An Example of Applying the Codesign Method MOOSE

Peter Green, Paul Rushton, and Ronnie Beggs

Systems Engineering Group, Department of Computation, UMIST, Manchester, UK

Abstract

An extended example of the application of a new method for the Object Oriented codesign of embedded systems (MOOSE) is presented. The example concerns an intelligent video system which is currently being developed using MOOSE. The paper highlights the notable features of the method (including executablity and the commitment process) with reference to the design of the video system, and presents tentative conclusions regarding the method's suitablity for embedded system codesign.

1. Introduction

A study in applying the hardware/software codesign method MOOSE (Model-based Object Oriented Systems Engineering) to the development of an embedded vision system is presented. MOOSE is a graphical/textual method which is geared towards the development of embedded computer systems and leads to codesign after the system as a whole has been investigated through the use of abstract, executable models (*uncommitted* models). Once the behaviour of the uncommitted model is deemed satisfactory, its components are gradually committed to hardware or software on the basis of 'systems engineering' decisions, based on non-functional requirements, on the results of analysing and executing the uncommitted model, and also on previous knowledge and experience. The result of this process is the *committed* model from which the hardware and software components of the system are implemented.

The vision system which is the subject of this paper, is still in the early stages of development, and so the results of applying the MOOSE method must be regarded as tentative. However, sufficient progress has been made to allow us to draw a number of conclusions about the method, and identify areas where further work is necessary (see Section 4).

2. Introduction to MOOSE

The early stages of the MOOSE method involve the construction of an abstract, executable model of the system under development (uncommitted model), which allows the architecture and behaviour of system to be evaluated. The basic ideas and motivation behind model execution are described in [8], and are also similar to those presented in [4] and [13].

Once the uncommitted model is deemed to be satisfactory, the various components of the system are committed to hardware or software, and detailed design begins. Thus the partitioning of a system into hardware and software subsystems only takes place after the system as a whole has been thoroughly investigated. This contrasts with the way in which embedded systems are often developed, where partitioning decisions are taken at a very early stage of development, and design and implementation in the two technologies proceeds separately. This approach clearly does not offer much flexibility in evaluating the relative merits of implementing sub-systems in hardware or software, a problem that MOOSE has been specifically designed to address.

MOOSE takes an Object Oriented approach to system development. Much has been written about the merits of Object Oriented software development (e.g. [1]) and many of these benefits are equally applicable to hardware. Briefly, Object Oriented approaches view a system as a loosely coupled set of objects which interact only through well-defined interfaces, and which conceal or encapsulate details of their inner workings from other objects within the system. Thus Object Orientation facilitates abstraction in analysis, design and implementation, simplifying the process of thinking about a system's overall characteristics, and limiting the effects that changes to the design of a particular object have on the rest of the system. These features, along with the capacity to create class hierarchies using inheritance, can also lead to the creation of reusable libraries of components which simplify the development of future systems.

0-8186-6315-4/94 $04.00 © 1994 IEEE

2.1 The MOOSE Method

Only a brief introduction to MOOSE is presented here and further details, including the semantics of the notation, are given in [9]. The application of the method begins with an analysis of the user requirements, and the separation of requirements into the various categories such as functional requirements (FRs), non-functional requirements (NFRs), design objectives (DOs) and design decisions (DDs), resulting in the Structured Requirements (see [9]). The development of the uncommitted model then proceeds as described below.

2.1.1 Uncommitted Models : The MOOSE notation allows an uncommitted system model to be expressed as a hierarchy of Object Interaction Diagrams (OIDs). This is similar to the diagram hierarchy of Structured Analysis [12], although the semantics are very different. Indeed, individual OIDs have more in common with the object diagrams of Booch [1], although OIDs provide more comprehensive ways of describing interaction and communication between objects, providing the additional flexibility necessary to model mixed hardware-software systems.

The elements from which *uncommitted* OIDs are constructed are shown in Figure 1a. External objects (rectangular boxes) exist beyond the boundary of the system under design, but interact directly with it, and so are used to model devices such as sensors and actuators which the system uses, as well as human users of the system.

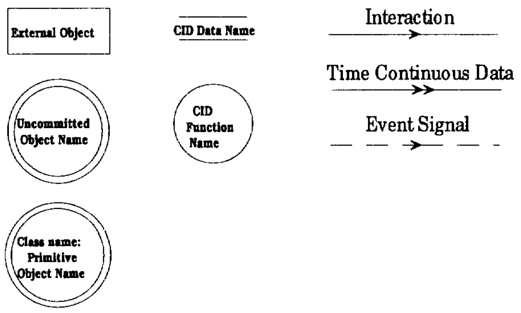

Figure 1a. MOOSE uncommitted notation.

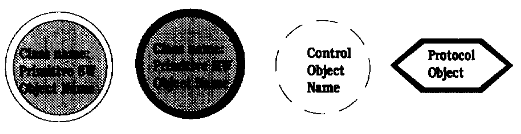

Figure 1b. Additional committed notation.

A number of different mechanisms are provided to describe the interaction between objects, within the uncommitted model. An *interaction* simply represents one object calling on another object to execute one of its operations (i.e. one of its procedures or functions). This is the form of interaction between objects familiar from most Object Oriented development methods. Interaction lines are annotated with the name of the called operation, and conventions exist in MOOSE with respect to the prefixing of interaction names to help in the interpretation of their semantics, e.g. objects may GET data, deliver or PUT data etc.

Interactions are sufficient to describe the communication between objects in software-only systems. However, in order to clearly and effectively model mixed hardware-software systems, additional forms of interaction between objects are needed, namely *time continuous data* and *event signals*. Interaction by time continuous data is provided for circumstances where an object needs to make information available continuously (for example, current time data) for other objects to access when necessary. This kind of situation is not well modelled by the simple interaction described above, and is sufficiently common in embedded systems to warrant a special piece of notation. Event signals model instantaneous occurrences and are a very important feature of embedded computer systems, and may arise both from the environment and within the system itself, and basic event signals cause the receiving object to execute a parameterless method. Events may also be parameterised as described in [9].

Now that the basic notation of uncommitted models has been described briefly, the hierarchy of OIDs can be considered. The first diagram in the hierarchy (the context diagram) represents the system as a single object, and depicts its interactions with the environment. The system object is then decomposed into a set of interacting objects, each of which has responsibility for some significant part of the system's overall functionality, as proposed in [11]. This OID shows how the objects interact with the externals and with each other. Each newly introduced object is then progressively decomposed, until the remaining objects are simple enough to be considered *primitive* i.e. simple enough to be directly implementable. Each object encountered in the decomposition process is fully documented in terms of its functional responsibilities, constraints on its implementation (for example response time constraints), and its interactions with other objects.

Primitive objects are described by class implementation diagrams (CIDs) which form the basis of class definitions for software objects. These resemble data flow diagrams with operations shown as processes and state information as data stores. The model is rendered executable by coding each operation in a simple procedural language.

Besides operations and state, other object classes may appear in CIDs, representing either inheritance or instantiation. The details of this aspect of CIDs, and the way in which class definitions may be synthesised for non-primitive objects, is dealt with in [10].

2.1.2 Committing Models : The aims of the commitment process are twofold. Firstly, decisions must be made as to whether objects are to implemented in hardware or software, and secondly issues regarding the number and type of processors to be used, the algorithms which are to be implemented etc. must be addressed. The way in which these decisions are taken is not tightly prescribed by the MOOSE method, allowing the most suitable approach for a particular system to be used. However, the existence of an executable uncommitted model allows feasibility studies and analysis to be undertaken, and provides a point of departure for constructing other more specialised models, e.g. performance models [5], which can be used to provide detailed information where conflicting NFRs exist (e.g. performance vs. cost). See [9] for further details.

Once objects have been committed, they are grouped into *soft* and *hard subsystems.* A soft subsystem comprises software objects which are to share the same processor, whereas a hard subsystem is made up from hardware objects which share data and control paths. Control objects may be added to coordinate the activities of the objects within either kind of subsystem. Communication between the various subsystems is through *protocol* objects which specify the interconnection rules between subsystems for the purpose of implementation, eg. a protocol object may define the communication mechanism between two soft subsystems running on different processors. Protocol objects are specified using CIDs, and in implementations appear as classes inherited by the interacting subsystems.

A subset of the notation used in the construction of committed models, appropriate to the partial models appearing later in the paper, is shown in Figure 1b.

2.2 Moving towards Implementation

If, after thorough investigation, the committed model is felt to satisfactorily meet the system's requirements (both functional and non-functional), then the detailed development of hardware and software can proceed. This development work will progress in a semi-independent manner, although the executable committed model will allow real implementations of software objects, and emulations of hardware objects to be tested within the framework of the overall system.

Work is currently in progress to automate the route from committed model to detailed design and this involves mapping soft subsystems into C++ class skeletons, and hard subsystems into VHDL.

3. The Intelligent Video System

The system which will be used to illustrate the application of MOOSE is a speaking LED/LCD display reader for blind people. This is one instance of a generic Intelligent Video System which is currently being developed with MOOSE. One of the objectives of this work is to create a library of reusable software and hardware objects which can be used to construct new video applications. However, for the purposes of this paper, attention will be restricted to the LCD/LED Display Reader.

The following sections describe the initial stages of the model and briefly explain its operation. One of the objects, the Pixel Filter, is then examined in detail. A discussion of the development of this object is presented to illustrate the MOOSE method.

3.1 The Display Reader

The requirements for the display reader system may be briefly (and incompletely) stated as follows :

A system is required to read the led/lcd displays on domestic electrical goods. The product is to be used by blind people so information should be transmitted to the user by a voice output. The device should have the following properties: it must be simple to use without the aid of sight, it should be inexpensive, robust, portable, reliable and operate in real time.

The first stage of the MOOSE method is to analyse such a product proposal and to create the Structured Requirements. These partition the overall requirements into functional and non-functional requirements, and design goals for the system, as discussed in [9]. They may be more detailed than the product proposal and seek to include information that may have been implicit or vaguely stated. For example, describing the display reader as 'portable' has been interpreted as meaning hand-held, lightweight and battery powered. Hand-held and lightweight are themselves interpreted to derive non-functional requirements (NFRs) relating to the maximum weight and size of the device. Recognising that the device should be 'battery powered' forces the issue of power consumption to the fore. Hence 'Must operate with minimum power consumption' becomes a design objective with a high priority. In terms of deriving an implementation which conforms to these objectives, it has been decided to implement the system on a single chip, and some of the design implications of this decision are discussed in Section 3.2.2.

Specifying that the device should 'operate in real-time' is, in this application, interpreted in terms of a maximum acceptable response time of 1 second from image acquisition to spoken output.

The way in which NFRs, DOs etc. are allowed to influence the development of the model is an important feature of MOOSE. However, as will be seen in the next 2 sub-sections, at present MOOSE only provides a framework to do this, and work on explicitly representing NFRs etc. on the OID decomposition is still in its early stages. Although they are formally stated in the Structured Requirements document, their impact on the OID hierarchy remains implicit.

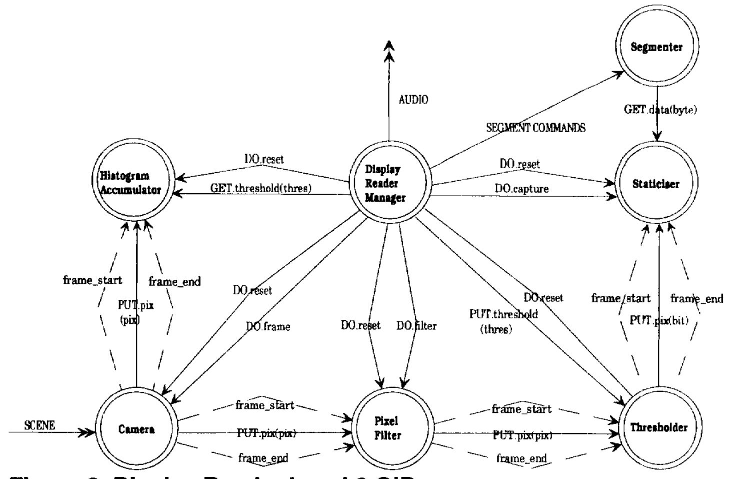

Figure 2. Display Reader Level 0 OID.

3.1.1 Operation of the Display Reader Model : The logical model of the Display Reader can be explained with reference to the level 0 OID in Figure 2. The Display Reader Manager object coordinates activity in the system, initiating the capture of an image (by sending the DO.frame message to the Camera object), 2-D filtering (DO.filter), and the calculation of a threshold grey-level value for distinguishing between character/icon pixels, and background pixels (GET.threshold). Once these tasks have been initiated the Thresholder object is instructed (via the PUT.threshold message) to create a binary image, which is then stored by the Staticiser. Using a binary image rather than a grey-level image is quite sufficient for the application [3], and helps to reduce the silicon area and the power consumption of the fabricated device. The Staticiser provides a stable platform for the Segmenter to detect and extract segments from the binary image. These are retrieved by the Display Reader Manager object which performs character and icon recognition based on the segment information.

Note that there is a potential timing problem with this architecture: as the Histogram Accumulator requires an entire frame to operate, if the Pixel Filter begins producing output pixels before the entire frame has been received by the object, the Thresholder will not have the valid threshold for this frame. This is indicative of the type of problem that could be encountered and must be solved during the commitment process.

3.2 Pixel Filter Decomposition

The Pixel Filter object is a low level generic filtering object that performs a 2-D convolution on a serial stream of pixels, producing a serial stream of convolved pixels at its output. Convolution of an image with a selected set of coefficients produces a transformed image which reflects the coefficients used. This is considered appropriate for reuse since by simply changing the coefficients used then different results can be obtained.

The decomposition and commitment of this object will be used to illustrate MOOSE and notable features of the method will be highlighted.

3.2.1 Uncommitted Decomposition of Pixel Filter : The decomposition of the Pixel Filter in the uncommitted model results in Figure 3. The Pixel Store object represents the need to buffer incoming pixels in order that the correct set of pixels for 2-D convolution are made available. The Convolver object is responsible for performing the convolution operation. The Filter Controller object is included to implement the interface interactions and control the filtering process. The PUT.status interaction effectively resets the store and allows it to store incoming pixels. The Convolver object is passed parameters by the Controller identifying the pixels to be filtered, and retrieves these pixels from the Pixel Store via a GET interaction.

The objects at this stage are declared as *primitive* since any further decomposition results in commitment decisions (see Section 2 and [9]). For example, decomposition of the Convolver would require decisions about whether it is to be implemented as hardware or software.

Thus, the construction of the uncommitted model allows the designer to encapsulate functionality and state within objects, postponing consideration of implementation details until later.

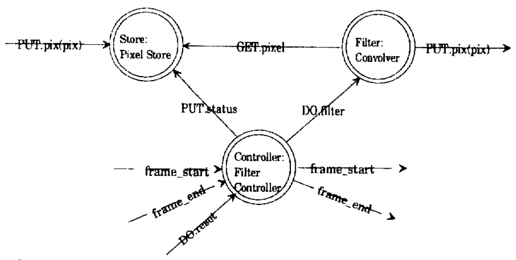

Figure 3. Uncommitted Pixel Filter.

3.2.2 Commitment : The committed model evolves from the uncommitted model via decisions made on the basis of NFRs, DOs etc. about whether a particular object would best be implemented in hardware or software. Considering, for simplicity, the Pixel Filter object and the NFRs relating to performance and silicon area, we can make a number of simple calculations to illustrate the commitment process.

We will assume an image size of 300 by 300 pixels, and that we will implement the filter using a parallel hardware architecture. With such an architecture, the major bottleneck in the convolution operation is in the multiplication time at each pixel in the filter matrix. Current CMOS technology can easily implement an 8 bit by 8 bit multiply within 200nS (where one set of eight bits represents the coefficient and the other represents the binary value of the pixel greyscale). Hence, including the latency delay to fill up the pipeline, such a hardware architecture could convolve the example image with a 3*3 filter matrix in approximately 0.02 seconds.

Now assume that the filtering is done in software. Figures for a typical embedded system processor, such as an ARM6 running on a 20 Mhz clock, indicate that each pixel requires 5 µS processing time, necessitating approximately 0.5 seconds to process the entire frame. (These figures were obtained from a knowledge of the number of machine cycles that each operation in the filtering process takes).

Now although the software implementation of filtering is well within the required 1 second, the bottleneck in the overall image processing time will be in the object recognition part of the Display Reader Manager object. Hence, in order to maximise the probability of the system meeting its performance target, the hardware implementation of the Pixel Filter should be used, so as to relax the performance constraints on the Display Reader Manager object as far as possible.

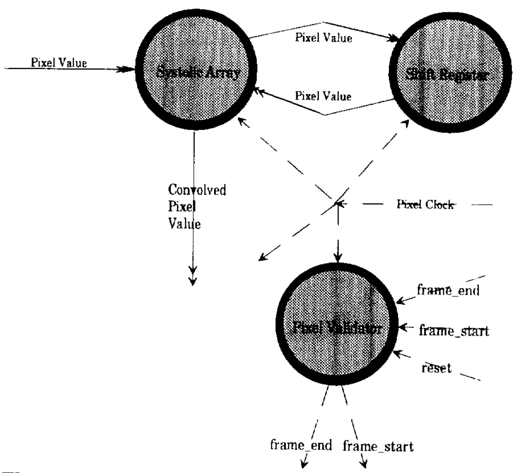

Figure 4. Committed Systolic Pixel Filter.

Now that it has been decided that the Pixel Filter should be implemented in hardware, it is necessary to consider different design options. Since this work is still in its early stages, only the systolic array [7] architecture shown in Figure 4 so far has been examined. The functionality of the Convolver in the uncommitted model maps to the Systolic Array, the Pixel Store maps to the Shift Register object and the Filter Controller maps to the Pixel Validator object. A systolic architecture, whilst not reducing the overall gate count in comparison with a more standard multiply-accumulate architecture, does present significant silicon real estate savings by virtue of its regularity and local interconnectivity[7].

The diagram shows how, in the process of commitment, the interactions etc. in the uncommitted model must be replaced by physical connections which implement the protocol of those interactions. Consequently the interaction PUT.pix has been mapped to a Pixel Clock event and a continuously available Pixel Value. In this way all possible interactions in the uncommitted model are replaced by their hardware committed equivalents.

The commitment of the Pixel Filter object to the systolic architecture illustrates additional features of the MOOSE method. In committing the Convolver to the Systolic Array and the Pixel Store to the Shift Register, there has been a small migration of responsibility, since besides the storage implicit in the Shift Register object, each processing element in the Systolic Array contains a small amount of storage in terms of a pixel latch (i.e. storage for a single pixel value). Thus, pixel latches are common to both the Systolic Array and the Shift Register (see Figure

5), and so are represented as a class in MOOSE, inherited by both objects.

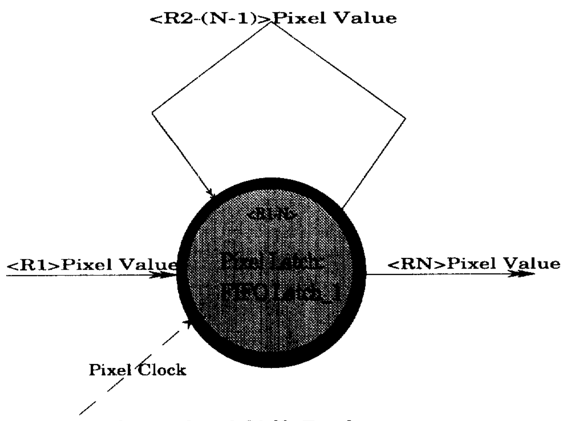

Figure 5. Committed Shift Register.

Figure 5 shows the use of MOOSE's repetition notation and how it can be achieved to a highly compact yet understandable representation of a repetitive structure. The Shift Register object is essentially a collection of Pixel Latches which are cascaded to form a synchronous sequential shift register. MOOSE allows a number of identical objects to be defined as a single object with the repetition notation defining the number of object instances. Repetition is denoted by the use of <R 1-N>, on the object bubble, where N is a repetition number. This is clearly an important feature of the notation, particularly for hardware commitment since repeated regular structures are most suitable for VLSI implementation. Note here that the CID of the FIFO Latch_1 object defines the class Pixel Latch.

The systolic array implementation of the Pixel Filter is simply one particular hardware design for this object which has been investigated. Alternative designs will be considered shortly.

3.3 Executing the Model

The executability of the Display Reader model has, to date, been valuable in two respects. It has provided a basis for experimentation with the uncommitted model architecture, and also has enabled errors in object design to be easily identified.

The initial work on the model involved a number of simple experiments, the purpose of which being twofold: firstly, to validate that the model operated correctly and secondly, to assess the effects of various thresholding and filtering algorithms. At this stage an actual camera could not be connected to the model so the model was executed using pre-captured scenes read from an image file. A figure of merit for each experiment was generated by calculating the probability of error in the thresholded image relative to a pre-determined 'best' configuration. Initial results were encouraging. The construction and execution of alternative algorithms and model configurations proved to be quick and straightforward. Work is now in progress to carry out a number of more sophisticated and rigorous experiments, to assess the operation of different image processing and segmentation techniques.

The execution of the model has also proved useful in identifying logical errors in the objects' design. For example, by executing the model, two important problems with the Pixel Filter object were discovered. Firstly, a deadlock situation was identified in the initial parallel architecture. Secondly, a subtle error in the convolution process was identified that only became apparent when the filtered image was examined in detail. The ability to identify and rectify such errors easily and at low cost clearly illustrates the advantages of an executable model.

The use of the executable model for evaluating commitment decisions is considered to be an important aspect of the MOOSE approach. However, work on the Display Reader system has not yet progressed enough for an evaluation of this aspect of the method to be made. However, we have been encouraged with the initial results based on experimentation with the uncommitted model and the systolic Pixel Filter. As a larger number of committed objects are completed, a more detailed evaluation the approach will be undertaken.

4. Discussion and Conclusions

We will now discuss some of the issues raised by the preceding study. It should be noted, however, that since this work is still in its early stages, the conclusions drawn are of a tentative nature.

To provide a framework for the discussion, it is useful to reflect on some of the features of a design method which make it suitable for the development of embedded systems. The features outlined below are not intended to be exhaustive, but do represent important issues which must be addressed by a design method for embedded systems.

Notation : A method's notation should be simple and clear, whilst being expressive enough to describe abstractions appropriate to mixed hardware/software systems.

Consistent progression between life cycle phases : It ought to be simple and straightforward to progress through the life cycle phases from analysis through to implementation, and the cost of the inevitable iteration between phases should be minimised. Revisiting previous life cycle phases during the development of embedded systems can be extremely costly.

Reasoning about a design : A method ought to provide a mechanism for analysing a design with respect to the system's requirements (both functional and non-functional). This mechanism will provide the basis for partitioning decisions.

Tool Support : Tools ought to be available to support all aspects of a method, from model capture, analysis and execution, through to the creation of skeleton implementations.

We will now consider how well MOOSE supports the features listed above.

As far as notation is concerned, both the introduction and the case study have demonstrated that though simple, the notation does support highly usable abstractions of commonly occurring mechanisms within embedded systems. Indeed the notation is as rich as that presented in [14], whilst at the same time enjoying the benefits of an Object Oriented basis. Representing a system in terms of a hierarchy of collaborating objects has been demonstrated to provide a number of benefits (see [1] and Section 2), and these are as relevant to hardware as to software. The augmentation of the basic Object Oriented communication mechanism (known as an interaction in MOOSE) with the other mechanisms discussed in Section 2.1.1, allows many common embedded system communication modes to be modelled in a realistic, but abstract, way.

The fact that MOOSE is based on Object Oriented principles means that the transition between life cycle phases is consistent and straightforward since there is no change in the underlying representation of the system, as there is with other kinds of approach (e.g. Structured Methods [2]). This allows the abstract uncommitted model to gradually evolve into a more detailed (committed) representation by means of the commitment process. The fact that a low-cost executable model of the system is available at an early stage in development helps to reduce the expense of iterating between life cycle phases by allowing a thorough investigation of the model prior to moving into the more detailed (and costly) stages of development.

The executability of MOOSE models also provides a basis for evaluating design alternatives as an aid to partitioning although, with respect to the Display Reader system, work has not yet progressed far enough to demonstrate this aspect of the method. Executability also allows a detailed assessment of the algorithms to be used in a system (in our case, image processing algorithms) in the context of a full system model, clearly a helpful feature.

At present, MOOSE provides a *framework* for making hardware/software partitioning decisions and a measure of support (via executability) for evaluating them in the context of the system's NFRs. Further work in this area is needed, specifically in connection with the formal representation of NFRs within models, and tool support for evaluating partitioning options on the basis of NFRs (perhaps through the use of cost functions) and it may be possible to include some elements of other published approaches to partitioning within MOOSE (for example, [6]). It should be noted, however, that this is not a straightforward issue, since the NFRs, DOs etc. requiring consideration during commitment will vary from system to system. Thus, we envisage an assistive tool which helps the designer evaluate the NFRs/DOs etc., rather than a wholly automated tool which mechanically evaluates all the constraints.

It is interesting to note that the commitment process may ultimately be simplified somewhat due to MOOSE's Object Oriented foundation. This is because one of the major aims of Object Oriented approaches is class (or object) reuse. Thus, if committed objects developed for one system are stored for future use along with characteristics such as performance, gate count etc., then the appropriate constraint information will be available when the object is considered for use during the commitment phase of a subsequent system.

As can be seen from the above, MOOSE is, to a certain extent, lacking in tool support, both in the area of commitment assistance, and also in the semi-automatic mapping from committed model into C++ and behavioural VHDL. However, work is in progress to automate the route from committed model into C++ and VHDL, and this will be reported shortly.

The results of the initial phase of this study appear to indicate that MOOSE has the potential to provide a comprehensive framework in which to design embedded systems, although further development work is necessary.

Acknowledgements

This work has been funded by the European Commission as part of the Open Microprocessor Systems Initiative (OMI) within the ESPRIT programme (Project OMI/DE-ARM 6909). We would like to acknowledge the input to this work from our colleagues who are involved in the development of the MOOSE method. In particular, thanks are extended to Professor Derrick Morris, Gareth Evans, Simon Schofield and Martin Bland.

References

[1] G. Booch, *Object-Oriented Design with Applications*, Redwood City, Ca: Benjamin-Cummings, 1991.

[2] P. Coad and E. Yourdon, "Object Oriented Analysis",

[3] D.G. Evans and P. Blenkhorn, "A Voice Output Reader for Displays on Video Cassette Recorders and other Domestic Products", to be published in *Journal of Re-habilitation Research and Development*, 1994.

[4] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Polti, R. Sherman, A. Shtull-Trauring and M Trakhtenbrot, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", *IEEE Trans. Soft. Eng.*, vol. 16, no. 4, pp. 403-414, 1990.

[5] P.P. Jain, S. Dhinga and J.C. Browne, "Bringing Top Down Synthesis into the Real World", *High Performance Systems*, vol. 10, no. 7, pp. 86-94, 1989.

[6] S. Kumar, J.H. Aylor, B.W. Johnson and Wm.A. Wulf, "A Framework for Hardware/Software Codesign", *IEEE Computer, pp. 39-45,* Dec. 1993.

[7] H.T. Kung, "Why Systolic Architectures?", *IEEE Computer*, pp. 37-46, Jan. 1982.

[8] D. Morris and D.G. Evans, "Modelling Parallel and Distributed Systems", *Parallel Computing,* vol.18, no. 7, pp. 793-806, 1992.

[9] D. Morris, D.G. Evans and P.N. Green, "Engineering Embedded Computer Systems", *to be published*, 1994.

[10] D. Morris, D.G. Evans and S. Schofield, "Model-based Object Oriented System Engineering (MOOSE): Part 1 - A Design Method and Notation, *to be published*, 1994.

[11] R. Wirfs-Brock and B. Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach", *SIGPLAN Notices*, vol. 24, no. 10, pp. 71-75, 1989.

[12] E. Yourdon, *Modern Structured Analysis,* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[13] P. Zave, "The Operational Versus the Conventional Approach to Software Development", *Comm. ACM*, vol. 27, no. 2, pp. 104-118, 1984.

[14] P.T. Ward and S.J. Mellor, *Structured Development for Real-Time Systems*, Englewood Cliffs, NJ: Yourdon Press Computing Series, 1985.