Modeling resources in a UML-based simulative environment

Hany H. Ammar, Vittorio Cortellessa, Alaa Ibrahim
Computer Science and Electrical Engineering Department
West Virginia University, Morgantown, WV, 26506-6109
email: {ammar,vittorio,Ibrahim}@csee.wvu.edu

Abstract

The importance of early performance assessment grows as software systems increase in terms of size, logical distribution and interaction complexity. Lack of time from the side of software developers, as well as distance among software model notations and performance model representation do not help to build an integrated software process that takes into account, from the early phases of the lifecycle, non functional requirements. In this paper we work towards filling this gap by extending the capabilities of a simulative environment developed for the UML notation. Out intent is to introduce new stereotypes representing performance related items, such as resource types and job dispatchers. They allow the software designers to homogeneously represent a software architecture integrated with a running platform as well as parameterized with the resource demand that the components require.

Keywords: Unified Modeling Language, Rose Real Time tool, performance model simulation, resource modeling.

1. Introduction

The validation of non functional requirements early in the lifecycle is an as important as difficult task to accomplish. Early performance assessment mostly allows to build, from the very beginning, software that better fulfills performance requirements, hence helps to reduce the risk of late rising of poor performance that would be hard to manage.

In the last few years Unified Modeling Language (UML) rapidly emerged as a standard notation for software modeling. Its success mostly relies on few elementary characteristics: different diagrams are provided (in an integrated framework) to represent the software model from different viewpoint, so explicitly specifying software aspects elsewhere hidden; the language is supported by a graphical representation, easy to use; no standard software development process is coupled to the notation, thus software designers may decide to use whatever subset of diagrams better fits its application requirements, and organize an application oriented software process.

The necessity to provide a standard representation of information related to the performance (e.g., resource demand) in the UML framework is therefore ever more severe [24]. As a consequent step, this would makes easier to transfer UML models from design to performance analysis tools [17]. Besides, UML was explicitly born as an "open" project [24], with the potential of embedding additional notations and tools to satisfy specific design requisites. Along this trace, the joint effort of Rational Software (the UML originator) and ObjecTime Limited (the Real-Time Object Oriented Modeling originator) has produced *Rational Rose Real Time* (RRT), an environment to run simulations of UML specified models, which is based on the UML-RT notation [25].

Several approaches for extending the UML notation to embed performance related information have been recently introduced.

In [16] Sequence Diagrams are considered and a simulation tool prototype(based on them) is presented. The re-

sulting simulation consists of an animated Sequence Diagram as a trace of events. A similar approach is presented in [1] where a simulation framework, SimML, is used to generate simulation programs from Class and Sequence Diagrams along with some random and statistics information.

In [5, 6] has been proposed the use of a Collaboration Diagram with State Diagrams of all possible objects embedded within them. All possible behaviors of the system are captured. Starting from this new combined diagram a Generalized Stochastic Petri Net model is generated. The direct generation of a continuous time Markov chain starting from Collaboration and State Diagrams is also investigated in [6] through a simple example.

In [10, 11] the extension of the UML notation to performance annotations (pa-UML) has been proposed to deal with performance of software systems. A set of transformation rules is then given to obtain Generalized Stochastic Petri nets from pa-UML diagrams. Performance indices are derived from classical analysis techniques.

A framework that allows UML diagrams to be used for building performance models is presented in [4]. Performance modeling is carried out basing on a precise textual notation, called Performance Modeling Language, to represent the UML characteristics relevant to performance models. These UML based performance models are then transformed into stochastic queueing networks with simultaneous resource possession.

A different type of performance annotation on UML diagrams is carried out in [3]. In this paper the component interconnection patterns of client/server systems are investigated (to derive performance information) by use of Class diagram and Collaboration Diagrams. These UML diagrams are annotated using an XML-type notation with parameters related to workload and service demand. A queueing model is then derived and analyzed to obtain the performance indices of interest.

Performance models and UML diagrams are also the topic of [2], where a methodology is proposed (and applied to distributed systems in [12]) to derive a queueing network based performance models from a set of UML diagrams. This methodology makes use of: the UML Use Case Diagram, the Sequence Diagrams and the Deployment Diagram. This set of diagrams is integrated with a set of parameters derived from the designer experience to obtain a quite accurate performance model.

Tailoring the derivation of a performance model on a specific application domain, such as Client-Server systems, is the goal of [9], where a methodology is introduced to make the distance between software developers and performance analysts shorter.

The derivation of performance models, based on Layered Queueing Networks (LQN), using graph transformation is presented in [13, 14, 15].

In this paper we exploit the simulative potential of the RRT tool to run software models that include items and parameters related to the performance of the model. The set of stereotypes that the UML-RT tool provides has been extended. The extension aims at building (a library of) new stereotypes that allow the representation of resource related items (such as CPUs, disks, etc.), in order to integrate in the same scheme the software structure and the resource requests of a software product.

2. An idea of UML-RT notation extension

UML-RT notation [25] essentially comes from merging the *Real-Time Object Oriented Modeling* (ROOM) [18] and the UML basic notations. ROOM models are intended for simulating the application execution scenarios and the complex object behavior. With the support of UML State Diagrams, executable design models are obtained, and simulation allows to infer their real-time properties, such as deadlines and scheduling constraints.

Three principal constructs are used to explicitly describe a software architecture, that are: Capsules, Ports and Connectors (their name are self-explaining their roles). Dynamic behavior is modeled by using Protocols and State Machines. A Protocol specifies the desired behavior over a connector. A State Machine specifies the internal behavior of a capsule, with the communication capability. ([1]).

A typical early model of a software product is known as a software architecture, that is essentially a graph whose

[1] This capability is achieved by explicitly introducing statements (e.g., *send* and *receive* primitives) in state transitions, such that those transitions may have an additional remote effect of sending a message and, therefore, firing a state transition in a different State Diagram.

nodes represent software components and arcs represent software connectors. In order to provide the potential to represent the same software at different levels of detail, a software architecture can be hierarchically structured.

UML notation does not explicitly provide a diagram to describe a software architecture, which is in fact not necessary. The RRT tool allows to build a diagram of components and connectors, where each component is represented by a capsule with ports to which connectors are associated to exchange messages with other capsules. A hierarchical structure is also provided to this software architecture representation, by allowing to detail the internal structure of a capsule with other capsules and connectors (see figure 1).

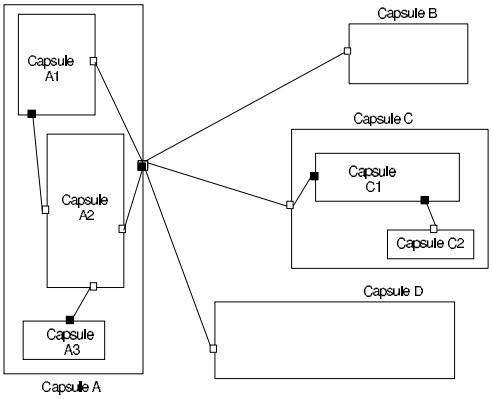

Figure 1. Capsules and connectors in RRT.

The simulative nature of the RRT tool requires as a minimum, in order to run such a scheme, a dynamic description of the behavior of each capsule belonging to the lowest levels of the hierarchy, that is each capsule that does not contain other capsules. This is achieved by providing State Diagrams of capsules, that follow the classical UML notation for State Diagrams.

In order to represent in the same capsule diagram the software architecture and the resources that the software components require, we have conceptually split the diagram in two sides: the *software* side and the *resource* side (see figure 2). Capsules are in both sides, but while the ones in the software side represent software components, the resource side capsules represent the resources that the considered architecture may need.

The strength of the RRT tool consists of using the dynamic description of the internal behavior of each capsule (i.e., the State Diagram of a capsule) to simulate the whole software behavior. This feature has been used for verification and validation of timing constraints [20], reliability analysis of component based software [21], dynamic complexity and dynamic coupling analysis [22], and architectural level risk assessment [23].

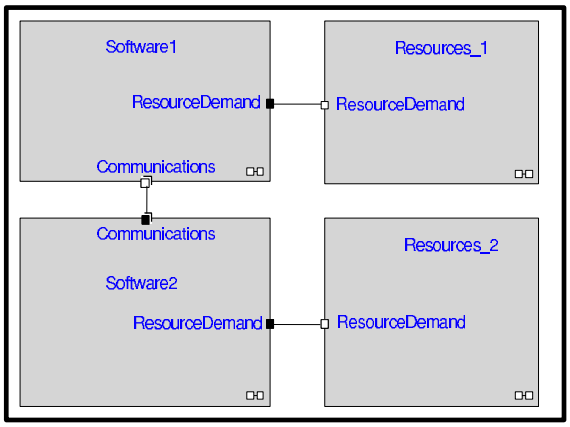

Figure 2. Two-sides capsule diagram.

Upon the extension of the software architecture illustrated by the scheme in figure 2, a properly parameterized simulation of such scheme allows to evaluate the performance of the combined software architecture/resource system. To achieve this objective we have built a basic structure of the resource side of the scheme, and we have started to provide standard capsule stereotypes to be used in the resource side.

In the upper side of figure 3 the capsule diagram of the basic structure that we propose for the resource side of the scheme has been drawn. This basic structure is intended to be used, as it is, wherever a resource side is necessarily to be coupled to a software side. So, for example, the capsule diagram represents the internal structure of both resource sides of figure 2, namely *Resource_1* and *Resource_2*. It is basically composed by a *Main Dispatcher* and a set of resource types.

The main dispatcher is the capsule ([2]) in charge of receiving resource requests from the software side. We suppose (like in a Software Performance Engineering approach

[2]From now on "capsule" and "component" are synonyms, even if the former is mostly a notation related name.

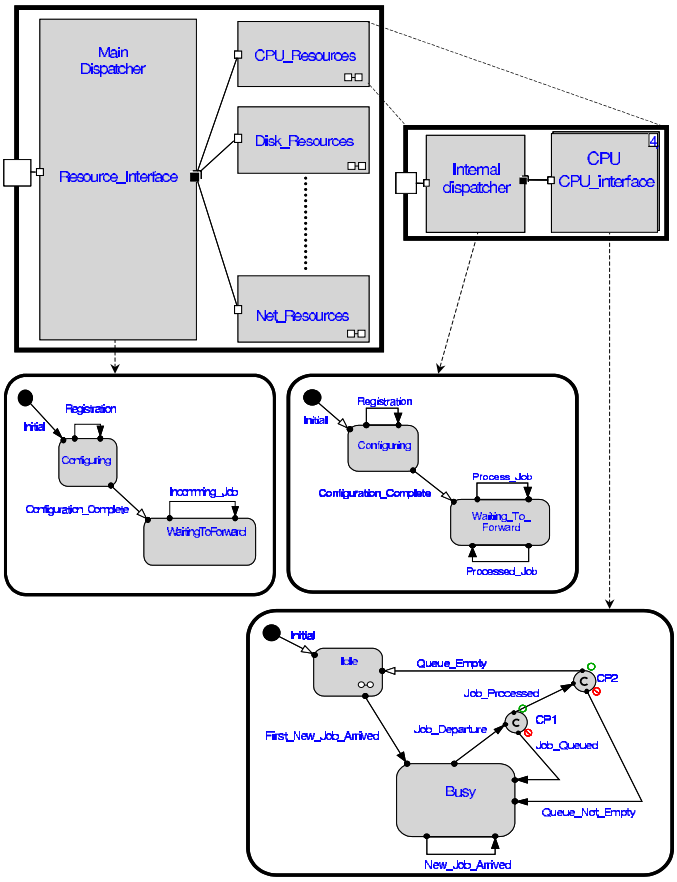

Figure 3. Basic structure of the resource side.

[19]) that every resource request has been produced by a software block (that is a set of operational steps), and includes the amount of every resource type needed to execute that software block (e.g., number of CPU instructions, number of disk blocks, bytes to be transferred on a network, etc.). Upon receiving a request, the dispatcher schedules the needed visits to the resource types. The *Resource_Interface* port in figure 3 is a multiple port, as this contributes to the generality of our scheme with regard to the number of resource types that can be considered. Figure 3 show examples of types of resources, and how this scheme allows to add (delete) a resource type by simply introducing (eliminating) a new capsule and modifying the *Resource_Interface* multiplicity.

The internal structure of any resource type capsule is quite standard as well. As shown in figure 3, where the *CPU_Resources* has been graphically expanded, every resource type capsule contains an *Internal Dispatcher* and a set of actual resource instances. In the figure we show, as an example, the case of four CPUs, where four is the multiplicity given to the *CPU* capsule (i.e., the number of resource instances) and the multiport connecting them to the *Internal Dispatcher*. Upon this "low level" dispatcher receiving a request of a specific amount of resource type it manages, basing on prior knowledge (e.g., speeds of different resource instances, queue lengths, previous request distribution) it schedules a job for a resource instance and notifies it by sending a message to the latter. When the requested amount has been consumed in the resource, the notification is sent back to the *Internal Dispatcher* and then forwarded to the *Main Dispatcher*; the latter checks whether the complete resource request of the software side has been satisfied or other resource types remain to be consumed.

Basically in figure 3 three new stereotypes (as capsules)

have been introduced: a high level dispatcher *Main Dispatcher*, a low level dispatcher *Internal Dispatcher*, and a *CPU* resource. In the lower side of the figure the State Diagrams of these stereotypes are shown.

For sake of conciseness and readability, we do not discuss the details of the dispatchers' State Diagrams, rather we focus on the *CPU* one. The CPU is modeled as a queued service center that extracts jobs from the queue following a quantum based round-robin strategy [7, 8]. In the "idle" state the queue is supposed to be empty and no job is being served. Upon the arrival of a job, the CPU becomes "busy" and it returns to the idle state in any moment the queue is idle and no job is being served. Two state transitions originate from the busy state. In case of a new job arrival the corresponding transition only serves as update of the queue length and contents. In case of a job departure from the service center (either due to the quantum expiration or due to the end of service requested) there are two conditions to be orderly checked, namely *CP1* and *CP2*. First the residual amount of resource requested is read: if zero then the job has been completely processed and it can leave the CPU, else it has to be queued again (i.e., round-robin strategy) in order to be served later for at least one more quantum. In case of job processed an additional check is needed: if there is at least one job waiting into the queue then the first job is extracted and processed (i.e., the CPU goes again in a busy state), else the CPU returns to the idle state.

In a similar way a capsule stereotype can be introduced for any type of resource type that contributes to build up a (possibly distributed) modern hardware platform (e.g., mass storage, wired network, etc.), provided that the corresponding State Diagram is also given. In any case the resource side of our scheme is open to represent whatever number of resource types with whatever number of instances, the only bound being the actual scalability of the modeled software/resources system.

3. Conclusion

We have introduced a new viewpoint in the early performance validation of software systems. Instead of starting from a software notation and translating it into a performance model (as most of the existing approaches do), we aim at migrating the resource representation into the software model notation. This opposite approach not only allows software designers to not modifying the modeling process, but also avoids a (probably heavy) transformation procedure to generate the performance model. By providing a set of prototypes representing different types of resources, we allow to "plug-in" resources into the software architectural model. The simulation of the integrated software/resource model gives the values of the performance indices of interest.

References

[1] Arief L.B. and Speirs, N.A."A UML Tool for an Automatic Generation of Simulation Programs", *Proc. of Second International Workshop on Software and Performance, WOSP2000, September 2000, Ottawa, Canada*, 2000.

[2] Cortellessa, V. and Mirandola R. "Deriving a Queueing Network based Performance Model from UML Diagrams" *Proc. of Second International Workshop on Software and Performance, WOSP2000, September 2000, Ottawa, Canada*, 2000.

[3] Gomaa, H. and Menasce, D.A."Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architecture", *Proc. of Second International Workshop on Software and Performance, WOSP2000, September 2000, Ottawa, Canada*, 2000.

[4] Kahkipuro P. " UML based Performance Modeling Framework for Object-Oriented Distributed Systems", *Proc. of Second International Conference on the Unified Modeling Language, October 28-30, 1999, USA, LNCS, Springer Verlag, vol.1723*, 1999.

[5] King, P. and Pooley, R. "Estimating the Performance of UML Models using Petri Nets", private communication, 1999.

[6] King, P. and Pooley, R. "Using UML to Derive Stochastic Petri Net Models", *Proceedings of the Fifteenth UK Performance Engineering Workshop, De-*

partment of Computer Science, The University of Bristol, N. Davies and J. Bradley, editors, UKPEW '99 July 1999.

[7] Lavenberg, S.S. "Computer Performance Modeling Handbook", *Academic Press, New York*, 1983.

[8] Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik K.C., "Quantitative system performance : computer system analysis using queueing network models", *Englewood Cliffs, N.J., Prentice-Hall*, 1984.

[9] Menasce', D.A., Gomaa, H. "A method for design and performance modeling of client/server systems", *IEEE Transactions on Software Engineering, vol.26, no.11*, November 2000.

[10] Merseguer, J., Campos, J. and Mena E. "Performance Evaluation for the design of Agent-based Systems: A Petri Net Approach", *Proc. of Software Engineering and Petri Nets (SEPN 2000), June 2000, Aarhus, Denmark*, 2000.

[11] Merseguer, J., Campos, J. and Mena E. "A Pattern-Based Approach to Model Software Performance", *Proc. of Second International Workshop on Software and Performance, WOSP2000, September 2000, Ottawa, Canada*, 2000.

[12] Mirandola R. and Cortellessa, V. "UML based Performance Modeling of Distributed Systems"*Proc. of Third International Conference on the Unified Modeling Language, October 2-6, 2000, York, UK, LNCS, Springer Verlag*, 2000.

[13] Petriu, D. "Deriving Performance Models from UML Models by Graph Transformations", *Tutorials, Second International Workshop on Software and Performance, WOSP2000, September 2000, Ottawa, Canada*, 2000.

[14] Petriu, D. Shousha, C., Jalnapurkar, A. "Architecture based Performance Analysis Applied to a Telecommunication System", *IEEE Transaction on Software Engineering, November 2000, to appear*, 2000.

[15] Petriu, D. and Wang, X. "Deriving Software Performance Models from Architectural Patterns by Graph Transformations", *Proc. of Theory and Applications of Graph transformations, TAGT'98, LNCS 1764, Springer Verlag*, 1998.

[16] Pooley, R. and C. Kabajunga, "Simulation of UML Sequence Diagrams" *Proc. of 14th UK Performance Engineering Workshop, Edinburgh,R. Pooley and N. Thomas Eds., UK PEW '98* July 1998.

[17] Selic, B. "A generic framework for modeling resources with UML", *IEEE Computer*, June 2000.

[18] Selic, B., Gullekson, G., Ward, P., "Real-Time Object Oriented Modeling", *John Wiley & Sons, Inc.*.

[19] Smith, C.U. "Performance Engineering of Software Systems", *Addison-Wesley, Reading, MA*, 1990.

[20] S. Yacoub, A. Ibrahim, H. Ammar, and K. Lateef, "Verification of UML Dynamic Specifications using Simulation-based Timing Analysis", *Proceedings of 6th International Conference on Reliability and Quality in Design, ISSAT, Orlando, Fl*, August, 2000, pp.65-69.

[21] S. Yacoub, B. Cukic, and H. Ammar. Scenario-based Reliability Analysis of Component-Based Software. Proceedings of the Tenth International Symposium on Software Reliability Engineering, ISSRE'99, Boca Raton, Florida USA, November 1-4 1999, pp.22-31.

[22] S. Yacoub, H. Ammar, and T. Robinson. Dynamic Metrics for Object Oriented Designs. Proceedings of the Sixth International Symposium on Software Metrics, Metrics'99, Boca Raton, Florida USA, November 4-6 1999, pp.50-61.

[23] S. Yacoub, H. Ammar, "A Methodology for Architectural-Level Risk Assessment using Dynamic Metrics," to appear in Proceedings of the 11th International Symposium on Software Reliability Engineering, ISSRE'00, IEEE Comp. Soc., October, 2000.

[24] http://www.omg.org.

[25] http://www.rational.com/uml/documentation.html.