Abstract—In this paper, we present the F4MS (Framework for Mixed Systems) which is a Computer Aided Design for software and hardware design, simulation and aided execution. Based on components which could be software or hardware, as well as all operational tools. This framework is an extension of the TI4CS framework (Tools Integration for Complex Software), which deals only with software components. In order to design a mixed system in this framework we use a general model (execution graph), which characterizes the integration phase in the design methodology. This methodology also covers specification and partitioning phases. Finally, to illustrate this work we propose an example of mixed system for the implementation of the VPN solutions.

Index Terms—F4MS, mixed system, TI4CS, design methodology, execution graph.

I. INTRODUCTION

The main focus for the development of a system remains in the partitioning of tasks. However the major challenges of mixed systems affect their development and use, taking into account the coexistence between software and hardware, as well as the multiple and complex interactions between various components.

The main goal of this work is not only the proposition of a design methodology (flexible) for the specification and the partitioning of software / hardware, but also provides a framework for the implementation of systems incorporating both hardware and software components, as well as the proposition of a general model for design and execution of mixed systems.

This paper is organized as follows. Firstly, section 2 presents the general structure of the complex software applications of the TI4CS framework. Then Section 3 presents the new extension (F4MS framework) for the design and execution of mixed systems. It describes the design methodology and the general model of these systems. Then, Section 4 illustrates the F4MS framework, the design and implementation of a mixed system for VPN. Finally, conclusion is presented in section 5.

II. TI4CS FRAMEWORK

TI4CS (Tools Integration for Complex Software) is a framework which helps to design software using a execution graph [1, 3, 4] in order to model and manage collaboration between several components [5]. This execution graph can model complex software based on scheduling, parallelism graph and interaction graph [1, 2].

a) Scheduling and parallelism graph

The scheduling and parallelism graph [2, 3] (Figure 1) is a directed graph which represents several possible scenarios of development, so that every scenario represents the same model application and describes the order of execution of components. It includes the connectors of scheduling and parallelisms or one them to establish indirect connections between components. The main activities ensured by these connectors are [3]: the Sequence, the parallelism, the exclusive choice and the synchronization.

The scheduling and parallelism graph represent all structures GOP of the form $GOP = (EC, L, δ_1, C_0, F)$ where:
- $EC$: Finite set of components.
- $L = CO \cup CP$ where $CO$: scheduling connectors set, $CP$: parallelism connectors set.
- $δ_1 : EC \times L \rightarrow P(EC)$ where $P(EC)$ is the parties set of $EC$.
- $C_0 \in EC$: Initial component.
- $F \subseteq EC$: Set of final components.

Several formal techniques can be used to model and validate the GOP: algebras of process, LOTOS or automats, but the technique of modeling which seems particularly suitable to model the GOP is the UML diagrams of Activity [6] because it makes it possible to describe in the form of graph, the sequence of activities and the behavior of the system or its components.

Fig. 1 – Activity diagram describes an example of scheduling and parallelism graph

b) Interaction graph

It is a complementary graph [3] to the scheduling and parallelism graph (Figure 2), its role is to ensure the transfer of data between heterogeneous components (interoperaibility). This operation is loaded by another type of connector called interaction connector.

The exchange of the data between the components is made on the interfaces level, which a component (Customer), in an application, needs, during his execution, a result of another component (supplier). It is necessary to connect the output interface of this last with the input interface of the component (Customer). Of course, it would be necessary to verify that any pair of components to be assembled is compatible of various points of view: syntactic and interactional.
The interaction graph reprint all structures GI of the form GI= (EC, CI, δ2) where:
- EC : Finite set of components.
- CI : set of interaction connectors.
- δ2 : EC × L × SEC → P (EC × EEC)
- SEC : Component outputs set.
- EEC: Component inputs set.

Fig. 2 – Interaction graph

III. FRAMEWORK FOR MIXED SYSTEMS

F4MS represents an extension of the TI4CS framework [1] (Tools Integration for Complex Software) which not include only software components but also hardware components. However the existing tools as SynDEx, Gaspard, StarSoc … are software for the design of the specific systems (real-time Systems, circuits dedicated ASIC, FPGA) containing only hardware components. This new extension is dedicated to the design of systems based on execution graph. The design of this type of systems by the F4MS framework is made in two stages: the first stage consists in defining all the components (Software and/or hardware) necessities for the system to realize. The second stage consists in integrating these components in the form of an execution graph, which contains the description of the final application, i.e how the components are organized (position of the component in the graph), and how they will interoperate with each other.

In the following, we present the characteristics of F4MS framework, and the design methodology.

A. Features of F4MS

F4MS has the same characteristics of TI4CS framework [1,3], at the same time gives the possibility of integrating two types of components completely different in terms of design and architecture.

We quote here a number of characteristics that we have considered during the design stage of the new version of that framework:
- A modeling of several levels of abstraction: The specification can use levels of abstraction combined to accelerate simulation, because in certain cases, any detail is not necessary to test the concept of a design and some of its functionalities
- A separation between the communication model and the treatment model describing the system: the size (format) has to allow a refinement of the communication network between independent modules and the optimization of the internal behavior of system components.
- Heterogeneity is the possibility to use multiple programming languages and hardware architectures (FPGA, ASIC, etc...), each one for a different part of the system. However, and since each one of these elements has these properties which can be exploited for the optimized analysis and implementation. These properties are different from a one model to another which inhibits the analysis and the optimization of the whole system beyond the limits of the language and hardware architecture, and thus the platform must resolve this problem.
- Distributed Validation: it makes it possible to distribute system modules to be validated through a network. It is then possible to simulate a whole system while using a suitable simulator for each development team. The simulators do not any more need to be all grouped on a machine or a particular site which generally requires a more computing power. The teams can then collaborate in the level of simulation and of the development. Of course, the simulation performances are dependent on the network used and its load during the simulation of the system.

B. Design methodology

The design methodology of mixed architectures [4, 5], grouped the technical aspects and the organization aspects. It coordinates the use of several combined tools of conception and the cooperation of several aspects bound at all level of a system development. The teams of the software and the teams of the hardware can work in parallel, in an environment of cooperation and collaboration which reduces the development cycle of the conception [6, 7, 8].

The designers must take several decisions to clarify the details of this architecture [9, 10], in a consistency that allows the best compromise driving performances / area / consumption / flexibility / reusability / time to market.

The main design steps are summarized below:

1) System specification
They are high level specifications of abstraction and allow expressing the needs of a designed system:
- Functional specifications describe with exactitude what the system will have to carry out (details of the operations to be done, relations between inputs and outputs, and results to produce).
- Non-functional Specifications describe the conditions under which the system must operate: meet
performances, consumption, average yield, the cost of manufacture,...etc.

2) Architecture

This stage consists of three steps:

− The partitioning or splitting of the system to software/hardware component based on the estimation performance: explore the alternatives of design to identify those which adapt best to the constraints of the system. This step carries out the transposition of the functions of the system on software and hardware components. The components are software entities, programmable processors, FPGAs, and memories.

− Development, Maintenance, Reuse

This step consist of implementing new components, maintain existing components and trying to adapt them to the context, or / and reuse of components developed by other agencies.

− The co-simulation is an important step in validating the behavior of a component after the Software/Hardware partitioning.

3) Integration

The execution graph \[2\] of F4MS is a workflow for the description of mixed systems architectures, based on softwares / hardwares components. He allows describing a system as being a set of components (monolithic or composite) which implement interfaces, connectors (interconnections between components) and their compositions.

A component of execution graph presented as a calculation unit, or a data warehouse. An interface specifies the services which the component provides. The connectors model the sequencing and interaction between components through their interfaces. A composition represents a graph of components, connected between them using connectors.

The execution graph is consists of two graph: the scheduling and parallelism graph (to organize the components and describe the sequence of execution) and the interaction graph (to ensure interoperability between components), it also includes information about the parameter setting and the configuration of the properties of execution with each component.

Fig. 3 – General model of mixed systems

IV. APPLICATION

To illustrate this work we propose an improved version [11] of a mixed system for the establishment of VPN solutions. In this section we describe the design phases according to our methodology.

A. VPN solutions specification:

This phase of design in summer already realized in our first version of VPN solutions, because we assume that there is always a software layer that will manage each hardware component (allocation, Resource repartition and recovery of data). Consequently we decided not to change the structure of the application.

The design that we proposed consists of two parts:

− The non functional part of the application which provides the security constraints to be respected (needs analysis in terms of cryptographic techniques, and in technological choices (OpenSSL and IPSec) for the implementation of the VPN solutions), the objectives and the advantages of these solutions compared to what already exists on the market in term of quality, speed of development, modularity, maintainability, portability, feasibility and most importantly ease of use.

− The functional part which is to define the organizational structure (the scheduling and parallelism graph) of VPN solutions in scenarios which represents the same application model and describes the order execution of components. We note that this phase of specification is made after the second stage of the design methodology that we will present later.

B. Architecture

1) Partitioning solutions

Partitioning is an operation charged by different skill levels (integrators) and having an expertise in specification, optimization, and integration. Generally the objective aimed by this phase of conception is to obtain a homogeneous architecture for low cost and which satisfy the constraints of execution of the application.

When partitioning, it is to answer the following questions:

− What are the components necessary to establish the VPN solutions?

− With which technology (software or hardware) must be used to implement each component?

− Which is the profit (performance, surface, and cost) obtained after an operation of refinement?

To answer these questions, we have studied the needs, which must answer a certain criterion that was listed.

2) Components development

The study of the needs led us to identify three types of components that we saw necessary to the establishment of solutions VPN (SSL and IPSec[12] ) :

− Components using the functions of encryption, authentication and certification from the OpenSSL library.

− Components configuration of the protocols: We implemented components allowing the parameter setting and the configuration necessary to the operation of the protocols to be used.

− Components to run the VPN connection at the end of the configuration of the solution chosen by the user.
Concerning the choice of implementations of components either in software or in hardware [13, 14], we held in account the following criteria: the cost in execution time, the type of treatment (management, calculation), the security level, surface or consumption. However, we propose the hardware implementation for the component of encoding and authentication of the data based on the IPSec protocol, because it requires an important allocation of the processor because of the complex nature of the calculations made by these operations, in order to offer both a low cost in execution time and a higher level of security.

C. Components Integration

It is the last phase of the design methodology, which consists in assembling the components software-software, software-software/hardware or even software-hardware in order to realize a usable application. For the example of the implementation of VPN solutions, this phase can be summarized in two parts:

- The first part is common between the two solutions to generate the certificates required for authentication by using the component 0 to create the certification authority and generate the certificates request that need to be signed by the component 2.

- The second part is to describe two scenarios of execution, so that each scenario represents a VPN (VPN SSL and VPN IPSec):
  - Scenario 1: Consist in to generate the necessary certificates to the authentication while using the component 0 to create the authority of certification and generate the certificates request by components 1, these request will be signed by the component 2. The components 3 and 4 will be executed in parallel because they do not interact between them. They allow generating respectively the files of the parameters Diffie Hellman and the shared-key. The component 5 to edit the file of configuration of the tool OpenVPN. The user can launch the connection VPN SSL while using the component6.
  - Scenario 2: In the second scenario, it is to use the hardware component 7 which implements the IPSec protocol to authenticate and encrypt the data.

### REFERENCES


[14] Nadia Nedjah and Luiza de Macedo Mourelle "Efficient and secure cryptographic systems based on addition chains: Hardware design vs. software/hardware co-design", Integration, the VLSI Journal, Volume 40, Issue 1, January 2007, Pages 36-44