# UML design for dynamically reconfigurable multiprocessor embedded systems

Jorgiano Vidal*, Florent de Lamotte*, Guy Gogniat*, Jean-Philippe Diguet*, Philippe Soulard†
* Lab-STICC – European University of Brittany - UBS – CNRS, UMR 3192
Centre de Recherche - BP 92116 – F-56321 Lorient Cedex - FRANCE
† SODIUS – 6 rue de Cornouaille – F-44300 NANTES - FRANCE

*Abstract*—**In this paper we propose a design methodology to explore partial and dynamic reconfiguration of modern FPGAs. We improve an UML based co-design methodology to allow dynamic properties in embedded systems. Our approach targets MPSoPC (Multiprocessor System on Programmable Chip) which allows area optimization through partial reconfiguration without performance penalty. In our case area reduction is achieved by reconfiguring co-processors connected to embedded processors. Most of the system is automatically generated by means of MDE techniques. Our modeling approach allows designers to target dynamic reconfiguration without being expert of modern FPGAs as many implementation details are hidden during the modeling step. Such a methodology allows design time speed-up and a significant reduction of the gap between hardware and software modeling. In order to validate our approach, an object tracking application has been implemented on a reconfigurable system composed of 4 embedded processors and 3 co-processors. Dynamic reconfiguration has been performed for one co-processor which dynamically implements 3 different computations.**

## I. Introduction

Advances in reconfigurable technologies allow entire multiprocessor systems to be implemented in a single FPGA (Multiprocessor System on Programmable Chip, MPSoPC). Designing such systems with existing tools will soon become unmanageable due to complexity and productivity reasons. One promising solution to mitigate designer task is to further increase abstraction levels in order to hide many implementation details. Such an approach will allow system designer to have access to various SW and HW technologies without being an expert of all of them. Several efforts have been performed these last years to promote UML (Unified Modeling Language) [1] in order to consider it as an efficient language to model multiprocessor systems. Considering a single language to design these systems is very interesting and allows to speed up design time and system integration. In [2], the authors have identified UML properties required to model embedded systems. More recently the MARTE profile [3] (Modeling and Analysis of Real-time and Embedded systems) which is in adoption by OMG (Object Management Group) addresses real-time embedded system modeling issues.

To build an efficient design flow it is mandatory to rely on a well defined model of platform. Typical MPSoPC systems (Figure I) are composed of a set of embedded processors executing tasks which communicate together to implement the system functionality. Specific IPs (Intellectual Property) can be considered to speed up the computation of intensive tasks. These IPs can be used as co-processors directly connected to processors or accelerators connected to internal buses. During the development process flexibility, performance and area correspond to key concerns for the designer. Existing FPGA technology (e.g. Xilinx Virtex devices) offers dynamic and partial reconfiguration (DPR) features that can be advantageously considered to address these points. DPR allows HW tasks to share the same resource if their execution is exclusive. Such a solution is very interesting as it reduces the total system area while still meeting performance constraints. Supporting DPR allows multiprocessor systems to replace co-processors or accelerators at run time which significantly increases platform flexibility. Unfortunately there is a lack of tools addressing the design of reconfigurable MPSoPCs at the abstraction level mentioned above. Thus in this paper we propose to reduce the gap between the design of reconfigurable MPSoPCs and the targeted technology (FPGA).

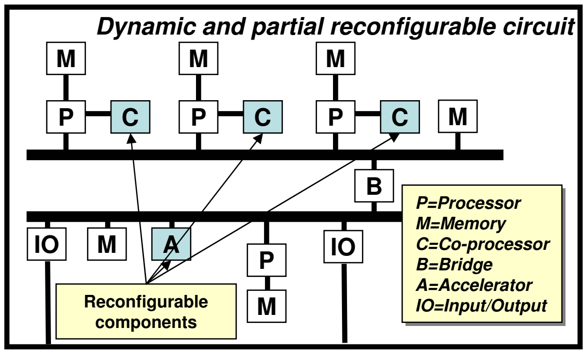


Fig. 1. Reconfigurable multiprocessor system: Accelerators and co-processors can be dynamically replaced.

In [4], an UML based co-design methodology for embedded systems has been presented. We propose to enhance this approach with reconfigurability modeling techniques. Our co-design approach considers application and platform models to be designed separately. A further allocation step is performed to obtain the allocated model, which represents the final embedded system. In our methodology we consider a platform model as the one shown in Figure I where we limit the reconfiguration to co-processors. Thus our approach allows the design of dynamically and partially reconfigurable multiprocessor systems, where HW tasks (co-processors) are replaced at run time. We use a reconfiguration service, described in [5] to perform DPR.

The paper is organized as follows: In Section II we discuss existing efforts. In Section III we introduce the execution scheme. In section IV we introduce our modeling methodology and detail reconfiguration modeling. In Section V we present our tools chain and detail the development flow. In Section VI we show some results on an object tracking application. Finally, in Section VII we conclude and present some future work.

## II. Related work

The use of model based approaches for co-design has been discussed in [6], which pointed out some advantages: cost decrease, silicon complexity handling, productivity increase, etc. Several works have also shown the benefit of using UML for embedded system modeling [7], [8], [9], [10], [11], [12], [13]. However most of them define specific profiles to model embedded systems. Using specific profiles limits a large adoption of these approaches as they do not rely on a standard. Furthermore proposed approaches mainly target system analysis and simulation. Few of them propose SystemC code generation [8], [14] which reduces the gap between the specification and the implementation but still requires further synthesis steps.

Dynamic and partial reconfiguration modeling using UML allows such methodologies to take advantage of dynamic reconfiguration capabilities of moderns FPGAs. Although there is a lot of work on embedded system modeling using UML, only few explore dynamic and partial reconfiguration capabilities [15], [16].

In [15], authors use UML sequence diagram with specific stereotypes to model dynamic reconfiguration. Their approach is very simple and efficient, but it lacks platform modeling. In their work the system platform is fixed: a processor with a reconfigurable device as an auxiliary computing unit. Also, it does not support dynamic and partial reconfiguration.

In [16], authors detail a dynamic reconfigurable system by extending UML/MARTE with specific stereotypes. Their approach is developed in a design environment called GASPARD [17], where VHDL code is generated. This approach is very target-dependent and requires a strong level of expertise as all elements of the Xilinx partial reconfiguration design process need to be modeled.

Compared to previous efforts, our approach only uses standard UML/MARTE elements. Specific properties are required by the code generation tool that is target dependent. In order to allow a large adoption of dynamic and partial reconfiguration we hide from system designer many technology details. Reconfiguration services and resources are automatically added to the system during the code generation step.

## III. Execution scheme

To define a dynamically reconfigurable MPSoPC we consider a platform architecture as described in Figure I. In our case the application does not exhibit reconfiguration features. The application execution is defined as a static one since the execution of the tasks are not data or control dependent. However when two tasks are executed exclusively they can share the same HW area. In that case the reconfiguration is performed at the end of the execution of the first task before launching the second one.

Data used by processors are stored locally into distributed memories as shown in Figure I. Communications between two tasks contain all required data to perform the corresponding computation. Communications between processors are implemented through a shared memory [18]. Communications between a processor and a co-processor are performed through a dedicated channel. A co-processor can only communicate with its associated processor. When a processor sends an event to a co-processor, it contains the event and the associated data. The same scheme is used when a co-processor sends

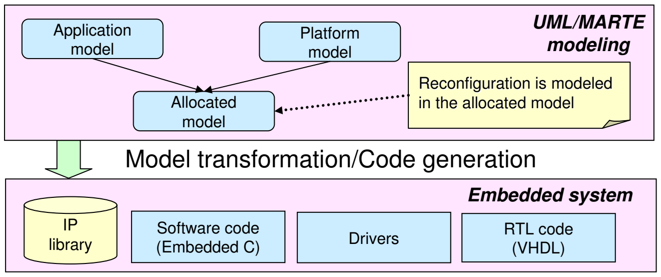

Application model
Platform model
UML/MARTE modeling
Reconfiguration is modeled in the allocated model
Allocated model
Model transformation/Code generation
Embedded system
IP library
Software code (Embedded C)
Drivers
RTL code (VHDL)

Fig. 2. Model transformation. Allocated model is used as input to the code generation tool.

an event back to a processor. Thus co-processors do not store any data between two executions. This execution scheme is considered to efficiently perform dynamic reconfiguration of co-processors.

In this paper we explore dynamic and partial reconfiguration of modern FPGAs. FPGA configuration is performed by loading a bitstream into the device. The bitstream contains the system to be implemented into the FPGA. When dynamic and partial reconfiguration is done a partial bitstream is required as only a part of the system is modified (in our case co-processors).

To perform dynamic reconfiguration we need some specific features. In the paper we call *reconfiguration service* all the elements used to perform dynamic and partial reconfiguration [5]. This service is composed of hardware and software elements (e.g. ethernet controller, Xilinx ICAP (Internal Reconfiguration Access Port), reconfiguration API) required to perform bitstream loading. In the next sections we rely our modeling techniques on these characteristics.

## IV. Modeling

Our co-design methodology takes into account three main elements to be modeled: **the application**, **the platform** and how application fits into the platform, what we call **the allocation**. The application is defined as a set of communicating tasks where each task performs some computations. The platform is defined as a set of connected IPs (components) and the allocation is a mapping between application tasks and platform components. Figure 2 exhibits the dependencies between the three models. The allocated model is used to generate the final system. The target platform is composed of a set of IPs where each IP can be defined in a library or designed during the development cycle. Embedded C code is used to implement software tasks. The code generation tool supports the set of transformation rules used to perform the system generation from the three previous models.

We achieve dynamic and partial reconfiguration through the allocated model, which explores a reconfigurable platform by performing a mapping between a static application (not reconfigurable) and a reconfigurable platform. We detail our modeling techniques in Sections IV-A (application modeling), IV-B (platform modeling) and IV-C (allocation modeling). In Section IV-D we introduce the reconfigurability modeling and associated code generation.

### A. Application

The application is defined as a set of tasks that perform computations and communicate to exchange data. Each task receives data, performs some computations and sends results

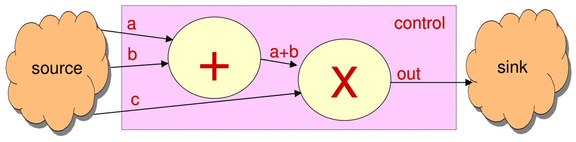


Fig. 3. Application tasks graph: two operations are performed sequentially.

to other tasks. Our modeling approach is well suitable for data flow systems (e.g. audio and video coders/decoders) that are generally specified by means of communicating tasks. Communications among tasks are done by events. An event contains a type and associated data. Upon an incoming event, a task performs computation and sends results to other tasks.

Figure 3 shows an application example with two tasks (in that case two operations). It takes three inputs: $a$, $b$ and $c$. The first two inputs are sent to the first task and the third input is sent to the second task with the result from the first task. As it can be observed, both tasks are exclusive i.e. they are not executed at the same time, as the second one needs a result from the first one. The designer can take benefit of this execution dependency to share the same HW area for both tasks.

In order to model such tasks using our methodology we use UML components. Each component behavior is performed by classes instances. The **main** behavior of a component is described by an active class, whose behavior is defined by a state machine. The only way to communicate with a component is by sending/receiving events to/from the component. The events (UML signals) must be sent through ports, that offer or require a service, defined in UML by means of interfaces. An interface defines a set of UML signals (events) that a component port accepts to receive and process. All events are sent from/to active class instances.

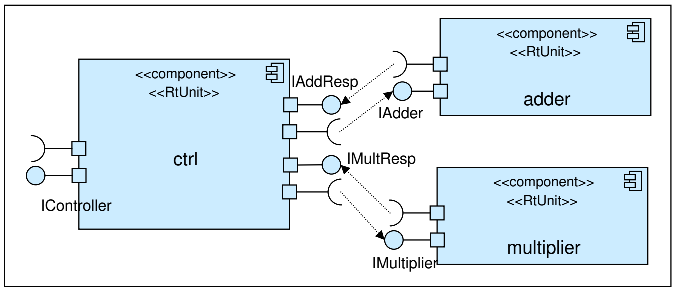
Figure 4 shows the example of Figure 3 with three components: one for a controller task, responsible for the control flow and the scheduling, and one component for each operation. Our controller task is used to send/receive events to/from the co-processors and to communicate with other processors or external components. The controller task receives the inputs and triggers both operations, one after the other. All communications are done by UML ports and use only UML signals, by sending events. Each port contains an associated interface, which will be explained later.

Figure 5 shows the interfaces and classes definition. An active class behavior is defined by means of a state machine. Passive classes contain operations that are called by active classes instances. A component contains exactly one active class instance and may contain several passive class instances.

Each component contains the MARTE HLAM RtUnit stereotype. The RtUnit indicates that the component is a real-time unit. Such information allows allocation of the application components to the platform components.

Figure 6 shows the Controller and Adder components with its objects. The controller contains one active class instance and the adder contains one active and one passive class instances.

Our application contains three components, where each one is driven by an active class instance. Each active class must realize at least one interface. The interface defines the events accepted by the component and used by the active class



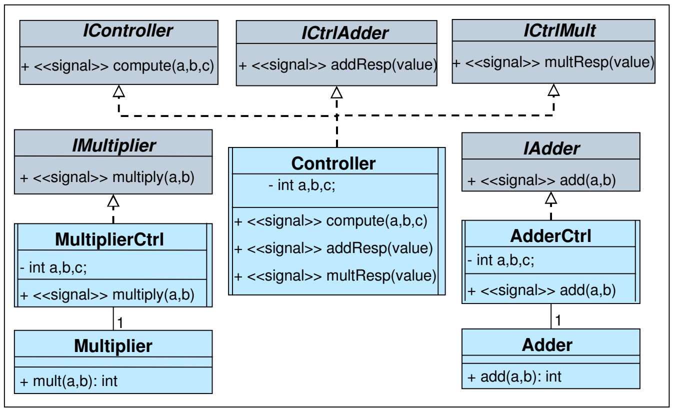

IController
+ <<signal>> compute(a,b,c)
ICtrlAdder
+ <<signal>> addResp(value)
ICtrlMult
+ <<signal>> multResp(value)
IMultiplier
+ <<signal>> multiply(a,b)
Controller
- int a,b,c;
+ <<signal>> compute(a,b,c)
+ <<signal>> addResp(value)
+ <<signal>> multResp(value)
IAdder
+ <<signal>> add(a,b)
MultiplierCtrl
- int a,b,c;
+ <<signal>> multiply(a,b)
1
Multiplier
+ mult(a,b): int
AdderCtrl
- int a,b,c;
+ <<signal>> add(a,b)
1
Adder
+ add(a,b): int

Fig. 5. Interface and class definition. The active class controls the component and uses passive classes to perform operations.

instance. Each interface is also associated with an UML port (shown in Figure 4. An outgoing event is sent by an UML port which is connected to a port in another component that accepts the event.

## *B. Platform*

The platform is defined as a set of IPs that can be reused from an IPs library or built during the development process. New IPs are built from the UML model specification [19].

UML components are used for platform modeling, where each component is an IP. We use UML MARTE HRM

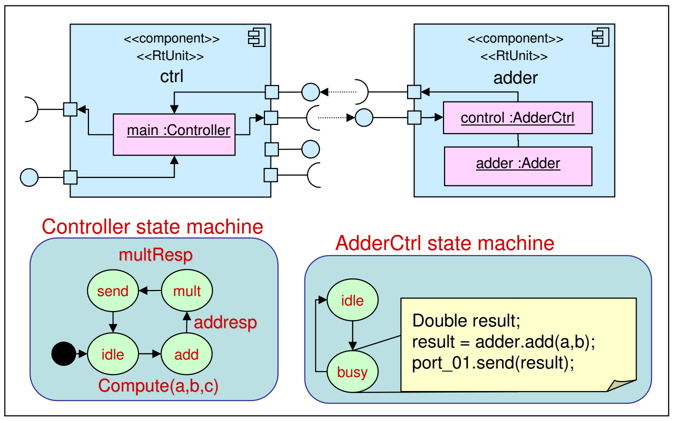


Fig. 6. Components details.

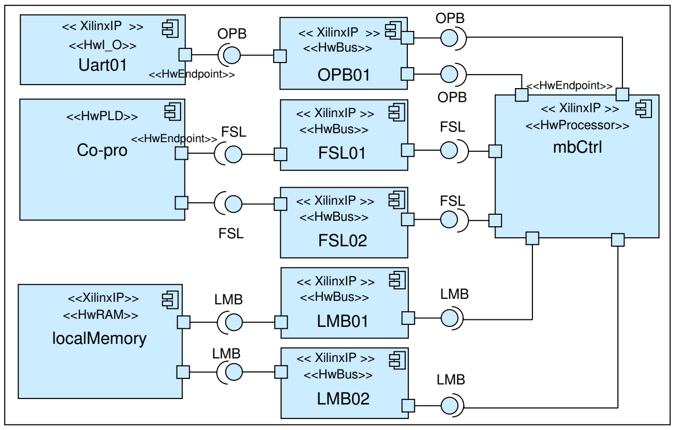


Fig. 7. Platform components.

profile elements to identify the IP characteristics e.g. version, name and address [18]. Thus specific stereotypes are used for each IP library. Figure 7 shows a typical platform in UML which contains one processor (`mbCtrl`), a co-processor (`co-pro`), an input/output component (`Uart01`), and a memory (`localMemory`). These components are connected together by buses (`OPB01`,`FSL01`,`FSL02`,`LMB01` and`LMB02`,). All the components except the co-processor contain the ≪XilinxIP≫ stereotype, with specific information for IP reuse. The co-processor can be dynamically replaced as we use FPGA dynamic and partial reconfiguration property. To identify such a property we use MARTE HRM HwPLD stereotype.

The reconfiguration service is not modeled by the designer in order to abstract the implementation details of such a technology. The service instantiation is handled by the code generation tool. This approach allows the model to target different technologies. Moreover, the reconfiguration service parameters can be specified in the code generation tool, leaving the system design clean from technology details.

## C. Allocation

Allocation is the key step in our methodology as it models the complete system to be implemented. Once the behavior is defined in the application model and the execution structure is defined in the platform model, we need to map the behavior to the platform, what we call the allocated model. This step is performed manually. All application components stereotyped **RtUnit** (from MARTE HLAM profile) are allocated to a platform computing component (processor, ASIC or PLD, as defined in MARTE HRM profile).

Figure 8 shows the allocated model of our example. The controller task is allocated to the processor and the operation tasks are both allocated to the same co-processor. The allocation of two application components to a same platform component with some specific information indicates reconfigurability and is explained in the next section. Our example requires two IPs: *adder* and *multiplier* that share the same HW area. This is a design decision, as their services are not requested at same time.

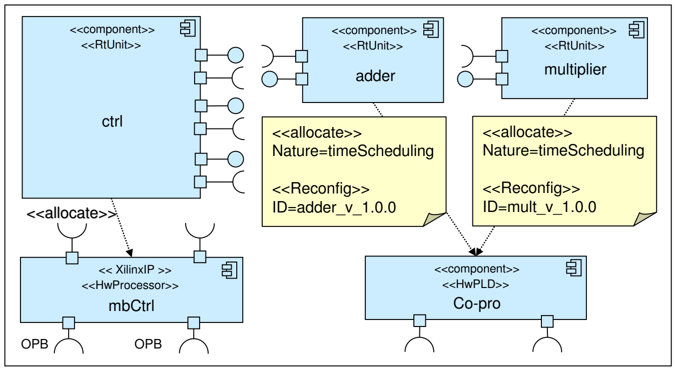


Fig. 8. Allocation of the application on the platform.

## D. Reconfiguration

Our approach to model reconfigurability is done in the allocation step where the application is mapped to the platform. We allow co-processors to be reconfigured at run time. In Figure 8 two RtUnit are allocated to the same co-processor. Specific stereotypes are used to indicate the reconfiguration modeling. As we can observe, the reconfigurability is related to the platform, whereas our application has a static behavior i.e. the application scheduling is static and predictable.

An allocation is an UML dependency stereotyped with MARTE ≪allocate≫ stereotype. *Nature* tag must be set to *timeScheduling* in order to inform the code generation tool that dynamic and partial reconfiguration is required to share HW area. For each allocation an $ID$ needs to be defined, which is used by the reconfiguration service to identify the right bitstream to be loaded upon request.

The HwPLD stereotype is used to indicate a reconfigurable component in the platform. This information is used by the code generation tool to generate the right information for the synthesis tools used to implement the system on the target platform (FPGA), this point is detailed in the next section. We have added the `Reconfig` stereotype to the allocation, to allow the code generation tool to insert the reconfiguration service call in the processor code. The `ID` tag is used to perform bitstream selection and downloading.

Figure 9 shows the generated code from a state of the controller that sends an event to a co-processor. The code generation tool searches in the controller state machine all events sent to co-processors. For each event it checks if it is sent to a reconfigurable component. If true a reconfiguration request is inserted just before sending the event. The reconfiguration command provides the bitstream `ID` to the reconfiguration service. The reconfig command is offered by the reconfiguration service. The execution of the application is aborted until reconfiguration is done. It is up to the reconfiguration service to verify configuration in order not to reconfigure a co-processor that is already implemented.

## V. DEVELOPMENT FLOW AND TOOLS

As we consider a high abstraction level for modeling we require implementation steps into existing technology and tools. Our development flow relies on existing tools for UML modeling, model transformation, embedded platform design, HW synthesis and bitstream generation. Xilinx FPGAs and

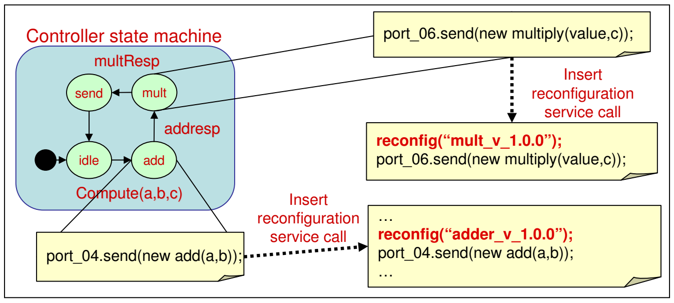

Controller state machine
multResp
send
mult
addresp
idle
add
Compute(a,b,c)
port_06.send(new multiply(value,c));
Insert reconfiguration service call
reconfig("mult_v_1.0.0");
port_06.send(new multiply(value,c));
Insert reconfiguration service call
port_04.send(new add(a,b));
…
reconfig("adder_v_1.0.0");
port_04.send(new add(a,b));
…

Fig. 9. Allocation with specific stereotypes models dynamic reconfiguration.

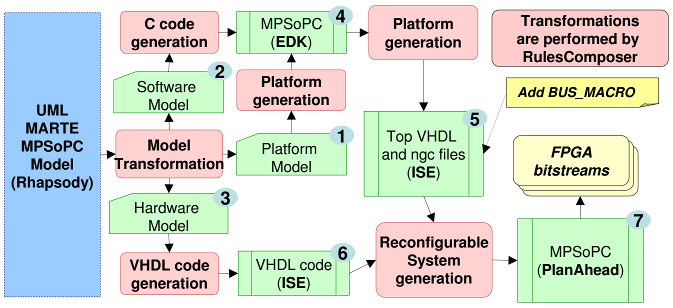


Fig. 10. Flow and used tools: The code generation tool generates projects and scripts.

associated tools are used, as they offer dynamic and partial reconfiguration technology. Figure 10 shows the development flow and considered tools.

Rhapsody UML modeler [20] is used for UML modeling. RulesComposer, a Rhapsody plugin, is used as model transformation/code generation tool. Through RulesComposer we can define transformation rules, that take as input an UML/MARTE model and generate implementation models. We have defined implementation models for platform, VHDL and C code. Code generation facility is built from the UML model, where a set of transformation rules creates a specific target model. As the UML model contains the complete system design, the first step is to extract the various system parts: platform with existing IPs (1), software code (2) and new IPs (3).

The platform and software models are used to automatically generate a Xilinx EDK project [18] (4), which is the Xilinx tool used for embedded system design. EDK defines a platform as a set of IPs and each new IP is generated with an empty behavior. Each processor contains a software project associated to it. The C model is used as input to generate processor software code in EDK. The code generation tool automatically inserts in the EDK project several IPs required to perform dynamic reconfiguration (i.e. ICAP, Ethernet controller). The EDK project is automatically transformed into an ISE project (5). Xilinx ISE is the tool used for hardware design, which accepts VHDL as input language. The EDK project exports a top VHDL file with all the components of the system.

Each VHDL model (new IPs) is used to generate ISE projects (6). At that step of the design flow we have one ISE project for the system and one VHDL project for each new IP. Then in the Xilinx flow we update the VHDL files with reconfiguration information. Each reconfigurable component must be connected through `BUS_MACRO`. A `BUS_MACRO` is a placement and routing constraint used to allow partial reconfiguration area in the target Xilinx FPGA. `BUS_MACRO` is inserted in the top VHDL file by the designer[1]. The set of ISE projects is used to generate a set of files that contains logical design data and constraints: the NGC files. Each NGC file is an IP that must be placed in the FPGA.

The NGC files are used as input to generate the PlanAhead [21] project (7) which is required to generate the set of bitstreams. With PlanAhead we place all IPs in the FPGA and mark which areas of the chip are dynamically reconfigurable. Then PlanAhead generates one bitstream for the initial configuration and a set of partial bitstreams for each possible configuration. Also, an empty bitstream is generated for each reconfigurable area in the design (for each co-processor in our architecture).

In our example 4 bitstreams were generated: **static_full.bit** (initial system configuration), **copro_add.bit** (adder component), **copro_mult.bit** (multiplier component) and an empty bitstream, **copro_blank.bit**.

Each bitstream is stored in a bitstream server on the network and contains an `ID`. At system start up the `static_full.bit` bitstream is loaded into the FPGA (with the empty co-processor by default). The reconfiguration service present in the system contains information about how to load the bitstream and where to load it from [5]. Reconfiguration commands inserted by our code generation tool uses the reconfiguration service to load the bitstreams specified by the `ID`.

## VI. Results

A more complex example was developed in order to validate our approach: An object tracking application. Figure 11 shows the UML model of the application. A video camera captures images and a set of operations is performed to track moving objects, which are then labeled and the result image is shown in a VGA screen.

The application is defined by 9 application components, with four main processing tasks: background substitution (BG subs), morphological transformation (morph trans), motion test and image update. Each one can use secondary tasks to perform compute intensive processing. Our reconfiguration design method is applied to the morphological transformation step. The morphological transformation task uses three auxiliary tasks: erosion, dilatation and reconstruction. They are performed in sequence and the next one needs the preceding result to process data.

The platform contains 60 components, including 4 processors (3 microblaze processors and one PowerPC), and is not shown due to space limitations. The PowerPC captures the next image and sends it to the first microblaze. The image processing is performed by the microblazes in sequence, and the image update and display is performed by the PowerPC.

The microblaze processor that performs morphological transformation is connected to a co-processor by two FSL BUS, one to send request and one to receive results. Such a co-processor is dynamically reconfigured to compute the

[1]Version 11 of Xilinx tools eliminates the need of BUS_MACRO, which simplifies this step. Our experiments are done in version 8.2i with EAPR (*Early Access Partial Reconfiguration*)

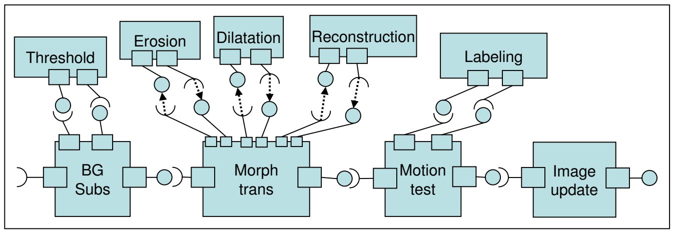


Fig. 11. Object tracking: a set of computation is done in order to track moving objects

three image processing tasks (erosion, dilatation and reconstruction). Three IPs were generated: `copro_erosion`, `copro_dilatation` and `copro_reconstruction`, each one performs an image processing step belonging to the morphological transformation. For each IP a bitstream is generated and a fourth one, the empty bitstream, is also generated. Partial bitstream is about 80KB for each co-processor. The reconfiguration throughput is close to 80Mb/s, thus total reconfiguration time is about 2ms. As three reconfigurations are performed by the morphological transformation, 6ms is the total time overhead added by reconfigurability.

The time constraint for each step in the processing is 40 milliseconds(25 images processed per second). Each of the morphological transformation step takes less than 10ms, which leads to 36ms to perform the morphological transformation: 30ms for the computation and 6ms for the reconfigurations. In that case partial dynamic reconfiguration allows a significant area optimization (we used 1 co-processor instead of 3) without performance penalties. The necessary memory used by the reconfiguration service executed on the PowerPC is less than 80KB (40KB for executable code and 32KB for local data).

## VII. Conclusion and future works

In this paper we have shown how to use a UML co-design approach to explore dynamic and partial reconfiguration characteristics of modern FPGAs. We propose a reconfigurability modeling technique which has been implemented into our co-design flow. Increasing modeling abstraction levels allows to hide implementation details to the designer, leaving focus on system requirements rather than implementation issues.

In order to achieve our goals the platform must support dynamic and partial reconfiguration. We have applied our methodology to Xilinx Virtex series FPGAs. Although our experimentations are target-specific, our approach can be applied to other dynamic and partial reconfigurable platforms. Our development flow allows a semi-automatic generation of dynamic reconfigurable MPSoPC. Results obtained demonstrated that replacing co-processors at run time allows area optimization without performance penalties.

We have explored dynamic reconfiguration by means of tasks allocation that share a same HW area. As future work we will explore reconfiguration at the application level when adaptive features are required. Such adaptability can also take advantage of FPGA reconfigurability for area optimization and needs to be modeled.

## Acknowledgment

The UML/MDA approach presented above is experimented in the RNTL06 research program MOPCOM SoC/SoPC supported by the French National Research Agency (ANR – contract 2006 TLOG 022 01), the ”cluster of clusters”, ”Media and Networks” and the Brittany and Pays de la Loire regions.

## References

[1] OMG, “The Unified Modeling Language (UML),” http://www.omg.org/uml. [Online]. Available: http://www.omg.org/uml
[2] G. Martin, “Overview of the mpsoc design challenge,” in *DAC ’06: Proceedings of the 43rd annual conference on Design automation*. New York, NY, USA: ACM, 2006, pp. 274–279.
[3] OMG, “Uml profile for marte, beta 1,” Object Management Group, Tech. Rep. ptc/07-08-04, 2007.
[4] A. Koudri, D. Vojtsiek, P. Soulard, C. Moy, J. Champeau, J. Vidal, and J.-c. Le lann, “Using marte in the mopcom soc/sopc methodology,” in *workshop MARTE*, 03 2008.
[5] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, “Ultra-fast downloading of partial bitstreams through ethernet,” in *ARCS*, ser. Lecture Notes in Computer Science, M. Berekovic, C. Müller-Schloer, C. Hochberger, and S. Wong, Eds., vol. 5455. Springer, 2009, pp. 72–83.
[6] J. Dekeyser, P. Boulet, P. Marquet, and S. Meftali, “Model driven engineering for soc co-design,” *IEEE-NEWCAS Conference, 2005. The 3rd International*, pp. 21–25, 2005.
[7] Y. Zhu, Z. Sun, A. Maxiaguine, and W.-F. Wong, “Using uml 2.0 for system level design of real time soc platforms for stream processing,” in *RTCSA ’05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA’05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 154–159.
[8] T. Wang, X.-G. Zhou, B. Zhou, L. Liang, and C.-L. Peng, “A MDA based SoC Modeling Apporach using UML and SystemC,” in *Proceedings of the sixth IEEE International Conference on Computer and Information Technology (CIT’06)*, september 2006, pp. 245–245.
[9] P. Kukkala, J. Riihimaki, M. Hannikainen, T. D. Hamalainen, and K. Kronlof, “Uml 2.0 profile for embedded system design,” in *DATE ’05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 710–715.
[10] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, “Designing a Unified Process for Embedded Systems,” in *Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*. IEEE Computer Science, mars 2007, pp. 77–90.
[11] M. Rieder, R. Steiner, C. Berthouzoz, F. Corthay, and T. Sterren, “Synthesized uml, a practical approach to map uml to vhdl,” in *RISE*, 2005, pp. 203–217.
[12] F. A. M. do Nascimento, M. F. S. Oliveira, and F. R. Wagner, “Modes: Embedded systems design methodology and tools based on mde,” in *Model-Based Methodologies for Pervasive and Embedded Software*, March 2007, pp. 67–76.
[13] S. Bocchio, A. Rosti, E. Riccobene, and P. Scandurra, “UML and MDA for Transactional Level Transactional Level Modeling,” in *UML-SoC*, 2007.
[14] E. Riccobene, A. Rosti, and P. Scandura, “Improving SoC Design Flow by means of MDA and UML Profiles,” in *3rd Workshop in Software Model Engineering (WiSME 2004)*, october 2004.
[15] C.-H. Tseng and P.-A. Hsiung, “UML-Based Design Flow and Partitioning Methodology for Dynamically Reconfigurable Computing Systems.” in *EUC*, 2005, pp. 479–488.
[16] Imran Rafiq Quadri, Samy Meftali, and Jean-Luc Dekeyser, “High level modeling of dynamic reconfigurable FPGAs,” *International Journal of Reconfigurable Computing*, vol. 2009, p. 15, 2009.
[17] F. WEST Team LIFL, Lille, “Graphical array specification for parallel and distributed computing (gaspard-2),” 2005. [Online]. Available: http://www2.lifl.fr/west/gaspard/
[18] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard, “Ip reuse in an mda mpsopc co-design approach,” in *ICM’09: Proceedings of the International Conference on Microeletronics*, 2009.
[19] 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 *DATE*. IEEE, 2009, pp. 226–231.
[20] Telelogic, “Rhapsody UML modeller,” www.telelogic.com/products/rhapsody, from Telelogic.
[21] X. Inc., “Planahead design and analysis tool.” [Online]. Available: http://www.xilinx.com/tools/planahead.htm/