AUTOMATIC SYNTHESIS OF EMBEDDED SW FROM UML/MARTE MODELS SUPPORTING MEMORY SPACE SEPARATION

…

ABSTRACT

The proposed approach presents a solution for automatically synthesizing the SW code of complex embedded systems from a model driven system specification. The solution is oriented to enable easy exploration and design of different allocation of SW components in heterogeneous platform, minimizing designer effort. The system is initially described following the UML/MARTE standard. Applying this standard, the system is modeled, describing its components, interfaces and communication links, the system memory spaces, the resource allocations and the HW architecture. From that information, a SW infrastructure containing the communication infrastructure is generated ad-hoc for the system depending on the HW architecture and the resource allocations evaluated. As a result, the infrastructure synthesized is *more* specific and simple than previous approaches using solutions such as CORBA or RMI. The consequent communication overhead reduction can result in an important advantage for system performance optimization.

Index Terms— System on Chip One, Software Synthesis, Design Space Exploration

1. INTRODUCTION

The evolution of fabrication technologies has enabled the development of powerful System on Chips containing multiple heterogeneous processors of different types, including CPUs, DSPs or GPUs. As a result, these systems can support large and complex functionalities. In order to handle this complexity design flows are evolving to start working at higher levels of abstraction. Designing at higher levels of abstraction is an effective way deal with large system complexities, selecting optimal configurations and verifying system constraints early in the design process. To do so, two main issues have to be solved. First, it is required to provide methodologies where designers can easily describe the system functionality, considering all the interactions among its functional components. Then, solutions capable of optimizing the implementation of this functional description are needed.

Model driven design methodologies are being commonly adopted to handle the design of large functionalities. Latest design methodologies start from high-level UML models combined with algorithmic codes (e.g. C, C++, Matlab, etc.) of the different system components [1]. In these models, the user defines the system functionality using a platform-independent model (PIM). Then, given a platform definition model (PDM), the PIM is translated to one or more platform-specific models (PSMs), where resource allocations are specified.

In order to achieve an optimal solution for the final system, the most promising platform-specific model has to be selected before the implementation process starts. Design space exploration (DSE) solutions have been proposed to perform this selection process. However, there is still much work required to develop solutions capable of minimizing the effort required to provide the accurate estimation metrics the DSE tools require to evaluate the quality of the different possibilities. To accurately classify a solution, design space exploration demands not only modeling and simulation techniques at the system level, but also a link to initial implementations. Lower level tools such as ISSs or virtualization tools are required to estimate design parameters like power, performance and cost. Furthermore, the translation to low level implementations is required for a rapid prototype generation.

However, the connection between high level modeling languages, such as UML, and the initial implementations required for performance evaluation currently implies large synthesis processes; processes that cannot be performed manually if designers want to enhance the productivity of the design cycle.

In order to enable the evaluation of various design implementation options, automatic synthesis of the system starting from the UML models is required. The proposed approach performs the synthesis of the different possible implementations to be explored combining the information provided in the PIM, PDM and PSM models. System components, interfaces and functional communication links are described in the PIM. Physical communication links are described in the PDM model. Architectural mappings among PIM components and PDM resources lead to PSM models.

In order to enable automatic synthesis of different solutions, the components are associated in the PIM model to different memory spaces. That way, it is possible to automatically allocate the functional codes to different resources since global variables and shared memory areas can only be used

by components in the same memory spaces. To do so, each entire memory space is allocated in a single HW resource.
From these models, the proposed approach generates the SW infrastructure required to interconnect the different memory spaces in the different platform resources using basic communication libraries developed for each physical communication channel in the HW platform.
The use of the information described in the UML model, enables the automatic generation of ad-hoc communications infrastructures supporting interconnection of the different system component. The synthesis of ad-hoc communication infrastructures produce more specific and simple results than previous approaches using solutions such as CORBA or Remote Method Invocation (RMI) [11], which reliy in more complex solutions, capable of being reused in a wide range of use cases. The use of simpler infrastructures provokes a reduction of communication overloads, which can give benefits when optimizing embedded system performance. Additionally, it avoids the effort of manually generating and filling the skeletons required to apply these generic communication infrastructures.
In order to present this approach, the paper is divided as follows. First, the state of the art is described. Second, the complete flow is presented. In section 4, the UML/MARTE methodology is shown. In section 5, the synthesis process is described. Then, an example is described in section 6. Finally, results, conclusions and future work are presented.

2. STATE OF THE ART

Automatic synthesis of code from high level models has obtained an important interest in last decade. For example, several works focused on synthesis for embedded SoCs design from SystemC approaches have appeared. In [11] a generic framework for HW/SW communication of functional tasks with shared resources, called Shared Objects is presented. Communication is implemented using a method-based interface realizing a RMI protocol. In order to analyze timing requirements of the HW/SW blocks separation and the bindings established among these HW/SW blocks, the authors propose a transparent communication mechanism and synthesis support for communication across the HW/SW boundary. In [12], a method for systematic embedded software generation is presented. There, the SW code (processes and process communication, including HW/SW interfaces) is systematically generated, from SystemC threads.
However, other non specific high-level modeling solutions, such as UML, have also been applied in that context. The application scope of UML [3] has evolved from object-oriented software systems modeling to cover different design domains. In this context, research to apply UML to the design of embedded systems has gained increasing interest, [1] [2], both in the application of the models in the design flows and in the evolution of the UML language itself [4].
Most of the efforts spent on the integration of UML within embedded design processes, have focused on synthesis. Several researches on synthesis based on UML models are characterized by the creation of state machine models or variations of them [13]. In [5], a formal design for reconfigurable, modular digital controller logic synthesis is presented. By means of UML state machines concurrent super-states are modeled, enabling the direct, automatic mapping on structured array of cells in FPGAs.
Nevertheless, not only state machine modes have been used for synthesis. In [6], a set of transformation rules for synthesis of code from UML activity diagrams are presented. UML Sequence diagrams are used to define control flow patterns, and then, they are transformed in Activity diagrams according to a different set of transformations rules.
Other relevant research area focuses on the development of HW/SW communications within UML-based methodologies. In [7] a semi-automatic solution for generation of HW/SW infrastructure from UML models is presented. This solution implements high-level programming interface (software drivers and hardware adapters) using Remote Method Invocation (RMI) semantics as the framework to unify the communication interfaces for all HW and SW components. The automatic generation is dealt with by means of a template-based mechanism.
In [8], a method is proposed for synthesizing interfaces for heterogeneous IP integration from UML models. The framework supports both interface protocol customization and glue logic generation, thereby maximizing IP integration. Additionally, the framework enables the generation of the communication links among the system blocks from UML profiles used to model the system level communication interfaces.
However, UML, as a completely generic language, usually lacks of all the semantics required to adequately model all the characteristics of embedded systems. In order to confront the challenge to cover the complete design flow of real-time embedded systems, the MARTE profile was created [4]. Taking MARTE-based models as input, several synthesis approaches have also been proposed. Gaspard2 [10] is a design environment for data-intensive applications which enables MARTE description of both, the application and the hardware platform, including MPSoC and regular structures. Through model transformations, Gaspard2 is able to generate an executable TLM SystemC platform at the timed programmers view (PVT) level.
In [9] the complete design flow to move from high level MARTE models to code generation, for implementation of dynamically reconfigurable SoCs is presented. In this paper, generic control semantics for the specification of adaptative and dynamic reconfigurable SoCs is presented. In [14] a design flow based on high level languages (SysMl, MARTE, SystemC…) enables the generation of the deterministic multi-threaded code for parallel implementations.

Nevertheless, all the previous solutions are oriented to generation of previously fixed models, leaving architectural decisions to rely on designers experience. However, with the improvement of evaluation tools such as virtualization (Qemu [15], OVP [16]) and DSE solutions [17], approaches oriented to support configurability, and especially different resource allocations are required. As a result, this paper focused on that area.

3. PROPOSED FLOW

The goal of the proposed flow is to enable selecting the most adequate allocation for the system under development with minimal design effort. Thus, it is required to provide a way to describe the system under design, and then, a solution capable of generating the inputs required by the simulation tool selected by the designer to estimate the performance metrics of the different alternatives. ISSs and Virtualization tools such as QEMU are usually selected for that task. In both cases, the inputs required are mainly the executable binary files that should run on the processing resources of the target platform. Additionally, rapid prototyping solutions can be also applied to evaluate the different design possibilities. In that case, similar binary files are also required. Finally, the resulting files for the selected allocation can be directly used in the final design or can be refined by the designer. Thus, the main challenge of this paper is to solve the automatic generation of these binary files though a synthesis process.

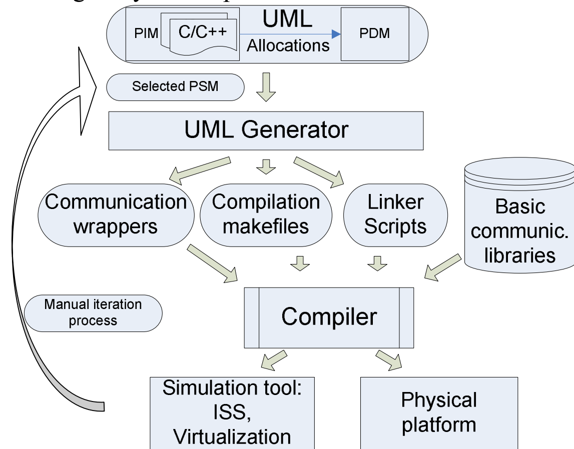

Fig. 1 *Proposed Synthesis Flow.*

The proposed design flow (figure 1) starts from the UML/MARTE model of the system. This model is provided by the designer. The model is composed of three main elements. The first one is the platform independent model (PIM) which describes the functional components, (their interfaces and the functional code) and the interconnections among them. Secondly, a platform description model (PDM) describing the HW platform composed of the available processing elements and their interconnection. And finally, the UML/MARTE model contains the architectural mappings to be evaluated, which are specified in the platform specific model (PSM) [8]. Additionally, the user must provide the functional C codes for all the system components of the PIM.

From this information, an infrastructure developed in Eclipse generates all the elements required to create the binary files required for simulation or physical execution. The elements generated can be grouped in three sets. First, the infrastructure generates the wrappers that communicate the interfaces of the components in different memory spaces using the resources of the platform. Secondly, makefiles are generated in order to enable automatic execution of the compilation processes. Finally, linker scripts are generated when needed, from the information of the PDM.

These elements are used by the compiler together with the functional C code provided by the user and a communication library already developed as part of the proposed flow. This library contains the basic solutions for communicating components depending on their allocations: different processes in the same OS, processes in different nodes communicated by TCP/IP connections, etc.

4. UML/MARTE MODELING

The system under design is specified by an UML/MARTE model before starting the flow. The graphical orientation of UML helps designers to handle large systems in an easy way. However, the UML/MARTE model has to contain all the relevant, essential information of the system, in order to enable performing the synthesis process. Thus, it is required to define a UML/MARTE methodology combining the benefits of a visual language with large amounts of information.

To solve that point, the information contained in a UML/MARTE model is separated in specific concerns, depending on their application area. Each concern is captured in a model view, which is represented using the UML diagrams that most fit the concern. Additionally, the views of the system model are grouped forming three different viewpoints: the Platform Independent Model, PIM), the Platform Description Model, (PDM), and Platform Specific Model (PSM). The PIM describes the system functionality (e.g. application, functional code, interfaces). The PDM describes the different HW and SW resources that form part of the system platform. Finally, the PSM describes the system architecture and the allocation of the application components into the platform resources.

PDM and PSM models can be solved using a single view for each one, since the information supported is not too wide: a view describing the HW components of the platform and their interconnections for the PDM and a view where memory spaces are mapped to HW resources for the PSM.

However, the description of the functionality requires much more detail, making the PIM to rely on the use of four views:

Functional view, Concurrency view, Communication view and Memory-Allocation view. As a result the designer obtains a complete system model that can be easily handled to support the system design.
First, the internals of the functional components of the systems are described using an UML package that is specified by the stereotype <<FunctionalView>>. *FunctionalView* includes both the specification of the functionality and the interfaces provided and required by each application components. Each application component has associated a set of C code files that define the component functionality. These files are modelled as UML artefacts using the UML standard stereotype <<file>>.
In a second step, internal concurrency of the system application components is modelled using the stereotype <<ConcurrencyView>>. The application components are modelled by the MARTE stereotype <<RtUnit>> included in the MARTE subprofile High-Level application modeling (HLAM). Each *RtUnit* component has their own execution thread, providing/requiring services to/from others application components by means of provided and required interfaces. The association among files and RtUnits is defined by means of a UML abstraction, specified by the MARTE stereotype <<allocated>>, included in the MARTE subprofile Allocation Modeling (Alloc). An example of view describing these relationships can be shown in Figure 2.

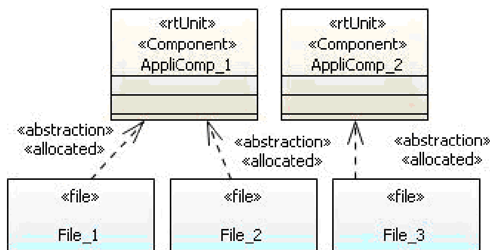

Figure 2. <<RtUnit>> components and their associated <<file>>

Additionally, the *ConcurrencyView* includes the application structure, defined by instances of the application components and the way there are interconnected. The application components are interconnected by means of UML connectors that represent communicating channels. Communications are established through UML ports, where the provided/required interfaces of each application component are defined.
The UML connectors are specified by means of the UML components defined in another model view, the <<CommunicationView>>, which is also part of the PIM. The *CommunicationView* includes the elements that define the semantics of the channels used to interconnect the application components. The MARTE stereotype used to specify these communicating components is the <<CommunicationMedia>>, included in the MARTE subprofile Generic Resource Modeling (GRM). The modeling of the set of specific communication semantics that a *CommunicationMedia* can capture is out of the scope of this paper.
Finally, the allocation of the application components into memory spaces is dealt with in a system view identified by the stereotype <<MemoryAllocView>>. The *MemoryAllocView* package contains the components that identify the different memory spaces that are used for the allocation process of the application components. These memory spaces are modeled by the MARTE stereotype <<MemoryPartition>>, which is included in the MARTE subprofile Software Resource Modeling (SRM).
After modelling the *MemoryPartition* components, the application components have to be allocated into these memory partitions. The mapping of the application components into memory partitions is dealt with in a UML composite structure diagram included in the *MemoryAllocView* package. In this diagram, the application instances defined in the *ConcurrencyView* are mapped into instances of *MemoryPartions* components. The application component instances are mapped to memory partition instances by means of UML abstractions specified by the MARTE stereotype <<allocate>> (Figure 3).

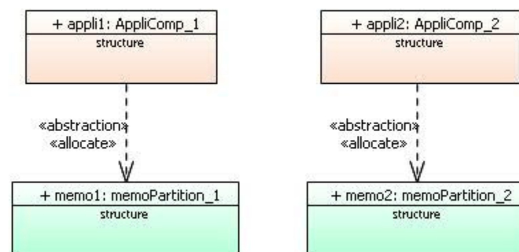

Figure 3. Mapping SW components to HW resources

5. AUTOMATIC BINARY GENERATION

From the UML model and original C code that implement the application functionality, the generator produce a set of files that includes C wrappers that enables the communication among the application components and the compilation scripts. The interface wrappers use the facilities provided by a communication library to implement the final communication mechanisms.

5.1. Interface wrappers

The code generator takes the UML model to extract the necessary information for application components and its communications.
This information must be reflected on the model and contained in the corresponding system views:

- the *ConcurrencyView* where the application components are defined, and the way they are interconnected; the required/provided interfaces and the channels

- the *CommunicationView* where the communicating components are defined
- the PDM view, where the physical communication mechanisms are displayed
- the PSM, where the resource allocations are described

All this information is used by the code generator to write the communication wrapper codes depending on the allocation of each component involved in each communication, details about the arguments that have to be transferred on each communication (data type, size, and direction: in, out or inout), and the physical resources the HW platform provides to communicate the processors.

5.2. Communication libraries

A library for communication between the application components was implemented. This library is based on the client-server paradigm.

On the client side when an application requires a service from other application, the client application generates a new thread for the request. Firstly, this thread generates, for each parameter of the service, a *parameter structure* with the information about the parameter: an unique identifier in the call, the size of the data parameter in bytes, the type (e.g. return, in, inout), a flag indicating if the parameter is pointer or not, and the pointer indicating where the data is in the process local memory. Then, the thread generates a *request structure* with the information of the request with the type (blocking or not blocking), the identifier of the function required, the number of parameters, and finally links into this *request structure* all *parameter structure* created before. The *request structure* is used by the interface communication to store data in the channel and to modify local data when the request is completed. Finally, if the request is non-blocking, the thread finishes, doing the opposite in the blocking case. Once the response is available, the thread modifies locally the original data for the new one and finishes.

On the other side, the server has an active part, which is in charge of constantly listening to each incoming communication channels waiting for the requests. When one request through a communication channel is received, the server registers the data and generates a new thread to attend it, and continues listening to the communication channel. So then, once the petition is completed, the server stores the new data in the communication channel.

5.3. Execution flow structure

As a result of the previous elements, several execution flows can be found in the resulting code. In addition to all execution flows required by the functional components to execute their functionality, each server application has one thread attending each channel which it is connected. Moreover, to attend each request a new thread will be generated, finishing when request is completed.

On the other hand, the client application has its own execution flow and generates a new thread when it needs require a service from other different application component. The request could be blocking or not, so it is necessary to allow that the application flow can wait the response or not and continue.

6.MULTI-PROCESS EXAMPLE

An application example has been developed to check the abilities of the proposed approach for exploration activities. The application consists on four application components. Two components acts like servers providing specific functionality through provided interfaces. In addition, the other two components acts like clients requiring, at some time, the functionalities that are provided by the servers; both servers are connected with both clients. Each client obtains a set of two matrixes of points from grey images and accesses the servers to manipulate them. More precisely, the clients use the first server to get the inverse image of the first matrix and then access to the other server to add this inverse image with the other image.

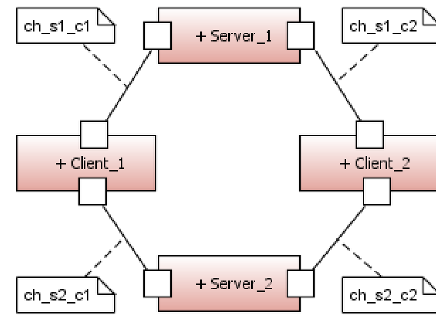

Figure 4. Architecture of the proposed example

Figure 4 shows the UML model that captures the architecture of the application with the four application components, the two server applications and the two client applications (*server_1, server_2* and *client_1*, *client_2*, respectively). The application components are interconnected by means of UML connectors that represent communicating channels (*ch_s1_c1, ch_s1_c2, ch_s2_c1* and *ch_s2_c2*). The communication is established through UML ports, where the provided/required interfaces of each application component are defined.

In the code synthesized from the UML model, a main program file is generated by each application. The main files contain the necessary code infrastructure for the implementation of the communication among the application components. Specifically for the server case, the files add an active element that is listening to the channel of communication. Additionally, for each system interface

included in the *FuntionalView* a file is generated in order to enable the calls to the functions provided by these interfaces. Finally, once the code files are generated, a makefile is generated to compile and link the whole application components with the communication interface to generate the target executable files.
The proposed infrastructure has been applied to explore two different implementations of the communications among clients and services: using fifos from the operating system or using TCP/IP sockets. The execution of the application components has been run on two different HW platforms. The first one was a common laptop (Intel Core 2 Duo @2.00GHz) and the second one was on a Panda Board (OMAP4430 Cortex-A9 @1.0GHz). The execution worked with 1920x1080 matrixes of integer values. From these simulations, the results of table 1 have been extracted.

App.	TIME	Board		Laptop	
		Fifo	Socket	Fifo	Socket
Client 1	REAL	2.127	2.835	0.338	0.829
	USER	0.750	0.797	0.124	0.128
	SYS	0.508	0.570	0.100	0.136
Client 2	REAL	2.201	2.990	0.356	0.732
	USER	0.852	0.781	0.136	0.132
	SYS	0.445	0.852	0.080	0.092

Table 1. Execution times of the proposed example

The proposed approach has enable performing the comparison without manual porting effort. As a result, it can be stated that fifo communications are faster than sockets, which can lead to an optimal implementation.

7. CONCLUSIONS AND FUTURE WORK

The proposed approach presents a solution for automatically synthesizing the SW code of complex embedded systems from a UML/MARTE models. The automatic synthesis process enables easy exploration of different allocation of SW components, since simulators such as ISSs, virtualization tools and rapid prototyping solutions can be performed with minimal designer effort.
The system is initially described following the UML/MARTE standard. The resulting model contains all information from functionality, HW platform and allocation required to perform the automatic synthesis. To do so while maintaining the enough simplicity in the visual diagrams, the information is displayed in several views.
From this model, a generator synthesize the communication wrappers completely ad-hoc for the application, reducing the overhead obtained with more generic solutions.
The approach enables easy exploration by focusing the system model on the definition of memory spaces. By identifying the memory spaces of the components, their interfaces and allocation, it is possible to generate binary files for different allocations from the same inputs.
Additionally, the generated wrappers can implement different communications using basic communication facilities. Communication facilities to connect memory spaces in the same OS and memory spaces in Oss connected through TCP-IP protocol have been implemented. Additional communication types, such as CPU-DSP communication will be solved in future works. Moreover, deep analysis on the effect of the proposed communication infrastructures in system performance, and their comparison with other infrastructures such as CORBA or RMI are still pending.

8. REFERENCES

[1] Y. Vanderperren, W. Mueller, and W. Dehaene, “UML for electronic systems design: a comprehensive overview,” Design Automation for Embedded Systems, vol. 12, no. 4, 2008
[2] L. Lavagno, G. Martin, B. Selic. “UML for real: design of embedded real-time systems”, ISBN 1-4020-7501-4.
[3] “UML profile for system on chip (SoC) specification”. 2006
[4] OMG: "UML Profile for MARTE", www.omgmarte.org, 2009.
[5] M. Adamski. Design of reconfigurable logic controllers from hierarchical UML state machines. 2009 4th IEEE Conference on Industrial Electronics and Applications, ICIEA 2009
[6] S. Kang, H. Kim, J. Baik, H. Choi, C. Keum.Transformation Rules for Synthesis of UML Activity Diagram from Scenario-Based Specification. IEEE Proceedings of 34th Annual Computer Software and Applications Conference (COMPSAC), 2010.
[7] J. Barba, F. Rincón, F. Moya, D. Villa, F.J. Villanueva, J.C. López. “Automatic HW/SW Interface Generation for Seamless Integration from Object-Oriented Models”, International Conference on Embedded Systems & Applications. ESA, 2009.
[8] S. Zhenxin, W. Weng-Fai. A UML-based approach for heterogeneous IP integration. Proceedings of ASP-DAC, 2009.
[9] I. R. Quadri, H. Yu, A. Gamatié, E. Rutten, S. Meftali, J-L. Dekeyser. Targeting reconfigurable FPGA based SoCs using the UML MARTE profile: From high abstraction levels to code generation. International Journal of Embedded Systems, 2010.
[10] É. Piel, R. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J.-L. Dekeyser, P. Boulet: "Gaspard2: from MARTE to SystemC Simulation", proc. of the DATE'08 workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile, 2008.
[11] P.A. Hartmann, K. Gruttner, P. Ittershagen, A. Rettberg.”A framework for generic HW/SW communication using remote method invocation”. ESLSyn, 2011.
[12] F. Herrera, H. Posadas, P. Sánchez, and E. Villar, “Systematic Embedded Software Generation from SystemC”, DATE, 2003.
[13] D. Harel, H. Kugler, and A. Pnueli, “Synthesis revisited: Generating statechart models from scenario-based requirements,” Formal Methods in Software and System Modeling, 2005.
[14] V. Papailiopoulou, et al: “From design-time concurrency to effective implementation parallelism: The multi-clock reactive case”. Electronic System Level Synthesis Conference, 2011
[15] QEMU, *www.qemu.org*
[16] Open Virtual Platforms, http://www.ovpworld.org/
[17] Multicube Explorer, http://home.dei.polimi.it/zaccaria/multicube_explorer_v1/Home.html