Separating Modeling and Simulation Aspects in Hardware/Software System Design

J. Lapalme E.M. Aboulhamid
Université de Montréal, Canada

G. Nicolescu
École Polytechnique de Montreal, Canada

F. Rousseau
TIMA, Grenoble, France

***Abstract* — Many are working on the productivity gap problem affecting the Electronic Design Automation. None of the current paths seems to be a silver bullet but object-oriented framework-based solutions such as SystemC are gaining a great deal of momentum and acceptance. Despite all the efforts which have been spent one the development of these types of solutions, as well as the numerous satellite tools which add a lot of value to the approach; very few efforts have been spent on the cornerstone task of investigating which software design techniques and technologies should be used to develop effective framework-based solutions. The main objective of the article is to present how new software engineering technologies may be used to create better framework-based modeling solutions by promoting separation of concerns between modeling and simulation aspects.**

Index Terms: Framework, Modeling, Simulation, System-Level Design

I. Introduction

As transistor integration reaches the order of billions, the already significant productivity gap which plagues the Electronic Design Automation (EDA) industry will only widen further. Many are working on the problem from different angles such as dedicate modeling languages, library/framework-based solutions and sophisticated EDA/Tools based on proprietary technology.

None of the current paths seems to be a silver bullet. However, object-oriented framework-based solutions such as SystemC [7] are gaining a great deal of momentum and acceptance by industry. Given this, we started, in late 2003 working, on a new .Net based methodology enabling fast and efficient creation of EDA tools for complex systems design. This methodology enabled the design of a new tool called ESys.Net (Embedded system design with .Net) [3].

Despite all the efforts which have been spent on the development of framework-based solutions, as well as the numerous satellite tools which add value to the approach by enabling sophisticated design flow, very few efforts have been spent on the cornerstone task of investigating which software design techniques and technologies should be used to develop effective solutions.

Software frameworks are quite difficult to build for their design has tremendous impact on: their effectiveness, their ease of use, their ability to promote good designs and their capability to be extended easily.

By combining the power and advanced capabilities of a modern object-oriented programming language such as C#/.Net, and the flexibility and elegance of modern software design patterns such as Inversion of Control and Proxy, it is possible to create a novel framework-based solution for hardware/software system modeling and simulation. We will present such a solution which permits a clear and unambiguous separation between the modeling, the verification, and the simulation aspects creating perfect separation of concerns. The level of separation of concern offered by the solution permits the elaboration and refinement of various simulation engines such as distributed and emulation without any modification to system models previously created.

The main objective of the article is to present how new software engineering technologies may be used to create better framework-based modeling solution. In this work we will present (1) interesting software engineering technologies, (2) show how current solutions lack “separation of concerns” in their design and the discuss the impact, (3) present a novel modeling framework based on the technologies presented earlier, (4) present a simulation framework for the modeling framework and (6) discuss the benefits of the solution regarding simulation, synthesis, and verification of the modeled hardware/software systems.

II. BACKGROUND

A. Separation of Concerns (SofC)

The basis of Separation of Concerns is the decomposition of a problem in sub-problems which are orthogonal to each other. It takes a classical divide-and-conquer approach to problem solving but relies on aspect decomposition instead of traditional functional decomposition. Within the context of system modeling and simulation, it can be said that the problem of modeling a system is orthogonal to the problem of simulating it.

B. C#/.Net 2.0 and Generics [7]

At the end of 2005, Microsoft released the next official version of .Net and the C# programming language, both versioned 2.0. Of the many enhancements made to .Net and C#, the implementation of generics types is especially important. The concept of generics was implemented as an extension of the metadata. Because of its implementation strategy, .Net generics are resolved at run-time and not compile time. This makes a big difference for at runtime, through reflection, it is possible to determine if an object is an instance of a generic type and the bound types of a generic instance. It is also possible to bind dynamically a generic type and create instances of that binding.

This implementation of generics which allows the runtime analysis of generic bindings, the creation of new bindings and instantiation of those bindings is a very powerful feature that we will exploit later in the article. These capabilities, to our knowledge, are unique for a statically typed programming environment.

III. Modeling and Simulation Framework

In the domain of system-on-chip modeling and simulation, many efforts have been invested and several contributions have been proposed. Designers currently have at their disposal efficient standard solutions for hardware modeling and simulation (e.g. VHDL, Verilog) but none are close to perfect. Most currently available modeling/simulation solutions fall into one of two categories: those based on a framework approach and those based on a domain specific language. The first group may be represented by solutions such as SystemC, ESys.Net, JHDL and Plotemy 2 [8]. The second group may be represented by solutions such as SpecC, VHDL, and SystemVerilog. In this paper we are concerned with the first group of solutions because they are constrained to the syntax and semantic limitations of a general programming language in order to define the necessary concept for system modeling and simulation.

A. Lack of Separation in Current Framework Based Solutions

Original HDLs such as Verilog and VHDL were developed from the start with modeling in mind; the simulation of models described with those languages was a secondary issue. This had a great influence on those standards for there are very little simulation semantics in them. This separation of concerns between modeling and simulation semantics is at the very opposite of environments such as SystemC, ESys.Net and Ptolemy. We intentionally omit the terminology of “language” to describe these solutions for they are truly simulation solutions and not modeling languages. Our reluctance to qualify the latter solutions as modeling languages lies in the fact that there exist no clear boundary between the aspects for simulations and modeling, one cannot model with these solutions and easily change the simulation implementation, especially after the model has been compiled with the simulation framework. In a perfect object-oriented framework a model should “at all times” only be dependent on the modeling aspects of the framework and not the simulation aspects. The “glue” element between the model and the simulation framework would be the modeling framework which would serve as a contractual interface.

In SystemC and ESys.Net, a designer has the responsibility to instantiate modeling concepts such as Event and Signal objects. The issue here is that since the objects are instantiated by the designer, the implementation of those objects are determined indirectly by the designer. Since the Event and Signal objects contain code that are directly related to the implementation of the simulation environment, the model created by the designer is indirectly tied to a simulator implementation. One could argue that it is possible to change the binding of a model to a certain simulator by changing the implementation to which it is linked (such as with SPACE) [9], but the model is compiled the binding is permanent. As stated earlier, a model should, at all times, only be dependent on the modeling constructs and not the simulation implementation. Since a SystemC or ESys.Net model is directly and indirectly dependant on a simulation implementation it does not respect true separation of concerns.

IV. SoFC Modeling Language (SoCML)

A. Overview

SoCML is a modeling framework inspired by SystemC and ESys.Net. For the moment it offers a subset of the latter but all the modeling semantics could easily be added. The objective here is not the implementation of a modeling framework but the demonstration on how to design a modeling framework which is separated from the implementation of a simulator for the modeling framework. The modeling framework contains the concepts of ports, signals, modules and channels with the same semantic meaning as SystemC and ESys.Net. The major difference is that the discrete-event model of computation semantics are defined only with abstract classes, virtual empty methods, interfaces and attributes.

```
  public class Consumer : BaseModule {
    public inEvent sync;
    public sinPort<int> input;

    [NonBlockableProcess, Sensitive("input")]
    protected void Consume() {
      Console.WriteLine(input.Value.ToString());
      Wait(sync);
    }
  }
```

```
  public class Producer : BaseModule {=
    public outEvent sync;
    public binPort clock;
    public outPort<int> output;
    private int i = 0;

    [BlockableProcess , Sensitive("clock")]
    protected void Produce() {
      output.Value = ++i;
      Wait(35);
      sync.Notify(25);
    }
}
```

```
[ClockDomain(20)]
  public class Model : BaseModule {
    private signal<int> sig;
    private clock clk;
    private Producer producer;
    private Consumer consumer;
    private biEvent sync;

    protected override void SectionPortBinding() {
      producer.clock = clk;
      producer.output = sig;
      consumer.input = sig;
      producer.sync = sync;
      consumer.sync = sync;
    }
 }
```

Code 1: Producer-Consumer Example

There are very little semantics differences between SoCML and SystemC.

It should be noted that there is no implementation code at all in the modeling framework. This is intentional for as stated earlier a modeling framework should only contain modeling

Top

semantics.

Code 1 is an example of a produce-consume model. The model is very similar to SystemC and ESys.Net. In the consumer module, the *NonBlockableProcess* attribute is equivalent to an SC_METHOD in SystemC. The Sensitive attribute indicates that the process is bound to the sensitive event of the signal bound to the port called input. The sensitive event has a clear semantics meaning which is "sensitive to writes on the signals no matter what the value. In the context of a *NonBlockableProcess*, the wait has the same meaning as a next_trigger in SystemC.

In the producer module, the Blockable attribute is equivalent to an SC_THREAD in SystemC. The semantics of the producer should be interpreted in the same way as it would be in SystemC.

Here in the model we can notice the use of the *SectionPortBinding*. The *ClockDomain* attribute defines a clock frequence for all clocks defined in its scope. A *ClockDomain* attribute may be assigned to a particular clock in order to override its parent's scope.

I should be noticed that there are no object instantiations in the model. This information is intentionally left out for two main reasons: object instantiation adds no information to the model and by delaying the instantiation of the objects until needed (such as at simulation time), we allow the implementation of the semantics to be chosen depending on the context.

B. *Design Constraints of SoCML*

In order to keep the framework "clean" of all simulation semantics and artifacts we started with the analysis of ESys.Net in order to determine the implementation elements that had to be eliminated from the framework.

There were two main types of elements which had to be eliminated: method bodies within the framework which were biased towards a certain simulation implementation and framework classes that had to be instantiated while containing code which was biased towards a certain simulation implementation. The two types of elements are clearly exemplified by the code in the Wait methods of the BaseModule class and the Event object.

The Wait methods contain code with pauses the current executing thread in order to switch with the simulation kernel thread. The methods also contain codes which make calls to internal methods of the simulation kernel. The Event class contains code which permits the scheduling of event through calls to the simulation kernel.

V. A SIMULATOR FOR SOCML

In order to complete the demonstration of our modeling framework design approach, we created a simulator for the SoCML. In order to achieve our implementation it was necessary to find solutions to the constraints we imposed on the modeling framework.

A. *Class Instantiation Problem*

The problem of instantiation of an implementation of a variable of a given type is basically the problem of class instantiation. The software design pattern called Inversion of Control is a perfect fit for the problem.

1) *Inversion of Control*

IoC is a design pattern which enables the decoupling between types. A type instance designed according to IoC does not instantiate objects which fulfil its dependency needs but rather delegates the instantiation responsibility to an execution environment and consumes the dependency through a interface contract which the dependency instance implements. The execution environment, through the use of a defined dependency, needs declaration between it and the type or type instance, locates an implementation for each required dependency and instantiates. Once the implementation instantiated, the environment gives the requester access to it through another defined convention.

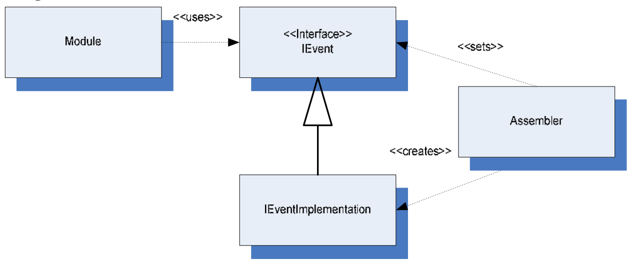

Figure 1: Module-Event IoC

Figure 1 represents a typical UML diagram depicting the dependencies between a **Module** and an **Event** using IoC. The **Module** object uses the IEvent interface in order to interact with an **Event** object but is must also instanciated the implementation of the **IEvent** interface. The **Module** object is the resquetor and the **Event** object is the dependency. In most current modeling solutions (ESys.Net and SystemC), the **IEvent** interface does not exist but is substituted for the implicit interface of their respective event classes.

In the diagram, the **Module** object doesn't have the responsibility of instantiating the implementation of the interface. The instantiation task is delegated to an **Assembler** object.

2) *Inversion of Control With Reflection*

The main stream techniques used to implement IoC are usually satisfactory in the concept of business applications, but they are not sufficiently transparent to be use in the context a modeling framework. It would be necessary to "pollute" the modeling framework with IoC implementation mechanisms which have nothing to do with modeling.

Our simulator of SoCML uses model analysis through reflection in order to act has a container. The core of the analysis is very similar to the one used in ESys.Net, the only significant difference is the instantiation upon detection of the modeling semantics declarations. The analysis and implementation strategy used by the simulator is made possible by the runtime resolution of generics in the .Net framework. The strategy could not have been used by an implementation in Java or C++.

B. *Virtual Method Implementation Problem*

The problem of implementing a virtual method without the user being aware is a problem which often arises in the context of distributed applications. In traditional distributed applications, a designer uses an object which impersonates a remote object. The responsibility of the impersonating objects is to offer a simple interface to the user and marshal the calls to the remote object. The same basic technique can be used for a virtual method implementation problem. The technique is based on the Proxy design pattern.

1) *Combining IoC and Proxy Design Pattern*

In order to solve the problem we used a combination of the IoC and proxy design patterns. In the simulation implementation that we propose, the implementation of the Wait methods of a module or channel are achieved by the use of a proxy which catches the calls and delegates them to the simulator. The interception of the calls is achieved by creating dynamically a derived type from a user defined module/channel type at runtime and overriding all the virtual Wait methods with an implementation which delegates the call directly/indirectly to the simulator.

We achieve the creation of the proxy class by using the DynamicProxy.Net [6] framework distributed by the Castle project. The framework supports the creation of proxy type from generic types.

VI. Benefits of our approach

Our modeling/simulation framework approach has many subtle but very important benefits that we present in this section. The benefits do not add significant value to the semantic capabilities of the solution but rather the news possibilities which become available because of the separation of concerns that we have achieved between the modeling and simulation aspects.

A. *Perfect Separation of Concerns*

We believe that compiled version of models and modules may serve as the backbone technology for simulation implementation agnostic IP block-based designs [1][5]. It is very difficult to achieve true IP block reuse in solutions such as SystemC, for once compiled, IP blocks are statically bound to a simulation implementation. Separation from simulation environments is fairly important, especially in the context of SystemC, for multiple implementation of the SystemC framework exist, each supports different tools. In this context, an IP block which is not independent of a simulation may probably not be used with certain tools which are simulation implementation dependant.

B. *Possible Interception For Verification*

The utilization of design patterns such as IoC and Proxy enable a simulation environment to create chains of interceptors which can monitor different aspects of a model under simulation without having to add the verification elements in the model itself or without having to create a complex verification enabled layer such as SVC. The combination of a flexible framework design and the reflective capabilities of the .Net framework offer many possibilities for the creation of effective tools at low cost [2].

C. *Transparency to alternative simulation implantation*

Since all models created with the modeling framework are separated from the implementation of the simulation framework it becomes possible to use a model with different simulators in order to take advantages of alternative implementations. An emulation solution could be implemented by using the same model analysis techniques based on reflection and IoC but could instantiate the user-defined modules types. The simulation would be achieved not by using a proxy pattern but by orchestrating the execution of multiples hardware elements in the simulation kernel and by updating the variables of the instantiate model giving the illusion that it is the user model which is being simulated by the kernel. In this context, the instantiated model serves the role of a projection of the emulation.

VII. Conclusion

In this paper we presented the current lack of separation of concerns which is common in most mainstream modeling/simulation framework-based solutions. We argue about the important of such separation and its benefits. We also present state of the art software design patterns and technologies which can promote better separation of concerns between modeling and simulation aspects. Finally we present a novel modeling framework called SoCML which follows our guidelines as well a simulation implemented for the modeling framework.

References

[1] DALPASSO, M., BENINI, L. and BOGLIOLO, A. 2002. "Virtual simulation of distributed IP-based designs." *Design & Test of Computers, IEEE,* 19, 92-104.

[2] N. Gorse, M. Metzger, J. Lapalme, E.M. Aboulhamid, Y. Savaria and G. Nicolescu 2004. "Enhancing ESys.Net with a Semi-Formal Verification Layer." *In Proceedings of the 16th IEEE Intl Conference on Microelectronics (ICM'04)*, Tunis, Tunisia, Dec. 2004, 388-391.

[3] J. Lapalme, E.M. Aboulhamid and G. Nicolescu 2005. "Leveraging Model Representations for System Level Design Tools." *In Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP'05)*, Montreal, Canada, June 2005, 33-39.

[4] MICROSOFT 2005. *Microsoft .NET Framework.* Available from: http://msdn.microsoft.com/netframework/.

[5] G. Nicolescu,, S. Yoo, A. Bouchhima, and A. Jerraya, 2002. "Validation in a component-based design flow for multicore SoCs." *In Proceedings of the 15th international Symposium on System Synthesis ISSS '02*, Kyoto, Japan, October 02 - 04, ACM Press, New York, NY, 162-167.

[6] The Castle Project - DynamicProxy.Net. http://www.castleproject.org/.

[7] THE OPEN SYSTEMC INITIATIVE (OSCI) 2005. *SystemC 2.1 Language Reference Manual.* Available from: http://www.systemc.org/.

[8] Ptolemy Project. http://ptolemy.eecs.berkeley.edu/

[9] J. Chevalier, M. de Nanclas, L. Filion, O. Benny, M. Rondonneau, G. Bois, and E. M. Aboulhamid, "A SystemC Refinement Methodology for Embedded Software," *Design & Test of Computers, IEEE*, vol. 23, no. 2, pp. 148-158, 2006.

Top