Design Automation for Embedded Systems, X, xx–xx (2000)
© 2000 Kluwer Academic Publisher, Manufactured in the Netherlands.

Interlanguage Communication Synthesis for Heterogeneous Specifications

FABIANO HESSEL[1] fabiano.hessel@imag.fr

PASCAL COSTE pascal.coste@imag.fr

PHILIPPE LE MARREC philippe.lemarrec@imag.fr

NACER-EDDINE ZERGAINOH nacer-eddine.zergainoh@imag.fr

GABRIELA NICOLESCU gabriela.nicolescu@imag.fr

TIMA Laboratory, 46 Av. Félix Viallet, 38031, Grenoble Cedex, France

JEAN-MARC DAVEAU[2]

IBM TJ WATSON RESEARCH CENTER. TACONIC PARKWAY & ROUTE 134. PO BOX 218. YORKTOWN HEIGHTS. NY 10592

AHMED JERRAYA ahmed.jerraya@imag.fr

TIMA Laboratory, 46 Av. Félix Viallet, 38031, Grenoble Cedex, France

Received xxxxx, 199X; Revised xxxxx, 199X

Editor: XX. XXXX

Abstract. Nowadays the design of complex systems requires the cooperation of several teams belonging to different cultures and using different languages. It is necessary to dispose of new design and verification methods to handle multilanguage approaches. This paper presents an approach for the interlanguage communication synthesis of heterogeneous specifications. The system is represented by a set of interconnected subsystems specified in different languages with different concepts, different interface types, different communication schemes and some of them can even be IP modules. The subsystems exchange data through abstract communication channels. The objective is to refine the abstract communication channels into an implementation. The result is a set of interconnected processors communicating through signals, buses and dedicated components. An example illustrates the usefulness of this approach for the design of an adaptive speed control system that was described in SDL and Matlab.

Keywords: Multilanguage Specifications, Communication Interface Synthesis, Multilanguage Codesign

[1] On leave from PUCRS/Brazil. Under grant supported by PUCRS-CAPES-COFECUB.
[2] Formerly at.

1. Introduction

The concept of multilanguage specification aims at coordinating different cultures through the unification of the languages, formalisms and notations. Nowadays, the use of more than one language corresponds to a current need in embedded system design. The design of large systems, like the electronic parts of an airplane or a car, may require the participation of several groups belonging to different companies and using different design methods, languages and tools. Besides, multilanguage specification is also driven by needs of modular and evolutive design because of the increasing design complexity. Modularity helps in mastering this complexity, promotes design re-use and more generally encourages concurrent engineering development.

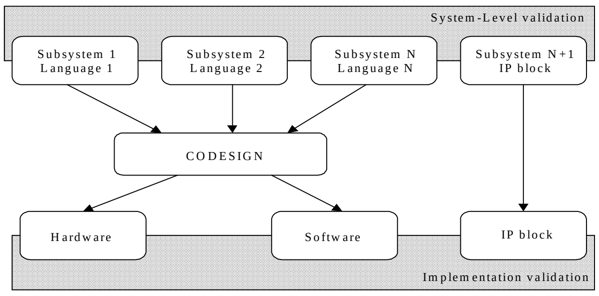

Figure 1. Multilanguage Codesign

Figure 1 shows a generic flow for codesign starting from a multilanguage specification. Each of the subsystems of the initial specification may need to be decomposed into hardware and software parts. The codesign process also has to tackle the refinement of interfaces and communications between subsystems.

As in the case of the heterogeneous specification for system architecture, the problems of interfacing and multilanguage validation need to be solved.

This brings up two distinct problems for design automation. The first problem is the global validation of the system at the system level. At this level, the specification of the system can be composed of several partial specifications (subsystems) that use different languages and the communication is described at the application level. The validation of the overall system may be done using the co-simulation-based approach [21]. The second problem is the synthesis of the interfaces between different subsystems. Since different languages may be based on different concepts for data exchange or the subsystems may be specified at different levels ranging from the implementation level to the application level, the refinement of the interfaces is generally a difficult task [20].

The main contribution of this paper is to present an approach to interlanguage communication synthesis of heterogeneous specifications.

In the following sections, we present our communication approach. The next section introduces the problem of the interlanguage communication synthesis for heterogeneous specifications. Section 3 specifies a multilanguage system. The objectives and previous work are presented in sections 4 and 5. Section 6 describes our communication model. The communication refinement problem is treated in section 7. In section 8, we will present an application example. Finally, we present our conclusions.

2. Definition of the Problem

When a large system has to be designed by separate groups, they may have different cultures and expertise with different modeling styles. The specification of such large designs may lead each group to use a different language which is more suitable for the specification of the subsystems they are designing according to its application domain and to their culture.

Figure 2 shows a heterogeneous system made of three subsystems that are designed by separate groups that may be geographically distributed. The subsystems are specified in different languages such as SDL, VHDL and Matlab, and an abstract communication channel realizes the data exchange. Each subsystem may use different communication interfaces, communication protocols and interconnection topologies depending on the specification language and the system needs.

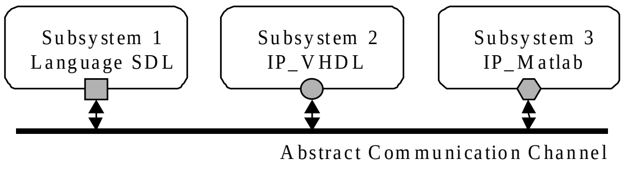

Figure 2. Heterogeneous System-Level Specification

The key issue of multilanguage codesign is the synthesis of communication interfaces between different subsystems. In fact, the inter-module communications are represented by a kind of high-level netlist that specifies the interconnections between different subsystems through abstract communication channels. Since different languages may be based on different concepts for data exchange and different communication interface types, the interpretation of the link between subsystems is generally a difficult task.

The problem of the communication interface synthesis is how to produce an adapted interface for each subsystem according to the subsystem characteristics in order to obtain a set of interconnected modules communicating through buses, signals and, eventually, a control unit. The interlanguage communication synthesis is realized in parallel with the module refinements as presented in Figure 3.

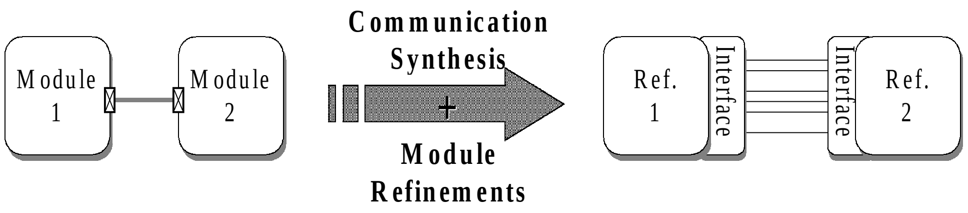

Figure 3. Interlanguage Communication Synthesis and Module Refinements

2.1 Application Domain

The multilanguage codesign methodology is used for the design of heterogeneous system specification. The design of each part of the system may lead the designers to use different specification languages. Of course, each subsystem may belong to different application classes such as control/data or discrete/continuous. Some examples of such systems are interfaces of mobile phones, or more generally end-user interfaces for communication products, set top boxes for digital TV, or control modules in cars, airplanes or buildings.

2.1.1 Component Reuse

The complexity of modern embedded system designs makes designers promote the reuse of both software and hardware IP modules [22]. Reuse is already a well-known concept at the electronic system level or in software development [16]. In order to reduce the time-to-market, it becomes necessary to apply the technique of reuse IP modules to the system-level. The IP modules are already described in a specific language. They have a specific and fixed interface, and a specific communication protocol, which objective is to satisfy the constraint of cost, performance and functionality of the module. Most communication interface synthesis approaches in the literature do not allow such a codesign model.

The communication interface synthesis approach proposed in this work allows integration and reuse of both software, hardware and IP components. This approach takes into account factors such as defined interfaces, their interconnection mechanisms and the communication protocol used to access the IP. Based on these factors, our approach produces the interconnections for the communication between the IP modules and the others parts of the system without getting into the specification details of the IP module. This allows the designer to modify the functionality of an existing component or substitute an existing component by another component without executing the communication interface synthesis anew. Of course, the IP interface and the communication protocol must be the same.

2.1.2 Interconnection of Heterogeneous Subsystems

A multilanguage system is defined by a set of subsystems based on different concepts and interconnected by abstract communication channels. Each subsystem may use different communication schemes, depending on the specification level of the subsystem. The complexity of the interface communication synthesis process depends on the level of the inter-module communication.

When the subsystems are specified at the system level, the communication is specified through high-level communication schemes such as high-level primitives (*send, receive, get, put*). In this case, the communication protocol may be hidden by the communication procedures. The communication protocol will be selected during the communication interface synthesis. At the next level, communication is carried out through read/write operation on I/O registers. At the lowest level, communication is performed using operations on simple wires with a well-known communication protocol.

3. Specification of a Multilanguage System

A multilanguage system is described as a set of communicating subsystems. Communication ports link the different subsystems. There are mainly three communication port types:

1. A wire connection, this corresponds to an HDL port,
2. A logical connection, this corresponds to a high-level access to an abstract channel,
3. A connection through a connector, this corresponds to an abstract view of a realization. A connector is a set of fixed wires accessed via a specific protocol.

Figure 4 shows different types of interconnections between ports:

1. **C1** is a communication channel that represents the abstract connection between two connectors. In this case, the channel is in charge of the protocol adaptation between the two connection points.
2. **C2** is a communication channel that represents the abstract connection between one connector and one logical connection point. In this case, the abstract communication channel offers the communication primitives to establish the communication of the logical connection point. Moreover, the abstract channel realizes the protocol adaptation between the logical connection point and the connector.
3. **C3** is a communication channel that represents the abstract connection between two logical connection points. In this case, the abstract communication channel offers a set of communication primitives that realizes the communication between the logical points. The realization of the channel is flexible and will depend on the communication protocol.

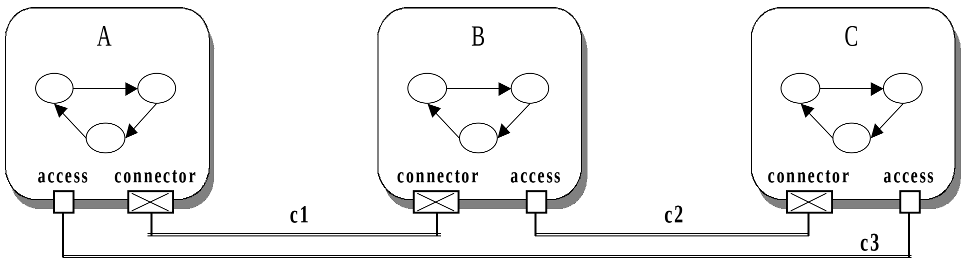

Figure 4. Different types of abstract communication channels

4. Objective

When designing heterogeneous embedded systems, communication interface synthesis is essential to multilanguage codesign as different subsystems inevitably need to communicate. Different communication schemes, communication protocols, interface types and protocol adaptation functions may be needed in multilanguage embedded systems as well as different connection topologies. Communication topologies and protocols greatly influence the overall system performance and may lead to a design impossible to implement if the designer underestimates the communication load. Therefore a large design space has to be explored to find a feasible solution. Our approach allows a wide range of communication protocols to be mapped on a synthesis-oriented approach for multilanguage system specifications. The main objectives of our approach are:

1. To be able to establish the communication synthesis of a multilanguage system,
2. To be able to model system behavior independent of communication. Abstract communication channels are used in a target-independent system specification to specify the inter-module communication,
3. To be able to reuse existing communication models through a library (e.g. buses such as PCI, CAN, etc., existing communication protocols such as FIFO, handshake, etc. or home-made communication protocols).
4. To be able to choose between different communication protocols. The designer may choose different communication protocols to realize an abstract communication channel and different implementations of the same communication protocol, according to the communication constraints.

5. Previous work

Most current approaches in communication synthesis for codesign have focused on interface synthesis for a homogeneous system specification [2,3,4,5]. In [6], Vahid presents an object-oriented communication library for hardware-software codesign. The PIA co-simulation tool [7] allows to specify multiple communication models for each interface of the system and to switch dynamically to another communication model during the simulation run. The CoWare [8] environment allows the designer to specify and simulate communication channels at various abstraction levels. However, these approaches realize the interface synthesis for a unified representation of the system.

Lots of previous works have focused on interface synthesis. In [9], Ecker presents a method for transforming and optimizing protocols. In [10], Narayan addresses the problem of bus interface generation between two different hardware modules of a partial specification. The focus is to optimize the use of a bus by keeping different point-to-point communications on it. As described in [11,12], Lin and Narayan consider the problem of interface synthesis with automatic protocol conversion with one or both sides having a fixed interface. In [13], a new design methodology based on interface design was proposed that treated behavior and communication as orthogonal elements. The goal is to make it easier to reuse communication components. SpecSyn [14] generates buses for communication between two processes using a technique called interface refinement. Analyzing the size of data communicated with rate of data generation, they determine the bandwidth of the generated bus and merge the communication channels onto the bus. Yen and Wolf [15] looked at communication synthesis in the context of derivation from heterogeneous distributed system architectures with an arbitrary topology. Ortega and Borriello [19] addressed the problem of synthesizing the communication for an arbitrary bus topology specified by the system architect. The synthesis processes make a map of high-level functions to the computational components of a particular architecture. An application-specific real-time operating system is generated for each processor in the system. An extension of this approach that takes into account IP modules is presented in [22]. Daveau [18] takes a behavioral description and automatically selects a protocol from a library to implement the communication. But, this approach solved only the problem for homogeneous systems. This paper extends the work of Daveau [18] to handle interlanguage communication.

The main advantages of our approach are:

1. Communication interface synthesis of multilanguage specifications,
2. A complete communication synthesis approach:
   - Network synthesis and protocol selection,
   - Interface synthesis and adaptation modules for different subsystem interfaces,
3. Reuse of existing communication protocols.

The limitations of our approach are:

1. The need of a communication library that must be provided by the designer,
2. The need of a cost function and communication estimator to guide network synthesis and protocol selection.

6. Communication Model

At the system level, a design is represented by a set of communicating subsystems that exchange data through communication channels. These subsystems are specified using different languages and some of them can be IP blocks [16], programmable processors, dedicated devices or software modules (Figure 5). The input into the communication synthesis process is composed of two parts: a specification model that describes the interconnection between subsystems and a library of communication unit model.

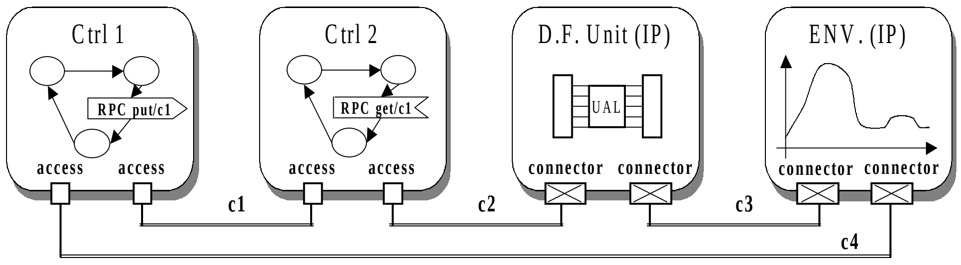

Figure 5. System-Level Netlist

6.1. Specification Model

The input to our communication interface synthesis tool is a global configuration of the system. This is a kind of system-level netlist that specifies the interconnections between different subsystems [20]. The result of communication synthesis is a system composed of a set of modules interconnected by signals. Figure 5 shows a system-level netlist that interconnects four subsystems through abstract communication channels. The inter-module interactions may be specified at different levels from the implementation level where the communication is performed through wires, to the application level where the communications is performed through high-level implementation-independent primitives.

An abstract communication channel is an entity able to execute a communication protocol independent of the final realization. The abstract communication channels offer a set of high-level communication primitives that allows the inter-module communication and, protocol adaptation and control functions that need to respect the communication protocol of each connector. The communication primitives (*send, receive, get, put*) and the connector types (*PCI, CAN, Shared Memory*) are the unique visible parts of an abstract communication channel. The set of communication primitives of an abstract channel is defined in accordance with the communication procedure calls of the interconnected subsystems. The connector types of an abstract channel are defined in accordance with the interconnected IP modules.

There is no predefined set of communication primitives or connector types. Each application may have different sets of communication primitives and different interface connector types. This model allows to hide the implementation details of the communication and separate communication from the rest of the design behavior. The detailed I/O structure and protocols are hidden in a library of communication components [18].

6.2 Communication Unit Model

We define a communication unit as an abstraction of a physical component. Communication units are selected from a library and instantiated during the communication synthesis process [18].

From a conceptual point of view, the communication unit is an object that can execute one or several communication primitives and can link one or several interface connector types with a specific protocol. These communication primitives and interface connectors may share some common resources (bus arbiter, buffering memories, buses) provided by the communication unit. The communication unit is composed of three main parts:

1. The communication primitives used by the logical connection points,
2. The control unit used, if necessary, to adapt the different communication protocols of the IP modules and to implement the control of a communication protocol,
3. The internal connections, that specify the interconnections between the interface ports of the communication unit.

The complexity of the control unit may range from a simple handshake to a complex layered protocol. A communication unit implements a communication channel using a specific protocol and realization. This model enables the user to describe a wide range of communication protocols and most system-level communications, such as message passing or shared memory. Figure 6 shows a communication unit that interconnects an abstract port and a connector, and another communication unit that interconnects two connectors.

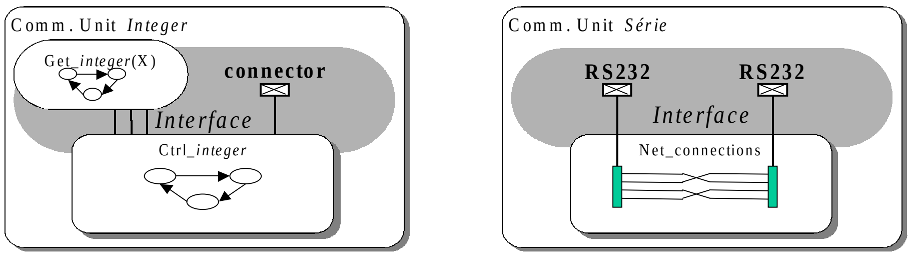

Figure 6. Heterogeneous Communication Units

7. Communication Refinement

In this paper we will use the protocol selection strategy described in [18]. The protocol selection starts with a set of subsystems communicating through communication channels (Figure 5) and a library of communication units, and chooses the appropriate set of communication units from the library in order to provide the communication primitives and interface connectors required by the communicating subsystems.

Communication synthesis selects an implementation for each communication unit from the implementation library and generates the required interfaces for all the subsystems using the communication units (Figure 7). The library may contain several implementations of the same communication unit according to data transfer rates, memory buffering capacity and adaptation functions for different subsystem interfaces and formalisms.

The interfaces of the logical connection are adapted according to the selected implementation and will be interconnected. The interfaces of the IP blocks are connected with the connectors of the communication unit. An approach for interfacing incompatible protocols is presented in [11]. The result of interface mapping is a set of interconnected processors communicating through signals, buses and an additional dedicated component (control unit) that establishes the inter-processor communication. With this approach it is possible to map the communication of a multilanguage specification onto any given protocol, from a simple handshake to a complex protocol. Starting from the system of Figure 5 and the communication units of Figure 6, the result of the interface mapping onto the abstract channels *c2* and *c3* is detailed in Figure 7.

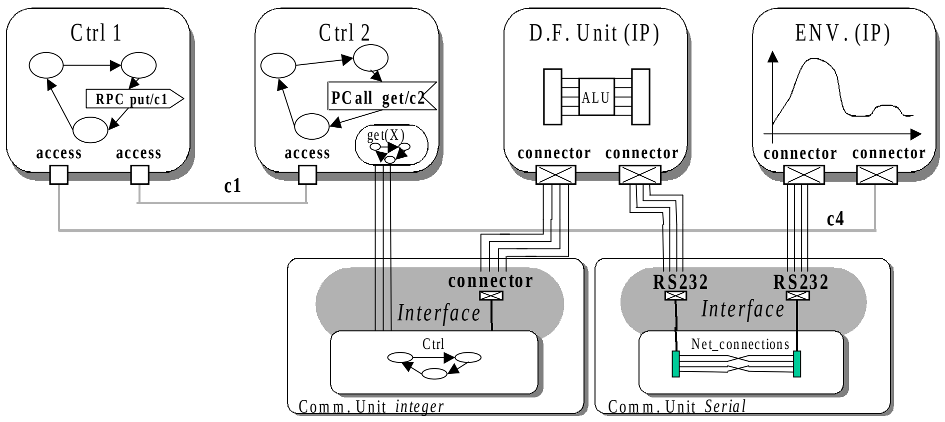

Figure 7. System after interface mapping

8. Results

In this section we show the results of our approach for the communication interface synthesis of multilanguage system-level specification.

The application is a robot-arm controller. The system can be divided into two parts, the motors of the robot arms and the controller. SDL is used to model the controller and Matlab is used to model the physical behavior of the motors. The robot arm controller can adjust the position and speed parameters up to eighteen motors belonging to the robot arm. This computation is intended to avoid discontinuous operation problems. In this example, only two motors are used to simplify the explanation. *Figure 8* shows the methodology used to produce and to validate the multilanguage system-level specification.

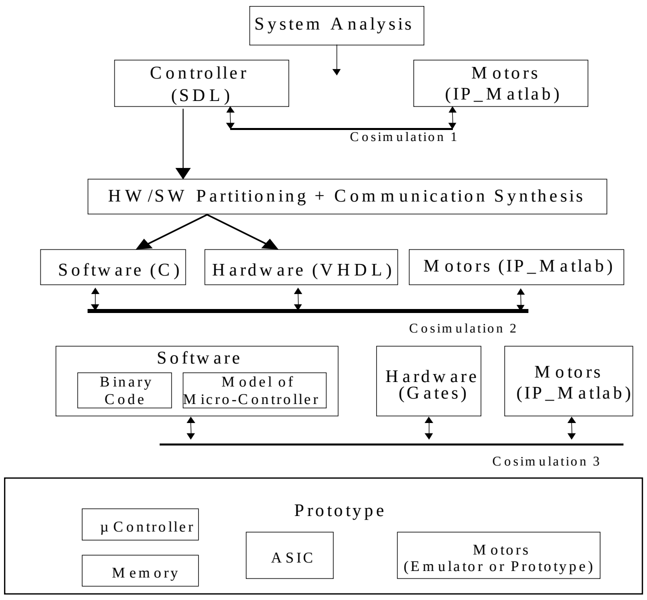

Figure 8. Multilanguage Design Flow

Four signals are exchanged between the Matlab subsystem and the SDL subsystem, two for each motor. The first signal controls the motor and the second one provides its current position. The basic block diagram of the system is shown in *Figure 9*.

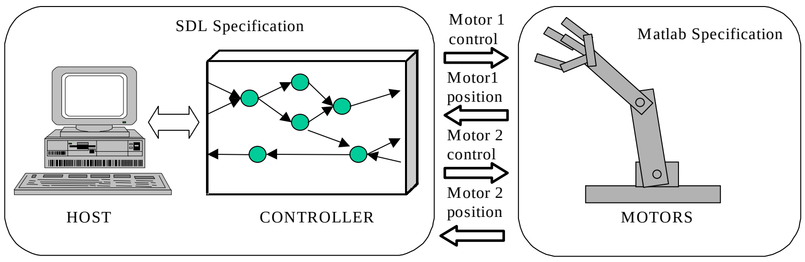

Figure 9. Robot Arm Controller System

The *Host* block computes a trajectory for all motors of the robot arm. The *Controller* block is in charge of giving orders to motors only when it is possible. In this case, a Matlab specification (*IP block*) is used to specify the environment. This model acts as a testbench during all the validation steps.

The graphical representation of the high-level netlist (subsystem interconnections) for the system is given in Figure 10. It is made of two subsystems interconnected through four communication channels (*s1, s2, s3 and s4*). In order to validate the global system specification at this stage of the design flow, an SDL-Matlab cosimulation is performed using MCI [21].

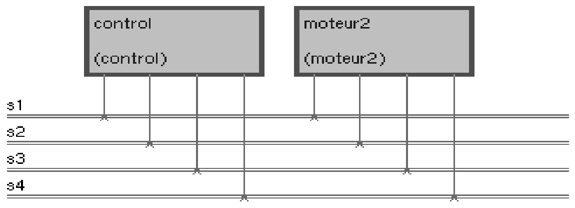

Figure 10. SDL-Matlab specification

The next steps make the refinements in this initial specification. This step translates the SDL specification into SOLAR [17] in order to apply system refinement transformations. The resulting specification is represented in Figure 11. This model is made of seven blocks. Six come from the SDL model (control) and the seventh corresponds to the Matlab subsystem (motors).

The next step consists in the communication synthesis. The last four communication channels will be physically implemented as simple rendez-vous (Communication between internal blocks). Communications between the Matlab environment (*IP block*) and the rest of the system will be implemented through a rendez-vous with the specific transceivers (*protocol adapter module*) able to connect heterogeneous subsystems.

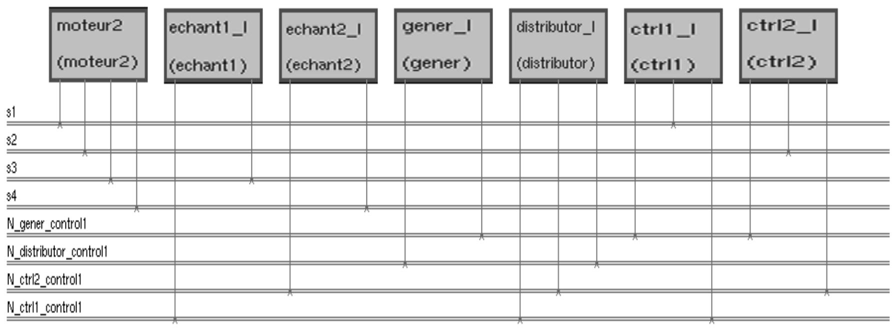

Figure 11. Specification after refinement steps

The result of the communication synthesis is shown in Figure 12. Four extra communication controllers (*Heterogene_real*, *Heterogene_int*) were inserted automatically during the communication synthesis. These controllers implemented the communication protocol and the adapter module.

The next step is to decompose the system into hardware (VHDL-RTL), software (C) and IP blocks. After this step, it becomes possible to make a cosimulation of the system at the implementation level for the validation of the above steps.

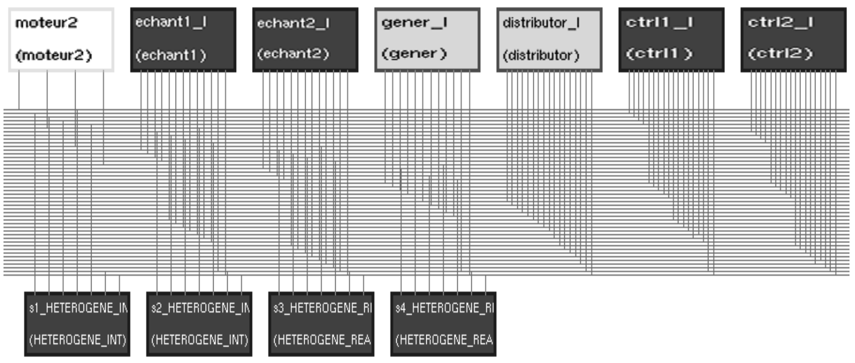

Figure 12. System after communication synthesis

9. Conclusion

This work presented a communication interface synthesis for multilanguage specifications. The interface synthesis task may be performed automatically and allows the designer to map high-level communication primitives described in different formalisms and notations, onto a physical implementation. Since no restrictions are imposed to the communication models, interface connectors and description languages, this process can be applied to a large class of multilanguage codesign applications.

References

1. W. Wolf, "Hardware-software co-design of embedded systems", *Proceedings of the IEEE*, vol. 82, pp. 967-989, July 1994.
2. Jean-Marc Daveau, Tarek Ben Ismail and Ahmed Amine Jerraya, "Synthesis of System-Level Communication by an allocation-Based Approach", in *Proc. of the 8th ISSS*, pp.150-155, 1995.
3. Jan Madsen and Bjarne Hald, "An Approach to Interface Synthesis", in *Proc. of the 8th ISSS*, pp. 16-21, 1995.
4. S.Narayan and Daniel Gajski, "Protocol Generation for Communication Channels", in *Proc. of the 31th DAC*, pp.547-548, 1994.
5. Michael Eisenring and Jurgen Teich, "Domain-specific interface generation from dataflow specifications", in *Proc. of the 6th International Workshop on Hardware/Software Codesign (CODES/CASHE'98)*, pp.43-47, 1998.
6. Frank Vahid and Linus Tauro, "Object-oriented communication library for hardware-software codesign", in *Proc. of the 5th International Workshop on Hardware/Software Codesign (CODES/CASHE'97)*, pp.81-86, 1997.
7. Ken Hines and Gaetano Borriello, "Dynamic communication models in embedded system co-simulation", in Proc. of the 34th DAC, pp.395-402, 1994.

8. D. Verkest, K. Van Rompaey, I. Bolsens and H. De Man, "CoWare – A Design environment for Heterogeneuos Hardware/software Systems", *Design Automation for embedded Systems*, vol. 1, no. 4, pp.357-386, 1996.

9. W. Ecker, M. Glesner and A. Vombach, "Protocol merging: A VHDL Based Method for clock cycle minimising and protocol preserving scheduling of IO operations", in *Proc. of the EDAC with Euro-VHDL*, pp.624-629, 1994.

10. S. Narayan and D. Gajski, "Synthesis of System-Level Bus Interfaces", in *Proc. of the EDAC with Euro-VHDL*, pp.395-399, 1994.

11. S. Narayan and D. Gajski, "Interfacing Incompatible Protocols Using Interface Process Generation", in *Proc. of the IEEE Design Automation Conference*, pp.157-164, 1995.

12. B. Lin and S. Vercauteren, "Synthesis of Concurrent System Interface Modules with Automatic Protocol Conversion Generation", in *Proc. of IEEE International Conference on Computer Aided Design*, pp.395-399, 1994.

13. J. Rowson and A. Sangiovanni-Vincetelli, "Interface-based design", in *Proc. of the Design Automation Conference*, pp.178-183, 1997.

14. D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded systems*. Prentice Hall, 1994.

15. T. Yen and W. Wolf, "Communication Synthesis for Distributed Embedded Systems", in *Proc. of International Conference on Computer Aided Design*, pp.288-294, 1995.

16. R. Ortega, L. Lavagno and G. Borriello, "Models and Methods for Hw/Sw Intellectual Property Interfacing", in *NATO-ANSI Workshop on System Level Synthesis*, 1998.

17. A.A. Jerraya and K. O'Brien, "SOLAR: An intermediate format for system-level design and specification", in *IFIP Inter. Workshop on Hardware/Software codesign*, Grassau, Germany, 1992.

18. J.M. Daveau, G.Marchioro, T. Ben-Ismail and A.A. Jerraya, "Protocol Selection and Interface Generation for Hw-Sw codesign", *IEEE Trans. on VLSI Systems*, vol. 5, no.1, pp.136-144, 1997.

19. R. Ortega and G. Borriello, "Communication Synthesis for Distributed Embedded Systems", in *Proc. of International Conference on Computer Aided Design*, pp.437-444, 1998.

20. A. A. Jerraya, M. Romdhani, C. A. Valderrama, Ph. Le Marrec, F. Hessel, G. Marchioro and J. M. Daveau, "Languages for system-level specification and design", in *Hardware/Software Co-design: Principles and Practice* (J. Staunstrup and W. Wolf eds.), pp. 307-357, Kluwer Academic Publishers, 1997.

21. F.Hessel, P.LeMarrec, C.A.Valderrama, M.Romdhani, A.A.Jerraya, "MCI-Multilanguage Distributed Co-Simulation Tool", Chapter in Distributed and Parallel Embedded Systems, DIPES'98 organized by IFIP WG10.3/WG10.5, edited by F. Rammig, pp. 191-200, Kluwer Academic Publishers, 1999

22. P. Chou, R. Ortega, K. Hines, G. Borriello, "IPChinook: An Integrated Ip-based Design Framework for Distributed Embedded Systems", in *Proc. of the Design Automation Conference*, 1999.