# Formal System-level Design Space Exploration

Daniel Knorreck and Ludovic Apvrille and Renaud Pacalet
System-on-Chip laboratory (LabSoC), Institut Telecom, Telecom ParisTech, LTCI CNRS
2229, routes des Crêtes, B.P. 193, F-06904 Sophia-Antipolis Cedex
Email: {daniel.knorreck, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr

***Abstract*—The paper focuses on the formal aspects of the DIPLODOCUS environment. DIPLODOCUS is a UML profile intended for the modeling and verification of real-time and embedded applications meant to be executed on complex Systems-on-Chip. Application tasks and architectural elements (e.g., CPUs, bus, memories) are described with a UML-based language, using an open-source toolkit named TTool. Those descriptions may be automatically transformed into a formal hardware and software specification. From that specification, model-checking techniques may be applied to evaluate several properties of the system, e.g., safety, schedulability, and performance properties. The approach is exemplified with an MPEG2 decoding application.**

## I. Introduction

A System-on-Chip (SoC) is a set of functions distributed over hardware computation elements (CPUs, hardware accelerators) interconnected with complex communication elements (e.g., NoC). Designers of SoC have to tackle advanced constraint issues because applications executed on SoC are more and more complex, typically like applications running on smart phones. While simulation techniques applied on low-level software and hardware models are widely used for verification purpose, formal verification is the only technique that can offer strong guarantees because it explores all possible system paths. Unfortunately, the well-known combinatorial explosion usually limits its use to very small subparts of low-level system (e.g., a bus controller). Additionally, the prohibitive cost of late SoC re-engineering advocates designs flows with early guarantees.

Design Space Exploration (DSE) is a major step in system design: it consists in selecting a software / hardware architecture complying to a set of functional and non-functional constraints (performance, power consumption, etc.). But, at Design Space Exploration stage, the complexity may already be non-manageable, and so, we suggest to perform that stage on very abstract models to pave the way for fast performance estimations. DIPLODOCUS is the environment we propose for addressing Design Space Exploration. As it relies on the Y-Chart approach, architecture and application are represented in an orthogonal fashion. DIPLODOCUS explicitly takes into account the hardware platform on which application tasks are meant to be executed. While modeling [1] and simulation [2] capabilities of DIPLODOCUS, as well as the toolkit supporting DIPLODOCUS [3] - TTool [4] - were already described in previous publications, the semantic support, one of the main strengths of DIPLODOCUS, has not been addressed so far. More precisely, for formal verification purpose, we have provided a formal semantics to all DIPLODOCUS diagrams (applications, hardware architectures, mapping of applications onto hardware architectures). Formal analysis is based on a process algebra named LOTOS [5] and on UPPAAL [6]. The paper is more particularly focused on abstractions made to describe tasks and hardware platforms, on how formal analysis can be done from this abstraction, and on the associated toolkit (TTool) in which all underlying formal techniques are totally masked to the designer (press-button approach).

The paper is organized as follows. Section II reviews related contributions. Section III recalls the DIPLODOCUS environment. Section IV focuses on the formal semantics in DIPLODOCUS, and more precisely on the abstractions offered by DIPLODOCUS to allow formal analysis with limited combinatory explosion. Section V presents the implemented support toolkit. Section VI illustrates our approach on a real-time multimedia application. Finally, section VII concludes the article.

## II. Related Work

Design Space Exploration (DSE) of system-on-Chip is the process of analyzing various functionally equivalent implementation alternatives to select an optimal solution [7]. The most suitable design is commonly chosen based on metrics such as functionality, performance, cost, power, reliability, and flexibility. At system-level, DSE is challenging because the system design space is extremely large and so usual simulation-based analysis techniques fail to efficiently observe the above mentioned metrics. Contributions on DSE environments [8]–[15] generally rely on a high-level language to describe application functions and architectures. For example, [13]–[15] rely on UML or MARTE diagrams. Functions are sometimes described with only their cost [16]. Unfortunately, in many of these environments, architecture and application concerns are not independent [10], making the study of alternative solutions more complex. Second, they propose a way to map functions onto hardware execution nodes. Lastly, they introduce simulation techniques to simulate the system built from the mapping of functions over hardware nodes. But the level of abstraction being commonly rather low, simulation may also be slow. For example, [8] relies on an Instruction Set Simulator which executes the real code of the application. In [12], hardware components are considered at micro-architecture level, hence leading to long simulation times. Otherwise, other environments offer formal exploration, but generally limited to sub-elements of the platform [17].



978-1-4244-7068-6/10/$26.00 ©2010 IEEE

SymTA/S [18] [19] and Real Time Calculus (RTC) rely on formal methods such as the real-time scheduling theory and deterministic queuing systems to determine characteristics of distributed systems. In SymTA/S the behavior of the environment is modeled by means of standard event arrival patterns including periodic and sporadic events with jitters or bursts. RTC imposes less restrictions by allowing deterministic event streams to be modeled with the aid of arrival curves denoting lower and upper bounds for event occurrences. Event streams are propagated among resources of distributed systems in a way that each resource may be analyzed separately with classical algorithms. However, the applicability of scheduling theories requires the task model to be simplistic and thus it merely reflects best case and worst case execution times. Control flow within tasks cannot be considered at all. For that reason it may be tedious if not impossible to model tasks exhibiting a data dependent or irregular behavior.

[20] relies on timed automata to analyze timeliness properties of embedded systems. The UPPAAL model checker is used to evaluate the automata which must be created manually. There is no automated translation routine from a high level language (UML,...) and thus the creation of the automata turns out to be error prone.

[21] provides means for formal and simulation based evaluation of UML/SysML models for performance analysis of Systems On Chip. UML Sequence diagrams constitute the starting point for the functional description. They are subsequently transformed into so-called communication dependency graphs (CDGs) which thus capture the control flow, synchronization dependencies and timing information. CDGs are in turn amenable to static analysis in order to determine key performance parameters like best case response times, worst case response times and I/O data rates. A drawback of this approach is that data flow independence has to be kept, thus preventing case distinctions and loops with variable bounds to be part of the application model.

[22] presents a framework for computation and communication refinement for multiprocessor Soc Design. Stochastic automata networks represent the application behavior and the authors claim that this formalism allows for fast analytical performance evaluations. When it comes to mapping an application on an architecture, transitions and states have to be added to the application model. Hence, application and architecture matters and not strictly handled in an orthogonal fashion. Due to a lack of data abstraction, the modeling of memory elements can quickly lead to state space explosion problem.

The PUMA [23] framework is a unified approach to software modeling. It provides an interface between high level input models (such as UML diagrams) and performance oriented models. For that purpose, input models are first translated into an intermediate format called CSM so as to filter out irrelevant information for performance evaluations. In a second step, CSM can be converted to Petri Nets, Markov models, etc and the resulting performance figures and design advice is fed back to the initial model. However, this framework concentrates on the modeling of software and thus does not yield a mapping where functionality is associated to software or hardware elements.

DIPLODOCUS offers a very clear separation between applications and architectures, and includes a high level of abstraction. Indeed, DIPLODOCUS is focused on control rather than on data, i.e., only abstract *samples* of data can be manipulated in the profile. Samples are untyped and carry no value: only their size is a relevant attribute. This high level of abstraction greatly reduces simulation times and makes formal proof techniques usable. In this paper, we apply these formal proof techniques to safety, performance and schedulability analysis purpose using the LOTOS process algebra. While LOTOS has already been successfully experimented for property proofs on hardware [24], we propose its use to more generic platforms (SoCs).

## III. The DIPLODOCUS UML Profile

A UML profile customizes the UML language [25] for a given domain of systems. It may extend the UML meta-model, according to *semantic variation points*, and may provide a methodology. The DIPLODOCUS UML profile targets the modeling and Design Space Exploration of system-on-chip at a high level of abstraction [1]. The DIPLODOCUS methodology, depicted in Figure 1, comprises three main steps described below.

### A. Application modeling

At first, the application is modeled using UML class and activity diagrams. Tasks are modeled as classes interconnected with *channels*, *events*, or *requests* to communicate. Data abstraction is a key point: channels do not convey values, but only a number of samples (data abstraction). Events are used for synchronization purpose, and requests are used to spawn tasks.

All tasks have a behavior modeled with UML-DIPLODOCUS activity diagrams. These diagrams come with usual control operators, complexity operators, operators to use communication medium (e.g., channels), and time operators.

### B. Architecture modeling

Second, a candidate architecture is modeled in terms of interconnected hardware components (or nodes) using UML components placed in UML deployment diagrams. Three kinds of nodes are available in DIPLODOCUS: *Execution nodes* (CPUs), *Communication nodes* (buses, bridges) and *Storage nodes* (e.g., memories). Each node has its own set of parameters such as *pipeline depth*, *miss-branching prediction rate*, *cache-miss rate*, etc. Also, all nodes - except buses - may be connected to one or several buses using UML links. For the time being, DMAs and hardware accelerators are represented by adequately parametrized CPU nodes.

### C. Mapping

The mapping is meant to study whether a given architecture can execute a given application according to constraints. Application tasks and channels may be distributed over hardware nodes. A UML deployment diagram is used for that

purpose. A given task may be mapped only on one execution node. Channels may be mapped on paths built upon links, communications nodes, and storage nodes.
One of the strengths of DIPLODOCUS relies on its ability to perform simulation and formal verification both at application and mapping stages. Formal verification at mapping stage is further discussed in the next section. Although common functional properties are usually studied at application level, performance properties are investigated after mapping, e.g., resource sharing, that is, the scheduling on CPUs (can the architecture execute tasks on time), bus load (can a bus transmit the required amount of data), and properties related to power consumption and silicon area.

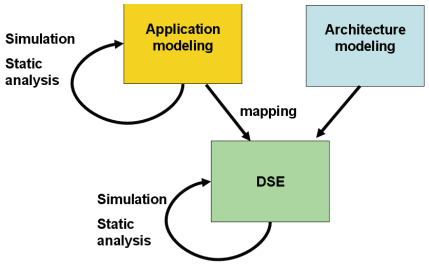


Fig. 1. Methodology for Design Space Exploration

## IV. Formal semantics and abstractions

### A. Formal support: LOTOS and UPPAAL

LOTOS [5] is an ISO-based Formal Description Technique for distributed system specification and design. A LOTOS specification, being itself a process, is structured into processes. A LOTOS process is a black box that communicates with its environment through gates using a multiway rendezvous offer. Values can be exchanged at synchronization time. That exchange can be mono- or bi-directional. LOTOS specifications may be formally verified with the CADP toolkit [26] using model-checking or reachability graph analysis techniques
The semantics of DIPLODOCUS is also defined in UPPAAL, but the latter is out of scope of this paper.

### B. Semantics at application level

*1) Tasks operators:* As described in previous section, an application is composed of a set of communicating tasks. Operators used to describe task behavior are of four types:

- **Communication operators**: **read** from a channel, **write** a sample to a channel, **notify** an event, **wait** for an event, know whether an event has been sent (**notified**), **request** a task.
- **Control operators**: usual control operators, such as **variable modifications**, **loops** and **tests**.
- **Complexity operators**: operators to model a number of operations on integers (**EXECI**), floats (**EXECF**) or custom (**EXECC**).
- **Temporal operators**: operators to model deterministic and non-deterministic physical **delay**.

This set of operators makes it possible to describe applications' communications and algorithms whilst forcing the modeler to abstract data, thanks to channels that merely account for the amount of transmitted data.
The LOTOS semantics of all task operators is further described in Table I, column "LOTOS Semantics before mapping".

*2) Communications between tasks:* Tasks communicate using channels, events and requests. While channels are used to model data stream between tasks - i.e., channels carry unvalued samples -, events and requests are meant to model synchronization schemes.
Three channel types have been defined: **BR-NBW** (Blocking Read - Non-Blocking Write, i.e., infinite FIFO), **BR-BW** (Blocking Read - Blocking Write, i.e., finite FIFO), **NBR-NBW** (Non-blocking Read - Non-Blocking Write, i.e., a set of elements, that is a memory).
The semantics of these channels is quite obvious to describe in LOTOS: since channels convey no value, but only a number of samples, the two first channels can easily be translated into a simple process (see Figure 2) sharing a natural value (which represents the number of elements in the FIFO) between two processes using two gates: one gate to add a sample ($wr__ch$), another one to remove a sample ($rd__ch$). The last channel type (NBR-NBW) is also translated into a similar LOTOS process apart from the fact that no counter is necessary - since its is always possible to read and write -, and so no guards (*[]* operator) are used before the actions on gates $wr__ch$ and $rd__ch$.

**Events are meant to model synchronization between tasks**. They can carry up to three parameters. Event communication semantics are the following: infinite FIFO, finite blocking FIFO, and non-blocking FIFO. The two first semantics have been selected because they reflect common synchronization schemes of embedded systems. The last one (Non-blocking finite FIFO) is particularly useful to model signal exchanges between tasks: indeed, software and hardware signals usually erase the previous one (e.g., Programmable Interrupt Controller, or UNIX signals). A separate LOTOS process accounts for each of the three semantics using the *Queue_nat* algebraic type. Figure 3 illustrates a non-blocking finite FIFO (the most complex case) for an event carrying only one natural parameter. Five cases have been taken into account:

1) The FIFO is not empty, and so, a *wait* action can be performed on the FIFO.
2) The FIFO is not full, and so, an event can be added to the FIFO (*notify*).
3) The FIFO is full, and so, an event can be added to the FIFO (*notify*) after the oldest one has been removed.
4) The FIFO is not empty, the *notified* action returns the value 1.
5) The FIFO is empty, the *notified* action returns the value 0.

Unlike channels and events which are one-to-one communications, **requests are many-to-one communications**. They

| Type | Task operators | LOTOS Semantics before mapping | LOTOS Semantics after mapping |
|---|---|---|---|
| **Channel** | Write $n$ samples to a channel | $n$ Write operations in FIFO, i.e., $n$ times action on gate $wr__ch$, see Figure 2 | $n$ cycles, and a request on a bus. |
| | Read $n$ samples from a channel | $n$ read operations from FIFO, i.e., $n$ times action on gate $rd__ch$, see Figure 2 | $n$ cycles and a request on a bus. |
| **Event** | Notify an event | Adds an event to the corresponding FIFO, i.e., performs an action on gate $notify__evt$, see Figure 3 | Same as before mapping. |
| | Wait for an event | Tries to get an event from a FIFO, i.e., performs an action on gate $wait__evt$, see Figure 3 | Same as before mapping. |
| | Notified | Returns the number of event in a FIFO using action $notified__evt$, see Figure 3 | Same as before mapping. |
| **Request** | Send a request (operator is called "request") | FIFO management is similar to the one used for events | Same as before mapping. |
| **Control** | loop, variable modifications, tests | Direct translation in LOTOS with corresponding LOTOS operators | Direct translation. Operators are executed in 0-cycle. |
| **Complexity** | EXECx $n$,$m$ i.e., between $n$ and $m$ integer instructions | No semantics before mapping, i.e., this operator is ignored | The task executes between $n * perf$ and $m * perf$ cycles with perf being a constant value depending on the hardware performance on which the task is mapped. |
| **Temporal** | Delay $d_{min}d_{max}$ unit | No semantics before mapping, i.e., this operator is ignored | The task is blocked for Between $n$ and $m$ cycles with $n = d_{min} * frequency$ and $m = d_{max} * frequency$. |

TABLE I
TASK OPERATORS

```
process ChannelBRBW__ch[rd__ch, wr__ch](samples:nat) : exit := (
[samples < 8] -> (wr__ch; ChannelBRBW__ch[rd__ch, wr__ch](samples + 1))
[]
[samples > 0] -> (rd__ch; ChannelBRBW__ch[rd__ch, wr__ch](samples - 1)) )
```

Fig. 2. Application-level LOTOS semantics for a BR-BW channel

rely on *n-to-one* infinite FIFO. The translation of requests is similar to the one of FIFO for events, apart from the fact that notification gates are instantiated n times, e.g., $notify_i$ with $i \in 1 \ldots n$.

### C. Semantics at mapping level

A mapping involves an application (i.e., a set of tasks and communications between those tasks), an architecture (i.e., a set of hardware nodes), a distribution of tasks onto hardware nodes (e.g., map the task *task1* onto the CPU *cpu1*), and a mapping of communication channels onto buses / memories. We have therefore defined a transformation function $tf()$ that takes as argument all above mentioned elements and generates a LOTOS specification (see Figure 4).

*1) Mapping issues:* The mapping phase is meant to answer whether a system made of an application executed on a given architecture satisfies a set of constraints, or not. More precisely, a mapping shall resolve contentions on shared resources (typically, a CPU, a bus, etc.) and therefore answer whether the computational and communication power offered by the architecture can execute the desired application, i.e., respect deadlines, etc. The LOTOS semantics is first defined with those issues in mind. As a consequence, the LOTOS specification of a mapping should take into account:

- The access control to shared resources, e.g., for tasks: access to CPUs, and for communication: access to buses. To take into account those accesses, we explicitly take into account operating systems' scheduling policies as well as arbitration policies of buses.
- The time taken by tasks to execute operators, and the time taken by communications, e.g., bus and memory latencies.

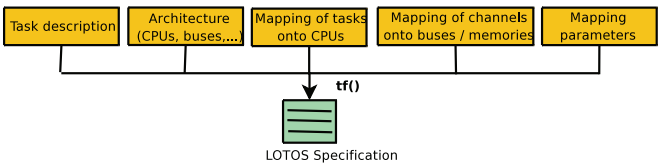


Fig. 4. General approach

*2) The Mapping-to-LOTOS transformation:* All task operators and hardware nodes parameters are taken into account by the Mapping-to-LOTOS transformation ($tf()$). However, we still do not take into account most recent proposals on resource sharing in DIPLODOCUS, e.g., we do not yet support hierarchical scheduling and virtual nodes [27] but no technical limitation has been identified which could hamper

```
process Event__evt[notify__evt, wait__evt, notified__evt](fifo_1:Queue_nat, fifo_val_1:nat, nb:nat, maxs:nat) : exit :=
  [not (Empty (fifo_1))] -> wait__evt !First(fifo_1);p_0_Event__evt[notify__evt, wait__evt, notified__evt](Dequeue(fifo_1),
fifo_val_1, nb-1, maxs)
  [] [nb<maxs] -> notify__evt ?fifo_val_1:nat;p_0_Event__evt[notify__evt, wait__evt, notified__evt](
Enqueue(fifo_val_1, fifo_1), fifo_val_1, nb+1, maxs)
  [] [nb == maxs] -> notify__evt ?fifo_val_1:nat;p_0_Event__evt[notify__evt, wait__evt, notified__evt](
Enqueue(fifo_val_1, Dequeue(fifo_1)), fifo_val_1, nb, maxs)
  [] [not (Empty (fifo_1))] -> notified__evt!1;p_0_Event__evt[notify__evt, wait__evt, notified__evt](
fifo_1, fifo_val_1, nb, maxs)
  [] [Empty (fifo_1)] -> notified__evt!0;p_0_Event__evt[notify__evt, wait__evt, notified__evt](
fifo_1, fifo_val_1, nb, maxs)
endproc
```

Fig. 3. Application-level LOTOS semantics for a Non-blocking Finite FIFO

their integration.

Basically, the LOTOS specification is built upon four functional blocks:

- The **Scheduling manager** schedules tasks on each CPU. $tf()$ transforms each task into a state machine modeled in LOTOS: preemption can occur when a task is blocked in a state, but never when a task performs a transition from one state to another.
- The **Communication manager** handles channel-based communication between tasks running on the same CPU, or on different CPUs. Events and requests are assumed not to take communication resources. Indeed, the amount of data represented by those two synchronization features are assumed to be negligible with regards to channel-based communications. Similar assumptions were made for the simulation semantics [2] (which is less abstract and more tailored to simulation runtime issues).
- The **Task execution manager** handles operators to execute in each task that is transitions between task states.
- The **Clock manager** handles clock cycles on hardware nodes, i.e., it activates necessary hardware nodes when a new cycle begins.

The main process of the LOTOS specification works as follows:

1) At first, an initialization phase is used to settle various data structures, for each CPU (e.g., all tasks of a CPU are put in "ready" state), and for the communication manager: data structures related to channels, queues related to events, and so on.
2) A main loop on clock cycles is started: The system waits for the next tick (*tick* is a LOTOS action). Then, each CPU plus its operating system are considered one after another. Basically, a CPU is meant to interpret DIPLODOCUS application-level operators of the selected task:
   a) Depending on its clock rate, the CPU is activated or not by the Clock manager.
   b) If it is activated, then a first test is performed to see whether one task is in *running* state, or not.
   c) If one task is in *running* state, then, the running state is activated from its former state. The task executes until either (i) it blocks (for example, it tries to receive one given event, and that event is not available): in that case, the scheduler is called, or (ii) it can perform an instruction consuming cycles (e.g., writing a sample to a non-full channel).
   d) When the scheduler is called, it first checks whether at least one task is runnable. If no task is runnable, the CPU goes *idle*. Otherwise, a scheduling algorithm - implemented in LOTOS - is called to select another task. Then, the state machine of that task may be called, and so on.
3) Once all CPUs have been selected, a communication manager resolves all inter-CPUs communication, i.e., all communications set-up by tasks in previous cycle (i.e., all *read*, *write*, *notify events*, etc.) are really performed only when all CPUs have terminated that cycle. This ensures (i) that a sample written on a CPU during a cycle may not be read by another CPU in the same cycle, and (ii) that the order of CPU evaluation has no impact on results.

The $tf()$ function may also generate debug information in the form of LOTOS actions performed at well-chosen points: actions to show scheduler data structures (e.g., list of runnable or blocked tasks), actions to monitor tasks states, actions to monitor the communication manager, etc.

Finally, $tf()$ has been defined with combinatory explosion in mind. Hence, $tf()$ tries to precompute possible synchronization between LOTOS processes: if possible, these synchronization are removed, and resulting processes put in sequence. Combinatory explosion may also be due to (i) non-deterministic elements: for example random, choice and temporal operators of tasks; (ii) Non-determinism in scheduling models: for example, in the round-robin scheduling policy, the possible indexes of tasks, in the tasks list. Abstraction is a key factor to reduce combinatory explosion. The main idea behind abstractions is to remove all software and hardware-related concerns that have no or little impact on evaluated properties (e.g., load on CPUs and buses). The next subsection is dedicated to abstractions.

### D. Abstractions

*1) Task abstraction (see Table I, column "LOTOS Semantics after mapping"):*

- **Communication operators**. These operations are given a cost (in clock cycles), and are executed by the execution manager along with the communication manager, to make

request on related buses. The cost in cycle depends of the hardware platform. For example, writing an 8-byte sample on a 32-bit processor takes two cycles. Also, the communication manager is involved for storing output samples, and for providing data to input operations. Note that these operations may be blocking, and so, the scheduling manager may also be involved.

- **Cost operators** are abstracted with a number of cycles depending on the hardware platform.
- **Other operators**: choice, loop, variable manipulation, etc. These operations are executed by the task execution manager. They take no cycle since there are used for control modeling only, i.e., the execution cost in DIPLODOCUS is modeled only with *EXECx* operations.
- **Temporal operators**: They are abstracted with a number of cycles.

*2) CPU abstractions:*

- **Parameters of CPU**: Data size (used for communication in channels), size of default integer and floating point data (used for EXEC operations), cost for each EXECx instructions, pipeline size (used for calculating the penalty induced by miss branching), miss branching rate, data cache-miss ratio and penalty, time to enter/leave the idle mode, clock ratio.
- **The Operating System** is taken into account with scheduling algorithms (e.g., Preemptive priority-based, round-robin), switching time, synchronization management (events, requests) and communication delay (buffering for handling channels).

*3) Communication abstractions (buses, memories):* Buses are meant to carry data samples with an arbitration policy between requests. The time a given transfer takes depends on the width of the bus. Bus arbitration is done on each cycle. Memory delays are modeled throughout bus latencies and cache-miss rates at CPU level, as proposed for the simulation semantics [2].

### E. Formal verification

LOTOS specifications may be derived either from an application modeling, or from a mapping of applications onto a given architecture (Figure 1).
At application level, tasks have a maximum concurrency between themselves, concurrency which is reduced when the mapping phase occurs: buses and CPUs are shared resources. For example, two tasks mapped on the same CPU do not execute in parallel any more. An interesting property would be that formal traces obtained after mapping are a subset of formal traces obtained before mapping, that is, a mapping only constrains possible application-level traces, without violating application-level safety properties (e.g., absence of deadlock). One of our ongoing work is to prove that scheduling and arbitration policies we have defined preserve safety properties proved at application level, that is a mapping is always *correct-by-construction* with regards to safety properties.

## V. TOOLKIT

### A. General overview

TTool [4] is an open-source toolkit initially developed for the TURTLE UML profile [28]. It now supports several other UML profiles such as the *CTTool* profile [29] and *DIPLODOCUS* [1]. TTool includes diagramming facilities, code generators (LOTOS, etc.) and graph analysis tools.
TTool includes a Graphical User Interface for drawing DIPLODOCUS UML diagrams. From those diagrams, simulation or formal analysis (see Figure 5) may be performed. Underlying simulation and validation languages (e.g., LOTOS) are totally hidden to DIPLODOCUS users. From LOTOS specification, TTool relies on CADP [26] to generate a reachability graph than can be analyzed directly in TTool (in particular, to detect deadlock situations), to minimize it, and to compare it with other graphs (bisimulations). UPPAAL offers simulation and model-checking capabilities. Furthermore, TTool has very fast simulation capabilities [2].

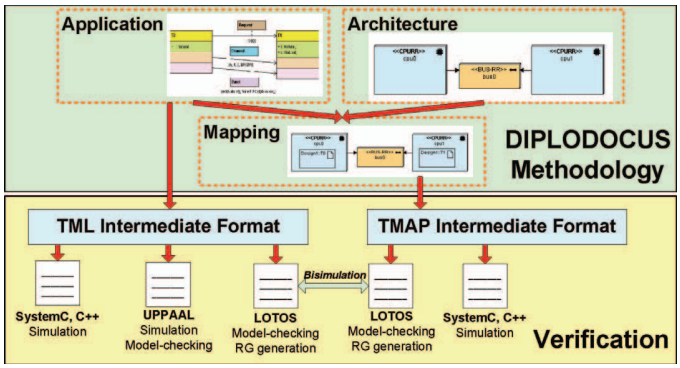


Fig. 5. TTool for DIPLODOCUS: code generation capabilities

### B. Property analysis with TTool and CADP

1) At first, an application is modeled (e.g., MPEG2 decoding, see next section). From that modeling a reachability graph is generated (let us call it $rga$), and model-checking techniques are used to prove a set $P$ of properties on the application itself.
2) A hardware architecture is described in terms of CPUs, buses, etc..
3) From the mapping (tasks onto CPUs, etc.), a LOTOS specification is generated, and from that specification, CADP is used to obtain a reachability graph $rg$. The following verification features are supported:
   - Minimizing the reachability graph to *tick* actions. From that minimization, the longest path of ticks is calculated, therefore resulting in a performance information on the application (e.g., *Worst Case Execution Time*).
   - Minimizing the reachability graph to *tick* and *transferOnBusX*. From that minimization, loads on buses can be deduced. Similar techniques can be used to compute CPU loads.

- Comparing $rg$ and the reachability graph generated at application level, i.e., to $rga$. To do so, a toolkit integrated in TTool first modifies $rg$ so as to make $rg$ action names compatible with the one of $rga$, then CADP minimizes the resulting graphs: if it is proved that $rg \subset rga$, then safety properties proved at application level are preserved.
- Of course, other usual model-checking techniques can be directly applied to $rg$ (e.g., using CADP).

## VI. Case Study

### A. MPEG2

MPEG2 video compression is based on motion estimation between pictures, discrete cosine transforms, quantization, and Huffman encoding. The case study initially introduced in [3] focused on the application and architecture modeling of an MPEG decoder. In this paper however, emphasis is put on proofs of functional properties.

### B. Application modeling

MPEG2 decoding is modeled with 6 tasks corresponding to the main MPEG2 functional blocks: a main task (*MPEG_Decoder*), a task to decode Huffman codes (*VLC*), a task to extract blocks (*Zigzag*), a task to perform the inverse quantization (*IQuantization*), a task to perform the inverse discrete cosine transform (*IDCT*) and finally a task to perform the motion compensation (*Motion_Compensation*). Basically, *MPEG_Decoder* drives all other tasks. *VLC* sends its results to *Zigzag* which itself forwards its output to *IQuantization* and then to *IDCT*. Pictures are then rebuilt based on *Motion_Compensation*. Channels are used between those tasks to exchange samples modeling pictures macroblocks - in MPEG2, images are cut into fixed-size subparts of pictures: those subparts are called macroblocks-, and events / requests are used between *MPEG_Decoder* for control purpose.

### C. Architecture and mapping

Several candidate architectures are experimented. A first basic mapping consists of one single CPU on which all tasks are mapped. A second option is to map the *MPEG_Decoder*, *Zigzag* and *Motion_Compensation* tasks on one CPU named *risc01*, and the others on another CPU named *risc02*: (see Figure 6: for space reasons, the mapping of channels, events and requests is not shown on the figure). In this second mapping, subsequent tasks are mapped onto different CPUs thus allowing for overlapping executions. A task can for instance read a few samples from memory before the previous task has terminated its computations. This remark is not 100% true for IDCT whose data-dependency constraints are both block-level and sample-level.

### D. Property analysis

The intended system is expected to decode 25 images / seconds. Images are of size 800x600. Thus, one image decoding is expected to be performed in less than 40ms. To ensure that the system is schedulable with that constraint, our testbed was the following:

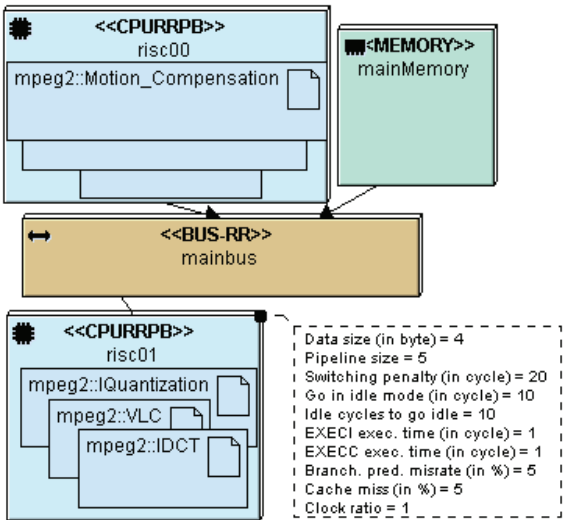


Fig. 6. Using TTool to map tasks onto two CPUs

- We have chosen a maximum frequency for the CPUs (200 Mhz). From this, we obtain a maximum number of cycles for each macroblock equal to 1066 cycles.
- Then, from the LOTOS specification of each mapping, CADP has computed a reachability graph (RG).
- Then, we have minimized each RG to *tick* actions: thus, we were able to count the maximum number of cycles taken by the system to decode one picture. To count that number, we have used a TTool feature that searches for the longest path on the RG.
  - For the mono-processor architecture: maximum path = 1486 cycles (i.e., at most 1486 cycles are necessary to perform the decoding of one macroblock).
  - For the two-processor architecture: maximal path = 1041 cycles.

  Finally, the system is schedulable with the 2-CPU approach, and not with only one CPU.

## VII. Conclusions and Future Work

The paper presents an environment - named DIPLODOCUS - for formal functional and performance analysis of complex embedded and distributed systems. A system is described with communicating tasks, hardware architectures and a mapping of tasks and channels onto a hardware architecture. A formal semantics is provided to tasks, communication between tasks and hardware architectures, making it possible to perform formal analysis before and after mapping. Moreover, DIPLODOCUS has been implemented in a UML-based open-source toolkit - named TTool. Formal analysis can be performed with absolutely no knowledge of formal techniques. The DIPLODOCUS environment has been experimented within the scope of several case studies, including an MPEG2 application, and industrial

case studies [27]. As stated in section IV, abstractions are a key factor of our models in order to alleviate combinatory explosion (and to greatly increase simulation speed [2]):

- *Data abstraction*: Only the amount of exchanged data between functional entities is taken into account. Data dependent behavior is made explicit within the task model by means of non-deterministic operators.
- *Control flow abtraction*: Symbolic instructions (EXECx) stand for operations to be executed on an execution device (like CPUs).
- *Hardware abstractions*: Hardware device-dependent behavior is captured by parameters like clock frequencies and cache miss probabilities.

Trading off accuracy against model complexity of hardware components will remain subject to our research. For example, instruction cache-misses and data cache-misses have been accounted for by static probabilities so far. Indeed, as algorithmic details are represented by symbolic instructions, the real code of the application is not available thus making state of the art cache models unsuited. Furthermore, the accuracy of bus and memory models shall be validated against a real embedded system. A fair comparison with a real implementation shall therefore reveal whether a set of parameters can be found to limit the inaccuracy to a reasonable percentage. To simplify the modeling of systems making extensive use of DMA engines, a specific UML stereotype could be introduced. This way, the designer would not have to model DMA transfers explicitly using a dedicated execution unit.

While being already operational, our environment will be enhanced with two main features. On the one hand, we will define a refinement process from the application modeling step to the after-mapping step, in order to preserve properties proved at application level. On the other hand, we intend to assess and adapt post-mapping hardware abstractions - e.g., the ones used for memories and buses - by confronting mapping results with real implementations.

## REFERENCES


[1] L. Apvrille *et al.*, "A UML-based environment for system design space exploration," in *13th IEEE International Conference on Electronics, Circuits and Systems (ICECS'2006)*, Nice, France, Dec 2006.

[2] D. Knorreck, L. Apvrille, and R. Pacalet, "Fast simulation techniques for design space exploration," in *Objects, Components, Models and Patterns*, ser. Lecture Notes in Business Information Processing, vol. 33. Springer Berlin Heidelberg, 2009, pp. 308–327. [Online]. Available: http://www.springerlink.com/content/r425w073k44k5441/

[3] L. Apvrille, "Ttool for diplodocus: An environment for design space exploration," in *8th annual international conference on New Technologies of Distributed Systems (NOTERE'2008)*, Lyon, France, jun 2008.

[4] LabSoc, "The TURTLE Toolkit," in *http://labsoc.comelec.enst.fr/turtle/ttool.html*.

[5] ISO-LOTOS, "A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour," in *Draft International Standard 8807, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection*, Geneva, July 1987.

[6] J. Bengtsson and W. Yi., "Timed automata: Semantics, algorithms and tools," in *Lecture Notes on Concurrency and Petri Nets*. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

[7] W. Muhammad *et al.*, "Abstract application modeling for system design space exploration," in *Euromicro Conference on Digital System Design (DSD'06)*, Dubrovnik, Croatia, August 2006.

[8] F. Balarin *et al.*, *Hardware-Software Co-Design of Embedded Systems, The POLIS Approach*, 5th ed. KLUWER ACADEMIC PUBLISHERS, 2003.

[9] Y. Watanabe, "Metropolis : An integrated environment for electronic system design." Cadence Berkeley labs, 2001.

[10] P. V. D. Wolf *et al.*, "A methodology for architecture exploration of heterogeneous signal processing systems," in *1999 IEEE Workshop on Signal Processing Systems (SiPS99)*, 1999.

[11] A. Chatelain *et al.*, "High-level architectural co-simulation using Esterel and C," in *Proc. of IEEE/ACM symposium on Hardware/software codesign*, April 2001.

[12] I. Assayad and S. Yovine, "A framework for modelling and performance analysis of multiprocessor embedded systems: Models and benefits," in *Proceedings of the 8th conference on Nouvelles Technologies de la Distribution (NOTERE'2007)*, Marrakech, Marocco, June 2007.

[13] T. Schattkowsky *et al.*, "A model-based approach for executable specifications on recon figurable hardware," in *Proc. of the Design, Automation and Test in Europe (DATE)*, Nov 2005, pp. 692–697.

[14] P. Kukkala *et al.*, "Performance Modeling and Reporting for the UML 2.0 Design of Embedded Systems," in *Proc. of the 2005 International Symposium on System-on-Chip)*, Nov 2005, pp. 50–53.

[15] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with uml and marte," in *Design, Automation and Test in Europe Conference and Exhibition, 2009. DATE'09.*, April 2009, pp. 226–231.

[16] B. Ristau, T. Limberg, and G. Fettweis, "A mapping framework for guided design space exploration of heterogeneous mp-socs," *Design, Automation and Test in Europe, 2008. DATE'08*, pp. 780–783, March 2008.

[17] K. Avnit and A. Sowmya, "A formal approach to design space exploration of protocol converters," in *Design, Automation and Test in Europe Conference and Exhibition, 2009. DATE'09*, April 2009, pp. 129–134.

[18] A. Hamann, M. Jersak, K. Richter, and R. Ernst, "A framework for modular analysis and exploration of heterogeneous embedded systems," *Real-Time Syst.*, vol. 33, no. 1-3, pp. 101–137, 2006.

[19] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques, IEE Proceedings -*, vol. 152, no. 2, pp. 148–166, Mar 2005.

[20] M. Hendriks and M. Verhoef, "Timed automata based analysis of embedded system architectures," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, April 2006, pp. 8 pp.–.

[21] A. Viehl, T. Schonwald, O. Bringmann, and W. Rosenstiel, "Formal performance analysis and simulation of UML/sysml models for esl design," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, pp. 1–6, March 2006.

[22] R. Marculescu, U. Y. Ogras, and N. H. Zamora, "Computation and communication refinement for multiprocessor soc design: A system-level perspective," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 3, pp. 564–592, 2006.

[23] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer, "Performance by unified model analysis (puma)," in *WOSP '05: Proceedings of the 5th international workshop on Software and performance*. New York, NY, USA: ACM, 2005, pp. 1–12.

[24] P. Wodey, G. Camarroque, F. Baray, R. Hersemeule, and J.-P. Cousin, "Lotos code generation for model checking of stbus based soc: the stbus interconnection," *This paper appears in: Formal Methods and Models for Co-Design, 2003. MEMOCODE '03. Proceedings. First ACM and IEEE International Conference on*, pp. 204–213, June 2003.

[25] OMG, "UML 2.0 Superstructure Specification," in *http://www.omg.org/docs/ptc/03-08-02.pdf*, Geneva, 2003.

[26] "The CADP toolkit," http://www.inrialpes.fr/vasy/cadp.

[27] C. Jaber, A. Kanstein, L. Apvrille, A. Baghdadi, P. L. Moenner, and R. Pacalet, "High-level system modeling for rapid hw/sw architecture exploration," in *Proc. of the 20th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP'2009)*, June 2009.

[28] L. Apvrille *et al.*, "TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit," in *IEEE transactions on Software Engineering*, vol. 30, no. 7, Jul 2004, pp. 473–487.

[29] S. Ahumada *et al.*, "Specifying Fractal and GCM components with UML," in *XXVI International Conference of the Chilean Computer Science Society (SCCC'07)*, Iquique, Chile, nov 2007.