Functional Validation of AADL Models via Model Transformation to SystemC with ATL

Pierre Bomel, Dominique Blouin, Mickael Lanoe, Eric Senn

Functional Validation of AADL Models via Model Transformation to SystemC with ATL

Pierre Bomel, Dominique Blouin, Mickael Lanoe, Eric Senn
Lab-STICC
Université de Bretagne Sud
Lorient, France
+33 (0)2 97 87 45 26
{pierre.bomel, dominique.blouin, mickael.lanoe, eric.senn}@univ-ubs.fr

ABSTRACT

In this paper, we put into action an ATL model transformation in order to automatically generate SystemC models from AADL models. The AADL models represent electronic systems to be embedded into FPGAs. Our contribution allows for an early analytical estimation of energetic needs and a rapid SystemC simulation before implementation. The transformation has been tested to simulate an existing video image processing system embedded into a Xilinx Virtex5 FPGA.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits, **Design Aids**]: Simulation

General Terms

Design, Languages

Keywords

AADL, MDE, Program Synthesis, ATL, SystemC, Simulation, FPGA, Functional Validation.

1. INTRODUCTION

Energy. To be able to use the huge quantity of hardware resources available inside today's FPGAs, new electronic system level (ESL) design methodologies and tools are necessary. Particularly, the ever increasing density of transistors, the complexity (number of gates) of assembled hardware functions and the apparition of new 3D ICs have the consequence that energetic needs are rising, and will drastically continue to do so. This is what the IRTS revealed when it added its "Energy" chapter in its annual report [1]. Therefore, the energy consumption can prevent systems to run for long because of heat dissipation problems or fast battery discharge.

HRMPSoC. Embedded systems are becoming more and more complex. They contain computing processors (microprocessors or IPcores), memory hierarchies (caches, scratchpads, local and external memories ...), communication links (point to point, bus, NoC) and rapid IO devices (Ethernet 1Gbit, real time video, network of sensors …).

ACES-MB'12, September 30 2012, Innsbruck, Austria.

These systems can be dynamically and totally or partially reconfigurable on the fly. They are heterogeneous (a mix of hardware and software functions) and may have "time variable architectures" depending on the ability of the application to react to environment changes. These systems are called HRMPSoC (Heterogeneous and Reconfigurable Multiprocessors Systems on a Chip), have a substantial processing power, are self-adaptative and are more and more numerous in a mobile and distributed environment (so called "ubiquitous").

These systems have three important qualities: huge number of transistors, heterogeneity of implemented functions and time variable architectures. Their co-simulation (co because of heterogeneity and time variability) at high abstraction levels is required and promoted because it is necessary to validate as quickly and as soon as possible the functional correctness of several candidate architectures. These architectures are built from a set of reused or synthesized on demand components. In such context, Trabelsi et al. [2] illustrate the fact that functional validation and early estimation of energetic needs by simulation are key factors in the choice of the best architecture. Moreover, it is methodologically efficient to tie both concerns inside a common specification environment to write once and then share several times the same system models.

It is proposed to federate analytical energy estimations with functional validation of electronic systems into an up-to-date and unique modeling environment based on the Eclipse IDE (Integrated Development Environment) and the SAE (Society of Automotive Engineers) Architecture Analysis and Design Language (AADL) [3]. AADL is an emerging standard architecture description language for real-time, fault-tolerant, scalable and embedded multiprocessor systems. It is component-centric and allows specifying both software and hardware parts of systems. A SystemC model is built by automatically assembling components previously grouped in a library in compliance with the architecture specified with AADL. Thus, having a unique AADL model of an FPGA based system helps designers to check two important constraints: 1) that the energetic needs do not exceed a given value, and 2) that the assembled system is functional.

This paper presents our work related to automatic generation of SystemC models from AADL models. Our automatic generation takes advantage of model transformation, which is expressed with the ATL language [4]. In section 2 we present the state of the art in the domain of automatic generation of models from AADL specifications. In section 3 we present our contributions: a

methodology, a semantic mapping between meta-models elements of AADL and SystemC languages and finally a set of ATL transformations. We validate our contributions in section 4 with a video processing system model. We conclude in section 5.

2. RELATED WORK

AADL enables the development and predictable integration of highly evolvable systems as well as analysis of existing systems. It supports early and repeated analyses of system architectures with respect to performance-critical properties through an extendable notation, a tool framework, and precisely defined semantics. In this section we inventory related work about analysis and/or generation of executable models, with or without the use of MDE techniques, from such AADL models. Most of this work concerns the verification of functional and non-functional system properties or the validation of systems by co-simulation in order to extract temporal estimations dynamically without the need for ISS (Instruction Set Simulators) and RTL level models like in complex and long simulations.

Ocarina [5]: Ocarina is a software tool which allows putting into action an evolutionary prototyping methodology based on AADL. Worst case execution time and dead-lock freedom are some of the non functional properties it can check. It also generates ADA or C executables on top of the high integrity POLYORB-HI middleware, in turn targeting ERC32 and LEON2 processors.

Cheddar [6]: Cheddar is an open source tool developed in ADA. With the help of simulations, it computes various performances criteria (schedulability analysis, time constraints, resources allocation, etc.). It accepts as input AADL models thanks to its embedded Ocarina API. Given the difficulties to apply schedulability theory, the authors have recently decided to exploit an MDE methodology to automatically generate, with the help of the Platypus tool, some decision support tools that will determine the relevant feasibility tests for a given architecture to evaluate. Platypus is a meta-model environment relying on the STEP standard (ISO 10303, EXPRESS language).

ACSR [7] and VERSA: The University of Pennsylvania, in collaboration with the Freemont Company, has developed a code generator that translates an AADL model into an ACSR model (Algebra of real-time process). This ACSR model can be analyzed with a tool in order to conduct schedulability analysis.

OSATE [8]: OSATE is a set of Eclipse plugins for the modeling of embedded electronic systems in AADL. It is based on EMF and contains a complete AADL meta-model. OSATE, as an extension of Eclipse, is itself an environment for integrating other tools that operate on AADL. Version 1.5, used for our work incorporates many analysis tools, but no real tools for code generation for executable models.

TASTE [9]: The TASTE toolset is the result of work of the ASSERT (IST 004033, 2004-2007) European project. It was developed by ESA (European Space Agency) with a set of partners in the aerospace field. It aims to define a development process of distributed real-time systems and is based on a tool chain which includes Cheddar and Ocarina. TASTE can build a system from heterogeneous software (MathLab, Ada, C, C++ ...). These codes are either generated automatically by using external tools or manually written. The overall system consistency is ensured by the use of two modeling languages: system modeling with AADL, and messages/data modeling between heterogeneous modules with ASN.1. Code generators are used during the modeling phases to produce software for a given target. TASTE does not generate a mixed executable model for co-simulating hardware-software. Neither does it currently include hardware features, although it seems to be part of future extensions.

Gaspard2 [10]. Gaspard2 is a modeling environment for real-time systems dedicated to intensive and regular data processing. These processes can be represented using a formalism derived from ARRAY-OL whose semantics has been adopted in the UML MARTE profile. It can generate a SystemC 2.0 TML-level simulation model. This model is based on the notion of virtual processor and allows representation of both software and hardware features. Finally, it incorporates the estimated consumption in the SystemC simulation model. However it does not accept AADL models as input and does not offer an analytical model to estimate the power consumption.

AADS, SCOPE [11]. AADS is a tool written in Java for the hardware/software co-simulation environment named SCOPE. It converts an AADL model into a SCOPE model. The SCOPE model is compatible with the Ravenscar computation model. SCOPE is a co-simulation environment written in SystemC, which provides time information on the various system tasks. To do this, no instruction sets simulator is used but time is estimated by executing an annotated native code. It specifically targets MicroC and POSIX OS operating systems and the LEON2 processor.

Apart from AADS, none of the work cited above does target both SystemC code generation and AADL modeling. One of the two languages is always missing. Finally, AADS does not use the MDE methodology to convert an AADL model into a SystemC model. Our contribution is to implement a model transformation in a standardized modeling environment (OSATE) targeting another standardized and highly flexible simulation environment (SystemC, IEEE 1666-2005).

3. CONTRIBUTIONS

In this section we present our design methodology, the set of AADL/SystemC semantic mappings and the ATL model transformations supporting the automated generation process.

3.1 Methodology

The methodology that we propose belongs to the category of "fast and evolutionary prototyping" [12]. It is based on a combination of modeling techniques, code generation and evaluation. It is shown in Figure 1 and is divided into six phases:

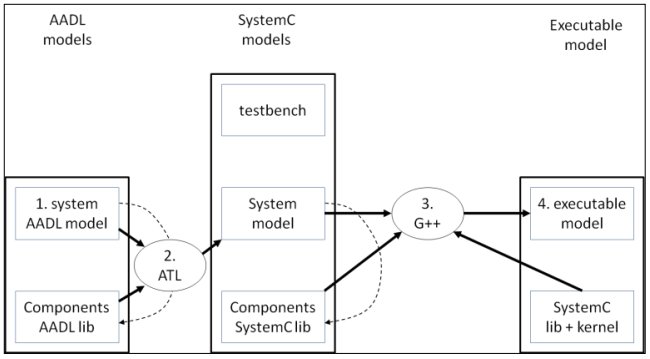

Figure 1. Model/Generate/Simulate methodology flow.

1. Use a library of components to model a system with AADL/OSATE. This library is enriched by
   - IP designers that provide AADL and SystemC models,

- and sub-systems previously modeled, generated and validated.

2. Automatically generate the complete system model in SystemC by means of a chain of ATL transformations. A simplified meta-model for SystemC has been developed including only the necessary concepts needed for C++ code generation from SystemC models.

3. Integrate the generated SystemC model in the system architect's test program. To do this, simply compile the code generated from the SystemC models of the assembled components and the test program, then link all with the SystemC simulation kernel.

4. Simulate the complete system with the resulting executable. The architect judges the validity of the system in light of the results based on provided inputs and expected outputs.

5. If the system is considered functional, the designer can estimate the energy consumption. But, he may as well start with the energy estimation and then check the functionality second. Energy estimation is performed using analysis models whose input often depend on both software and hardware parameters. Besides functional validation, SystemC simulation can also be used to obtain estimates for some of these input parameters.

6. When the system is functional and "energy correct", we can then move on to the detailed design phase or repeat this method to evaluate a different architecture, or the same architecture with another components library. The amount of effort needed for the detailed design phase depends on the available component libraries. If RTL components already exist, they can be reused. Otherwise, they must be developed, which may require significant efforts.

The two dashed arrows in Figure 1 indicate that the obtained AADL and SystemC models can be respectively added to the AADL models libraries and SystemC components library. This methodology allows the building of libraries of increasingly complex components.

The components are initially designed to represent a computable artifact of the behavior of functions. They do not necessarily represent their final implementation. As such, they can represent both hardware or software functions. Anyway, there is nothing that prevents the existence of several SystemC models of the same function. Therefore, they could represent the same function with different implementation types or different abstraction levels and, as long as their interface with the system remains the same, they can coexist in the libraries.

3.2 AADL / SystemC Mapping

AADL permits the modeling of an electronic system in terms of software and hardware components 1) which communicate with each other and 2) with the placement of interconnected software components over the hardware execution platform. The hardware is itself made out of a set of connected hardware components

In the scope of our methodology, the objective is the rapid functional validation of a components assembly, each component having a functional representation in SystemC. The AADL subset we have chosen for this methodology allows the description of data types, interface components, system architectures, shared data, and communications between components and the external interface of the complete system. The link between AADL and SystemC entities is defined thanks to annotations added in the AADL model. Finally, the model transformation must consider the incompatibilities between the rules for naming identifiers. Unambiguous AADL to SystemC conversion rules are needed.

Data types. All types of data processed by components have matched AADL and SystemC models. Let *CppX* be the name of a C or C++ data type, and *AadlY* the name of the corresponding AADL data type. Then the AADL data type *AadlY* has the form shown in Figure 2:

```
data AadlY
  properties  Type_Source_Name => "CppX";
end AadlY;
```

Figure 2. AADL model of a data type.

The AADL model is reduced to the creation of an AADL component of type *data* with the name *AadlY*. The value of the property *Type_Source_Name* is the annotation that indicates the semantic mapping between *CppX* and *AadlY*.

Components. Our AADL components are black boxes for which only the interface is known. They are represented by AADL *threads*. Their interface consists of communication *ports* and *accesses* to shared data.

As shown in Figure 3, the AADL model contains a description of a *thread* and its *implementation*. Inside its *features* section, the *thread* contains a list of ports of type *event data port* when some typed data transit and of type *event port* when it comes to digital only signals. It also contains a list of shared variables that it must have access to. This is expressed via a *requires data access* clause. The mapping with the SystemC module *CppThread*, which represents the true functionality of the *thread*, is declared with an annotation: we use the value of the property *Source_Text* in the *implementation* of the *thread*. Note here the implicit identity between the AADL ports and SystemC ports of both models. Finally, the notion of shared data is also implicitly synonymous to a C++ global variable that is shared by the codes of the SC_METHOD or SC_THREAD SystemC processes declared in the SC_MODULE.

```
thread AadlThread
  features
    id : in  event data port AadlY;
    od : out event data port AadlY;
    i  : in  event port;
    o  : out event port;
    d    : requires data access AadlY;
end AadlThread;

thread implementation AadlThread.impl
  properties  Source_Text => "CppThread";
end AadlThread.impl;
```

Figure 3. AADL model of a functional component.

Architecture, Shared Data and Communications. To represent the functional architecture of the system, we use an AADL component of type *process* and its associated *implementation*. We declare in the *subcomponents* clause of the *implementation* as many threads subcomponents as we need as well as all the shared data subcomponents that *threads* need to read/write from. Finally we connect the *ports*. Figure 4 illustrates the architecture of such a *process* inside which N *threads* of type *AadlThread* are chained together and the ends of the chain are connected to the *ports* of the

process. It also creates the shared data *d*, of type *AadlY*, and indicates that all *threads* have access to it.

System Interface. The complete top level system is modeled using an AADL *system* component type. It has the same type of interface than the assembled components. The *implementation* of the system declares an instance of the *process* modeled earlier and connects its ports to those of the top level system (Figure 5).

The identity of the interface of AADL components of type *process* and *system* allows for repeatedly enriching the libraries from the AADL modeling process. During the generation of SystemC modules, the same top level system name is created and becomes a reusable and valid SC_MODULE. This name will be available for future annotations via the *Source_Text* property. Thus, *Aadlsyst* is a module that can be added to the SystemC components library and can be reused for future AADL models.

```
process SystemArch
  features
    id : in  event data port AadlY;
    od : out event data port AadlY;
    i  : in  event port;
    o  : out event port;
end SystemArch;

process implementation SystemArch.impl
  subcomponents
    t1  : thread AadlThread.impl;
    …
    tN  : thread AadlThread.impl;
    d   : data AadlY;
  connections
    c1a : event data port id   -> t1.id;
    c1b : event port      i    -> t1.i;
    d1  : data access     d    -> t1.d;
    …
    cNa : event data port t(N-1).od -> tN.id;
    cNb : event port      t(N-1).o  -> tN.i;
    dN  : data access     d         -> tN.d;
    cNc : event data port tN.od     -> od ;
    cNd : event port      tN.o      -> o ;
end SystemArch.impl;
```

Figure 4. AADL model of the architecture of the system.

Refinement and Implementation. The AADL concepts of refinement (*refines*) and *implementation* are both naturally represented in the generated SystemC models by the C++ mechanism of inheritance.

```
system AadlSyst
  features
    id : in  event data port AadlY;
    od : out event data port AadlY;
    i  : in  event port;
    o  : out event port;
end AadlSyst;

system implementation AadlSyst.impl
  subcomponents
    arch : process SystemArch.impl;
  connections
    c1 : event data port id      -> arch.id;
    c2 : event data port arch.od -> od;
    c3 : event data port i       -> arch.i;
    c4 : event data port arch.o  -> o;
end AadlSyst.impl;
```

Figure 5. AADL model of the top level interface.

Transformation Rules for Identifiers. SystemC is a language sensitive to uppercase and lowercase while AADL is not. So, `ABCD` and `abcd` are identical in AADL, but not in C or C++. We need to agree on rules for processing AADL identifiers into new identifiers that:

- are legal in C++,
- are not identical to C, C++ or SystemC reserved keywords and macros,
- and are never duplicated.

For this, we followed the AADL SAE's recommendations about the C language [13] and have extended them to the case of SystemC and C++. They are listed here.

- The AADL namespace exists. It contains the names of all executable objects that are equivalent to AADL concepts. These names are located in a SystemC runtime library that contains all types and all classes required for the generation of C++ and SystemC models.
- To every AADL package corresponds a C++ namespace. As an example, Figure 6 shows an AADL package named *AadlPack* in which all components, data types and systems mentioned in this article are defined.
- All AADL identifiers are converted to lowercase and a mechanism for automatic prefixing with "PREFIX_" avoids duplications or collisions with keywords of C, C++ or SystemC. In addition, all characters "." are replaced by "_DOT_", and all sequences "::" are replaced by "_PATH_". Figure 7 shows all possible translation cases.

```
package AadlPack
  data AadlY …
  end AadlY;
  thread AadlThread …
  end AadlThread;
  …
  system AadlSyst …
  end AadlSyst;
  system implementation AadlSyst.impl …
  end AadlSyst.impl;
end AadlPack;
```

Figure 6. AADL package.

```
Idf             -> idf
IDF             -> idf
break           -> PREFIX_break
a.b             -> a_DOT_b
c_DOT_d         -> c_DOT_d
c.d             -> PREFIX_c_DOT_d
a::b            -> a_PATH_b
c_PATH_d        -> c_PATH_d
c::d            -> PREFIX_c_PATH_d
```

Figure 7. Identifier conversion examples.

3.3 ATL Model Transformations

A chain of five model transformations in ATL has been developed to generate the SystemC model. In any case, at least two transformations were needed for first transforming the AADL model into a SystemC model, and then the SystemC model into C++ code. Breaking the transformation into smaller pieces allowed reducing the complexity of the global transformation.

These transformations are based on a source AADL meta-model and a target meta-model named scMM, which is the C++ subset that represents our minimum needs to generate SystemC models. It is smaller and easier to manage than a full set of C++ and SystemC meta-models syntactically complete. Because we do not target all the C++ and SystemC specificities like compilers do, we

do not require a complete meta-model. Moreover the genericity of scMM allows us to retarget to any other object-oriented language. Figure 8 shows the scMM meta-model. The C++ concepts are namespaces (*Namespace*), classes (*ClassList*, *Class*, *ClassSection* and *ClassMember*), identifiers and builders of connections (*ConnectionId*, *ConstructorConnectionInit*, and *Binding*) and finally the identification of the system model (TopLevel).

The five transformations are (Figure 9):

- ***a2s.atl*** is the essential exogenous transformation that converts our AADL subset into its scMM equivalent.
- ***updateRefs.atl*** and ***updateRef2.atl*** are two endogenous scMM model transformations updating internal references that could not be computed in the initial processing by a2s.atl.
- ***orderClasses.atl*** is the endogenous transformation whose role is to sort all classes and types in an order consistent with a compilation process.
- ***sc2txt.atl*** is the transformation that converts the scMM model, with all its internal references properly updated and rearranged into a compilable ASCII text. It supports the syntax of the C++ object-oriented target language.

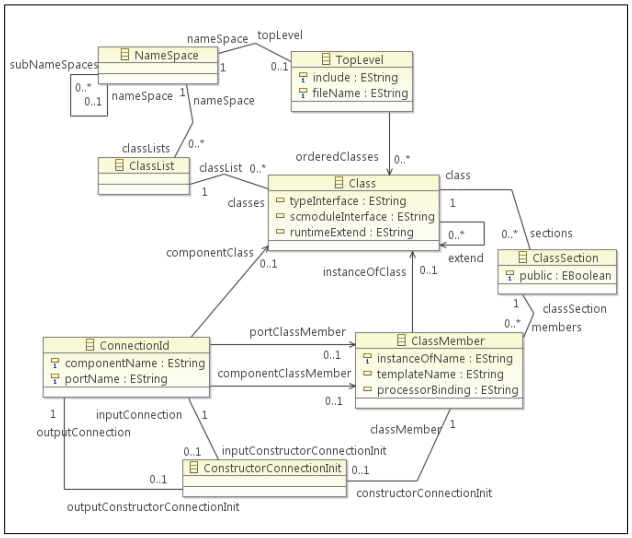

Figure 8. scMM meta-model.

4. RESULTS

The presented results have been tested in the following technical context: Eclipse 3.6, ATL 3.1.1, AADL/OSATE 1.5, SystemC 2.2.0 and Eclipse C Development Tools (CDT) 7.0.2. Our transformation chain has been integrated in the Eclipse IDE as a plugin whose code was partially generated by the ATL development toolkit. Users can select the AADL files to be transformed, and a directory of a predefined CDT project into which the generated C++ files will be put, properly configured for SystemC for quick simulation of the system.

We have modeled an existing image processing system with AADL that can process a 25 frames/s VGA video image stream. It is embedded into a Xilinx Virtex5 FPGA. Image capture and display are performed by hardware blocks respectively interfaced with a camcorder and an LCD screen. The image real-time processing is performed by a program executed by a synthesizable MicroBlaze processor. This system can be easily customized and serves many research and project activities. It has been developed thanks to the MOPCOM project.

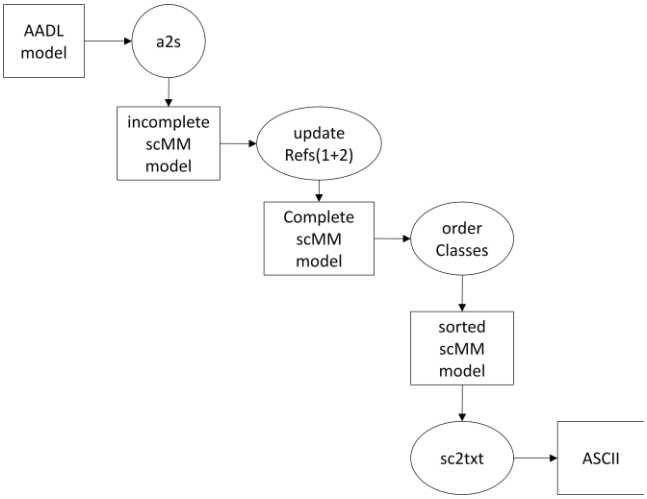

Figure 9. ATL model transformation chain.

For didactic purposes (black and white paper print) we have chosen to reverse the three color components (RGB) of the received images. We have transmitted the image of Lena as a very well known test input so that readers feel familiar with the presented results.

Figure 10 and Figure 11 show the graphical and AADL architecture of the system. It consists of four components whose names are meaningful: capture, processing, display and global synchronization. The synchronization block performs the permutation of the images accesses indices and schedules the image processing at a given frame rate. Capture and display blocks operate at the pixel clock. A shared memory stores a buffer of three images inside which the blocks can make reads and writes through a shared bus. In the AADL model, one can see the instantiation of the four components *Synchro0*, *Capture0*, *Display0* and *Processing0*, the *imageArray* image buffer, and the connections needed to connect the *ports* and provide access to *imageArray* to all *threads*.

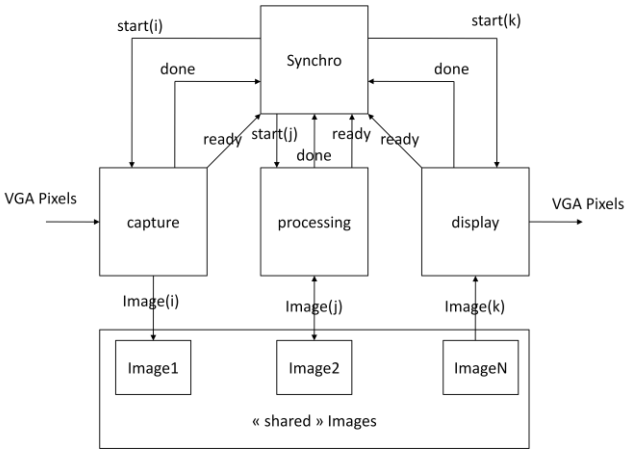

Figure 10. Video processing system architecture.

The simulation of this architecture proves that the system is functional. The resulting images are depicted in Figure 12. While the real system is a real-time one running at a rate of 25 frames / sec, the SystemC model is simulated at a rate of only one image every four to five seconds. So we have a ratio of about 100

between the simulation speed and the real time processing rate expressed in images per seconds.

```
process SoftwareArchitecture
  features
    pixel_in: in event data port VGAPixelType;
    pixel_out: out event data port VGAPixelType;
end SoftwareArchitecture;

process implementation SoftwareArchitecture.impl
  subcomponents
    Synchro0   : thread Synchro.impl;
    Capture0   : thread Capture.impl;
    Processing0: thread Processing.impl;
    Display0   : thread Display.impl;
    imageArray : data   ImageArrayType;
  connections
    pixels_in       : event data port pixel_in       -> Capture0.pixel;
    pixels_out      : event data port Display0.pixel -> pixel_out;

    capture_ready   : event port Capture0.ready    -> Synchro0.capture_ready;
    processing_ready: event port Processing0.ready -> Synchro0.processing_ready;
    display_ready   : event port Display0.ready    -> Synchro0.display_ready;

    capture_start   : event data port Synchro0.capture_start    -> Capture0.start;
    processing_start: event data port Synchro0.processing_start -> Processing0.start;
    display_start   : event data port Synchro0.display_start    -> Display0.start;

    capture_done    : event port Capture0.done    -> Synchro0.capture_done;
    processing_done : event port Processing0.done -> Synchro0.processing_done;
    display_done    : event port Display0.done    -> Synchro0.display_done;

    -- images accesses
    Capture0_image   : data access imageArray -> Capture0.images;
    Processing0_image: data access imageArray -> Processing0.images;
    Display0_image   : data access imageArray -> Display0.images;
end SoftwareArchitecture.impl;
```

Figure 11. AADL model of the system internal architecture.

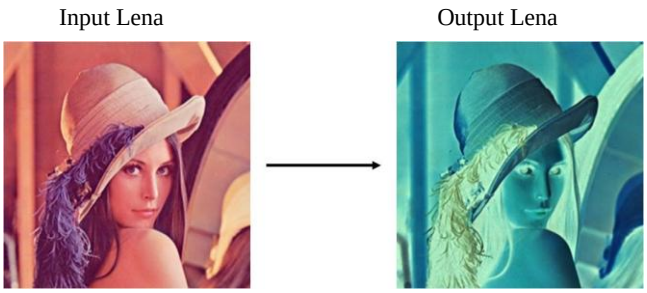

Figure 12. Simulated processed images.

5. CONCLUSION AND PERSPECTIVES

In this article we presented our work about the transformation of AADL models into SystemC for electronic systems embedded into FPGAs. Our contributions, which have been validated by the modeling of a real time image processing system, the code generation and the SystemC simulation (consistent with the expected behavior), show that it is possible and efficient to combine in the same Eclipse meta-modeling environment the analytical estimation of power consumption and the functional validation by simulation. By reusing the same models, the two methodologies reduce the modeling efforts imposed to the system architect. Finally, this rapid generation and simulation design process allows considering a broader exploration of the architectural design space.

During this work, we have identified that the use of incomplete AADL specifications (keyword *refines*) enables a generic modeling and a late binding mechanism during the modeling process. This mechanism seems very close to the C++ *template* concept. We intend to study it and integrate it in the ATL transformations. With this modeling feature, it will be possible to model generic architectures and refine them only when needed. Hence, functional components AND generic architectural components will be both available in our AADL and SystemC libraries.

6. ACKNOWLEDGMENTS

This work is part of the Open-PEOPLE project (26/12/2008-25/12/2011) and is currently funded by the French Research Agency (ANR). It is labeled by the “Images et Réseaux” ("Media and Networks”) Brittany pole. The MOPCOM project has been supported by the ANR (contract 2006 TLOG 022 01), the “Images et Réseaux” pole and the Bretagne and Pays de la Loire regions.

7. REFERENCES

[1] "International Technology Roadmap for Semiconductors, 2010 update", www.itrs.net

[2] C. Trabelsi, R. B. Atitallah, S. Meftali, J.-L. Dekeyser and A. Jemai, "Model-Driven Approach for Hybrid Power Estimation in Embedded Systems Design", EURASIP Journal on Embedded Systems, vol. 2011, id. 569031, Hindawi Publishing, 2011.

[3] SAE. "Architecture Analysis & Design Language" (AADL). AS5506A.

[4] F. Jouault and I Kurtev. "Transforming models with ATL". Satellite Events at the MoDELS 2005 Conference, 2005, pp. 128-138. http://wiki.eclipse.org/M2M/ATL

[5] J. Hugues, B. Zalila, L. Pautet, and F. Kordon. "From the prototype to the final embedded system using the Ocarina AADL tool suite". ACM Transactions on Embedded Computing Systems, TECS 2008, vol. 7, n° 4, article 42, july 2008.

[6] M. Kerboeuf, A. Plantec, F. Singhoff, A. Schach and P. Dissaux. "Comparison of six ways to extend the scope of Cheddar to AADL v2 with Osate". 5th international workshop on AADL and UML. Oxford, UK, March 2010, pp. 367-372

[7] O. Sokolsky, I. Lee and D. Clark. "Schedulability Analysis of AADL models". Procs of the 20th Intl. Parallel and Distributed Processing Symposium, IPDPS 2006, vol. 2006, april 2006, article 1639421.

[8] OSATE. Carnegie Mellon Software Engineering Institute (SEI), "Open Source AADL Tool Environment" (OSATE), http://www.aadl.info/aadl/currentsite/tool/osate-down.html

[9] TASTE. M. Perrotin, E. Conquet, P. Dissaux, T. Tsiodras and J. Hugues. "The TASTE Toolset : turning human designed heterogeneous systems into computer built homogeneous software". Procs. of the Intl. Conf. Embedded Real Time Software Systems (ERTSS), may 2010, Toulouse, France.

[10] R. Ben Atitallah R., E. Piel, S. Niar, P. Marquet and J.-L. Dekeyser J-L. "A Fast MPSoC Virtual Prototyping for Intensive Signal Processing Applications". Microprocessors and Microsystems Embedded Hardware Design Journal (MICPRO) 2011.

[11] Roberto Varona, Eugenio Villar and A-I. Rodríguez. "Ravenscar Computational Model compliant AADL Simulation on Leon2". Procs. of the International Symposium on Information System and Software Engineering, ISSE 2011, March 2011, Prague, Czech Republic.

[12] F. Kordon and Luqi. "An Introduction to Rapid System Prototyping". IEEE Transactions on Software Engineering, 70(3), pp. 817-821, 2002

[13] SAE, "Language Compliance and application program Interface". The AADL specification, 2005, annex volume 1, annex D.