Challenges for embedded software development

Michael V. Woodward, *Member, IEEE*, Pieter J. Mosterman, *Member, IEEE*

***Abstract*—Embedded software development is becoming increasingly difficult because of technical and commercial pressures. This paper assesses the state of the embedded software development discipline, examines where it is now by posing a set of challenges, evaluates the state of development of key technologies, and lastly presents a vision for how embedded software development might proceed in fifteen years time.**

***Index Terms*—embedded software development, Model-Based Design**

I. Introduction

Embedded software developers work in one of the most difficult technical and commercial environments; projects are becoming increasingly complex and difficult to complete, whilst competition is intensifying. For example, high-end mobile phones are now designed with sophisticated video and audio capabilities and often include PDA-style functionality, however despite the technical difficulties of delivering these products, there are many companies poised to enter world markets [1]. The notorious "Law of Observed Functionality" [2] in consumer electronics holds that while system value (utility) increases linearly, transistor count increases exponentially. This exponential increase in transistor count gives rise to the well known "Design Gap" [3] which increases design costs, causing the ITRS (International Technology Roadmap for Semiconductors) to say that "cost of design is the greatest threat to continuation of the semiconductor roadmap" [4].

Traditional approaches to embedded project development have viewed quality, cost, and time as three vertices on a triangle and to "…choose any two" to optimize at the expense of the third [5]. Given the commercial and technical forces at work in the embedded software market, this approach is increasingly untenable and projects need to improve quality, cost *and* time. To do this will require disruptive changes to the way that we design, develop, and test embedded systems.

Much promising work has been done in analyzing the design process in terms of Models of Computation (MoC) [6]. For purposes of this work we can define a MoC as an executable model of the system aimed at a specific purpose, for example there might be an MoC that acts as a specification, an MoC for implementation, and so on. Model-Based Design [7] is one such methodology that fits within the MoC framework and orients the design process around a system model. This system model acts as an executable specification and a basis for design, implementation, and test and verification. This work contends that the use of Model-Based Design is likely to be critical to improving the embedded software development process.

This paper presents some of the challenges that keep embedded software development locked in the time-cost-quality trade-off. We will examine the state of development of some important technologies that may help break the deadlock, and lastly we offer a view of what the development process might be like in 15 years time.

M.V. Woodward and P.J. Mosterman are with The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA, 01760, USA (e-mail: [mike.woodward|pieter.mosterman]@mathworks.com).

II. The Challenges

We identify five challenges which we believe need to be overcome to break project development trade-offs. We have labeled these challenges *Complexity*, *Optimization*, *Interdependency*, *Verification*, and *Tools*.

A. Complexity

1) Description: complexity is the outcome of several trends. It arises because of the combination of more and more functionality onto a single system, increasingly complex standards (particularly in wireless and media applications), and the availability of more and more transistors to design with. Furthermore, as networking capabilities are becoming persuasive in embedded systems, a design becomes a system of systems, adding yet another layer of complexity [8]. To quote the ITRS "…together, the silicon and system complexity challenges imply *superexponentially increasing complexity* of the design process" [4].

2) Impact: the ITRS recognizes complexity as one of the key drivers behind increasing system design costs [4]. Currently increased system complexity can lead to increased project delivery times and quality issues.

3) Solution properties: some means of affordably coping with increased complexity must be developed. As complexity is increasing exponentially, this requires any solution to scale sub-linearly with complexity if it is to be affordable.

B. Optimization

1) Description: optimization is essentially a cycle of design, evaluation, and design again. It becomes a difficult issue because of the length of time taken to build a functioning system model or prototype to evaluate. An embedded system contains both hardware and software elements that interact in complex ways with one another which can make selecting the 'best' combination difficult. Currently the only way this can be achieved is by prototyping the different solutions or by

guesswork.

2) Impact: it is often the case that only one design can be optimized (due to the cost of building a functioning design), hence severely limiting the optimization possibilities. Choosing an incorrect or sub-optimal software/hardware combination can lead to project delays or even project failure.

3) Solution properties: it must be easier and faster to develop functioning system models or prototypes that enable a wide range of design options to be quickly and accurately evaluated. It must be possible to simulate the interaction of hardware and software.

C. Interdependency

1) Description: different parts of the design process are becoming increasingly interdependent, however today's design processes keep design domains separate. As a result we cannot explicit handle these interdependencies or take advantage of them. Perhaps the best known example is the hardware/software co-design challenge which has been well described elsewhere [9]. Another example is the relationship between behavioral modeling and implementation; the choice of algorithm and system design used in behavioral modeling is often partly dependent on the hardware chosen, and conversely the behavioral model often helps determine the choice of hardware, however there is no clear mechanism for including implementation details in behavioral models.

2) Impact: the lack of a clear way of handling interdependencies makes it difficult to make cross domain design decisions, for example to evaluate the trade-off of using a digital pre-distortion filer with a cheaper rf (radio frequency) power amplifier, or to make decisions about implementing algorithms in hardware or software. The consequence of separate and non-communicating design domains gives rise to multiple versions of a design that are not synchronized and it is not uncommon to find multiple and separate MoCs within a design process, e.g., a behavioral model, a fixed-point model, and an implementation model. In the worst cases, differences in MoCs do not become known until the production stage is reached, for example the differences in wiring diagrams used in the A380 [10]. These multiple design process 'truths' must all be maintained separately, adding to project costs.

3) Solution properties: any solution must take a more holistic view of the design process, allowing for cross-domain design and explicitly encouraging centralized configuration management. Some means of reducing multiple design 'truths' or at least better synchronizing different design version must be developed.

D. Verification

1) Description: verification is a complex and wide ranging topic. At its simplest it asks whether the system as implemented meets the specification or not, however much more complex demands are possible. For example for safety critical systems there may be a requirement to demonstrate that the system never moves through a dangerous state or to prove that some system states are unreachable under certain operating conditions.

2) Impact: verification costs are well known to be rising sharply, with some authors maintaining that verification will consume 50-70% or more of project development time [11]. This situation is likely to worsen unless verification methodologies change. Even with automated verification the phenomena of state-space explosion [12] for complex systems will continue to make automation a difficult task and reduce the likelihood of achieving full code and model coverage in testing. Current design processes also tend to find errors late in the development process when they are most expensive to correct [13].

3) Solution properties: some form of verification automation using the system specification is almost certain. This in turn has implications for the form of the system specification as any automated tests must be derived from a more formal description or MoC which may be an executable specification. An executable specification (or the more general use of Model-Based Design) will help with the early detection of errors [14] and is estimated to have a substantial impact on design productivity [4].

E. Tools

1) Description: the development tools used by embedded software developers are much less sophisticated than those used by desktop application developers for example. In general, current IDEs (Integrated Development Environments) are good at low level development, but don't offer more sophisticated development techniques such as refactoring. Furthermore these tools concentrate on just one stage of the design process or on one design domain and there is often limited integration between tools, for example between an IDE and verification tools.

2) Impact: the 'point solution' nature of current design tools prevents coherent tool-chain and workflow integration, for example making cross domain optimization (e.g. rf and digital) difficult or impossible. All of the challenges listed in this paper will require changes in the nature and type of development tools used for embedded software development. Failure of tools to evolve will halt progress.

3) Solution properties: tools will be required to provide high levels of interoperability. This can be achieved at the model content level by the development of interoperability standards. Alternatively, at the numerical interface level, a cosimulation infrastructure may allow the integration of the dynamics modeled by different tools. In response to this challenge there has been a shift in research priorities in the field of modeling and simulation towards the promotion of tool interoperability and more general workflow issues [see for example 15].

III. Key Technologies

A number of technologies under development offer the potential to resolve one or more of the challenges listed above. We have selected a few of the more promising ones to discuss in more detail.

Increasing the level of abstraction has long been touted as

one of the key solutions for coping with complexity [4, 11]. Historically each time the level of abstraction has been raised there have been concerns about the efficiency of the new abstraction levels and about its applicability. Each time these concerns were overcome by a combination of technology advances and improved productivity [16]. Currently, SystemC and more generally TLM are areas of considerable promise for improving design productivity through abstraction and have been extensively discussed elsewhere [17]. Model-Based Design is complementary to TLM and, likewise, holds considerable promise for improving design productivity.

The central idea of Model-Based Design is the use of executable system models as the basis of specification, design, implementation and testing [18]. By systematic model elaboration, an initial design can be captured as a core model that is then gradually extended by increasingly including implementation functionality. In order to allow a single model to be used for multiple purposes, Model-Based Design environments must allow for multi-domain simulation from the same model, e.g. it must be possible to model digital, analog, and rf sub-systems in the same simulation run. This addresses the interdependency challenge while the reduction of development time that results from Model-Based Design [19] addresses the optimization challenge. Model-Based Design has been adopted in some industries for embedded system development, most notably automotive and defense [19-23], where it has met with considerable success, but has not yet seen widespread adoption through the signal processing embedded software community. We are perhaps five years away from such widespread adoption.

One of the key aspects of Model-Based Design is the generation of code from an executable system model. The advantages of code generation are well-known [22], and very similar to the benefits of the extensive use of high-level programming languages with compilers automating the tedious task of obtaining assembly code. Code generators operate faster than humanly possible and can handle optimization complexity that far exceeds human cognitive abilities, without being as error-prone. This is a similar issue in some respects to raising the level of abstraction referred to earlier. Certainly code generation will speed up development, so helping with the optimization challenge.

Increasing the level of abstraction takes a design further from the underlying hardware. Ultimately, however, the design depends on the modality of the hardware selected. To address this challenge, it is contended that some form of formalized automated mapping from abstract model to hardware must be developed. Such a formalized mapping would enable the investigation and control of the impact of hardware and design changes on the overall system. This facilitates choosing the best combination of system design and underlying hardware earlier in the design process. The formalized mapping must contain a description of the underlying hardware and the features it contains, for example a description must include details of the processor speed, memory, cache, and any on-board specialized processing units (e.g. A/D conversion, FFT processors etc.). Any moderately complicated system is likely to consist of heterogeneous hardware and such hardware models will be crucial to deciding which hardware elements should implement which parts of the system functionality.

IV. Vision of the Future

To better illustrate the concepts we have discussed here, we will look into the future into the day in the life of an embedded software development team in fifteen years time.

A home electronics company is producing a security system and needs a visual identity verification sub-system. The entire sub-system must be created from scratch and delivered in under two months. Because development times are now so short for embedded systems, the company has approached two consulting teams to build prototypes. The team with the best prototype wins the contract.

It is Monday morning in Montreal, late into the project, and the team leader of one of the consulting firms opens his mail. The first item is a formal notice from the client that the specification has changed and that the system must now include input from an IR monitor. The e-mail contains a description of the IR data format and details of how it is to be added into the system. The deadlines haven't been extended.

A team meeting is called and the work is split amongst the team of three, the algorithm engineer, the system engineer, and the hardware engineer.

The algorithm engineer searches for the appropriate image processing algorithms to process the IR signal. It takes him several hours, but he finds and buys appropriate algorithms and starts to construct a processing chain in their design tool. After running a number of tests, he believes he has created a working sub-system that will fit the requirements.

Meanwhile the system engineer modifies the design to allow for another monitor input and changes the user interface for the design appropriately. He also updates the executable specification with the new design changes, using as much of the client supplied specification as possible. When the new design is ready, he runs an automated regression test generated from the specification. A few errors are found, but they are quickly corrected.

The previous hardware was at the limits of its performance so the hardware engineer searches for new targets. She finds some hardware which she thinks offers the best price/performance ratio for the expected volume of sales for the system. She downloads the chip specification, an optimization map, and a virtual simulator of the chip. She then constructs a virtual hardware model of the multi-chip system.

The behavioral model is used to generate code for the target hardware. The mapping process maps items in the behavioral model onto co-processors etc. on the underlying hardware using the optimization map supplied by the chip manufacturer and 'hints' created by the team. Multiple different mappings are attempted until one that looks to have the right performance characteristics is accepted.

The generated code is downloaded to the virtual platform running the hardware simulation. Once again a regression test suite is run, which shows up no errors. The new tests

generated from the requirements specification show up no failures, but do indicate a possible cause for concern under some operating conditions. The team notes the issue for later.

The team receives another e-mail from the client correcting a bug in the altered requirement specification. The IR spectrum specification is a little wrong and out by 100nm.

The algorithm engineer makes a minor tweak to the algorithm and updates the design. The design environment reports the impact of the change and indicates which regression tests need to be re-run.

The team generates code for their virtual platform and run the selected regression tests. The tests indicate that with the code change, the system is functioning within specification.

It is five o'clock and the team turn off the lights and leave for the night. It's been a long eight hours since the original e-mail arrived from the client.

V. Conclusion

Embedded software development is likely to remain a challenging domain over the foreseeable future, however there are many technological and methodological developments which may yield substantial benefits in the years ahead. These developments are by no means certain to be successful or to be widely adopted in their entirety, however it is highly probable that the design processes used by embedded software developers will change radically in the next ten to fifteen years.

References

[1] "Chinese Fashion Phones Come to U.S. Market", Wireless Week, March 30, 2007

[2] T. Claasen, "The logarithmic law of usefulness", Semiconductor International, July 1998

[3] P. Belanovic, M. Holzer, D. Micusık, and M. Rupp, "Design Methodology of Signal Processing Algorithms in Wireless Systems", CCCT'03, pp. 288-291

[4] International Technology Roadmap for Semiconductors – Design, 2005 Edition

[5] R. Atkinson, "Project management: cost, time and quality, two best guesses and a phenomenon, it's time to accept other success criteria", International Journal of Project Management Vol. 17, No. 6, pp. 337-342, 1999

[6] A. Jantsch, I. Sander, "Models of computation and languages for embedded system design", IEE Proc. Comput. Digit. Tech., Vol 152, No 2, March 2005

[7] A. Mulpur, "Model-Based Design Begins Paying Off For Signal Processing Design", Comms Design, February 2004

[8] P. J. Mosterman, "Networked Embedded Systems", Proceedings of Beyond SCADA: Cyber Physical Systems Meeting, Pittsburgh, PA, November, 2006

[9] W. Wolf, "A decade of hardware/software codesign", IEEE Computer, April 2003

[10] "Airbus Vows Computers Will Speak Same Language After A380 Delay", Bloomberg News, September 29, 2006

[11] MEDEA+ Design Automation Roadmap, Version 5, July 2005

[12] El-Far I.K., Whittaker J.A., "Model-Based Software Testing", Encyclopedia on Software Engineering, Wiley, 2001

[13] J. Dabney, "Return on Investment for Independent Verification & Validation – Phase 2B Final Report", NASA, 2004

[14] "Executable specifications: creating testable, enforceable designs", Microsoft Research, 2001

[15] SPIRIT Consortium, www.spiritconsortium.org

[16] M. Balcer, S. Mellor, "Exploring the role of Executable UML in Model-Driven Architecture", Addison-Wesley Professional, 2002

[17] F. Ghenassia (ed), "Transaction-level modeling with SystemC", Springer, 2005

[18] A. Behboodian, "Model-Based Design", DSP Magazine, October 2005

[19] G. Hodge, J. Ye, and W. Stuart, "Multi-Target Modeling for Embedded Software Development for Automotive Applications," SAE Paper 2004-01-0269

[20] P. Barnard, "Graphical techniques for aircraft dynamic model development", AIAA Modeling and Simulation Technologies Conference, August 2004

[21] J. R. Ghidella, P. J. Mosterman, "Requirements-based testing in aircraft control design", Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit 2005, San Francisco, CA, Aug. 2005

[22] J. Thate, L. Kendrick, and S. Nadarajah, "Caterpillar Automatic Code Generation," SAE Paper 2004-01-0894.

[23] C. Davey and J. Friedman, "Software Systems Engineering with Model-Based Design," International Conference on Software Engineering, 2007