Model-driven Generative Techniques for Scalable Performability Analysis of Distributed Systems*

Arundhati Kogekar[1], Dimple Kaul[1], Aniruddha Gokhale[1],
Paul Vandal[2], Upsorn Praphamontripong[2], Swapna Gokhale[2],
Jing Zhang[3], Yuehua Lin[3], Jeff Gray[3]

[1]Vanderbilt University
Dept. of Electrical Engineering
and Computer Science
Nashville, TN 37235 USA
{akogekar,dkaul,gokhale}@dre.vanderbilt.edu

[2]University of Connecticut
Dept. of Computer Science
and Engineering
Storrs, CT 06269 USA
{ssg}@engr.uconn.edu

[3]University of Alabama at Birmingham
Dept of Computer and Information Science
Birmingham, AL USA
{zhangj,liny,gray}@cis.uab.edu

Abstract

The ever increasing societal demand for the timely availability of newer and feature-rich but highly dependable network-centric applications imposes the need for these applications to be constructed by the composition, assembly and deployment of off-the-shelf infrastructure and domain-specific services building blocks. Service Oriented Architecture (SOA) is an emerging paradigm to build applications in this manner by defining a choreography of loosely coupled building blocks. However, current research in SOA does not yet address the performability (i.e., performance and dependability) challenges of these modern applications. Our research is developing novel mechanisms to address these challenges. We initially focus on the composition and configuration of the infrastructure hosting the individual services. We illustrate the use of domain-specific modeling languages and model weavers to model infrastructure composition using middleware building blocks, and to enhance these models with the desired performability attributes. We also demonstrate the use of generative tools that synthesize metadata from these models for performability validation using analytical, simulation and empirical benchmarking tools.

*This research was supported in part by a collaborative grant from the National Science Foundation (CSR-SMA).

Keywords: Model driven development, Generative programming, Performability

1. Introduction

Society is increasingly reliant on a wide array of applications (e.g., electric power grid, mobile communications, health care, entertainment) provided by distributed networked systems. Rapid advances in networking and hardware technologies, and increased competition, is requiring service providers to rapidly introduce newer applications to the market. Service providers, however, now have to deal with two major forces. On the one hand they must reduce the *time to market* while on the other hand they must ensure that the applications continue to provide high performance and dependability – the so called *performability* of the applications. The Service Oriented Architecture (SOA) [22] provides a mechanism by which new applications can be defined as a choreography of loosely coupled services.

Recent trends in software design indicate that the use of prefabricated building blocks for software development is on the rise. The prefabricated artifacts are the off-the-shelf (COTS) software infrastructure and domain-specific service components that one can ac-

quire from different vendors and assemble them to deploy large-scale applications. Modern applications designed using SOA use these concepts to form component assemblies that are deployed on distributed resources. Component middleware technologies, such as J2EE, .NET and CORBA Component Model (CCM), coupled with advances in technologies, such as SOAP, WSDL and XML, are increasingly being used to provision such applications since they enable the construction of application functionality dynamically by connecting individual component functionalities spread across distributed resources.

Although these advances in software development are necessary for the rapid development of new applications, service providers are now required to ascertain whether the choice of the components, and their configurations and compositions will provide the required levels of performance and dependability. This analysis must be performed in the early phases of the application's life cycle [23] (i.e., at design time). Design-time performability analysis is almost invariably based on a model that represents application behavior. For example, in order to enable model-based performability analysis of event-driven applications, it is necessary to build a model of the underlying event demultiplexing framework that is ubiquitous in such applications. In the past, design-time performability analysis has been conducted on a per-application basis, but this methodology does not scale as service providers are required to deploy newer applications.

Our focus is on developing and implementing novel techniques to conduct design-time performability analysis of applications built using the SOA approach. This problem requires a two-level design-time performability approach. At the first level it is necessary to ascertain that the configuration and composition of individual infrastructures at different nodes in the distributed system provide the desired performability guarantees. At the second level, we require the need for performability analysis for the assembly of services. Our previous work [4, 5] applied analytical methods for design-time performability analysis of event demultiplexing patterns in network services, such as virtual private networks. In this previous work we described the manual development and validation of analytical models for patterns-based middleware building blocks.

In this paper we first describe the manual process in developing a Stochastic Reward Net (SRN) [14], which is an analytical performability model for patterns-based building blocks. We then show how the manual development process becomes infeasible and cannot scale since the middleware building blocks provide numerous configuration parameters that affect their behavior. This problem is further complicated when such building blocks are composed to form larger systems (Note: Other kinds of performability analysis, such as simulations or empirical benchmarking will exhibit similar challenges). To address these concerns we describe a framework comprising model-driven generative tools [15,21], which enables the automatic synthesis of scalable SRN performability models, simulations and empirical benchmarks for services that are modeled as a composition of patterns-based middleware building blocks.

This paper is organized as follows: Section 2 discusses the process of building a performability model of the reactor pattern using the SRN modeling paradigm; Section 3 describes how the performability analysis process can be scaled and automated using a model-driven generative framework; Section 4 discusses our work comparing it to related work; and finally Section 5 provides concluding remarks outlining the lessons learned from this study, as well as directions for future research.

2. Developing a Performability Model of a Middleware Building Block

In this section we describe the process of constructing a Stochastic Reward Net (SRN) [14] model of a representative middleware building block. We focus on the reactor pattern [18], which provides synchronous demultiplexing and dispatching capabilities by decoupling these from event handling. Such decoupling is useful for building event-driven systems. In developing a model for a reactor requires identifying the different characteristics and performability measures of the reactor. A SRN model of the reactor pattern is then presented along with a discussion of how the performability measures can be obtained by assigning reward rates at the net level. We conclude the section with a discussion of the scalability challenges involved in constructing the performability models.

2.1. Characteristics of the Reactor Performability Model

In our performability model of the reactor, we consider a single-threaded, select-based implementation of the reactor pattern with the following characteristics, as shown in Figure 1:

- The reactor receives two types of input events with one event handler for each type of event registered with the reactor.

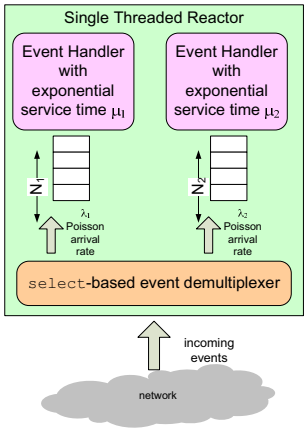

Figure 1. Characteristics of the Reactor

- Each event type has a separate queue, which holds the incoming events of that type. The buffer capacity for the queue of type #1 events is denoted N_1 and of type #2 events is denoted N_2.
- Event arrivals for both types of events follow a Poisson distribution with rates λ_1 and λ_2, while the service times of the events are exponentially distributed with rates μ_1 and μ_2.
- In a given snapshot, if the event handles corresponding to both the event types are enabled, then they are serviced in no particular order. In other words, the order in which the events are handled is non-deterministic.

The following performability metrics are of interest for each one of the event types in the reactor pattern model:

- **Expected throughput** – which provides an estimate of the number of events that can be processed by the single threaded event demultiplexing framework. These estimates are important for many applications, such as telecommunications call processing.
- **Expected queue length** – which provides an estimate of the queuing for each of the event handler queues. These estimates are important to develop appropriate scheduling policies for applications with real-time requirements.
- **Expected total number of events** – which provides an estimate of the total number of events in a system. These estimates are also tied to scheduling decisions. In addition, these estimates will determine the right levels of resource provisioning required to sustain the system demands.
- **Probability of event loss** – which indicates how many events will have to be discarded due to lack of buffer space. These estimates are important particularly for safety-critical systems, which cannot afford to lose events. These also provide an estimate of the desired levels of resource provisioning.

2.2. SRN Model of the Reactor Pattern

SRNs represent a powerful modeling technique that is concise in its specification and whose form is closer to a designer's intuition about what a model should look like. Since a SRN specification is closer to a designer's intuition of system behavior, it is also easier to transfer the results obtained from solving the models and interpret them in terms of the entities that exist in the system being modeled. An overview of SRNs can be found in [16]. Stochastic reward nets have been extensively used for performability, reliability and performability analysis of a variety of systems [9–11, 14, 17, 20]. The work closest to our work is reported by Ramani *et al.* [17], where SRNs are used for the performability analysis of the CORBA event service. The CORBA event service is yet another pattern that provides publish/subscribe services.

Description of the net: Figure 2 shows the SRN model for the Reactor pattern. The net is comprised of two parts. Part (a) models the arrival, queuing, and service of the two types of events as explained below. Transitions $A1$ and $A2$ represent the arrivals of the events of types one and two, respectively. Places $B1$ and $B2$ represent the queue for the two types of events. Transitions $Sn1$ and $Sn2$ are immediate transitions which are enabled when a snapshot is taken. Places $S1$ and $S2$ represent the enabled handles of the two types of events, whereas transitions $Sr1$ and transition $Sr2$ represent the execution of the enabled event handlers of the two types of events. An inhibitor arc from place $B1$ to transition $A1$ with multiplicity $N1$ prevents the firing of transition $A1$ when there are $N1$ tokens in the place $B1$. The presence of $N1$ tokens in the place $B1$ indicates that the buffer space to hold the incoming input events of the first type is full, and no additional incoming events can be accepted. The inhibitor arc from place $B2$ to transition $A2$ achieves the same purpose for type two events.

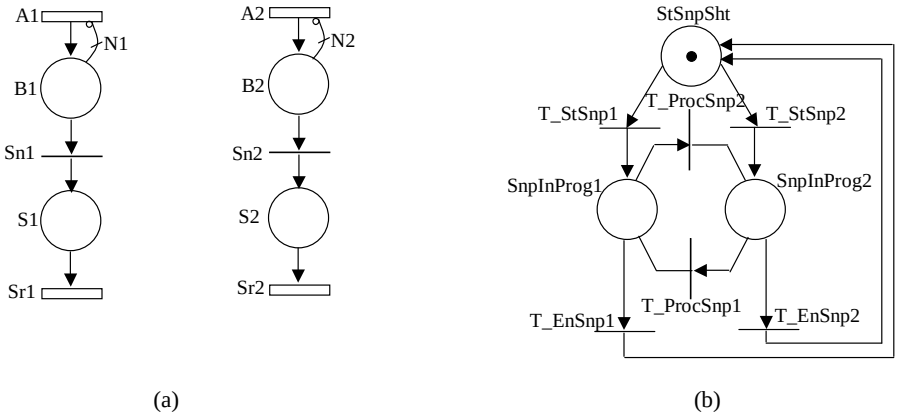

Figure 2. SRN model of the Reactor pattern

Part (b) models the process of taking successive snapshots and non-deterministic service of event handles in each snapshot as explained below. Transition $Sn1$ is enabled when there are one or more tokens in place $B1$, a token in place $StSnpSht$, and no token in place $S1$. Similarly, transition $Sn2$ is enabled when there are one or more tokens in place $B2$, a token in place $StSnpSht$ and no token in place $S2$. Transition T_StSnp1 and T_StSnp2 are enabled when there is a token in either place $S1$ or place $S2$ or both. Transitions T_EnSnp1 and T_EnSnp2 are enabled when there are no tokens in both places $S1$ and $S2$. Transition $T_ProcSnp2$ is enabled when there is no token in place $S1$, and a token in place $S2$. Similarly, transition $T_ProcSnp2$ is enabled when there is no token in place $S2$ and a token in place $S1$. Transitions $Sr1$ is enabled when there is a token in place $SnpInProg1$, and transition $Sr2$ is enabled when there is a token in place $SnpInProg2$.

Assignment of reward rates: The performability measures described in Section 2.1 can be computed by assigning reward rates at the net level as summarized in Table 1. The throughputs T_1 and T_2 are respectively given by the rate at which transitions $Sr1$ and $Sr2$ fire. The queue lengths Q_1 and Q_2 are given by the average number of tokens in places $B1$ and $B2$, respectively. The total number of events E_1 is given by the sum of the number of tokens in places $B1$ and $S1$. Similarly, the total number of events E_2 is given by the sum of the number of tokens in places $B2$ and $S2$. The loss probability L_1 is given by the probability of $N1$ tokens in place $B1$. Similarly, the loss probability L_2 is given by the probability of $N2$ tokens in place $B2$.

Table 1. Reward assignments to obtain performability measures

Performability metric	Reward rate
T_1	return rate($Sr1$)
T_2	return rate($Sr2$)
Q_1	return ($\#B1$)
Q_2	return ($\#B2$)
L_1	return ($\#B1 == N1?1 : 0$)
L_2	return ($\#B2 == N2?1 : 0$)
E_1	return($\#B1 + \#S1$)
E_2	return($\#B2 + \#S2$)

After the model is developed, it must be solved to determine how the system will perform for the given workloads. Our previous work [4, 5] describes the results of solving the SRN models for the reactor pattern with event handling priorities. Validation of the performability measures obtained from SRN model using CSIM is also described in our earlier work [5].

2.3. Scalability Challenges

The previous section described the manual process of constructing a SRN model for the reactor pattern and assigning reward rates to obtain the different performability measures. Many different variations of the reactor pattern are possible depending on the configuration parameters used. These variations stem from the different event demultiplexing and event handling

strategies used in a reactor. For example, in network-centric applications, networking events can be demultiplexed using operating system calls, such as `select` or `poll`. For graphical user interfaces (GUIs), these events could be due primarily to mouse clicks and can be handled by GUI frameworks like Qt or Tk. On the other hand, the event handling mechanisms could involve a single thread of control that demultiplexes and handles an event, or each event could be handled concurrently using worker threads in a thread pool or by thread on demand.

Other variations stem from the number of event types handled, the buffer space available for queuing events, input workloads and event service rates. To enable design-time performability analysis for an application employing a variant of the reactor pattern, the SRN model of the variant needs to constructed manually. This process is cumbersome, tedious and time-consuming. These challenges are further complicated when systems are composed of several middleware building blocks. There is a need to automatically generate performability models for different variants based on the performability model, which captures the invariant characteristics of the pattern and specializing them with the possible variations. Section 3 describes our solution to address these challenges.

3. Scaling and Automating the Performability Modeling Process

The previous sections described how a SRN model for performability of the desired application can be developed. The process described until now focuses on manually developing these models and the associated input scripts used by the model solvers, such as SPNP [8]. In this section, we describe the model-driven development (MDD) [15, 21] approach, which allows the user to scale the base SRN models and automate the process of performability analysis.

3.1. Modeling Languages for Performability Analysis

In this section we describe the ideas based on a model-driven [15] generative programming [3] framework we are developing to address the aforementioned challenges. Our modeling framework comprises modeling languages we have developed using the Generic Modeling Environment (GME) [13]. GME is a tool that enables domain experts to develop visual modeling languages and generative tools associated with those languages. The modeling languages in GME are represented as metamodels. A metamodel in GME depicts a class diagram using UML-like constructs showcasing the elements of the modeling language and how they are associated with each other. The GME environment can then be used by application developers to model examples that conform to the syntax and semantics of the modeling language captured in the metamodels.

We have developed two modeling languages that provide the visual syntactic and semantic elements required to model the systems compositions and their performability models (Note: Formal descriptions of these modeling languages is beyond the scope of this paper). The first language is called POSAML (Patterns-Oriented Software Architecture Modeling Language), which models the patterns described in the POSA [18] pattern language. The POSA pattern language is a vocabulary describing a set of related patterns used to develop network services. The other modeling language is called SRNML (Stochastic Reward Net Modeling Language), which enables a user to model the behavior of individual patterns as a SRN. Additional details on these languages and their capability are described in our recent work [12].

Figure 3 illustrates the benchmarking view of the metamodel for the POSAML language. POSAML allows a user to describe the middleware as a collection of patterns. In the benchmarking view of the system the modeling language enables users to model the workload characterization for the composed system. For example, it is possible to model the input arrival patterns of requests, the service rates of the event handlers in the systems, and sizes of thread pools.

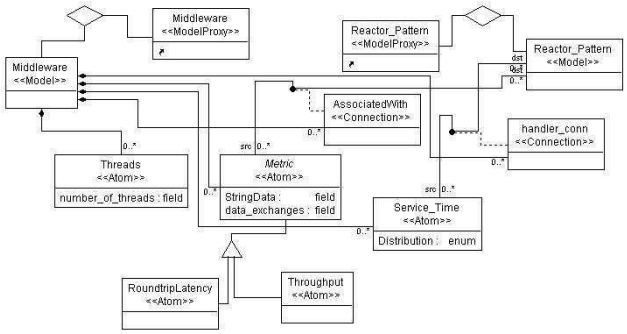

Figure 3. POSAML Metamodel: Benchmarking View

Built in constraint checking in POSAML enables correct by construction structure of system compositions since any erroneous associations between different patterns are flagged as an error at modeling time. Moreover, the feature modeling view and benchmarking view of the language also flags errors when a model

engineer commits errors. For example, if an engineer chooses a single threaded reactor demultiplexing strategy in the feature view but provides a thread pool configuration in the benchmarking view, then the constraint checker flags this condition as an error.

Figure 4 illustrates an example middleware composition comprising three patterns of the POSAML language.

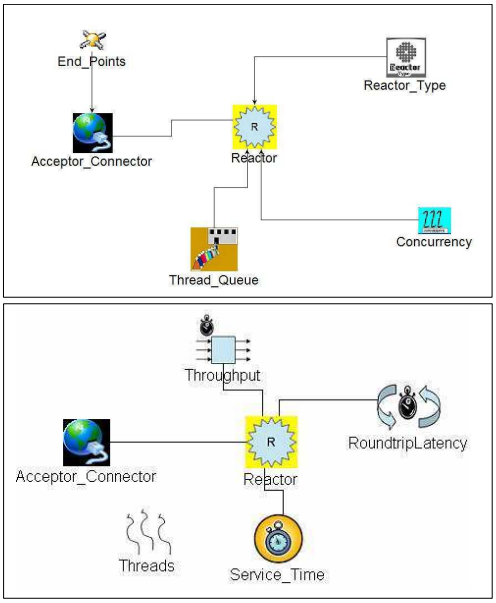

Figure 4. POSAML GME Model for the Reactor Pattern

A second language we have developed is the Stochastic Reward Net Modeling Language (SRNML). Parts of SRNML are shown in Figure 5. SRNML provides the artifacts necessary to model a system as a SRN.

Figure 6 illustrates a sample the SRN model using SRNML.

3.2. Generative Tools and Model Scalability

The GME environment allows metamodel developers to plugin model interpreters that can provide various capabilities, such as code generation or configuration information.

In POSAML to date we have included two model interpreters with generative capabilities. The benchmark interpreter described here traverses all the elements of the model (i.e., Pattern, Feature and Benchmarking), extracting relevant information and generating an XML file to drive the simulations and benchmarking. The relevant information captured in the synthesized metadata includes:

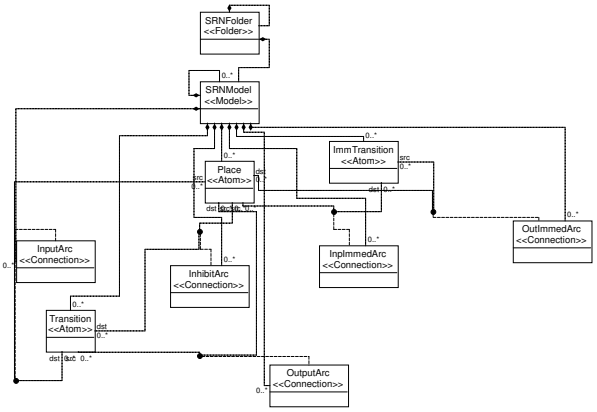

Figure 5. SRN Metamodel

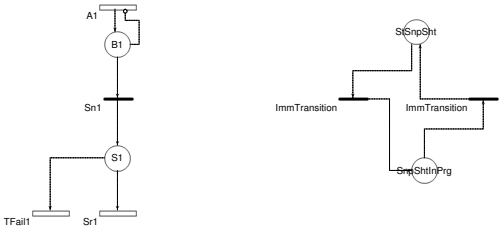

Figure 6. SRN GME Model of the reactor pattern

1. the Reactor type (extracted from the Feature Aspect), such as a single threaded or multi-threaded.
2. the desired metrics, such as whether the measurements should include latency or throughput or both.
3. the number of data exchanges (which is basically the number of times one client thread generates events for the reactor).
4. the number of client threads (or connections).
5. the number of event handlers (extracted from the Pattern Aspect)
6. the service time required by a handler specified as uniform distribution or exponential distribution.
7. arrival pattern of requests modeled as Poisson or periodic arrivals.

Following is a snippet of the XML file generated by the benchmark interpreter:

```
<benchmark_inputs>
   <connections>5</connections>
   <data>ABCDEF</data>
   <data_exchanges>200</data_exchanges>
   <reactor_inputs>
   <reactor_type>wfmo</reactor>
      <handlers>2</handlers>
   </reactor_inputs>
 </benchmark_inputs>
```

Model scalability is addressed using a model replicator tool we have developed previously called C-SAW (Constraint-Specification Aspect Weaver) [6]. The model replication and scalability was discussed in an earlier work of ours [7]. We have designed C-SAW to provide support for modularizing crosscutting modeling concerns as well as scaling models in the GME. This weaver operates on the internal representation of a model (similar to an abstract syntax tree of a compiler). GME provides a framework that allows metamodel developers to register custom actions and hooks with the environment. These hooks can read and write the elements of a model during the modeling stage. GME also provides an introspection API, which provides knowledge about the types and instances of a model, without *a priori* knowledge about the underlying metamodel. Utilizing this feature of GME, we have implemented C-SAW as a "plug-in," which is GME terminology for a metamodel independent hook.

4. Related Work

Performance and dependability analysis of some middleware services and patterns has been addressed by some researchers. Ramani *et al.* [17] develop a SRN model for the performability analysis of a CORBA event service, which is a pattern that provides publish/-subscribe service. Aldred *et al.* [1] develop Colored Petri Net (CPN) models for different types of coupling between the application components and with the underlying middleware. They also define the composition rules for composing the CPN models if multiple types of coupling is used simultaneously in an application. A dominant aspect of these works are related to application-specific performability modeling. In contrast we are concerned with determining how the underlying middleware that is composed for the systems they host will perform.

With the growing complexity of component-based systems, composing system-level performance and dependability attributes using the component attributes and system architecture is gaining attention. Crnkovic *et al.* [2] classify the quality attributes according to the possibility of predicting the attributes of the compositions based on the attributes of the components and the influence of other factors such as the architecture and the environment. However, they do not propose any methods for composing the system-level attributes.

At the model-driven development and program transformation level, the work by Shen and Petriu [19] investigated the use of aspect-oriented modeling techniques to address performability concerns that are weaved into a primary UML model of functional behavior. It has been observed that an improved separation of the performability description from the core behavior enables various design alternatives to be considered more readily (i.e., after separation, a specific performance concern can be represented as a variability measure that can be modified to examine the overall systemic effect). The performability concerns are specified in the UML profile for Schedulability, Performance, and Time (SPT) with underlying analysis performed by a Layered Queueing Network (LQN) solver.

A disadvantage of the approach is that UML forces a specific modeling language. The SPT profile also forces performability concerns to be specified in a manner than limits the ability to be tailored to a specific performability analysis methodology. As an alternative, domain-specific modeling supports the ability to provide a model engineer with a notation that fits the domain of interest, which improves the level of abstraction of the performability modeling process. Our work falls in the category of developing domain-specific models for performability analysis.

5. Concluding Remarks

Time to market pressures and economic reasons are requiring the next generation of distributed networked services to be developed via composition and assembly of off-the-shelf reusable components. The service providers are now required to reason about the performance and dependability of such composed systems. This paper discusses a framework for design-time performability analysis and validation of services that are composed, configured and deployed using patterns-based middleware building blocks. The performability analysis models in our study use Stochastic Reward Nets. However, as shown due to the substantial variability that exists within every building block and in their compositions, it is infeasible to manually develop performability models for large systems. We presented a model-driven generative framework that can be used to automatically synthesize complex SRN models, simulations and empirical benchmarks for the composed systems.

Current shortcomings in our work will be addressed as we extend POSAML to include the additional patterns used in network systems and address correctness of the compositions. Additional focus will be on integration of POSAML and SRNML. The goal is to use this integrated framework to drive performability analysis using analytical models, simulations and empirical benchmarking of large systems. Additional concerns, such as dependability and security, will also be addressed in combination with the performability dimension addressed in this paper. Ultimately, we want to scale our models to handle the performability evaluation of the distributed assemblies envisioned in the SOA approach.

References

[1] L. Aldred, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. "On the notion of coupling in communication middleware". In *Proc. of Intl. Symposium on Distributed Objects and Applications (DOA)*, Agia Napa, Cyprus, 2005.

[2] I. Crnkovic, M. Larsson, and O. Preiss. *Book on Architecting Dependable Systems III, R. de Lemos (Eds.)*, chapter "Concerning predictability in dependable component-based systems: Classification of quality attributes", pages 257–278. Springer-Verlag, 2005.

[3] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, 2000.

[4] S. Gokhale, A. S. Gokhale, and J. Gray. Response time analysis of middleware event demultiplexing pattern for network services. In *Proc. of IEEE Globecom, Advances for Networks and Internet Track*, St. Louis, MO, December 2005.

[5] S. Gokhale, P. Vandal, U. Praphamontripong, A. Gokhale, and J. Gray. Performance analysis of the reactor pattern in network services. In *Proceedings of the 5th International Workshop on Performance Modeling, Evaluation and Optimization of Parallel and Distributed Systems (PMEO-PDS 2006)*, Rhodes Island, Greece, April 2006.

[6] J. Gray, T. Bapty, and S. Neema. Handling Crosscutting Constraints in Domain-Specific Modeling. *Communications of the ACM*, pages 87–93, October 2001.

[7] J. Gray, Y. Lin, J. Zhang, S. Nordstrom, A. Gokhale, S. Neema, and S. Gokhale. Replicators: Transformations to Address Model Scalability. In *Lecture Notes in Computer Science: Proceedings of 8th International Conference Model Driven Engineering Languages and Systems (MoDELS 2005)*, pages 295–308, Montego Bay, Jamaica, Nov. 2005. Springer Verlag.

[8] C. Hirel, B. Tuffin, and K. S. Trivedi. SPNP: Stochastic Petri Nets. Version 6.0. *Lecture Notes in Computer Science 1786*, 2000.

[9] O. Ibe, A. Sathaye, R. Howe, and K. S. Trivedi. Stochastic Petri net modeling of VAXCluster availability. In *Proc. of Third International Workshop on Petri Nets and Performance Models*, pages 142–151, Kyoto, Japan, 1989.

[10] O. Ibe and K. S. Trivedi. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, December 1990.

[11] O. Ibe and K. S. Trivedi. Stochastic Petri net analysis of finite–population queueing systems. *Queueing Systems: Theory and Applications*, 8(2):111–128, 1991.

[12] A. Kogekar, D. Kaul, A. Gokhale, P. Vandal, U. P. S. Gokhale, , and J. Gray. Investigating the use of model-driven generative techniques for middleware performance analysis. In *Submitted to Dependable Networks and Systems - Performance and Dependability Symposium (DSN-PDS 2006)*, Philadephia, PA, June 2006.

[13] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, pages 44–51, November 2001.

[14] J. Muppala, G. Ciardo, and K. S. Trivedi. Stochastic reward nets for reliability prediction. *Communications in Reliability, Maintainability and Serviceability: An International Journal Published by SAE Internationa*, 1(2):9–20, July 1994.

[15] Object Management Group. *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.

[16] A. Puliafito, M. Telek, and K. S. Trivedi. The evolution of stochastic Petri nets. In *Proc. of World Congress on Systems Simulation*, pages 3–15, Singapore, September 1997.

[17] S. Ramani, K. S. Trivedi, and B. Dasarathy. Performance analysis of the CORBA event service using stochastic reward nets. In *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems*, pages 238–247, October 2000.

[18] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.

[19] H. Shen and D. C. Petriu. Performance analysis of uml models using aspect-oriented modeling techniques. In *Proc. of Model Driven Engineering Languages and Systems (MoDELS 2005), Springer LNCS 3713*, pages 156–170, Montego Bay, Jamaica, October 2005.

[20] H. Sun, X. Zang, and K. S. Trivedi. A stochastic reward net model for performance analysis of prioritized DQDB MAN. *Computer Communications, Elsevier Science*, 22(9):858–870, June 1999.

[21] J. Sztipanovits and G. Karsai. Model-Integrated Computing. *IEEE Computer*, 30(4):110–112, Apr. 1997.

[22] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2006.

[23] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley, 2001.