A Novel Codesign Methodology for Real-Time Embedded COTS Multiprocessor-Based Signal Processing Systems

Randall S. Janka
Georgia Tech Research Institute
Georgia Institute of Technology
Atlanta, GA 30332-0856 USA
1-770-528-3165
randall.janka@gtri.gatech.edu

Linda M. Wills
School of Electrical & Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250 USA
1-404-894-4565
linda.wills@ee.gatech.edu

ABSTRACT

The process of designing large real-time embedded signal processing systems is plagued by a lack of coherent specification and design methodology (SDM). Powerful frameworks exist for each individual phase of this canonical design process, but no single methodology exists which enables these frameworks to work together coherently, i.e., allowing the output of a framework used in one phase to be consumed by a different framework used in the next phase. A specification and design methodology (SDM) known as "Search-Explore-Refine" (SER) was developed by Gajski, Vahid, et al. for an application and technology domain that is different from that of real-time embedded signal processing systems implemented with commercial-off-the-shelf multiprocessing hardware and software. However, due to similarities between the fundamental design objects of these two domains, a new SDM was developed and prototyped based on SER known as the MAGIC SDM. The "tools and rules" of the MAGIC SDM are presented. The MAGIC SDM achieves a high degree of model continuity based largely on its use of standards-based computation (VSIPL) and communication (MPI) middleware.

Keywords

Specification and design methodology, COTS, embedded, multiprocessing, middleware, MAGIC, VSIPL, MPI, MPI/RT.

1. INTRODUCTION

The process of designing large real-time embedded signal processing systems is plagued by a lack of coherent specification and design methodology (SDM). A canonical waterfall design process is commonly used to specify, design, and implement these systems with commercial-off-the-shelf (COTS) multiprocessing (MP) hardware and software. Powerful frameworks exist for each individual phase of this canonical design process, but no single methodology exists which enables these frameworks to work together coherently, i.e., allowing the output of a framework used in one phase to be consumed by a different framework used in the next phase. This lack of coherence usually leads to design errors that are not caught until well into the implementation phase. Since the cost of redesign increases as the design moves through these three stages, redesign is the most expensive if not performed until the implementation phase, thus making the current incoherent methodology costly. This paper shows how designs targeting COTS MP technologies can be improved by providing a coherent coupling between these frameworks, a quality known as "model continuity."

2. SPECIFICATION & DESIGN OF COTS MULTIPROCESSING-BASED SYSTEMS

2.1 COTS MP Technology

In recent years both market forces and technological requirements have been driving the design process to limit hardware options to COTS hardware. In the radar signal processing domain, this means using a PCI or VME chassis containing multiprocessor boards with high-speed interprocessor bandwidth and C language support. Despite limiting the design space to a finite number of hardware options, the design process has still been challenging, given compressing development cycles that increase software development productivity requirements, implicitly requiring that software be portable, so that previous design and development efforts can be reused. This productivity and portability must be achievable without an appreciable loss of system performance.

2.2 CASE Frameworks

A partial response to this design challenge of real-time multiprocessor digital signal processing systems has been the development of different frameworks of tools, such as GEDAE[1], RIPPEN[2], and PGM ACT[3], to provide computer-aided system engineering (CASE) support for system implementation. In particular, these frameworks offer code generation that reduces the complexity of system configuration and communication coding, a quality known as "complexity control." Yet no one single framework or one single language can cover the entire design process. Powerful implementation tools can generate deployable application code, but are weak in capturing

[1] GEDAE–Graphical Entry, Distributed Application Environment from Lockheed Martin ATL.

[2] RIPPEN–Real-time Interactive Programming and Processing Environment from ORINCON.

[3] PGM ACT–Autocoding Toolset (ACT) using the Processing Graph Method (PGM).

requirements and are difficult to use in exploring architecture design alternatives. Some languages, such as MATLAB, are powerful in capturing computational requirements, but do not readily lend themselves to being used for deployed implementations.

2.3 Ideal COTS MP SDM Flow

Ideally, the information flow for specification and design would occur as shown in Figure 1. In this ideal SDM, the executable specification model is passed into the design analysis phase, and the design model in the form of an executable design specification is then passed into the implementation phase, where the physical implementation occurs.

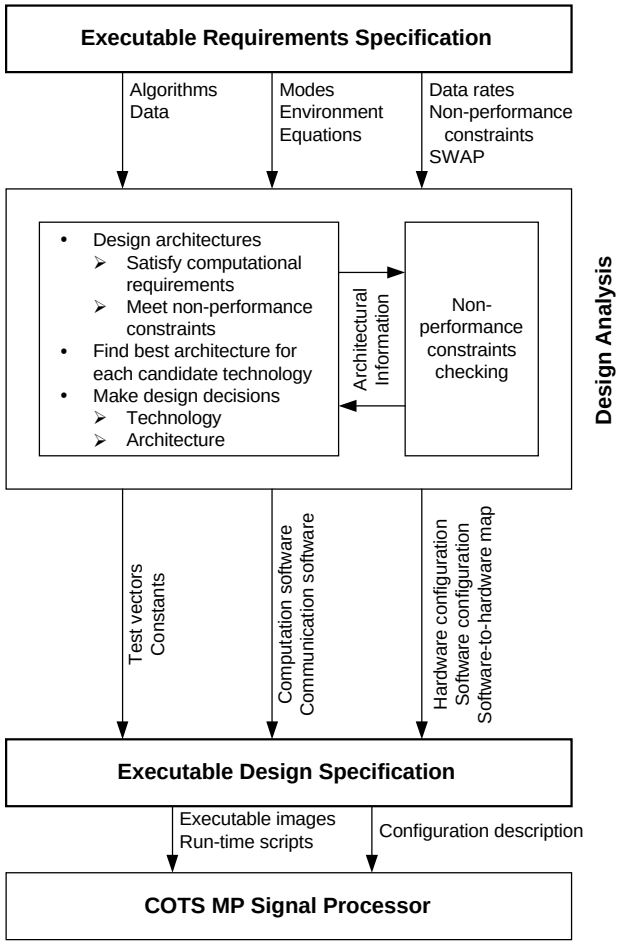

Figure 1. Basic flow of information needed to support model continuity.

2.4 Current COTS MP SDM Flow

Unfortunately, the current state of practice in this domain does not support model continuity, which is illustrated in Figure 2. While the different frameworks described in §2.2 have evolved to support the design and rapid system protyping of signal processing systems using COTS MP technology, they are sophisticated software development environments (SDEs) providing a powerful framework for the codesign of a COTS MP-based system. Codesign in this application and technology domain consists of the following:

- Board-level hardware description capture
- Software process and function description capture
- Software-to-hardware mapping
- Generation of code, makefiles, and run-time scripts

Consequently, despite having powerful frameworks to support codesign, specification and design still suffers from model *dis*continuity as illustrated in Figure 2.

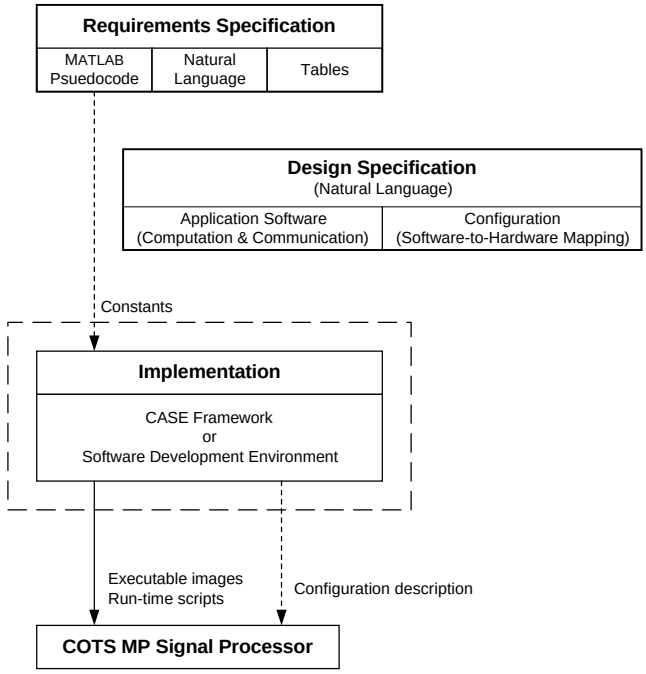

Figure 2. How model continuity is currently lacking in current COTS MP SDM.

3. THE SER SDM OF GAJSKI, VAHID et al.

Gajski, Vahid, et al. developed a hardware/software codesign methodology appropriate to their application domain, along with associated system design language with supporting tools [1, 2], which they called "Search-Explore-Refine" (SER). Their methodology is similar to describe-and-synthesize, except that the requirements specification is captured in an executable language called SpecCharts, which they developed. Their methodology raised the level of abstraction in an attempt to achieve higher productivity. The SER application domain (small-scale reactive and embedded digital systems with at most a single processor) differs from ours (streaming input data transformational radar signal processing applications using parallel and distributed processing). But, the design objects we use are logical extensions of those considered by Gajski, Vahid, et al. See Table 1 (after Figure 1 in [1]) for how we have extended this table to include the COTS MP board-level hardware (in italics) used in our domain. The SER methdology is shown in Figure 3 (after Figure 2 in [1]) along with how we've extended it to our domain ("Board-level SER").

Table 1. Extending SER design representation and abstraction levels to our ADoI (Board Level).

Levels	Behavioral Forms	Structural Components	Physical Objects
Processor	Executable specification(s), programs.	Processors, controllers, memories, ASICs.	PCB's, MCM's.
Board	***Executable specification(s), programs.***	***Components, connections, ports***	***SBC' s, MP boards, I/O boards, high-speed interconnections***

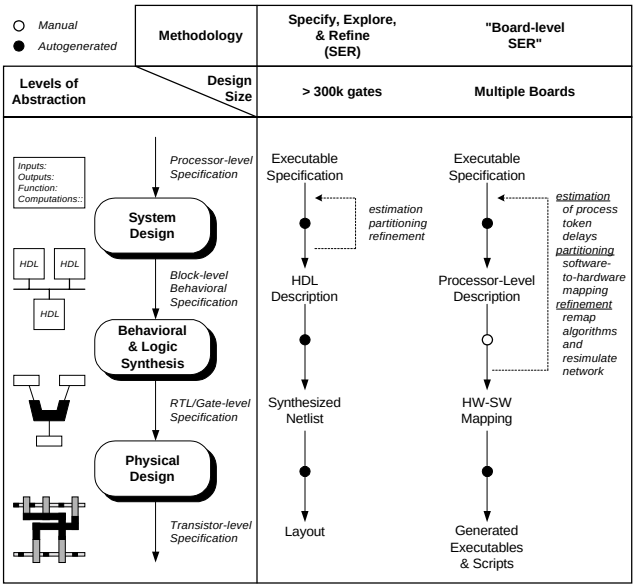

Figure 3. Extending SER to the board-level domain of COTS MP-based systems.

The *specify* phase involves the capture of the system requirements using an executable language. This language should be able to accurately and completely capture the requirements, be easy to understand, and be able to interface to CAD tools in order to support modeling and analysis.

The *explore* phase consists of different mappings of the system functionality to different hardware and software components in an attempt to best satisfy design constraints.

The *refine* phase translates the explore phase decisions into updates in the system specification. Refinement migrates the design from a pure functional spec toward a structural implementation. E.g., behaviors must be added to maintain correct functionality, while defined behaviors may need to be distributed over multiple processors. This requires variables such as data vectors and matrices to be mapped into shared memory buffers and communication protocols to be established, such as defining and assigning semaphores for process synchronization.

4. THE MAGIC SDM

The new SDM that we have developed is the MAGIC SDM, which stands for the "methodology applying generation, integration, and continuity." [3] The origin of this moniker will become clear as we discuss how the MAGIC SDM is used.

Any SDM will start with some human language text requirements specification document. The goal of SDMs is to go from this inexact document to a design and implementation in a manner that minimizes propagation of specification and/or design errors. We do this with an integration of tools guided by sound rules to capture the requirements in a format to make sure there are no conflicts or absence of requirements, then proceed on through a vendor-independent design phase. Without first committing to a vendor, alternate architectures can be considered and an optimum one decided upon. We then take code generated from specification and software-to-hardware mapping determined from design to provide inputs to an implementation framework.

The starting point for specification and design in our ADoI is the set of computation requirements. These are algorithms and data "specified" by MATLAB code, including different scenarios of inputs and their associated outputs. The MATLAB code serves well as an input to a framework that can use it to create an executable specification. The scenarios will provide valuable inputs for the generation of test data to be used downstream in the implementation phase. Communication and control requirements typically refer to data I/O rates as well as the signal processor modes and the control signals that determine the processor's mode (state). Processors in our ADoI have few states; often there are two: one state for initialization and setup ("outer loop") and one state for steady state data transformation ("inner loop"). These modes must be defined and described, preferably in an executable model. Constraints include SWAP, latencies, reliability, and other "illities," which are usually tabulated. It would be useful to have these data encapsulated in a fashion that allows us to include their verification during the specification and design iterations. Recalling how we are extending the SER SDM to our domain with a "Board-level SER" (cf. Figure 3), we now redraw it as our new SDM, as shown in a simplified diagram of the specification and design flow in Figure 4.

A specificiation and design methodology is comprised of "tools and rules," so we now lay out the frameworks (§4.1) and middleware (§4.2) used to implement the MAGIC SDM, and then delineate the rules (§4.3).

4.1 MAGIC SDM Tools

We have chosen the following frameworks to integrate into the MAGIC SDM. We did not choose them because they are perfect; almost all frameworks targeting complex systems are more accurately described not as "frameworks" but as "frame*works-in-progress*." We have chosen the frameworks described in this section because they are well-matched to our application and technology domain, as well as stable commercial products.

For requirements capture and modeling we have chosen the DSP Workstation (DSPW) from The MathWorks as well as Excel and Excel Link (also from The MathWorks). For design exploration we chose eArchitect from Viewlogic, which provides a performance modeling framework with the necessary COTS MP component models. Characterization and features driving the selection of The MathWorks and Viewlogic frameworks are given in the following paragraphs where we discuss these frameworks.

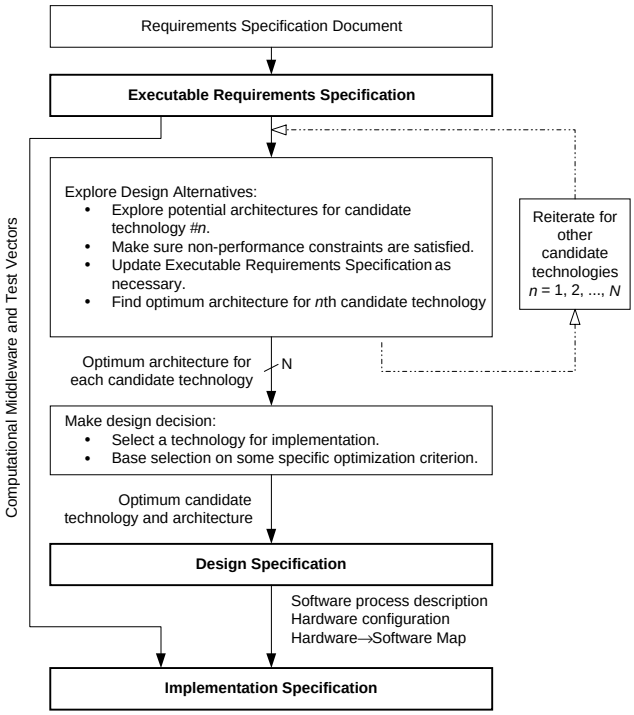

Figure 4. Simplified view of new MAGIC SDM.

4.1.1 DSP Workstation

MATLAB is the de facto lingua franca of algorithm developers, including radar signal processing system analysts. Simulink is a system modeling framework strongly tied to MATLAB, allowing MATLAB expressions to be used explicitly in Simulink blocks. Supplemented by the DSP Blockset, Simulink has become a viable rapid prototyping environment for DSP applications. The Real-Time Workshop (RTW) is the C code generation facility complementing Simulink. It is especially useful because The MathWorks is moving towards adding VSIPL computational middleware support to RTW, an effort we are supporting.

4.1.2 Excel and Matlab Excel Link

Excel provides the framework needed for requirements tabulation and analysis. The Excel spreadsheet is a commodity productivity application familiar to all, and in the same way that many analysis and design frameworks provide support for MATLAB, tabular data-oriented frameworks provide support for Excel. Excel Link is a facility rather than a framework. It is a channel to allow Excel to copy data into MATLAB and execute on it in MATLAB while remaining in Excel. Many requirements are easily "captured" in a spreadsheet, and depending on the sophistication of the computation required to iterate between requirements modeling and design analysis, Excel or MATLAB may be required. Excel Link allows the specifier to remain in a single framework.

4.1.3 eArchitect

Performance modeling was chosen for design exploration and analysis since it supports architectural trade-off analysis without prematurely committing to a given vendor's hardware and software. The COTS performance modeling framework that is best matched to our domain will provide support for the technologies most likely to be used for implementing the signal processor. There are few embedded multiprocessing performance modeling frameworks available commercially. We are only aware of one that supports VME and at least two of the COTS MP interconnection technologies (RACEway and Myrinet), and that is eArchitect from Viewlogic.

4.2 Model Continuity via Middleware

Model continuity is achieved in large part through the use of middleware for computation and communication. Open standards-based middleware supports computation and communication software portability, which means that middleware written for one vendor's hardware should run on another vendor's platform. Consequently, middleware code that constitutes the inner-loop software implementation can be used for different vendors' platforms for design analysis using performance modeling. Critical to making the use of middleware a strong thread of model continuity is the autogeneration of middleware code, since automating the generation of software by a framework that is correct in specification reduces the chance of error in the design and implementation.

Simulink's Real-Time Workshop generates middleware for computation using VSIPL, MPI for communication, and/or MPI/RT for communication and control produces code for both design and implementation. The generated middleware is then used to quantify process delays in the performance model framework and as the core for signal processing implementation application software.

Our reasons for choosing VSIPL and MPI are very similar to our reasons for choosing the frameworks discussed above. They are stated here in order of importance with the most important reason stated first:

- Acceptable performance–These middlewares deliver high-performance because they are tightly integrated with the vendors' computation and communication libraries.
- Standards-based–Since all the COTS MP vendors in our domain space support these middleware and actively participate in their standardization processes, frameworks that generate VSIPL and MPI code will be consumable by all of the hardware vendors' SDEs considered in the design phase.
- COTS–They are now becoming commercially available and therefore stable and supported.

VSIPL is an API supporting portability for COTS users of real-time embedded multicomputers that has been produced by a national forum of government, academia, and industry participants [4]. VSIPL is computational middleware, which also supports interoperability with interprocessor communication (IPC) middleware such as MPI and MPI/RT. Commercial implementations are just now becoming available (early 2000). Earnest consideration by various defense programs as well as other commercial projects is underway and early adoption has begun. The VSIPL API standard provides hundreds of functions to the application software developer to support computation on scalars, vectors, or dense rectangular arrays.

Message passing is a powerful and very general method of expressing parallelism and can be used to create extremely efficient parallel applications. High-performance implementations

of MPI are now available, including implementations for COTS MP platforms. The leading vendor is MPI Software Technology, Inc. (MSTI) who provides high-performance implementations of MPI under the commercial trademark MPI/PRO, including two of the three leading COTS MP vendors in our technology space (RACEway and Myrinet). There is another standards effort underway to specify a real-time version of MPI with a guaranteed quality-of-service (QoS) called MPI/RT [5]. Non-QoS beta versions of MPI/RT are just now (early 2000) beginning to appear.

To generate the steady-state inner-loop middleware-based C code from Simulink, the DSP Blockset is translated into VSIPL or MPI function calls with the arguments determined by the parameters contained in the Simulink blocks. Basically, Simulink "boxes" are transformed into VSIPL computation function calls, while the "arrows" are transformed into MPI communication function calls.

4.3 MAGIC SDM Rules

We lay out here the specification and design rules of the MAGIC SDM. We assume that a natural language (e.g., English) requirements specification document exists that contains the system requirements, interfaces, data rates, etc. We do not assume that the algorithms have been coded in MATLAB, though it would be very unusual for them not to be.

1) Tabulate requirements–Identify and cull details of the requirements from the requirements specification document.
2) Capture non-constraint requirements in an executable model–Describe computation, communication, and control requirements in an executable model.
3) Build executable workbook with requirements–Put all the requirements into a tabular form to facilitate computational manipulation, e.g., in a worksheet/workbook environment such as Excel.
4) Gather benchmarks for tokens–Gather benchmarks of the middleware functions that are likely to be used in design and implementation and enter them into the executable workbook.
5) Explore alternative architectures and technologies–Use performance modeling to explore potential architectures for a given technology, then determine the best architecture for that technology. Repeat as necessary for other candidate technologies.
6) Make design decisions–Decide which technology and architecture to use in implementing the signal processor.
7) Create implementation specification–Pass along architectural details to the system implementation specification based on the design exploration.

5. CONCLUSION

A new specification and design methodology has been developed and prototyped. The MAGIC SDM has been analytically evaluated and benchmarked using a synthetic array radar application. It has been shown to possess a high degree of both model continuity and complexity control [3]. It effectively allows the designer to evaluate candidate architectures and technologies *before* committing to a technology and be confident that an optimum design has been obtained.

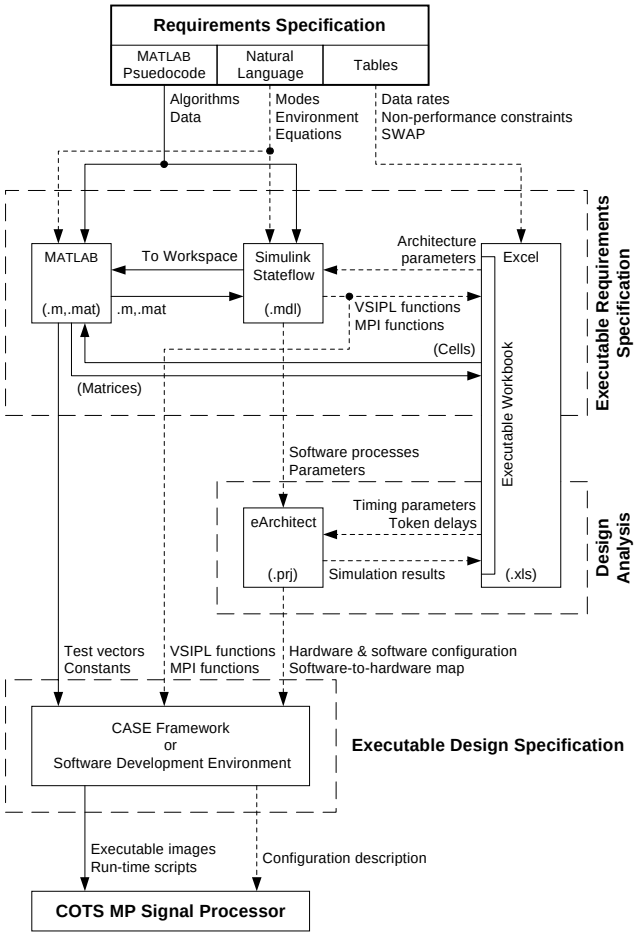

Figure 5. MAGIC SDM information flow and illustration of model continuity.

6. REFERENCES

[1] D. D. Gajski, S. Narayan, L. Ramachandran, F. Vahid, and P. Fung, "System Design Methodologies: Aiming at the 100 h Design Cycle," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, pp. 70-82, 1996.

[2] F. Vahid, S. Narayan, and D. D. Gajski, "SpecCharts: A VHDL Front-End for Embedded Systems," *IEEE Transactions on CAD*, vol. 14, pp. 694-706, 1996.

[3] R. S. Janka, "A Model-Continuous Specification and Design Methodology for Embedded Multiprocessor Signal Processing Systems," a Ph.D. dissertation in the School of Electrical and Computer Engineering. Atlanta, Georgia: Georgia Institute of Technology, 1999, pp. xxiii, 225.

[4] VSIPL Forum, "VSIPL v1.0 API Standard Specification," DARPA and the Navy, Draft http://www.vsipl.org/PubInfo/pubdrftrev.html, 1999.

[5] Real-Time Message Passing Interface (MPI/RT) Forum, "Document for the Real-Time Message Passing Interface (MPI/RT-1.0) Draft Standard," DARPA, Draft http://www.mpirt.org/drafts.html, February 1, 1999.