Heterogeneous Multi-core Systems: UML Profiles vs. DSM Approaches

David McKean
Advanced Fusion Technologies
mckean@aft-worldwide.com

Jonathan Sprinkle
University of Arizona
sprinkle@ECE.Arizona.Edu

Abstract

This paper examines tradeoffs in extending a UML profile, or developing a new DSML for a particular domain. Questions are presented which help to make an objective decision. A case study is presented to address system design in heterogeneous multi-core systems. Since throughput, application-specific data streams, and the desired output executable language are each complicated by design constraints or process constraints, this decision is weighed against timeliness of delivery, and the availability of tools to analyze, design, and validate the solution.

General Terms Design

Keywords Domain-specific modeling, UML profiles, MARTE, multi-core design, high-performance computing

1. Introduction

Model-based design represents the future of systems design, due to the ability of models to scale to complex systems, and the practice to synthesize large portions of a codebase. However, for someone just learning to leverage model-based approaches for their own domain problems, there is a pivotal question that must be answered early in any system's design: *"Do I design a domain-specific modeling language, or take advantage of UML profiles?"* This question is not easy to answer. This position paper aims to describe how some of the tradeoffs can be made, with discussion relating these questions to the domain of multi-core, high-performance computing.

2. Approach

In order to discuss the tradeoffs, the key axes of comparison must be identified. In this paper, we consider the following metrics:

- Effort required to adopt the schema (i.e., profile or language)
- Effort required to design/build models in the adopted schema
- Analysis methods available in the schema
- Output artifacts available from the schema
- Reuse available with the schema

2.1 Effort required to adopt the schema

Effort for adoption includes training on the new technology, as well as expertise necessary to build/adapt the Unified Modeling Language (UML) profile or domain-specific modeling language (DSML). For many domains this question is difficult to answer at the beginning of the design: nonetheless, it is worth considering the following questions:

- Does anyone in the organization have previous experience with profiles?
- Does anyone in the organization have experience with domain-specific modeling?

If the answer to only one question is "yes," then that answer tips the balance in favor of that response. Note that it does *not* necessarily imply that none of the other questions should be answered!

2.2 Effort required to design/build models

The core value of domain-specific modeling is that it elevates elements of the system design to types, thus reducing the effort to design a problem through abstraction techniques. For example, within the context of a heterogeneous application domain model the domain-specific types will include `Device`, `Kernel`, `Program` and `Memory`, and their association rules will constrain how the models may be built.

The UML profile approach will have similar types, but a profile is fundamentally an annotation of existing UML types, even if those types are a subset of UML's types. If the modeler takes into account the UML profile when creating the new models, then it is fundamentally a domain-specific approach. If the UML profile is not known when the models are created, there may be additional effort in constructing the various models prior to annotating them with profile types.

If the number of types is small, then a DSM approach is favored. However, if the number of types is large, then a UML profile has its benefits because those types will actually be implicitly realized as varying objects and associations, not as new types in a UML profile. This is perhaps counterintuitive, but is related to the effort required to develop semantic maps for a DSM approach, and it depends on a UML profile's reliance on existing maps.

It is worth noting, though, that a large number of types, with complex associations, will be difficult to build correctly in a UML profile approach, and leaves open the possibility that design errors will not be caught until later in the lifecycle.

2.3 Analysis methods available

A DSM approach traditionally utilizes analysis methods that are either performed in the framework, or performed on generated system artifacts. This means that some form of code generator or model transformation is required when taking a DSM approach. This fact is not news to those in the DSM field, where it is well-known that the definition of the semantics of a modeling language is by far the highest cost in terms of effort.

A UML profile approach may benefit from the availability of analysis methods that operate just on the state models, sequence models, or class models of the design. However, note that the analysis methods for many profiles still require some sort of model transformation to put the models in an acceptable structure for analysis. Thus, the UML profile technique usually requires some semantic attachment.

The key question here is: what kind of analysis is most important to my output? If an analysis tool already exists but just requires reformatting in order to obtain results, then each approach has about equal effort. If the analysis tool does not already exist, then a DSM approach has an edge, depending on how difficult it is to define the analysis methods, because the (generally) fewer number of created objects will make defining the analysis algorithms easier.

2.4 Output artifacts available

Similar to the analysis question, here the most important question is: what will I do with these models when the design is complete? Modeling approaches are gaining ground because of their ability to synthesize executable systems (or high-valued configuration files) from the models. There is no straightforward way to answer this question, except that if a definition for valuable output artifacts already exists (e.g., an XML schema, or a template for code synthesis), then the specific details required to generate that output artifact will help the modeler understand which approach is more useful.

Generally, DSM approaches are more flexible in how they generate their output artifacts, but UML profiles have an edge if the desired output artifact is stub code, rather than fully functional code, because of the large number of commercial code skeleton generators available for UML tools (e.g., tools by Altova, eUML, Microsoft, AlphaSimple). Note this is most applicable if the existing code generators are suitable.

2.5 Potential for Reuse

By its nature, DSM attacks a specific problem and does not lend itself to model reuse in the same way as UML. Because UML profiles are (typically) annotation models for existing model designs, then portions of a UML design can be used for many different profiles, if that makes sense (semantically). For DSM, reuse is generally obtained when languages are composed (e.g., state models that contain dynamic system models). Even in that case, composition of models only rarely results in straightforward semantic composition.

If reuse of models (to other applications) is important to the problem, then this tips the scale dramatically toward UML profiles. If the design is intended to be used to solve a specific problem, then the reuse question is obviously of less importance

3. Context: Heterogeneous Application Design

The core question addressed by this paper is: "Will the best approach be a UML profile, or a domain-specific language?" In order to concretely discuss how this tradeoff can be made, some discussion of the domain in question is merited. The relevant UML profile that would be evaluated is also described.

3.1 Heterogeneous programming

The past decade has seen the evolution of affordable parallel computer architectures in the Personal Computer (PC) marketplace. This is evidenced by the emergence of multi-core Processors (MCPs) and Massively Parallel Processors (MPPs). A current MCP example is the Intel Core i7-800. It has 4 cores and 2 virtual processors for a total of 8 processor threads [4]. The processors use a Multiple Instruction, Multiple Data (MIMD) architecture, as defined by Flynn's taxonomy [2], where each processor has separate instruction and data access to shared program and data memory.

A current example of a MPP is the Nvidia Kepler GK110, which contains 15 Streaming Multiprocessors (SMX). Each SMX contains 192 single-precision Compute Unified Device Architecture (CUDA) cores [6]. The SMXs use a Single Instruction, Multiple Data (SIMD) architecture, as defined by Flynn's taxonomy [2], where all processors execute the same instructions from a shared program memory but each processor has both a private memory and access to shared data memory. Within the next few years, MCPs will evolve to many-core CPUs (100s of physical processors with 2+ virtual processors) and GPUs will evolve to General Purpose GPUs (GPGPUs). The combination of multi-/many- CPUs with GPUs/GPGPUs will be referred to as a node. Programming in a node environment is referred to as *heterogeneous programming* [1]. Various programming models using a shared memory model have been identified [1, 8] (such as OpenCL, OpenMP, Array Building Blocks (ArBB), etc.) that facilitate implementation of software applications in a heterogeneous node environment.

3.2 Interconnection Networks

In addition, future software applications will require the use of multiple nodes that communicate via an interconnection network [9]. Programming in a multi-node, or cluster, environment is referred to as *hybrid programming*. Hybrid programming models support the concept of a Partitioned Global Address Space (PGAS) using Distributed Shared Memory (DSM) as discussed in [1]. DSM models are built on a message passing parallel programming model, such as the Message Passing Interface (MPI). The PGAS DSM model is partitioned into a runtime layer (e.g. Global-Address Space Networking (GASNet), Aggregate Remote Copy Interface (ARMCI), Kernel Lattice Parallelism (KeLP)) and programming languages that support the PGAS DSM model [1] (e.g. Unified Parallel C, Titanium, X10, Chapel, Fortress).

3.3 OpenCL

OpenCL is an open industry standard for programming a heterogeneous collection of CPUs, GPUs, and other discrete computing devices organized into a single application platform. OpenCL is a framework for parallel programming that includes a language, API, libraries and a runtime system to support software application development [3, 5].

The Platform Model for OpenCL consists of a host connected to one or more OpenCL devices. An OpenCL device is divided into one or more compute units (CUs) which are further divided into one or more processing elements (PEs). Computations on a device occur within the processing elements. An OpenCL application runs on a host according to the Platform Independent Model (PIM) native to the host platform. The OpenCL application submits commands (via a command queue) from the host to execute computations on the processing elements within a device. If constructed, the proposed HAD UML profile will extend the MARTE UML profile [7] to support the OpenCL framework.

3.4 HAD UML Profile

The state of heterogeneous application development in today's environment is uncertain with no less than 20 programming models available for single node (MCP/MPP) computer configurations. However, modern software engineering makes a distinction between the design of a software application and its implementation (i.e. programming). This paper proposes a design approach for heterogeneous applications that uses Unified Modeling Language (UML) [10] semantics for design representation. The UML semantics are collected into multiple Platform Specific Models (PSMs) with each PSM targeting a specific domain area (e.g. OpenCL PSM targets, single node MPPs, openMP targets, single node MCPs, MPI targets, multiple nodes). The PSMs are collected into a proposed Heterogeneous Application Domain (HAD) UML Profile.

All HAD software applications must employ concurrent design and implementation principles. In fact, HAD application design considerations extend many of the design considerations for Real-Time Embedded System (RTES) applications. The Object Management Group (OMG) has defined a Modeling and Analysis of

Real-Time Embedded Systems (MARTE) UML Profile [7] to support model-based analysis and design of RTES applications[1]. The MARTE profile provides semantics to annotate models with information necessary to perform performance and schedulability analyses. It also provides an analysis framework that allows refinement/specialization to other kinds of analysis (e.g. power analysis). The profile provides the following:

- A common method of modeling hardware and software aspects of an RTES application to improve communication among developers.
- Interoperability among tools used for specification, design, code generation, etc.
- The construction of models that enable derivation of quantitative predictions of various real-time design characteristics which consider both hardware and software

The MARTE UML profile defines precise semantics for time and resource modeling, which will enable automatic model transformation to lower abstraction levels (e.g. C++ source code). The HAD UML profile extension to MARTE defines precise semantics for parallel frameworks (e.g. OpenCL, such as in [**?**]) and also enables automatic model transformation to lower abstraction levels.

4. Discussion

With the previously defined metrics in mind, and now a discussion of the context of the question for this paper, we can describe how the answers are traded off.

4.1 Effort required for adoption

In this case, team members have expert knowledge of domain-specific modeling, as well as the MARTE UML profile. No clear advantage goes in either direction here, as one member has expertise in the MARTE UML profile, while the other is a researcher for the last decade in DSM and has produced numerous DSM tools. Regardless of which technology is chosen, an expert will be available.

4.2 Effort required to design/build models

The HAD designs will be compositions of several different PSM models and their structures. This means that a relatively large number of types will be used, but not an excessive number (e.g., perhaps 10-20 types in total). Since a significant amount of the semantics (whether a DSM or UML profile approach is taken) will be achieved through the inclusion of software blocks for existing algorithm implementations, the emphasis for design will be on the correct association of objects when instantiated.

This tips the scale slightly in favor of a DSM approach, because of the ability of DSM to reduce the number of design blocks based on abstractions of the problem space. If models already exist in UML, then the DSM advantage would be lessened or could disappear.

4.3 Analysis methods available

The MARTE profile has significant analysis that already exists for schedulability, performance, and power analysis. Clearly, the DSM approach could require significant effort to duplicate this analysis, so most DSM experts would generate a system artifact that would be ingestible by those analysis tools. If the analysis (within MARTE) takes place inside the modeling tool, this would be problematic for DSM to generate. However, if MARTE profiles also synthesize an artifact, the format of that artifact could be used to guide the design of the DSM generators.

[1] The MARTE UML profile replaces the Schedulability, Performance, and Time (SPT) UML profile

4.4 Output artifacts available

The MARTE profile provides significant output artifacts, and support for the verification/validation stages. However, in order to be used in the HAD designs, additional output artifacts will be needed, and it is not clear exactly how these can be generated. For example, one goal is to use OpenCL as an execution semantics for the generated code, and this is not clearly available from UML's MARTE.

This particular case tips the favor to DSM, because the DSM and UML profile approaches will require some additional artifact synthesis to be designed, and the DSM approach permits fewer models (due to abstraction) which results in more generated code per model built. This implies that the code generator is a bit easier to write, which is true for DSM experts.

4.5 Potential for Reuse

In this case, model reuse (that is, reusing a model for a purpose not intended for this domain) is not important. That is, most models will be built for a particular application, and when a new application is built, those models may be easily reused, whether the approach used is MARTE or a DSM. However, if the models may be used in a completely different domain (say, client/server models), then UML's profiles carry an advantage.

Here, the favor tips again to a DSM approach. When a model is built in UML, it should have both class models, and state models. Those states and classes (if renamed) could have different application. However, in DSM, usually the state models are either fixed for each type (in which case, they are taken care of by the code generator) or are generated based on context. This is an apples-to-oranges comparison to UML's profiling approach, because the approaches are fundamentally different. However, it reinforces that the goals of DSM—which include raising the level of abstraction—frequently contradict with model reuse in a different problem domain.

5. Conclusion

The presented context for discussion gives a slight edge to a DSM approach, since the output artifacts are not readily generated. However, a likely approach could be to generate UML within a MARTE profile from a DSM, and simultaneously generate the OpenCL output from the same DSM model. This can be produced with standard DSM model transformation approaches, and takes (in a sense) the best of both approaches.

This paper has examined how to determine whether a UML profile, or DSM, approach should be considered for a specific problem. While the answers are always dependent on the expertise of those involved, and the specifics of the problem, this paper presents a series of questions to be asked, which will help to weigh the decision before making an effort investment. Finally it is important to note that these questions and decisions are based on the authors' experience. Future work may indicate whether these are a comprehensive set.

This work is supported by the National Science Foundation, under award #CNS 0930919. The authors thank the reviewers for their suggestions to improve the quality of the manuscript.

References

[1] A. W. De Oliveira Rodrigues, F. Guyomarc'H, and J.-L. Dekeyser. An MDE Approach for Automatic Code Generation from MARTE to OpenCL. Rapport de recherche RR-7525, INRIA, Feb. 2011.

[2] J. Diaz, C. Muñoz-Caro, and A. Niño. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on Parallel and Distributed Systems*, 23, 2012.

[3] M. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 23(9), 1972.

[4] B. Gaster, L. Howes, D. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous Computing with OpenCL*. Elsevier, 2012.

[5] D. Marr, F. Binus, D. Hill, G. Hinton, D. Konfaty, J. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1), 2002.

[6] A. Munshi, B. Gaster, T. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. Addison-Wessley, 2012.

[7] NVIDIA. Nvidia's next generation cuda compute architecture. Technical Report Kepler GK110.

[8] Object Management Group. UML profile for modeling and analysis of real-time and embedded systems (MARTE).

[9] T. Rauber and G. Runger. *Parallel Programming for Multicore and Cluster Systems*. Springer, 2010.

[10] A. Varbanescu, P. Hijma, R. van Nieuwpoort, and H. Bal. Towards an effective unified programming model for many-cores. In *IEEE International Parallel Distributed Processing Symposium*, pages 681–692, 2011.