Position Papers from The 8th Workshop for PhD Students in Object-Oriented Systems

Frank Gerhardt, Luigi Benedicenti, Erik Ernst (Eds.)

Contents

Run-Time Reusability in Object-Oriented Schematic Capture

David Parsons, Southampton Institute, U.K.

1 Introduction

This position paper describes a graphical tool for electronic circuit design that exploits a number of object-oriented techniques to generate code in the VHDL-AMS mixed mode hardware description language. In particular it uses a lightweight 'reflective' architecture to allow run time creation of new component types, and encapsulates some complex aspects of selecting component models behind simple visual representations. Although the current system has been developed using C++, its architecture is designed to be independent of any particular object-oriented language or software tool.

2 Schematic Capture

'Schematic capture' means the translation of the graphical representation of an electronic circuit (a schematic) into some kind of textual representation such as a netlist, hardware description language or simulation code. This in turn can be used to generate graphical output representing the electronic waveforms within the circuit and/or for circuit synthesis, for building a physical version of the circuit (figure 1.).

Schematic capture of electronic circuit designs is a well-established software application within the field of Electronic Design Automation (EDA). The standard graphical symbols (such as ANSI and IEC) for describing the components and connections in electronic circuits transfer easily to the domain of a Graphical User Interface (GUI). The circuit schematics created in this way can then be translated into a range of non-graphical representations, including digital hardware description languages such as VHDL (VHSIC Hardware Design Language), specialist electronic simulation languages

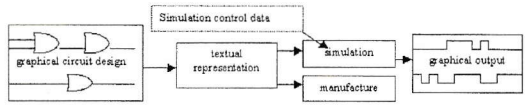

Figure 1: Schematic capture systems convert the graphical representation of a circuit into a text representation that may be used for simulation or design for manufacture.

such as SPICE (Simulation Program with Integrated Circuit Emphasis) or more general simulation languages such as Modelica.

3 The Need for New Schematic Capture Systems

Although existing schematic capture and simulation systems are adequate for many design tasks, there are domains where they are found wanting. One limitation of many traditional systems is the inability to model components at various levels of abstraction. We might wish to model a component as a single entity, but many EDA systems rely on structural modelling; they require higher level components to be modelled as aggregations of discrete components. The power switch in figure 3, for example, is modelled here as a collection of separate internal components: A switch, two resistors and a capacitor. However, we might prefer, for reasons of efficiency and simplicity, to model the whole component as a single higher level entity, treating it as a 'black box' with a behavioural rather than a structural description.

The same principle applies to the modelling of entirely new components, where a designer may wish to simulate the proposed behaviour of objects that have yet to be manufactured. Behavioural modelling facilities make this straightforward. While there are a number of systems that do allow for behavioural modelling, these are typically only applicable to digital circuits. Systems that do allow for mixed mode (analogue and digital) modelling are not always effective in simulating circuits that are highly complex, where the simulation time is too long and/or the results are inaccurate.

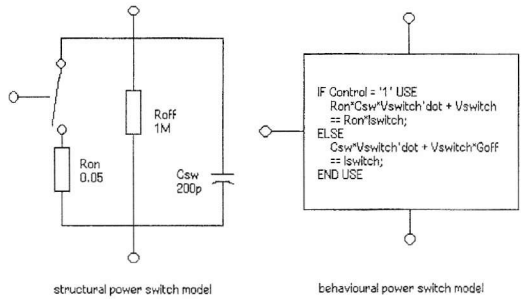

Figure 2: Modelling a switch using structural aggregation versus a 'black box' behavioural abstraction.

4 Schematic Capture for VHDL-AMS

Recent advances in languages for hardware design and simulation, particularly addressing the issues raised above, have stimulated research into more powerful and flexible schematic capture interfaces. One key development has been the publication of standard analogue extensions to the IEEE standard VHDL called VHDL 1076.1 (IEEE, 1997). The complete language, including the full VHDL 93 subset is also known as VHDL-AMS (VHSIC Hardware Design Language - Analogue and Mixed Signal). This language is intended for the design and simulation of mixed mode (analogue and digital) circuits, and provides a number of challenges for the schematic capture interface designer. A major aspect is the need to provide behavioural modelling facilities for both digital and analogue components via the interface. Although new components can be described at the code level by manually writing VHDL-AMS entity descriptions into a library file, this alone does not address the issue of schematic capture where a visual representation of any newly defined component must be available to be manipulated and integrated into circuit schematics. It is essential that the power to build components structurally at run time via the schematic editor (i.e. simply assemble components from a library into larger objects) is matched by the power to model new components behaviourally at run time.

One of the key aspects of mixed mode modelling in this language is the

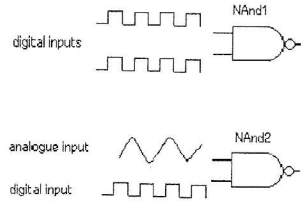

Figure 3: NAnd gates with different types of input signal requiring different VHDL-AMS model invocations.

added layer of complexity that it introduces to the underlying representational syntax, a complexity that a graphical interface can usefully render more transparent. The most obvious example of this is that different models of the same physical component may have to be used depending on the types of components they are connected to. Figure 3 shows two 'NAnd' gates that indicate how connectivity may vary for different components of the same type.

'NAnd1' has two digital inputs, whereas 'NAnd2' has one digital and one analogue input. From the circuit designer's perspective, these are objects of exactly the same type. However in the context of VHDL-AMS code generation these two gates must be invoked using two separate models, one for a fully digital gate and one for a gate with mixed mode input. A similar difference in models occurs where the output from digital components has an analogue connection leading to a proliferation of possible models that could be selected to represent a single physical object. Clearly, this is the kind of complexity that a schematic capture system could usefully encapsulate behind a more transparent interface.

5 Applying Object Technology

Previous work in building schematic capture interfaces suggests that object-orientation provides an appropriate paradigm. Object technology has been applied in many earlier schematic capture systems, using a range of tools such as Smalltalk (Walker, 1991), C++ (Raghavan, 1990) and object-oriented databases (Gupta et al, 1989). This is because there is an apparently sim-

ple mapping between the hardware components of an electronic circuit and the conceptual objects of object oriented design and it has also offered the widely promoted benefits of reusability and extensibility. Object technology provides the basis for extensibility at the coding stage using techniques such as specialisation inheritance and run time polymorphism (dynamic binding), and it is these traditional architectures that have typically been applied to modelling electronic components, an approach exemplified by Mak (1991). Using these techniques it is easy for an object-oriented programmer to add new classes/types to an application during development or maintenance.

Despite the clear benefits of an object-oriented approach, there are some areas where problems persist. Once a piece of software is delivered to the user, access to source code extensibility features is no longer available. Instead of a flexible and extensible environment, users may find themselves working with an inflexible monolith, providing no facilities for adding new types of object. Nevertheless, in many cases the ability for the user of a system to introduce new types at run time is desirable, even necessary. Electronic circuit design is a good example of this because a user may wish to use behavioural modelling to design and simulate physically complex components that are not provided in a standard library.

Some difficulties can also arise when building an object-oriented interface to an underlying system that is not itself object-oriented. VHDL-AMS, while exhibiting some of the characteristics of an object oriented language, does not have inherent polymorphism. This means that there can be no one-to-one mapping between a hierarchy of application classes in a schematic capture interface and the hardware oriented 'objects' being modelled in the underlying hardware design language. The nature of mixed mode modelling requires a solution that addresses the potential for high syntax complexity in the generated code while maintaining the simplicity of the user interface.

The VHDL-AMS schematic capture system addresses both system extensibility and syntax complexity by bringing some innovative aspects of object technology to bear. In particular, it applies an approach to object oriented design that allows the maximum flexibility in adding new types of object to a running system, while encapsulating some of the complexities of mixed mode simulation behind an intuitive interface. In both cases, aspects of polymorphism are used to build objects that are semantically rich.

6 Virtual Polymorphism

To provide the facility to add new component types at run time the software is designed in a somewhat unconventional manner, rejecting the traditional model of dynamically bound application classes in favour of what might be termed 'virtual polymorphism'. This term is used here to describe an architecture where the semantic richness normally associated with a classification hierarchy is collapsed into a single heavily configurable class. Objects of this single class are able to exhibit apparently polymorphic behaviours, flagged by a mechanism similar to a 'virtual constructor' (Coplien, 1992) where type definition is hidden behind a base class with a parameterised constructor. The implementation of these polymorphic behaviours is achieved by a form of reflection (Buschmann, 1996), where metadata is held separately from the domain classes that use it to enable it to be more easily modified and extended. In this instance the metadata provides all the characteristics of electronic component representations in schematic capture including their images, the nature of their connections (analogue or digital) and the ability to generate code. By adding to the meta data we can easily add new component types. The design strategy uses a number of aspects of reflection, notably 'property lists' (Sommerlad, 1997) and dynamic binding of predictable primitive types (such as graphics primitives and basic data types) rather than unpredictable domain level objects (new types of component).

7 Visual Polymorphism

In terms of the schematic capture graphical user interface, a second form of polymorphism is applied to reduce complexity. This might be termed 'visual polymorphism', where a single visual representation may be used to invoke underlying VHDL-AMS models of different types, depending on the context of connectivity to other objects (Parsons and Kazmierski, 1997). This is specifically applied in the context of designing mixed mode circuits, where the connectivity of components affects the way they must be described in VHDL-AMS. Because of the non-polymorphic nature of the language, a single digital component may have a number of possible models for different combinations of connectivity (ref. figure 3) but from a design perspective only one type of component is being used. One of the key features of the schematic capture interface is that it renders the selection of appropriate models transparent to

the user by passing this responsibility to objects within the system. A digital component object is 'aware' of the nature of the nodes to which it connects, and when it contributes to the generation of code will invoke the appropriate model by creating a 'code token' object that matches its particular circumstances. The type of model invoked by a particular component in the circuit can change dynamically if the nature of its connections change.

8 Summary

One of the key requirements of the VHDL-AMS interface has been to provide some of the power of an underlying language through the graphical user interface, including the user's ability to integrate new component types into the run time system. In essence, a form of visual programming interface is provided to allow the user to easily create new types of object without having to modify the program source. The software is thus a combination of application and programming environments, the lines between which have become increasingly blurred in recent years. However, this extensibility is provided without incurring an overhead of non-proprietary tools or dependence on specialist language or platform specific techniques. Rather, a reflective architecture is provided using standard C++ features that could easily be translated to other languages. The system demonstrates that object technology can provide run time support for reuse of an existing architecture to model new components and integrate them into a mixed-mode modelling environment. In addition, the complexity of mixed mode simulation has been encapsulated behind visual component representations that exhibit their own form of polymorphism, namely the ability to respond to their electronic context when generating VHDL-AMS code. The system is unique in providing a schematic capture interface specifically tailored to exploiting the mixed mode modelling capabilities of VHDL-AMS. It also demonstrates the breadth of techniques available with object technology for solving problems even in very specialised application domains.

References

[1] Buschmann, F. *Reflection, in Pattern Languages of Program Design 2* ed. Vlissides, J., Coplien, J. and Kerth, N. Addison-Wesley, Reading,

Mass., 1996.

[2] Coplien, J. *Advanced C++ Programming Styles and Idioms* Addison-Wesley, Reading, Mass, 1992.

[3] Gupta, Rajiv, Cheng, W., Gupta, Rajesh, Hardong, I. and Breuer, M. *An Object-Oriented VLSI CAD Framework - a case study in rapid prototyping* IEEE Computer, May 1989, vol. 22, no. 5, p.28-37.

[4] IEEE. *1076.1 Working Document - Definition of Analog Extensions to IEEE Standard VHDL* IEEE 1076.1 Working Group, May 1997.

[5] Kazmierski, T. *Fuzzy-Logic Digital-Analogue Interfaces for Accurate Mixed-Signal Simulation* Proceedings of Design Automation and Test in Europe (DATE), Paris, February 23-26, 1998, p.941-4.

[6] Mak, V. *DOSE: A Modular and Reusable Object-Oriented Simulation Environment* Proceedings of the SCS Multiconference on Object-Oriented Simulation, vol. 23, no. 3, San Diego, 1991, p.4-11

[7] Parsons, D. and Kazmierski, T. *Visual Polymorphism in Schematic Capture for VHDL-AMS* Proceedings of the 1997 IEEE/VIUF International Workshop on Behavioral Modeling and Simulation (BMAS'97), Arlington, Virginia., October 20-21, 1997.

[8] Raghavan, R. *Building Interactive Graphical Applications Using C++ in Applications of Object-Oriented Programming* ed. Pinson L. and Weiner R. Addison-Wesley, Reading, Mass., 1990.

[9] Sommerlad, P. *Self-aware Software: Understand and Build Reflective Architectures* Proceedings of OOPSLA 97, 1997.

[10] Walker, I. *A Smalltalk/V VLSI CAD Application* Computer-Aided Engineering Journal, April 1991, vol. 8, no. 2, p.47-53, 1991.