.00	1368.67	0.00	100.00	30.41	1024.0
BusInInterface	0.00	4500.00	1213.17	0.00	100.00	26.96	1024.0
VideoSequenceOutInterface	0.00	4500.00	2.95	0.00	100.00	0.07	1024.0
FctInstOfCPU	0.00	4500.00	475.89	0.00	100.00	10.58	0.0
MP3Player	0.00	0.00	0.00	0.00	0.00	0.00	0.0
MP3Reader	0.00	0.00	0.00	0.00	0.00	0.00	1024.0
MP3Decoder	0.00	0.00	0.00	0.00	0.00	0.00	1024.0
Clock	0.00	0.00	0.00	0.00	0.00	0.00	1024.0
VideoConsumer	0.00	4500.00	455.65	0.00	100.00	10.13	1024.0
VideoController	0.00	4500.00	4.51	0.00	100.00	0.10	1024.0
VideoProducer	0.00	4500.00	14.75	0.00	100.00	0.33	1024.0
ScenarioController	0.00	4500.00	0.98	0.00	100.00	0.02	1024.0
CoProcessor	0.00	162000.00	48546.73	0.00	200.00	59.93	5120.0

Start Time : 0 us End Time : 60000.0 us Power Consume uW Utilization % Memory Size B Cost €

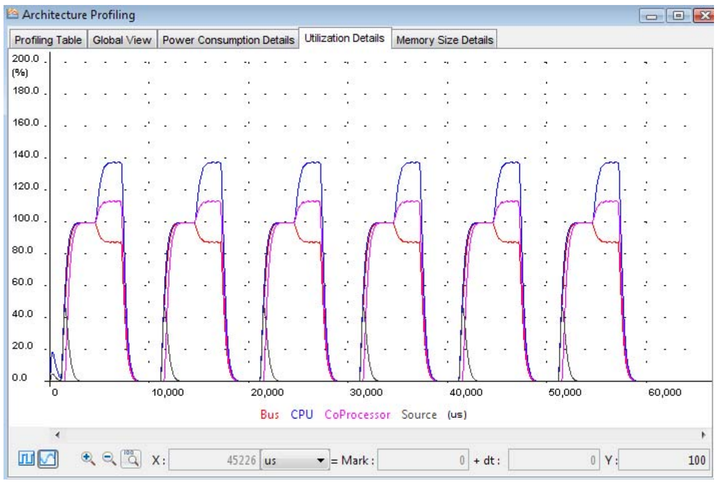


Figure 13: Performance profiling windows in CoFluent Studio

Performance profiling results include power, resource utilization (loads in percentage of Mcycles/s), and memory. For instance, the simulation indicates a *CPU* use of 70.84% and a *Bus* use of 56.26%.

If the *CoProcessor* was not used and all the functions were mapped onto the *CPU*, the *CPU* utilization would increase and the bus utilization would decrease. This kind of mapping change is easily done in CoFluent Studio, enabling fast “what if” analysis.

6 CONCLUSION

This paper describes an embedded system architecting flow that utilizes virtual system models obtained from SysML/MARTE specifications. The specifications are translated into executable SystemC transaction-level models. Virtual system technology overcomes many of the limitations of virtual platforms, since it is accessible when specialized hardware and software IP models or code are not yet available. It provides the simulation capability for performance analysis, behavioral and architectural verification, and use cases and tradeoff analysis.

The use of standard notations of SysML/MARTE profiles to describe virtual systems enables system architects to store and exchange design information internally and between third parties in a standard format. Furthermore, virtual systems allow fast and accurate evaluation of different use cases and design scenarios by allowing the execution of UML specifications. Automatic SystemC TLM code generation allows reusing models in other SystemC-based virtual platform and verification environments.

CoFluent Design Europe
24, rue Jean Duplessis
78150 Le Chesnay – France

info@cofluentdesign.com

CoFluent Design Japan
Grace Takanawa Building 9F
2-14-17 Takanawa, Minato-ku
Tokyo 108-0074 – Japan

info-jp@cofluentdesign.com

CoFluent Design, Inc.
2290 N. First Street
Suite 304
San Jose, CA 95131 – USA

info-us@cofluentdesign.com