Performance Modeling for Software Integration

Kang G. Shin and Shige Wang
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
email:{wangsg,kgshin}@eecs.umich.edu

Abstract — Software construction with reusable components and legacy code has proven useful for fast and low-cost software development in many real world applications. However, since this methodology is often applied to domains with stringent performance and resource constraints such as embedded systems, its ignorance of non-functional constraints makes it very difficult and expensive to meet the constraints at a post-integration stage.

In this paper, we propose a new methodology for (i) integrating a performance model with functional models during the design phase, and (ii) using the performance model parameters along with functional components throughout all SW development phases that usually rely on different tools. The performance model is constructed by examining the performance requirements and characteristics of functional components and their interaction patterns, and by associating them with functional components and patterns as performance parameters. The advantage of such modeling lies in that the performance model is easily integratable with functional components and can be reused just like reusable functional components. The proposed performance modeling, therefore, solves the cross-cutting issues of performance-aware functional design, and makes the consistent performance information available across the different phases of software lifecycle. The main technical issues for such performance modeling include how to partition the performance model along the function models and software architecture, how to determine and systematically measure the performance parameters of software components in each partition for reuse in constructing different performance models, and representing the performance parameters for use across different tools, and building a performance model for design-time analysis and runtime QoS management.

1 Introduction

New software engineering technologies such as component-based software integration [6] and pattern-oriented software architecture [3] have been used in many real world applications, and proved beneficial for fast and low-cost software development. However, recent practices of using these technologies for performance-constrained applications, such as networked multimedia and embedded controls, have shown that only limited benefits can be gained from using these technologies alone. One of their main difficulties is that these applications have to interact with the physical world, and are, therefore, subject to stringent physical performance constraints. These constraints are traditionally hidden, or abstracted away, and hence left unaddressed by the above-mentioned SW development technologies. Models, design methods, and architectures used in current practices are geared towards functionality with little attention paid to the non-functional constraints; that is, they focus only on functional partitioning, design, implementation and integration. Such a design and integration views performance issues as implementation issues, and addresses them by simulation in a later phase of the development, which may be too late and/or too costly to change the design if the implementation cannot meet the performance requirements. When a component is reused in software integration, only its functionality and the corresponding interfaces are reused. Such reuse will not be feasible if the performance constraints change, which are common for embedded real-time systems. On the other hand, performance analysis is carried out in an ad hoc way, and is normally viewed as an art, not an engineering process [4]. Thus, the results obtained vary dramatically from engineer to engineer, depending on the issues he is interested in, and methodologies used to validate the performance.

All of the above issues require a performance model that can capture non-functional aspects and can be used along with functional components throughout the entire software development cycle from the system requirements specification to system design, implementation, testing, even runtime analysis and reconfiguration. To this end, we propose an approach to modeling system performance of each individual reusable component as a set of performance parameters at each level associated with the corresponding functionalities. The performance parameters are designed to be measurable and stable in a given environment, and are viewed as static

properties of functional components. The performance model of SW integration can then be constructed using the performance parameters of constituent functional components, component execution environments and their interaction patterns in the integration. Modeling performance this way enables a designer to reuse the performance model with functional components/subsystems (by choosing different implementations and interaction patterns for the same functions). Such performance parameters can also be used across different tools in a tool chain despite the fact that the different tools may represent different component functions and interfaces.

The rest of the paper is organized as follows. Section 2 describes the performance model and issues related to its construction, including determination of the performance parameters for each component, measurement of the parameters under some representative environment, and their specification with the components for reuse and analysis. Section 3 discusses software integration using the proposed performance model to meet the application performance requirements. Section 4 describes our previous experiences on performance-constrained software development, and identifies the remaining research issues to complete performance modeling. The paper concludes with Section 5.

2 Modeling Performance for Software Integration

Performance requirements are usually expressed as some functional constraints and properties that will ensure the dynamic behaviors of functions to be correct and timely. So, we can treat the performance model of integrated software as a combination of the performance of individual components used in the integration. We describe the performance of each component with parameters. To reuse the performance information and build the performance model of integrated software with each component's performance, the performance parameters should have the following properties.

- Performance information should be included in the component model as attributes associated with its functionalities. Current designs are basically a process of partitioning end-to-end functions into small pieces, and map them to the functions supported by some components. Existing reusable software components are all designed and implemented based on their functionality, which is a primary factor for component selection and reuse. So, performance information has to be used along with those components that will meet the intended functional requirements.

- Performance information should be expressed in some abstract and general form so that it may be used for different tools in a tool chain throughout the different development phases. As software gets more complicated, it is difficult to use only one tool throughout all development phases. Multiple tools with different underlying assumptions are usually used to design and analyze different aspects of the system. Reusable components in different tools may be represented in different formats, and hence, each needs a different representation of performance information. For example, the timing information in a tool for design may be represented as an end-to-end deadline, while in a tool for implementation it may be represented as a rate of execution.

- The performance model should include each component's performance characteristics (such as its worst-case execution time and memory usage). Such performance characteristics can usually be measured on a platform, independently of its interactions with other software components. The ability of independent/isolated measurement ensures the performance of a component is self-contained and can be reused in different software integrations. The deployment-related parameters of a performance model can be derived through analysis using the component performance characteristics together with their interactions. For example, the blocking time of a component's execution can be derived from the execution time of a component itself and the sum of execution times of other components it has to wait for.

- The performance model should also capture performance requirements. The performance requirements of a component impose constraints (such as deadlines) on the component, which has to be met for the system to behave correctly. Such performance requirements usually depend on the component's context, and varies from configuration to configuration. It is also common that requirements change dynamically during runtime when the execution mode changes. So, unlike the performance characteristics which are fixed for a given platform, component performance requirements should be externally adjustable to achieve different levels of performance.

To build a performance model with all of the above properties, we model components to have not only functional specifications such as their behaviors and interfaces, but also performance specifications, as shown in Figure 1. Note that components can be any reusable elements in software design and implementation, including application models and code, patterns of component interactions and communications, and the underlying infrastructure services (middleware, OS, and network protocols).

In such a structure, each component can perform several functions by accessing the corresponding function interfaces (e.g., method calls and event triggers). Each function has a set of performance parameters. Some of the parameters are static attributes representing the performance characteristics,

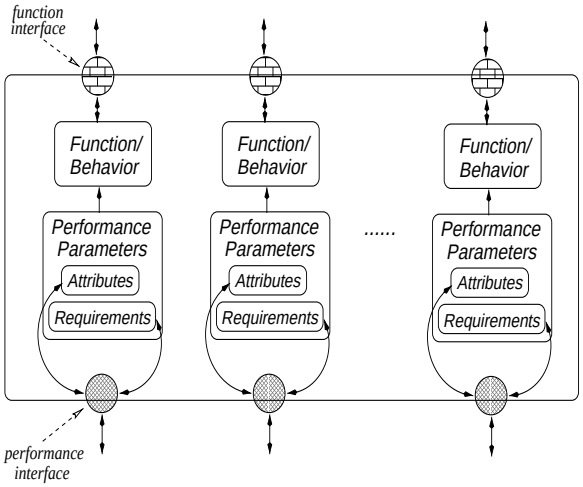

Figure 1: The reusable component structure with performance parameters.

while others are the requirements representing performance constraints. Both can be changed by accessing the corresponding performance interfaces.

An assumption for software integration using this model is the existence of implementation polymorphism, meaning that each functional component should have multiple implementations with different levels of performance. To reuse performance parameters in multiple software integrations, the following issues need to be addressed.

Determine performance parameters. Which performance parameters can be reused and customized, should be determined when a component is designed. Parameters for performance attributes usually depend on the nature of functions and target analysis. For example, latency is a proper attributes for communication components, while execution time is more suitable for computation components for timing analysis. It may also depend on the performance attributes of the underlying support. Parameters of performance requirements usually depend on component interactions and higher-level performance requirements. Since both types of parameters are highly domain-specific, the component designer in a particular domain would be able to determine them based on which products/applications will use the component and how it can possibly be used.

Performance parameter measurement. After determining the parameters essential for constructing the performance model for integration, the associated performance attributes can be measured and reused. The measurements should be taken in the context where the component can possibly be used, including different configurations of the underlying services and possible interactions with the components at the same layer of hierarchy.[1] The parameters should be measured in a systematic manner with representative codes to ensure that the results are reproducible and comparable. Results of these measurements should be represented in a way that the context will be used as a parameter when the performance attributes are derived.

Hierarchical performance model construction. Components are usually integrated hierarchically to reduce complexity. Consequentally, the performance model of integrated software should also be constructed hierarchically using the performance parameters of each component. The performance parameters of a higher-level component can be derived from a combination of performance parameters of all constituent lower-level components, as shown in Figure 2. Existing research [1, 2] has demonstrated that such a hierarchical model can be constructed and will be beneficial, although the proposed models are either too low-level (such as at instruction set level) to be useful in practice, or constructed in an ad hoc way for a target application.

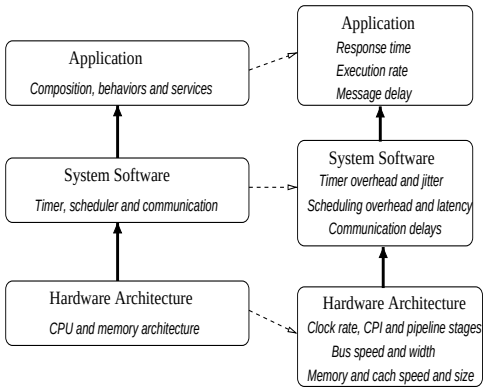

Figure 2: Deriving performance parameters from lower-level components.

Performance parameters for formal verification. Since performance-constrained applications are usually mission/safety-critical applications, formal verification is desired to ensure their correctness. Modeling the performance of components as a set of performance parameters makes it easy to integrate the performance information with formal methods in analysis. The behavioral model of a reusable component can be constructed using a basic model (e.g., continuous or discrete event) with the performance parameters added to it. The verification and analysis can then be done based on the performance parameters of each component. On the other hand, the results proven after verification can also be mapped to proper performance parameters of each component for implementation.

[1] A typical hierarchy consists of operating system, middleware, and application.

3 Software Construction with Performance Model

Performance requirements of software are normally given as the end-to-end performance constraints derived from physical constraints during the system design. For example, computation associated with the sensing of an object flying into an area, determination of its identity as an enemy missile/fighter jet, and launching of an anti-missile should be done within a bounded time *D*. In the current describe-and-synthesis development process [7], software functions are partitioned according to some criteria, and then integrated by choosing proper existing components that perform the desired functionality, and then linking them together. Since there is no performance information associated with components in current component design and implementation, the performance evaluation of the integrated software heavily depends on traditional ad hoc simulations.

Software construction using the proposed performance model in Section 2 along with the describe-and-synthesis process will enable the performance analysis to start as early as in design phase, hence eliminating or reducing the development cost due to error detection in simulation at a later stage. One approach to detecting a performance failure before implementation is to use performance-proven integration. In this approach, after components are selected and integrated to satisfy the functional requirements, the performance model can be constructed, and performance analysis is conducted based on the model to verify the satisfiability of the performance constraints. Once the constraints are all satisfiable, the system can be implemented with selected components, and the integrated software is guaranteed to be correct in both functionality and performance. Otherwise, changes have to be made by either re-selecting components with different performance numbers or re-organizing components using different interaction patterns with different performance figures.

An alternative to software construction using the performance model is performance-directed partitioning and component selection. Instead of partitioning the system purely based on functionality, the partitioning can be done with a combination of both function and performance considerations. Once a subsystem is generated, a set of performance sub-constraints for the subsystem are also generated. Then, components and implementations can be selected depending on whether they will meet the sub-constraints or not. Such an approach may be suitable for applications with pre-defined performance constraints for some subsystems.

In both the software construction approaches, performance parameters are reused. The performance model (or part thereof) is reused as components/integrations are reused, or the same application performance requirements are reused. Performance attributes are reused along with component functions, while performance requirements are reused when substituting a component in the same context. Unlike reuse of performance attributes related only to one component, reuse of performance requirements may imply use of the same software architecture and patterns. With reuse of a partial or whole performance model of integrated software, incremental or differential performance analysis is possible when only part of the software is changed.

Modeling performance with component performance parameters also supports performance-aware software integration with multiple tools as a tool chain. In current complex software development with multiple tools involved in different development phases and used for designing and analyzing different system aspects, performance information should be able to flow from one tool to another without losing interesting/useful information. Many existing research results have shown that a meta-model is required for such a tool chain integration [5]. The performance parameters can be defined in the meta-model and used for different tools.

4 Experiences and Unresolved Issues

Modeling the performance for use with software components for integration is motivated by our previous experience with manufacturing and embedded system SW design. We have developed a reconfigurable software architecture for the open architecture controllers since 1995. Although the results show that object-oriented modeling and separate behavior specification provide the end users more flexibility in controller software design and implementation, meeting the performance constraints was the most difficult and time-consuming issue. More than a half of the development efforts were made on performance verification and tuning in machine tool controller software to ensure timely completion of all critical system operations, even on a resource-rich platform. The root cause of this was the lack of performance information for reusable components; the performance constraints have to be checked in an ad hoc case-by-case way after all functions are implemented. So, an analytic model with systematically-measured performance parameters will help reduce the development time significantly in such software construction. The preliminary results of our ongoing model-based real-time embedded software integration have indicated that a reusable performance model will help accelerate software development. The performance analysis algorithms and methodologies we have developed and integrated into the GME environment, have used some of the performance parameters of reusable components to verify the correctness of design, and support automatic performance parameter value assignments. These performance parameters have also been defined in the meta-model that will be used across tools in an integrated experiment.

Although modeling the performance of components and reusing the model along with the functional components is a promising approach to fast and correct software construction,

there remain several issues that warrant further research. One of them is the need to develop methods for constructing the performance model for integrated software. The difficulty arises from the complex relationships between the components in a real application. Such relationships include all types of interactions of the underlying services and mechanisms, data/information dependencies among the components at the same level as well as between lower and higher levels. According to our experience in modeling OS services, interactions between components can become arbitrarily complicated due to the inter-dependencies among components. Consequently, the performance of a component may have cascaded dependencies with multiple other components. This too will make it complicated to construct in the performance model. Although component interactions usually follow some patterns, the performance of software patterns is neither well-defined nor clear in current research.

Another issue is related to using the performance model with heterogeneous models with different modeling assumptions such as synchronous/asynchronous communication, time-driven or event-driven operation, etc. Depending on the designer's interest, the software can be modeled using an object-oriented model such as UML class diagrams, or a formal behavior model like finite-state machines, or some continuous function blocks such as block diagrams in Matlab/Simulink. Due to the different characteristics and the underlying assumptions of different models, performance parameters may have different formats to be used for these models. Transformation of performance information from one format to another while maintaining the consistency between performance parameters is another issue that needs to be studied.

A limitation of our performance model is that it can only be constructed when all performance parameters are available. The performance analysis and verification for a model with only partial performance information will have almost the same difficulty as the case of no information because a performance model with partial information doesn't help at all in analysis. This implies that some form of implementations have to exist for all components before making the performance analysis, which is not always the case. Although it is much better than ad hoc simulation, performance analysis and verification based on a model with partial information is still desired.

Some other issues such as developing toolkits for systematic performance measurements, representing a performance model for online decision and reconfiguration, and integration of performance modeling with the meta-model should also be studied to capitalize on the performance model.

5 Conclusions

Performance-aware software development is critical for future software construction in many application domains, such as distributed embedded systems and real-time control systems. The software development will be accelerated if the performance of product can be specified and analyzed during early phases of development, and the information can be passed between different development stages that use different tools. We propose performance modeling with the parameters associated with functional components and patterns in order to achieve early performance analysis and reuse in software integration. The performance model of integrated software using the proposed approach can be constructed hierarchically at different function levels with systematically-measured performance parameters of components. The model can facilitate both early phase performance analysis and component selection & integration. Although our experiences have shown that such a performance model would accelerate software development without losing software quality, further research needs to be done on such issues as performance for different software patterns, information transformation among heterogeneous models, and model construction with partial performance information.

References

[1] K. Bradley. *A framework for incorporating real-time analysis into system design processes*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, December 1998.

[2] A. B. Brown and M. I. Seltzer. Operating system benchmarking in the wake of lmbench: A case study of the performance of netbsd on the intel x86 architecture. In *Proceedings of the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–224, June 1997.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[4] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[5] G. Karsa, *et al*. Model-integrated system development: models, architecture and process. In *Proceedings of the 21st Annual International Computer Software and Application Conference (COMPSAC'97)*, Bethesda, MD, August 1997.

[6] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Publishing Company, 1997.

[7] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification aand Design of Embedded Systems*. Prentice Hall, 1994.