An object-oriented approach for modeling interfaces in a codesign environment

Mouloud KOUDIL, Karima BENATCHBA

Institut National de formation en Informatique, BP 68M, 16270,Oued Smar, Algérie.

Email : k_benatchba@ini.dz

RESUME. Cet article introduit une approche orientée-objet pour ma modélisation d'interfaces, dans un environnement de codesign. Le but de cette approche est d'offrir à l'utilisateur la possibilité de concevoir son application sans avoir à se préoccuper des détails de bas niveau de l'opération de synthèse de la communication. En effet, le concepteur a simplement à choisir les composants de l'architecture cible, les protocoles de communication et éventuellement des systèmes d'exploitation dans une bibliothèque puis l'opération de synthèse se fait automatiquement par l'insertion du code (partie logicielle) et des circuits (partie matérielle) nécessaires à la communication entre les composants processeurs.

ABSTRACT. This paper describes an object oriented approach for interface modeling in a codesign environment. The aim of this work is to offer the user the opportunity to design his application without being worried with low level details of the synthesis operation. In fact, the designer has just to pick target architecture components, communication protocols and operating systems if necessary, and then synthesis operation automatically takes place by inserting the code (software part) and the circuitry (hardware part) needed by the communication between processor components.

MOTS-CLÉS : Codesign, modélisation d'interfaces, Partitionnement, communication.

KEYWORDS: Codesign, interface modeling, partitioning, communication.

1. Introduction

Codesign is a technique that allows to design mixed systems (containing both hardware and software parts) in a unique process that allows reducing integration problem and, consequently, time-to-market delays. The different parts of such systems are, generally, independent from a processing point of view. However, it frequently happens that they have to exchange information through mechanisms that are dedicated to communication. These mechanisms include both hardware (buses, controllers…) and software (drivers). In order to communicate, these different parts must use a common protocol that defines the rules used to exchange information through a communication channel.

Different communication protocols are used in the reported work dealing with codesign. Some authors recommend using "standard" (or at least well known) protocols, while others develop their own ones.

In order to design such systems, designers are often confronted with the difficulty of finding efficient design tools for building complex systems. The tasks that are the most prone to errors and the most consuming in design time are the processes consisting of interfacing hardware and software components in order to insure communication between them. This is due to the fact that the communication mechanisms can be complex in managing the hardware part (that can require to manage interrupt systems as well as different controllers), and the software part (managing synchronization, device drivers…).

This paper presents an automatic modeling approach for interface parts, in applications built using codesign. This approach takes advantage of the object-oriented paradigm and offers the user the opportunity to describe his application using classical C++ language. It dispenses users to master the low-level details of the interface components. He just has to choose a target architecture, a communication protocol in a library, using an interactive tool, and then launches the automatic interface communication synthesis process. The different steps of the synthesis operation are then totally transparent. The presented approach neither aims at presenting a formal model, nor using it to do formal proof. It rather aims at uniformizing all the entities handled by the designer (software and hardware components), as well as the processes ran on them (specification of the system under design). Section 2 of the paper briefly presents some of the most important works reported in the domain of interface synthesis for codesign. In Section 3, a generic model for the component of the library is introduced, as well as the way to specialize it for the main component types in a mixed application (containing hardware and software). The two following sections present the successive steps in our approach for communication interface automatic synthesis.

2. Previous works

It is possible to classify the different works reported that deal with interface synthesis in the following categories: some of them implement standard protocols [1, 2, 3] such as communication through shared memories, communication by message exchange [4] or by "rendez-vous" [5-7]. Some other works propose the implementation of non-standard communication protocols [4, 8-10] or approaches for message scheduling [15]. The use of real-time operating systems is proposed by [11], while [12] proposes the use of interrupt systems. The works of [13] deal with the specification of communication protocols by users and several works propose the use of library components [5, 7, 9, 14-16]. The last class includes works that develop specification languages for bus interface synthesis [8, 14, 18, 19].

It appears that more and more research teams are interested in communication interface synthesis for codesign. However, no one seems to have fully tackled all the problems related to this activity, and only few implementation details are given [17]. Number of these efforts either does not consider the global properties of communication links, or perform the allocation using exclusively non-standard protocols [4]. In the following section, the approach proposed for the synthesis of target architecture and automatic communication mechanism (between the different parts of an a codesign application) is introduced.

3. Presentation of the component generic model

Our approach [20], following the example of several other works, uses a library of generic components that allow deriving all the objects handled by the environment. In this section, the model and the manner to specialize it for application objects (specified in C++) is introduced. The specialization procedure is also presented for architectural and communication ones as well, allowing to synthesize the target architecture and its behavior from a processing and a communication point of view. The library contains reusable components, that can be instantiated to compose a target architecture on which mixed applications can be ran. It is integrated in an interactive environment that includes a tool allowing the composition of mixed architectures and a second tool (allowing the administration of the library). The latest one allows to "tailor" the components that the user wants to include in the library. Starting from a basic generic component, it is the possible to modify each of the three parts of the component (figure 1). The mixed system design tool proposes the use of a component model on which all the components of the environment (corresponding to the unique model described in figure 1) are stacked. To be adapted to all the components, variations are done on the different units of the initial model.

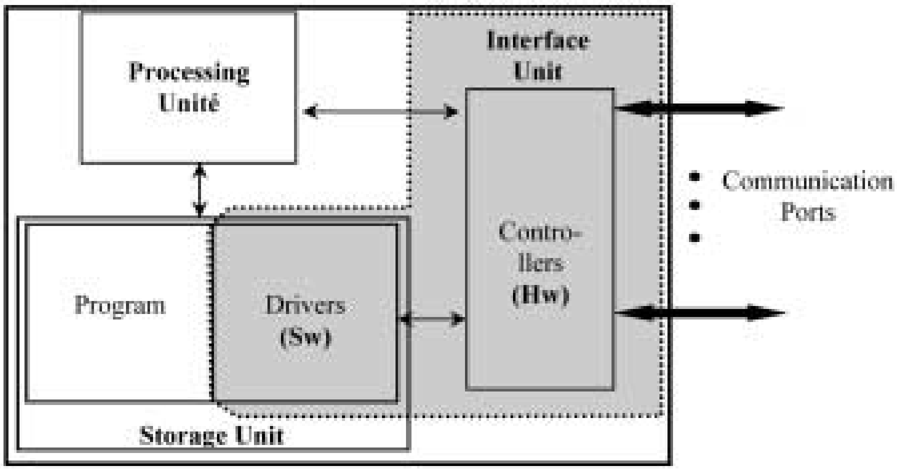

Figure 1. *Unique model representing the different units of a component.*

The components are implemented as C++ classes with the three parts corresponding to the units of the components as described in figure 2 (processing, storage and interface units).

```
class TComponent
{ private:
     1-    //declaration of the storage unit and its parameters;
     2-    //description of the storage unit (control part);
     3-    //declaration and description of private methods of the processing unit;
     4-    // declaration and descriptionof the ports and their parameters;
           //declaration of the control part of the interface unit (communication protocol...)
public:
     6- // declaration and description of the access points to the component
                              (communication primitive : send and receive)
};
```

Figure 2. *Description of a generic component.*

The parameters characterizing the different units contain, for example: information concerning the size of the storage unit, the communication ports, the processing (type of processor, clock frequency, number of interrupt levels…). These information are stored in the library and linked to the concerned component. In the majority of codesign approaches, it exists three types of components: the objects composing the application, the processing components (hardware processors and general purpose processors), and communication components. The remainder of this section presents the different generic components as well as instantiation and adaptation examples of these components.

3.1. Modeling of the software components of the application

The software components of the application are classical objects (in C++). They are considered as a box encapsulating data reachable only through ports that are materialized by the methods. It is thus easy to stack these objects on the generic class, in order to create a software components: the storage unit includes the data of the object; the processing unit represents the processes executed by the different private methods (internal behavior of the object) and the body of public methods. The interface unit represents the entry points to the object (declaration of "calling" or "called" public methods detected in the object during the analysis step of C++ code). The public methods (that constitute the entry points to the object) as well as the methods used by the object to call other objects, are converted into ports and communication primitives, as described in section 4.

3.2. Modeling of processing components

The processing components are used to compose the target architecture on which the user application is executed. These components materialize the different processors. The processing unit can be a general purpose processor or a hardware processor (FPGA, ASIC...). In the following, the specialization step of the different units, used to realize a processing component, is introduced.

3.2.1. Processing units

It implements the processes realized by the processor. It is composed of a control part and parameters that guide the synthesis operation. In the case of a general purpose processor, that aims at executing the software part of the application, the processor is characterized by a data path and an instruction set. Only the characteristics are stored in the model, since the control part (behavioral description of the processor) is useful only during the simulation step. In the case of dedicated hardware processors, the control part is definitely known only after the partitioning step (that consist of splitting the different entities of the application onto the target architecture processors), because these processors are meant to realize the hardware parts of the mixed application (containing both hardware and software). After the synthesis step, the control oar is automatically translated in VHDL. The new description is the provided, during the synthesis step, to a hardware synthesis tool. The parameters of the processing unit include, among others, the processor type, the timing characteristics of the instruction set (number of cycles for execution), its space characteristics, the clock frequency, the size of the different buses, the number of interrupt lines, the type of each line (vectorized, non programmable, maskable or not…).

3.2.2. Storage units

It represents the cells of a memory block, in the case of an instruction-set processor, or a register file in the case of a dedicated processor. It can require the use of a control part that describes the decoding logic and the memory controller. The parameters of the storage unit are, essentially, the memory capacity (number of memory words), the size of a memory word, the mean access time, the internal memory space dedicated to code (for programs) or the external memory space (case of peripherals considered as memory addresses by some instruction-set processors)…

3.2.3. Interface units

An interface can include for essential parts: a bus interface (set of ports with their characteristics), a register file, a description of the behavior of the device and access primitives to the component. The interface unit is parameterized by the number of ports, the size and direction of each port, the type of the protocol used, the transfer speed, the transfer delays, etc. It is automatically synthesized during the synthesis step, according to the results of the partitioning step and the parameters provided by the user through a graphical interface for target architecture parameterization.

The interrupt system is also taken into account. For non-programmable interrupts, all that is necessary is to connect the ports of the interrupt signals of the processing unit (processor) directly to the ports of the interface unit. Concerning the vertorized interrupts (if the modeled processor allows it), the administrator can program them by inserting (in the storage unit) procedures corresponding to the task to execute and storing the starting address of the procedures in the interrupt vector. He also has to establish a connection between each such interrupt port and the communication port allocated to it in the interface unit. In our model, the component uses the interface unit ports to send or receive a data. However, the instruction set of the processor or the wired functions constitute the set of private methods of the processing unit that are non-reachable from outside the component.

3.3. Modeling the communication components

To insure the communication between two components, it is sometimes necessary to have the use of specialized communication components (bus, controllers…). It is possible, thanks to the generic models, to create such components by specializing a generic processor model, in order to compose a communication channel. For example, in the case of Direct Memory Access (DMA), the storage unit contains the component internal registers: registers for data, command, state; specific registers… The processing unit implements the control part of a component (description of the behavior and the dialog with the concurrent processor that competes for the access to the memory). The interface unit, implements the controller communication ports.

3.4. Description of communication protocols

Our approach [20] allows the designer to choose and implement one of the standard protocols contained in the library, or to describe himself his own protocol. These protocols are written in C++ and use "send" and "receive" virtual primitives to communicate through cirtual ports. It is thus easy to integrate them in a component by specializing, during the synthesis step, the available ports on the chosen processing component or communication component. The proposed approach allows as well to describe new protocols by simply modifying the generic component corresponding to the protocol described I C++ in the processing unit (simply by changing the code of the two primitives, as well as the virtual ports used). The following section described the procedure that stacks the C++ objects (of the initial specification of the application under design) onto generic software components of the library in order to get an homogeneous internal modeling of the different components.

4. Application object modeling step

Our interface synthesis approach proceeds by successive steps. In the first step, the user has to choose processing and communication components for the target architecture, and precise their parameters. He then has, during the second step, to choose the communication protocol(s), describing the development of communication operations. The third step is automatic, and consists of associating, to each object of the original application C++ specification, a software object component of the generic model. These objects correspond to the model described in section3. Virtual ports are declared and adjusted according to the needs in communication of the object. The headers of the object methods and the calls to other objects (usual communication tools of C++ objects), are replaced, in the model, by virtual communication primitives ("send" and "receive") that are refined along the synthesis process. The partitioning step determines the objects that must be implemented in software and those that must be realized in hardware. For the interface modeling step, we wanted to use a communication approach that allows starting with the C++ high level specification, and automatically refining it in a transparent way for the user, until low level details are reached. From this point of view, the model of Remote Procedure Calls (RPC) [6, 8, 21] is a mechanism that allows to communicate through communication channels. The connection mechanisms are transparent to the users and the access to the communication components are controlled by a set of primitives called methods or services. As in an object oriented approach, the communication can be done only through the access to a method. The rest of the component is transparent to the user and consist of a set of ports linking the parameters of the method to the channel controller. A component can, thus, activate a method in an other component. The advantage of the RPC approach lies in the encapsulation of the processes and the fact that the user does not have to know the implementation details of the communication components. It is this approach that we decided to implement in our environment. In our approach, the interface unit of each component, implements the communication facilities through send and receive primitives that allow normalizing the communication through the different architectural and communication components of the library. The protocol uses these primitives for external communication in order to link the external ports of the components to the internal ports of the processor (processing and storage units) and describe the behavior of the communication layer. Besides, the software components that model the objects of the initial specification of the application, communicate through public methods that constitute the only access points to the internal data of the objects. To insure the correspondence between the primitives of the interface unit of the architectural component on the one hand, and the methods used by the application C++ object on the other hand, it is necessary to operate transformations in the source code of the application objects. These modifications aim at normalizing the communication points in order to let them directly stack on the virtual primitives and ports of the unit interfaces of the target architecture components. This is made possible by the adoption of the object approach from beginning to end of the codesign process. The object notions allow to isolate and clearly locate the communication operations between the application objects, that can be done only through public method calls. The RPC notions allow keeping the notion of encapsulation, and avoid the user "immersing" in low level implementation details.

4.1. Modeling and inserting virtual ports and primitives

For the insertion of virtual ports, one have to perform an analysis of the objects in the C++ code that allows to build the communication graph between the objects. An instance of software object component of the generic model is created for each object.

Besides, the method calls between the application objects are located and the analysis of calling or called methods allows extracting all the characteristics of the communication such as input and data, or the size of the emitted or received data. Theses characteristics allow determining the needs in ports and virtual primitives of the model object components. The first step of the virtual port insertion process consists of associating a virtual port to each method header. The analysis of the type of parameters provided in the call of the method allows deciding on the type of port to insert: input (Port_In); output (Port_Out) or input/output (Port_InOut).

For a calling method, the calling object provides its parameters and sends an activation signal to the method of the called object. Generally, it waits then for a return parameter. In the same way, for a called object, it receives, at least, an activation signal for its method that is often accompanied with input parameters. It then answers by output parameters. One must take into account the low level dialog lines in order to implement the adopted communication protocol.

The second step consists of inserting "send" and "receive" virtual communication primitives. The approach proceeds by successive refinements. In this way, at each analysis step, the extracted parameters are inserted in the port and primitive descriptions. The tool automatically modifies the primitives to add different parameters (for example: port size, transfer direction…), becoming closer, at each transformation, to the physical level.

4.1.1. Case of called objects

1st case: the called object is in software

– If the method is called only by other software objects, there is no hardware-software communication, and the method does not undergo any change;

– If the method is called only by hardware objects, it is automatically modified without keeping an initial copy of it for software communication;

– If the method is called by both object types (hardware and software):

- Create a generic component copy of the object;

- Copy the method in order to use the over-definition and allow the two type of communication quoted above;

- Eliminate the header parameters from the new copy of the method for the hardware/software communication and insert virtual ports;

- Insert a ***RECEIVE*** (paramètre_i, size_i...) primitive in the method input;

- Insert a ***SEND*** (paramètre j, size_j...) primitive in the method output (before the "***return***") and eliminate the parameters of the return if there exist.

Figure 3 presents an example of method in a called software object.

```
Method1 (Par1, Par2, Par3) // par2 is an input and Par1, Par3 are outputs
{   ...   return (Par1);
}

The method is copied, its copy is replaced by:
Method1()
{   //private declarations Par1, Par2, Par3... ,
    //public declarations of the ports according to the parameters Pari;
    RECEIVE Par2;
    ... /* body of the method;
    SEND Par1, Par3;
    Return();
}
```

Figure 3. A method in a called software object.

2nd case: the called object is in hardware

The object undergoes the same process, but there is no duplication nor over-definition, since the hardware/hardware communication uses the same communication mechanism if the two methods are on different processing components, and the internal communication is transparent if the methods lie on the same component.

4.1.2. Case of calling objects

1st case: the calling object is in Hardware:

The modification is automatic whatever the communication type is (hardware-hardware or hardware-software) and the modifications are the following:

– The method call is replaced by:

SEND (called_object_name, method_name, param_i, size_i...);

RECEIVE (called_object_name, method_name, param_j, size_j...);

– Virtual ports are inserted and an RPC primitive in inserted for the activation of the called method.

2nd case: the called object is in Software:

The modification is performed only if the called object is in a different partition (the communication between software objects of a same processor does not require to transit by a a communication channel). The modification is the same as the preceding case.

5. Conclusion

This paper presented an automatic modeling approach for the interfaces in application designed using the codesign method. This approach takes advantage of the object-oriented paradigm. In fact, there is no need, for the user, to master a new language, nor a complex syntax. He does not have to know much of the interface details either. He just has to specify his application in classical object oriented C++, and the tool automatically replaces all the communication operations. The generic component model presented here, allows to model in a quite homogeneous way all the entities handled by the environment, and to store them in a library of components: ranging from processing components of the target architecture, to the software components of the user application and the communication protocols. One of the advantages of this model is that it allows the encapsulation of the processes and the data inside the model, isolating the different units, preventing from untimely modifications and making easier to locate the different access points to the components. The environment takes in charge the automatic generation of the C++ classes corresponding to the different objects modeling both architecture and application.

The approach presented here is integrated in a codesign environment called CECOOC (Codesign Environment from C++ Object Oriented Cospecifications) [20, 21].

References

1 GUPTA R.K. AND DE MICHELI G., "Hardware/Software cosynthesis for digital systems", *IEEE Design and Test of Computers, Vol. 10, N°3*, Sept. 1993, pp.29-41.

2 COGNIAT G., "Etude et modélisation de support de communication dans les systèmes mixtes logiciel/matériel", *Rapport de DEA, I3S*, Université de Nice, 1994.

3 UPENDER B.P. AND KOOPMAN P.J., "Communication protocols for embedded systems", *Embedded Systems Programming, Vol.7, N°11*, November 1994, pp.46-58.

4 ORTEGA R.B., LAVAGNO L., AND BORRIELLO G., "Models and methods for hardware/software intellectual property interfacing", *In 1998 NATO ASI on System-level Synthesis*, 1998.

5 BENISMAIL T., ABID M., O'BRIEN K. AND JERRAYA A.A., "An approach for hardware-software codesign", *Proc. of the Fifth Int. Workshop on Rapid System Prototyping*,1994, pp. 73-80.

6 JERRAYA A.A. AND O'BRIEN K., "SOLAR: An Intermediate Format for Systéme-Level Modeling and Synthesis", *Co-Design Computer aided HW/SW Engineering*, IEEE Press, 1994.

7 VERCAUTEREN S., LIN B. AND DE MAN H., "Constructing application-specific heterogeneous embedded architectures from custom hardware/software applications", *33rd Design Automation Conference*, 1996.

8 GAJSKI D., VAHID F., NARAYAN S., AND GONG J., *"Specification and Design of Embedded Systems", Prentice-Hall*, 1994.

9 DAVEAU J-M., MARCHIORO G.F., BEN-ISMAIL T., AND JERRAYA A.A., "Protocol selection and interface generation for hardware-software codesign", *IEEE Trans. on Very Large Scale Integration, Vol.5, N°1*, March 1997, pp.136-144.

10 ROWSON J.A. AND SANGIOVANNI-VINCENTELLI A., "Interface-based design", *In Proc. of the Design Automation Conf.*, 1997, pp.178-83.

11 KANDLUR D.D., SHIN K.G., AND FERRARI D., "Real-time communication in multihop networks", *IEEE Trans. on Parallel and Distributed Systems Vol.5, N°*10, 1994, pp.1044-1055.

12 CHIODO M., GIUSTO P., JURECSKA A., LAVAGNO L., HSIEH H.C.AND SANGIOVANNI-VINCENTELLI A., "Synthesis of mixed Hw/Sw implementation for CFSM specifications", *Handouts of the Int. Workshop on Hw/Sw Codesign*, 1993.

13 WENBAN A., O'LEARY J. AND BROWN G., "Codesign of communication protocols", *IEEE computer, No 12*, 1993, pp.46-52.

14 CHOU P., ORTEGA R., AND BORRIELLO G., "Synthesis of the hardware/software interface in microcontroller-based systems", *In Proc. of the International Conference on Computer-Aided Design*, Nov. 1992.

15 ERNST R. AND BENNER T., "Communication, constraints, and user-directives in COSYMA", *Technical Report TM CY-94-2*, Technical University of Braunschardwareeig, June 1994.

16 VAHID F. AND TAURO L., "An object-oriented communication library for hardware-software codesign", *In Proc. of the 5th Int. Workshop on Hardware/Software Codesign*, March 1997.

17 M. Eisenring and J. Teich, "Domain-specific interface gneration from dataflow specifications", *Proc. of Codes/CASHE'98, the 6th Int. Workshop on Hardware/Software Codesign*, Seattle, Washington (U.S.A.), 1998, pp. 43-47.

18 O'NILS M., "Specification, synthesis and validation of hardware/software interfaces", *Technical Doctor Thesis, School of Electrical Engineering*, Royal Institute of Technology, Sweden, 1999.

19 LIN B. AND VERCAUTEREN S., "Synthesis of concurent system interface modules with automatic protocol conversion generation"', *IEEE International Conference on Computer-Aided Design*, November 1994.

20 KOUDIL M., "Une approche orientée objet pour le codesign ", *Thèse de Doctorat d'Etat*, INI,2002.

21 KOUDIL M., BENATCHBA K., TOUATI S.A., KHERFI M.L., "CECOOC: Environnement de Codesign à partir de Co-spécifications C++ Orientées Objet", *4ième Colloque Africain sur la Recherche en Informatique, CARI'98*, Octobre 1998.