Object oriented development method for reconfigurable embedded systems

P.N.Green and M.D.Edwards

Abstract: The authors present a novel method for developing reconfigurable systems targeted at embedded system applications. The paper shows how an existing object oriented design method (MOOSE) has been adapted to include reconfigurable hardware (FPGAs). Previous research on reconfigurable computing has concentrated on the efficient mapping of algorithms to FPGAs. It must be realised that reconfigurable hardware usually forms part of much larger embedded systems, which include other hardware components, memories, and processors executing significant software tasks. The work represents a significant advance over current embedded system design methods in that it integrates the use of reconfigurable hardware components with a systematic design method for complete systems including both hardware and software. The development techniques are illustrated by a practical example. Although the enhanced methodology is applicable to some reconfigurable computing systems, further work is planned to extend our design technique to exploit the potential benefits of embedded systems which can be dynamically reconfigured at system run-time. The final objective is to produce an object oriented design methodology where system objects can be seamlessly implemented in either software or reconfigurable hardware.

1 Introduction

Embedded systems are proliferating in areas as diverse as high performance networking applications and high volume consumer products [1]. The management of design constraints is common to the development of all embedded systems, with factors such as cost, performance and power consumption having a profound effect on a product. Such constraints encourage a detailed consideration of implementation options, since many aspects of a system can be realised in a number of ways, each with different costs and benefits. For example, a data compression system may be implemented in software on a microcontroller, or in hardware on custom logic. If the hardware option is selected, designers may target either an ASIC solution, or one based on FPGAs. Programmable hardware solutions are becoming increasingly attractive [2] due to recent increases in logic capacities, improvements in performance, and the ability of some devices to be wholly or partly reconfigured during the run-time of a system [3, 4].

Design constraints typically relate to the system as a whole, and it is necessary to evaluate hardware and software implementation options in the context of the whole system. As a consequence a system design must be represented in an implementation-independent form to provide a context in which to evaluate implementation choices against design constraints. It is also crucial that a well defined route exists from the implementation-independent model to the chosen implementation technologies.

This paper is concerned with a development method for embedded systems that provides an implementation-independent modelling capability, and facilitates system implementation in any combination of software and hardware. The method, MOOSE (model-based object oriented systems engineering) [5], was initially developed to target software and/or ASIC implementations, and the focus of this paper is on the extensions that are necessary to target reconfigurable hardware as an additional implementation option.

2 Developing embedded systems with reconfigurable components

The use of FPGAs in the implementation of reconfigurable computing systems has been studied since the early 1990s and a wide range of FPGA-based platforms is listed in [6]. In this paper, we are not concerned with any particular computing platform, but with the provision of methods and tools which allow reconfigurable hardware to be incorporated into an existing embedded system development process.

There has been little, if any, previous work on design methods that are specifically targeted at the development of systems containing reconfigurable hardware. Current reported methods tend to advocate the separate design and implementation of software and reconfigurable hardware with some form of run-time support [7–9]. Even when design methods for embedded systems are considered, there is little work which is directly applicable to reconfigurable systems. A key problem is that although

IEE Proceedings online no. 20000483

DOI: 10.1049/ip-cdt:20000483

Paper first received 2nd September 1999 and in revised form 31st March 2000

The authors are with the Department of Computation, UMIST, PO Box 88, Manchester M60 1QD, UK

both software and hardware development are reasonably well understood, there are relatively few techniques that apply to the design of systems as a whole, especially when considering the integration of software and hardware.

Nevertheless, whole system approaches have been proposed which provide a seamless approach to the development of complete systems [10, 11]. These techniques have the advantage of using well tried methods but at the expense of tool and method integration problems. Alternative paradigms devise homogenous methods and tools that provide an integrated route to system development [12, 13].

The MOOSE method [5] adopts a homogenous approach based on object oriented (OO) principles. The arguments supporting the use of OO techniques for software are well known [14, 15] and MOOSE extends these principles to apply to hardware as well as software. Previous work [16], and that of others [17, 18] has been applied to both complete systems, and to fixed hardware alone. We are taking a natural step to extend MOOSE to model systems which also contain reconfigurable hardware.

3 The MOOSE method

The approach (see Fig. 1) starts with an informal product specification and ends when the system product can be implemented from descriptions of the hardware and software parts of the system during the 'product implementation' stage. The product idea is formalised into a specification in the 'develop architecture' stage, through the development of the behavioural model. This model is a graphical, hierarchical representation of the objects in a system, showing how they collaborate to satisfy the functional requirements. Complex objects are decomposed into primitive objects, which at this stage are uncommitted to either hardware or software implementations.

The behaviour of the primitive objects is specified in the next stage by partially synthesising an executable model, in C++, so that the system functionality can be validated against its requirements. This model can be validated with a number of scenarios to achieve a satisfactory behaviour.

'Transformational codesign' is concerned with committing objects to software or hardware implementations. The implementation technology for each object is identified, via a consideration of the design constraints using 'time-aware' model execution [19] that supports limited timing/performance analysis. In addition, the external interface of the system is completed, the numbers and types of processors are determined, hardware components are associated with processors, and concurrent threads are identified. The result is a committed model that satisfies the functional requirements and design constraints of the system. However, the implementation is incomplete as further hardware and software objects are required to support the operation of the system. The platform model adds implementation details such as processor, bus, and memory, together with operating system components, and interfaces between hardware and software objects. In the next stage, source code is synthesised from the committed and platform models. Software in C++ is produced for each processor, and a VHDL framework is developed [16] for hardware.

The necessary modifications to the MOOSE paradigm to sustain reconfigurable hardware are mainly concerned with the management of FPGA components so that they can be timeshared between hardware objects during the run-time of a system. The changes necessary to the design methodology and the alterations to the platform model are discussed in Section 5.

4 Hardware objects and run-time support

The concept of an object as a problem domain entity encapsulating state and behaviour [14] has long been recognised as being applicable to hardware as well as software [20]. This is also true of the notion of class as an abstraction defining the behaviour and state shared by a set of objects. In terms of reconfigurable hardware, behaviour is provided by the FPGA configuration, and state is held either in the configuration itself (for example in FPGA registers), or in system memory. Hence a class definition is essentially a description of the configuration, along with a scheme for storing object state. Common class relationships such as inheritance and association [21], may be given straightforward interpretations in this context.

The notions of class, objects and inheritance, while important to the goal of implementing OO systems in reconfigurable hardware, only address the static part of the OO model of computation. Of equal importance are the key dynamic aspects of OO systems, namely, interobject communication, object creation and destruction. Message passing mechanisms identified for software OO models, for example, blocking communications, are not always directly appropriate for modelling messages passed between hardware objects or hardware and software objects. In this case, message passing must be defined in terms that are closer to the actual mechanisms employed. Hardware objects typically exchange data via signals through ports, as defined by specialised protocols. Communication mechanisms have already developed in MOOSE for modelling communication between hardware objects (see Section 5), and there

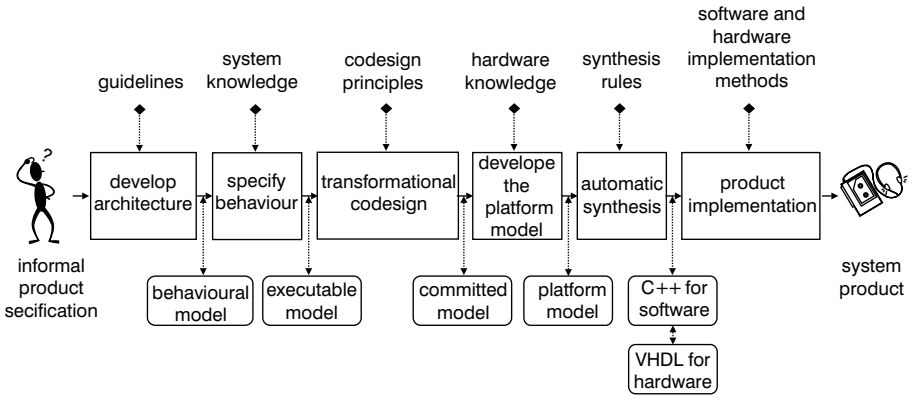

Fig. 1 *MOOSE method*

are well defined mappings between high-level message passing primitives and these low-level mechanisms.

Run-time reconfigurable hardware offers unique support for the dynamic creation and destruction of objects. The basic model of dynamic behaviour views an FPGA as a host to a time varying population of objects that are created, compute and communicate, may be scheduled/descheduled, and are eventually destroyed. A key difference with the OO software model is that these activities are concurrent and must be managed appropriately. This task is similar to that performed by an operating system, and we assume that FPGA management is performed by additional system software, the run-time support system (RTSS), that runs on one of the system processors.

The RTSS is responsible for creating objects, scheduling them for execution on an FPGA, and subsequently configuring the FPGA. It should also able to preempt hardware objects and restore their state. In addition, the RTSS has the responsibility for facilitating interobject communications involving FPGA-based objects, as described in Section 5.2.2. An advantage of this scheme is that it can also be used to implement the late binding [14] of method calls between a software and reconfigurable hardware object, or between two reconfigurable hardware objects, since the RTSS can determine the class of the target object in the same way as a run-time support system does for an OO programming language.

5 Extending MOOSE to support reconfigurable hardware

The MOOSE behavioural and executable models for a application are independent of hardware and software implementations and, therefore, no changes need to be made to the early stages of the development process. In the derivation of the committed model, however, the MOOSE notation must be extended to specify objects that will be implemented in reconfigurable hardware. Fig. 2 illustrates the notation for identifying the different types of objects in the committed model, including an extension for reconfigurable hardware objects which identifies the FPGA on which an object will run and the processor to which it is interfaced. Fig. 2 also indicates the MOOSE notation for the different communication mechanisms that are required in the various models. Modifications are required to the platform model to include facilities for scheduling the objects on an FPGA and to provide communications between objects running in reconfigurable hardware and the rest of the system.

The changes to the committed and platform models are illustrated by an example. The application is a distributed video surveillance system (VSS) [22, 23] which is used in buildings and consists of a set of embedded video controllers (EVCs) and a single supervisor control centre (SCS). The EVCs are distributed around a building for capturing, compressing, buffering, and transmitting images over a network to the SCS for decompression and display.

5.1 MOOSE committed model

For the purposes of this paper, it is more important to appreciate that a committed model represents the system as a set of objects whose implementation technologies and processor allocations have already been identified during the transformational codesign stage rather than to dwell on the derivation of the model itself. A simplified committed model of an EVC object, which captures, compresses and buffers images from a camera under the control of the frame manager object, is shown in Fig. 3. The model has three software objects sharing an ARM processor, and a reconfigurable hardware object running on FPGA_1 which is interfaced to the processor.

Fig. 4 shows a refinement of the compressor object as a set of four reconfigurable hardware objects. This design is based on a wavelet image compression algorithm and is adapted from the 'versatility' benchmark described in [24]. The four primitive objects run on the same FPGA and their executions are scheduled by the RTSS in a pipeline-like fashion.

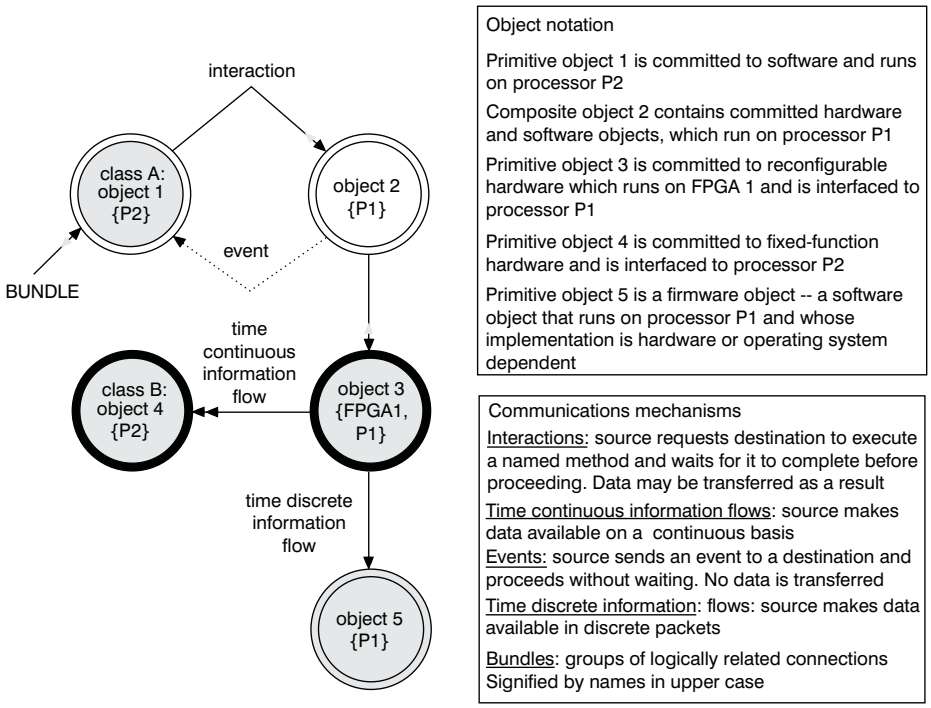

Fig. 2 *Extended MOOSE committed model notation and communications mechanisms*

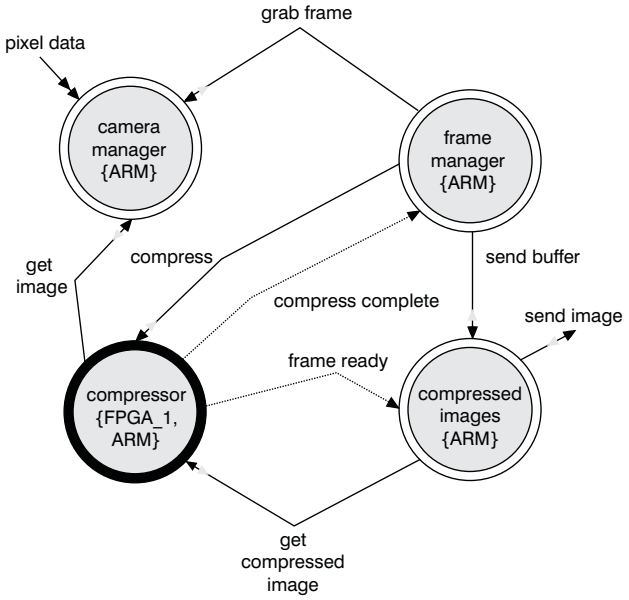

Fig. 3 *Simplified committed model of embedded video controller*

5.2 The MOOSE platform model

A MOOSE platform model identifies the hardware and software execution environment of a system. Platform model development is a complex process [16] and involves many factors including the generation and interconnection of hardware objects that support the execution of software (processor + memory + bus + system controller), and the development of suitable interfaces for the hardware objects together with their software drivers. In extending MOOSE to handle reconfigurable hardware, it is necessary to distinguish between the configuration of an object and the FPGA that runs the object. This is analogous to the allocation of a software object to a processor.

The structure of a platform model can be defined from three related viewpoints: communications, software interface and hardware. The communications view provides a high-level interface between software objects running on different processors. The software interface view supplies the software drivers required to control hardware objects, and a scheduler for each processor. Finally, there are a set of hardware views, which identify the hardware objects associated with each processor. The high-level view of a platform model for the ARM processor contains three objects corresponding to these three viewpoints, as shown in Fig. 5.

5.2.1 The communications view: The committed model denotes object communications as logical connections and the purpose of the communications view object is to map these connections to the communications mechanisms provided by the platform. The communications view object in Fig. 5 typically contains a number of simpler objects, which act as proxies [21] for the remote objects. To facilitate communication between reconfigurable hardware objects and software objects, an FPGA communications object may be added to the communications view object to service FPGA-based objects. This new communications object will run on the FPGA, and will receive application-specific calls from its resident configurations and map them to the hardware communications mechanisms provided by the platform.

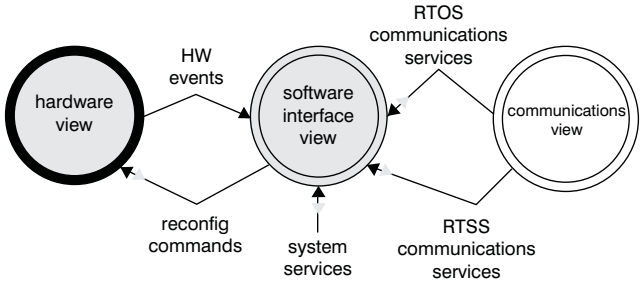

Fig. 5 *Simplified high-level view of ARM processor platform*

Fig. 3 shows event communications from reconfigurable objects to software objects (the 'frame ready' and 'compress complete' events). These are intended to trigger activity in the software objects, for example, 'frame ready', causes the compressed images object to load a compressed image into an internal queue. This event is raised by a child of the compressor object (entropy coded image), and is sent to the FPGA communications object as an interrupt to the ARM processor. The associated interrupt handler invokes an operation within the compressed images object that results in the call 'get compressed image' being issued.

Software to reconfigurable object communication must also be managed by the FPGA communication object, for example, consider the compress interaction issued by the frame manager which causes the wavelet transformed image to read and compress a new image. Depending on the actual scheduling of objects on the FPGA, this object may not be resident on the FPGA when the interaction is initiated. If this is the case, the FPGA must be reconfigured to contain the object. The RTSS would be invoked to reconfigure the FPGA and complete the interaction.

Object communications of this kind fit neatly into the MOOSE platform modelling framework, whereby calls from software to FPGA objects are directed to the communications view. The appropriate part of the communications

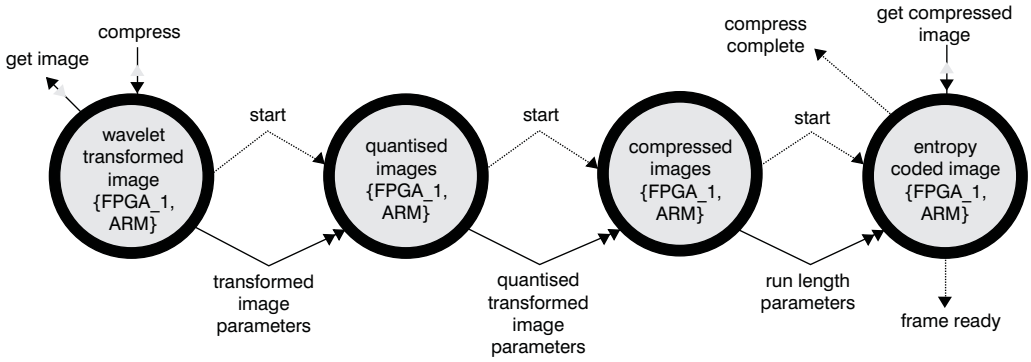

Fig. 4 *Committed model of compressor*

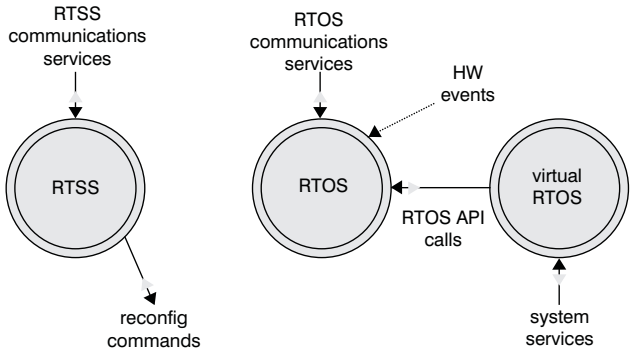

Fig. 6 *Software interface view of ARM processor platform*

view maps the calls to low-level communications encapsulated within the software interface view object. This mapping is indicted by the bundle, RTSS communications services, in Fig. 5. The bundle, RTOS communications services, would be used for communication between software objects on different processors.

5.2.2 The software interface view: The software interface view, shown in Fig. 6, contains software objects that control hardware objects. A real time operating system (RTOS) provides a software scheduling mechanism and device drivers to interface with hardware objects. For systems with reconfigurable components this software is supplemented with the RTSS to manage the FPGA.

The virtual RTOS object provides an abstraction layer between the application software and the RTOS itself. The interfacing of software to fixed-function hardware is performed by interface objects in the RTOS while the RTSS has been added to perform similar operations for interfacing software to FPGA objects.

When communicating from software to FPGA objects, (for example, the compress interaction in Fig. 3), requests are routed through the appropriate communications object which checks with the RTSS to determine whether or not communication is possible (for example, to confirm that the wavelet transformed image is currently running). If so, the RTSS allows the call and performs the interaction with the FPGA. If the destination FPGA object is not resident, the call from the software either returns indicating failure, or is suspended by the RTOS and resumed when the RTSS object has reconfigured the FPGA.

FPGA objects will typically communicate with software objects via interrupts. For example, the frame ready event generated by the compressor would be implemented as an interrupt (mapped to part of HW events in Fig. 6) handled by the RTOS. The handler would trigger the frame ready operation within compressed images.

5.2.3 The hardware view: The initial form of the hardware view can be synthesised from the committed model and consists of the processor, fixed-function hardware, FPGAs and their interfaces. This model is developed further to include the physical interconnections between the hardware objects. A full system implementation may be developed by adding further detail using standard hardware design techniques, supported by VHDL synthesis tools. A partly developed hardware view of the ARM processor platform model is shown in Fig. 7 and includes the FPGA_1 and associated configuration memory objects. The configuration memory object stores the configurations for the reconfigurable hardware objects that run on the associated FPGA. Device reconfiguration is managed by the RTSS via the bundle, reconfig commands.

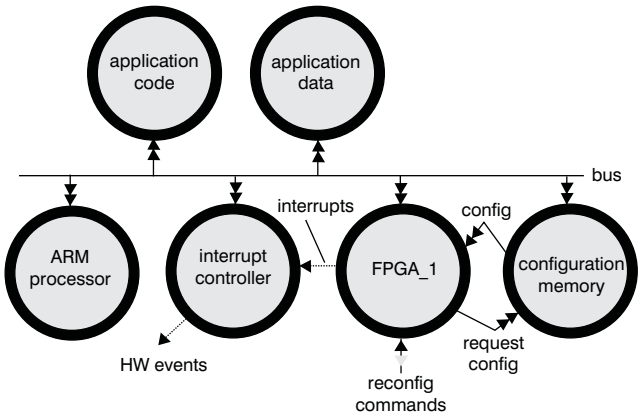

Fig. 7 *Initial hardware view of ARM processor platform*

6 Conclusions

We have presented the enhanced MOOSE approach to the development of complete embedded systems containing software, fixed-function and reconfigurable hardware. A uniform framework is provided to permit evaluation of different implementation options on the basis of design constraints. Clear and well defined routes to hardware and software implementations are provided through model capture and synthesis of the platform model. Furthermore, we have indicated, through an example, how the static and dynamic aspects of OO design may be incorporated into hardware as well as software objects.

Further enhancements to MOOSE will include further consideration of reactive systems where reconfigurable hardware objects can be scheduled in response to events from a system's external environment. This will include the idea of 'object preemption' which involves saving and restoring the state of preempted hardware objects.

Finally, we believe that our unified approach to system development will permit the widespread adoption of OO methods for embedded systems. Designers will be able to use such techniques without being specifically concerned with whether an object is implemented in hardware or software, a situation that is not true with currently available methods and tools. We believe we have made some progress towards this goal through the work described in this paper.

7 References

1 OSTERBERG, L.: 'Standards in real-time systems: enabling market growth', *in* 'Business and work in the information society' ed: ROGER, J.-Y., STANFORD-SMITH, B., and KIDD, P.T.: (IOS Press, 1999)
2 EDWARDS, M.D., and FORREST, J.: 'Software acceleration using programmable hardware devices', *IEE Proc., Comput. Digit. Tech.*, 1996, **143**, (1), pp. 55–63
3 'Virtex 2.5V FPGA series (XCV00) data sheet' (Xilinx Inc, 1998)
4 'AT40K FPGA data sheet' (Atmel Corporation, 1999)
5 MORRIS, D., EVANS, D.G., GREEN, P.N., and THEAKER, C.J.: 'Object oriented computer system engineering' (Springer Verlag, Berlin, 1996)
6 GUCCIONE, S.: 'List of FPGA-based computing machines' http://www.io.com/~guccione/HW_list.html
7 'ACEcard(tm)' (TSI Telsys Inc, 1998)
8 FLEISCHMANN, J., BUCHENRIEDER, K., and KRESS, R.: 'A hardware/software prototyping environment for dynamically reconfigurable embedded systems'. 6th International Workshop on Hardware/Software Codesign, 1998, Seattle, USA, pp. 105–109
9 MACKINLAY, P.I., CHEUNG, P.Y.K., LUK, W., and SANDIFORD, R.: 'Riley-2: a flexible platform for codesign and dynamic reconfigurable computing research'. 7th International Workshop on Field-programmable Logic and Applications, 1997, London, UK, pp. 91–100
10 KRONLOFF, K.: 'Method integration: concepts and case studies' (Wiley, 1993)
11 THOME, B.: 'Systems engineering: principles and practice of computer-based systems engineering' (Wiley, 1993)

12 HAREL, D.: 'Biting the silver bullet: towards a brighter future for system development', *IEEE Comput.*, 1992, **25**, (1), pp. 8–20
13 WOO, N.S., DUNLOP, A.E., and WOLF, W.: 'Codesign from cospecification', *IEEE Comput.*, 1994, **27**, (1), pp. 42–47
14 BOOCH, G.: 'Object oriented design and analysis' (Benjamin/Cummings, 1994)
15 RUMBAUGH, J., BLAHA, M., PREMERLANI, E., EDDY, E., and LORENSEN, W.: 'Object-oriented modeling and design' (Prentice-Hall, 1991)
16 EVANS, D.G., GREEN, P.N., MORRIS, D., and JAMES-ROXBY, P.B.: 'A systems approach to embedded system development', *in* 'Embedded microprocessor systems' ed: MULLER-SCHLOER, C., GEERINCKX, B., STANFORD-SMITH, B., and VAN RIET, R. (IOS Press, 1996)
17 PEETERS, J., JADOUL, M., HOLZ, E., WASOWSKI, M., WITASEK, D., and DELPIROUX, J.: 'Hardware/software co-design and the simulation of a multimedia application'. 7th European Simulation Symposium, 1995, Erlangen, Germany
18 WOLF, W.: 'Object oriented co-specification for embedded systems', *Microprocess. Microsyst.*, 1996, **20**, (3), pp. 141–147
19 MORRIS, D., EVANS, D.G., and SCHOFIELD, S.: 'Simulating the behaviour of computer systems: co-simulation of hardware/software', *Comput. J.*, 1997, **40**, (10), pp. 617–629
20 SCHUMACHER, G., and NEBEL, W.: 'Object-oriented hardware modelling – where to apply and what are the objects?'. EURO-DAC'96, 1996, Geneva, Switzerland, pp. 428–433
21 DOUGLASS, B.P.: 'Real time UML' (Addison-Wesley, 1998)
22 GREEN, P. N.: 'MOOSE models of the video surveillance system'. ESPRIT Project 20.592 (OMI/MODES), Deliverable TR6.2.5, 1998
23 GREEN, P.N.: 'Developing and implementing embedded systems from object models', *in* 'Business and work in the information society' ed: ROGER, J.-Y., STANFORD-SMITH, B., and KIDD, P.T. (IOS Press, 1999)
24 'Benchmarking tools and assessment environment for configurable computing: benchmark specification document – versatility stressmark' (Honeywell Technology Center, 1997)