A Simulation-based Software Design Framework for Network-Centric and Parallel Systems

Hussain M. J. Almohri Osman Balci

Department of Computer Science
Virginia Polytechnic Institute and State University (Virginia Tech)
Blacksburg, Virginia 24061, U.S.A

almohri, balci@vt.edu

Abstract

In this paper we discuss a software design framework that is capable of realizing network-centricity and the rising multicore technology. Instead of producing static design documents in the form of UML diagrams, we propose automatic generation of a visual simulation model, which represents the target system design. We discuss a design environment that is responsible for the generation and execution of the simulation model.

Categories and Subject Descriptors
D.2.10 [*Software Engineering*]: Design

General Terms Design, Verification

Keywords Design, simulation, modeling, network-centric, parallel processing

1. Introduction

One purpose of software design as a process within the software development life cycle is to develop a conceptual design model representing the software. The resulting design model is intended to enable verification and validation of the design as well as planning for the corresponding code-level implementation.

The verification and validation are essential to assess the accuracy of the design. We need to be able to refine existing requirements by uncovering new requirements and modify (or remove) the existing ones. In addition, it is important to check the design for its feasibility, perform various analyses and test for possible software faults, failure and malfunction.

It is crucial to plan for an accurate code-level implementation at the design stage. Since a software design model is an abstract representation of the target software system, it must provide design decisions for every aspect of the software. That does not mean interference with the implementation of the design by providing very low level details of the software system. However, the design model has to be comprehensive and expressive enough and yet remain high-level, abstract and mostly platform independent.

By examining the existing approaches to software design, we have realized the lack of expressiveness and the ability to perform verification and validation on the design documents. Some of the existing research has addressed the idea of executable software design. In particular, the research in this area has centered on UML [22], the widely used object-oriented software modeling language. However, we believe that to perform a useful execution of the software design, there is a need for a design framework that can realize issues related to the execution from early stages of the design.

In fact, we observed that the existing modeling languages are not expressive enough to address the needs of today's complex software systems. Most large and complex systems need to be distributed and thus become a network-centric system. In addition, we have seen the rapidly increasing number of cores on commonly used computing platforms. Therefore, in order to utilize the available distributed and parallel computing power, we need to have an expressive and detailed design framework that can provide the required abstractions to design such systems.

In our ongoing work, we propose the development of a Design Framework (DF) from which we can automatically generate a corresponding visual simulation model representing the software design. We envision a DF that is capable of addressing the needs for network-centric and parallel software systems. These two elements of our DF will assist in enabling verification and validation of the design as well as planning for an accurate code-level implementation of the design.

Throughout this paper we discuss some of the ideas related to the Design Framework, a survey of related work and our plans for future work.

2. Design Framework

The Design Framework is defined as an underlying structure and organization of ideas, which constitute the anatomy and basic skeleton that, guide a software designer in representing a network-centric and parallel software-based system in the form of a visual simulation model.

We do not tie the DF to a specific software design paradigm or an existing modeling language. Instead, using the DF a designer shall be able to use major software design paradigms such as Procedural, Object-Oriented, and Service-Oriented, to design the target software system.

In our DF, we define a Conceptual Construct (CC) as a design-level concept, which represents an abstract element of software systems that is independent from a specific paradigm or a platform. In this section we introduce some of the conceptual constructs that are fundamental as well as those related to parallel and network-centric systems.

2.1 Fundamental Conceptual Constructs

The fundamental CCs form a set of basic CCs that can be used to describe a generic software system. The fundamental CCs provide a basis for other kinds of CCs such as those for Parallel and Network-Centric.

Within the fundamental CC, we define a *block* as an abstract CC that represents a software component, module, service, or class. A block can be *simple* or *nested*. A nested block consists of sub-blocks that are designed to be as part of their container block. The sub-blocks of a nested block are not able to directly interact with outside world. They are special blocks that assist the container block in accomplishing its tasks. Thus, the container block can always control the behavior of its sub-blocks.

A simple block cannot contain sub-blocks. In Object-Oriented terms, a simple block is similar to a class without having other classes defined as part of it.

A block can be executable or non-executable. Those blocks that define their *execution logic* can be executed during the simulation runtime. The non-executable blocks can also be part of the simulation runtime when the instantiation of these blocks is simulated.

Blocks can participate in *block interactions*. Blocks can interact as part of a service request and response mechanism. For instance, a block can request invocation of particular execution logic as a service from another block. This way a Remote Procedure Call, a Remote Method Invocation or a local procedure call can be modeled. A list of parameters is associated with each interaction established between two blocks. The parameters include the requester and the responder, the requested execution logic, and the synchronization method. Block interactions are specified without being part of an execution context. Instead, the specified interactions can be put into action as part of execution logics within various blocks as far as the interaction requirements are fulfilled.

2.2 Multicore Parallelism

Our literature review indicates that, except for a few trials [14], the research community has paid little attention to parallelism at the design stage. We believe parallelism is a problem to be dealt with at both the design and the implementation stages of software development life cycle. That is because there are design decisions that have to be made on the way parallelism takes place within the software execution.

With today's rising multicore technology, the designers of complex software systems need to be able to appropriately design the software with utilization of the multicore technology. It is important to be able to model physical cores, their availability, and the way various software modules would use them.

The Design Framework defines CCs for a number of physical hardware resources that are associated with the execution of a software system. These CCs are *machine*, *processor* and *core*, which abstract physical machines, processors and cores respectively. The level of abstraction here is determined by the specification needs of the software. As designers, we need to be able to define new machines, processors and cores to be selected as execution platforms for specific executable blocks.

The designer can direct the execution to a specific core or a group of cores on a particular machine. For example, a block can be designated to run on a specific core on a specific machine. Alternatively, the designer can choose to run the class on a machine without specifying the core. This is important in the case that explicit parallelization is not important at the conceptual level.

Another element of parallelization in our framework is the *availability*. For example, the designers can indicate the level of availability of a core to its execution block. An execution block can occupy a core all the time, or it can occupy some parts of it for a period of time. Therefore, we also add a usage *capacity* to each core to indicate how much of core's capacity is available to a particular execution block.

2.3 Network-Centricity

Our Design Framework needs to capture basic elements of a network-centric software system. Network-centric systems are not limited to a collection of distributed (and perhaps homogeneous) software systems. Network-centric software can be thought of as a network of heterogeneous software systems organized under what is called a system of systems [1]. The Design Framework focuses on the distributed nature of network-centric software systems.

We include abstractions to enable high-level design of the networking functionality needed by the system. The choice of network protocol, service requests and responses and the connectivity between software blocks are among essential networking abstractions.

We abstract *geographic regions* to be used as part of the design. Machine cores as defined in Section 2.2 can be assigned to different geographic regions. This makes it necessary to include the networking abstractions discussed earlier. A software block uses the networking abstractions to model a communication channel established between the *distributed blocks*. A distributed block is an execution block that resides on two or more geographically dispersed cores.

The Design Framework defines synchronization constructs that can be used to specify the synchronization mechanism of an interaction between two distributed blocks. For example, an interaction of two distributed blocks can be set to asynchronous. At simulation runtime, we will simulate the appropriate behavior corresponding to an asynchronous request. Further, this request can be set to a timeout limit or can be set to unlimited timeout. In this case, it is important to note the possibility of race conditions and deadlocks. As part of our tool implementation, we aim to provide a design analysis tool that can capture such conditions before the design model is ready for simulation.

3. Integrated Visual Simulation-based Design Environment

In order to provide the required tools for software design using our DF, we propose an Integrated Visual Simulation-based Design Environment (IVSDE). Within IVSDE, there are a number of tools (as depicted in Figure 1) that assist the designer in producing the final work product. Guided by the DF, a designer uses a tool called Visual Software Designer to describe the design. This tool uses a graphical modeling language based on the DF with the appropriate GUI to capture the design. Once the designers specified the design, it will be fed into the next tool called Design Analyzer. The tool's task is to perform static analysis on the design and look for possible structural errors. That is related to the appropriate use of the graphical modeling language as well as the integration of the design in general. At this stage we also look for race conditions and possible deadlocks that may appear as a result of using network-centric and parallel conceptual constructs.

When various kinds of analyses are performed on the design, if the design is approved and free of errors, we proceed with the design towards Visual Simulation Model Generator. This tool uses automated code generation techniques to produce the executable visual simulation model. We aim to have minimal designer interference in this process to make it more usable.

At the final stage, the executable visual simulation model is given to the Software Design Simulator (SDS). This tool is our simulation runtime. It uses designer specified parameters to run the simulation model. The simulation will be in a visualized format. The designer can see the flow of data and the execution of the specified logic during the runtime. Further, the designer can ask the SDS to provide current system information at specific checkpoints provided with a pause or continue option. This enables a convenient way of debugging the model at the runtime.

Using the SDS, the designers can perform dynamic analysis on the design and further refine it as needed. Among possible analyses are:

- "What if" analysis
- Feasibility of the design
- Correctness and completeness of the design
- System performance examination

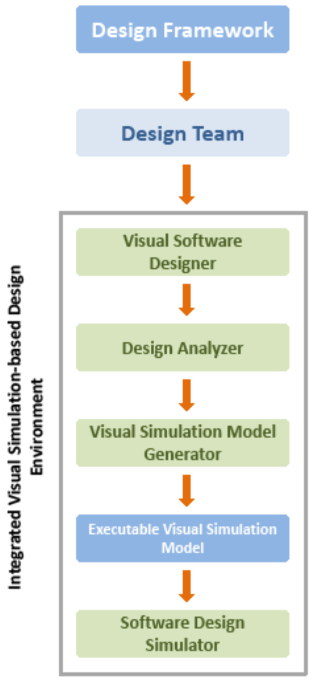
Design Framework
Design Team
Integrated Visual Simulation-based Design Environment
Visual Software Designer
Design Analyzer
Visual Simulation Model Generator
Executable Visual Simulation Model
Software Design Simulator

Figure 1. High-level view of IVSDE.

4. Related Work

Existing work on simulation and execution of software design is centered on UML diagrams. Some of previous works have discussed executable UML diagrams [4–6, 10–13, 15, 20, 25]. In fact, Object Management Group describes execution semantics for a subset of UML (called fUML) that is considered to be generic enough [21]. Working on enabling execution out of UML diagrams is valuable due to wide usage of UML diagrams as an industry standard in the area of software design.

Another interesting modeling language is called Coloured Petri nets [16] (a variation of the Petri nets [23]). Coloured Petri nets are used to model concurrent and distributed systems and can be simulated [2, 19]. There has also been an attempt to translate UML 2.0 state diagrams into Coloured Petri nets [9].

Fundamental Modeling Concepts (FMC) provides a complement to UML for modeling concurrent and distributed aspects of a system [18, 24]. It introduces the idea of virtual locations and agents interacting with a system. A separation of behavior and compositional structure is emphasized in this work. However, we believe that for the purpose of modeling parallelization and providing a rich executable environment, UML and FMC do not provide the needed ground.

Other works explicitly mention simulation and modeling of UML diagrams [3, 8, 17]. Similar to execution of UML diagrams, simulation of these diagrams can be quite valuable and provide a basis for our framework.

Behavior tree is a modeling formalism, which automates the process of generating a software design out of requirements expressed in natural language [7]. These trees can also be simulated.

5. Future Work

This work is part of a broader project for developing a complete design framework with automatic simulation capabilities. We are trying to improve the design framework to include more necessary design elements and support it with a clear formalism.

In our research, we plan to focus on supporting parallel software design realizing the multicore technology. We also plan to emphasize on defining appropriate abstractions to capture the network-centricity, which is an essential part of today's complex systems.

Our objective is to have a design framework that is capable of acknowledging widely used design methodologies. That helps the framework to be applicable in a wider range of applications.

The IVSDE is at its conceptual development stage. We have not yet developed a prototype to show the actual capabilities of our framework and our vision in terms of the way a design model will be executed. Our target is a prototype tool that can generate visual simulation models and provides complete support for our design framework.

References

[1] Balci, O. and W. Ormsby (2006), "Quality Assessment of Modeling and Simulation of Network-Centric Military Systems," In *Modeling and Simulation Tools for Emerging Telecommunication Networks*, Springer-Verlag, Berlin, Germany, 365-382.

[2] Breton, E. and J. Bzivin (2007), "Towards an Understanding of Model Executability," In *Proceedings of the 2001 International Conference on Formal Ontology in Information Systems*, ACM, New York, NY, pp. 70-80.

[3] Campbell, L., B. Cheng, W. McUmber and R. Stirewalt (2000), "Automatically Detecting and Visualizing Errors in UML Diagrams," *Requirements Engineering Journal 7*, 264-287.

[4] Crane, M. and J. Dingel (2008), "Towards a Formal Account of a Foundational Subset for Executable UML Models," In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, Springer-Verlag, Berlin, Germany, pp. 675-689.

[5] Curtis, D. (2006), "SPARK Annotations Within Executable UML," In *Proceedings of the 11th Ada-Europe International Conference on Reliable Software Technologies*, Springer-Verlag, Berlin, Germany, pp. 83-93.

[6] Dobrzanski, C. and L. Kuzniarz (2006), "An Approach to Refactoring of Executable UML Models," In *Proceedings of the 2006 ACM Symposium on Applied Computing*, ACM, New York, NY, pp. 1273-1279.

[7] Dromey, R. G. (2003), "From Requirements to Design: Formalizing the Key Steps," In *Proceedings of the 1st International Conference on Software Engineering and Formal Methods*, IEEE Computer Society Press, Washington, DC, pp. 2-11.

[8] Ermel, C., K. Holscher, S. Kuske and P. Ziemann (2005), "Animated Simulation of Integrated UML Behavioral Models Based on Graph Transformation," In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE Computer Society Press, Washington, DC, pp. 125-133.

[9] Fernandes, J. M., S. Tjell, J. B. Jorgensen and O. Ribeiro (2007), "Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net," In *Proceedings of the 6th International Workshop on Scenarios and State Machines*, IEEE Computer Society Press, Washington, DC, pp. 2-11.

[10] Flint, S., H. Gardner and C. Boughton (2004), "Executable/Translatable UML in Computing Education," In *Proceedings of the 6th Conference on Australasian Computing Education*, ACM, New York, NY, pp. 69-75.

[11] Fuentes L. and P. Sanchez (2007), "Towards Executable Aspect-Oriented UML Models," In *Proceedings of the 10th International Workshop on Aspect-oriented Modeling*, ACM, New York, NY, pp. 28-34.

[12] Fuentes L. and P. Sanchez (2008), "Execution and Simulation of (Profiled) UML Models Using Populo," In *Proceedings of the 2008 International Workshop on Models in Software Engineering*, ACM, New York, NY, pp. 75-81.

[13] Golfarelli, M. and R. Stefano (2008), "UML-Based Modeling for What-If Analysis," In *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*, Springer-Verlag, Berlin, Germany, pp. 1-12.

[14] Gorton, I., J. P. Gray and I. Jelly (1995), "Object-Based Modeling of Parallel Programs," *IEEE Parallel and Distributed Technology: Systems & Applications 3*, 52-63.

[15] Hansen, H, J. Ketema, B. Luttik, M. Mousavi and J. van de Pol (2010), "Towards Model Checking Executable UML Specifications in mCRL2," *Innovations in Systems and Software Engineering 6*, 1, 83-90.

[16] Jensen, K., L. M. Kristensen and L. Wells (2007), "Coloured Petri Nets and CPN Tools for Modeling and Validation of Concurrent Systems," *International Journal on Software Tools for Technology Transfer 9*, 213-254.

[17] Ji, Y., K. H. Chang and P. O. Bobbie (2004), "Interactive Software Architecture Design with Modeling and Simulation," In *Proceedings of the 42nd Annual Southeast Regional Conference*, ACM, New York, NY, pp. 305-306.

[18] Keller, F. and S. Wendt (2003), "FMC: An Approach towards Architecture-Centric System Development," In *Proceedings of the 10th IEEE International Conference and Workshop on Engineering of Computer-Based Systems*, IEEE Computer Society Press, Washington, DC, pp. 173-182.

[19] Kristensen, L. M., S. Christensen and K. Jensen (1998), "The practitioner's Guide to Coloured Petri Nets," In International *Journal on Software Tools for Technology Transfer*, Springer-Verlag, Berlin, Germany, pp. 98-132.

[20] Mooney, J. and H. Sarjoughian (2009), "A Framework for Executable UML Models," In *Proceedings of the 2009 Spring Simulation Multiconference, Society for Computer Simulation International*, San Diego, CA, Article No. 160.

[21] Mellor, S. J. and M. Balcer (2002), "Executable UML: A Foundation for Model-Driven Architectures," Addison-Wesley, Boston, MA.

[22] Object Management Group (2010), "Introduction to OMG's Unified Modeling Language (UML)," http://www.omg.org/gettingstarted/what is uml.htm.

[23] Reisigs, W. and G. Rozenberg (1998), "Informal Introduction to Petri Nets," *Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science 1491*, 1-11.

[24] Tabeling, P. (2002), "Multi-level Modeling of Concurrent and Distributed Systems," In *Proceedings of the 2002 International Conference on Software Engineering Research and Practice*, CSREA Press, Las Vegas, NV, pp. 94-100.

[25] Waheed, T., M. Iqbal and Z. Malik (2008), "Data Flow Analysis of UML Action Semantics for Executable Models," In *Proceedings of the 4th European Conference on Model Driven Architecture Foundations and Applications*, Springer-Verlag, Berlin, Germany, pp. 79-93.