Investigating the Use of Model-driven Generative Techniques for Middleware Performance Analysis*

Arundhati Kogekar, Dimple Kaul
and Aniruddha S. Gokhale
Dept of EECS
Vanderbilt U
Nashville, TN 37235
{akogekar,dkaul,gokhale}@dre.vanderbilt.edu

Paul Vandal, Upsorn Praphamontripong
and Swapna S. Gokhale
Dept. of CSE
U of Connecticut
Storrs, CT 06269
{ssg}@engr.uconn.edu

Jeffrey G. Gray
Dept of CIS
Univ of Alabama at Birmingham
Birmingham, AL
gray@cis.uab.edu

Abstract

The increasing demand for faster availability of new, feature-rich and high performance network-centric services imposes the need for these services to be rapidly developed via composition, configuration and assembly of standards middleware building blocks. However, the decisions on the systems compositions must be validated for performance guarantees at design time, which is addressed in this paper. First, we describe the manual process in developing a performance model for a middleware building block and illustrate the challenges in scaling such models for large-scale compositions. Second, we show how model-driven development (MDD) can play a key role in addressing these scalability challenges. We describe a MDD performance evaluation framework we are developing. Third, we illustrate our framework's capabilities on a simple case study demonstrating the viability of this approach.

Keywords: performance models, simulations, model-driven generative technologies.

1 Introduction

Society is increasingly reliant on a wide array of services (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 services 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 services continue to provide high performance and dependability.

One predominant feature of these networked systems is their event-driven capability requiring efficient demultiplexing, dispatching and servicing of the events [13]. The event-driven style constitutes a provider/listener

*This research was supported in part by a collaborative grant from the National Science Foundation on the CSR-SMA Program

model [5], where the application listens for service requests or "events" and provides the necessary services in response to these requests or events. Event-driven systems provide many advantages, the most prominent advantage being evolvability, which is enabled by the separation of event demultiplexing and dispatching from event handling. Although the handling of events is specific to the service itself, the event demultiplexing and dispatching functionality is more or less uniform across all the services that follow the event-driven style, and can be codified into a reusable building block or a pattern [4]. The *Reactor* pattern [19] embodies the demultiplexing and dispatching capabilities that could be reused to facilitate the development of event-driven applications.

There are numerous additional commonly occurring patterns in such services that address issues, such as concurrency, connection management, prioritized service and data (de)marshaling. Standards middleware, such as CORBA, J2EE and .NET, provides the building blocks that address these issues. Middleware building blocks thus provide the means to rapidly compose, configure, assemble and deploy the services that service providers want to rapidly bring to market.

Service providers are, however, now required to ascertain whether the choice of building blocks, 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 system's life cycle [24] (i.e., at design time). Design-time performance analysis is almost invariably based on a model that represents application behavior. For example, in order to enable model-based performance 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 performance analysis has been conducted for a specific application. In some cases, analysis methodology for generic architecture styles that are applicable at the level of the application architecture have been developed [25].

Our previous work [6] described our early work on applying analytical methods for design-time performance analysis of event demultiplexing in network services, such as virtual private networks. This paper enhances our previous work describing an approach to conducting design-time performance analysis of systems composed, configured, assembled and deployed from patterns-based middleware building blocks. We describe the manual process in developing a Stochastic Reward Net (SRN) [15] analytical performance model for patterns-based building blocks. The manual development process, however, 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.[1] To address these concerns we describe a framework comprising model-driven generative tools [16, 23], which enables the automatic synthesis of scalable SRN performance 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 performance model of the reactor pattern using the SRN modeling paradigm; Section 3 describes how the performance analysis process can be scaled and automated using a model-driven generative framework; Section 4 illustrates the use of our MDD framework for performance analysis of a application on mobile devices; Section 5 discusses our work comparing it to related work; and finally Section 6 provides concluding remarks outlining the lessons learned from this study, as well as directions for future research.

2 Developing a Performance Model of a Middleware Building Block

In this section we describe the process of constructing a Stochastic Reward Net (SRN) [15] model of a middleware building block. We focus on the reactor pattern, which provides synchronous demultiplexing and dispatching capabilities by decoupling these from event handling. Such decoupling is useful for building event-driven systems. Initially we describe the characteristics of the reactor pattern and the relevant performance measures. A SRN model of the reactor pattern is then presented along with a discussion of how the performance measures can be

[1] Other kinds of performance analysis, such as simulations or empirical benchmarking will exhibit similar challenges.

obtained by assigning reward rates at the net level. We conclude the section with a discussion of the scalability challenges involved in constructing the performance models.

2.1 Characteristics of the Reactor Pattern

Event demultiplexing, dispatching and event handling is a hallmark of event-driven systems. The development of these systems is increasingly relying on the use of well-documented patterns [4, 19]. The use of patterns offers the potential for improving the reliability as well as enabling faster time-to-market by fostering reuse. Fundamental to the design and development of event-driven systems is the Reactor pattern [19]. Therefore, performance analysis of the event-driven aspects of systems boils down to the performance analysis of the reactor pattern. This section therefore provides background information on the reactor pattern and its dynamics.

Figure 1 depicts a typical event demultiplexing and dispatching mechanism documented in the reactor pattern [19]. The application registers an event handler with the event demultiplexer and delegates to it the responsibility of listening to incoming events. On the occurrence of an event, the demultiplexer dispatches the event by making a callback to the correct application-supplied event handler. This is the idea behind the reactor pattern, which provides synchronous event demultiplexing and dispatching capabilities.

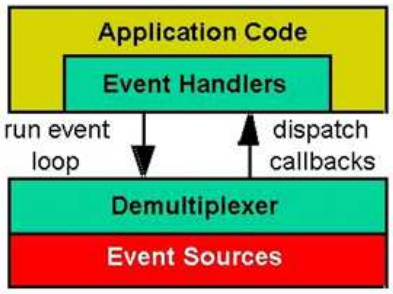

Figure 1. Event Demultiplexing Pattern

Figure 2 illustrates the reactor pattern dynamics. We categorize these dynamics into two phases described below.

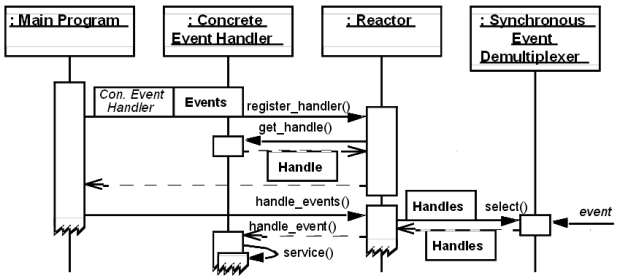

Figure 2. Reactor Pattern Dynamics

1. **Registration phase:** In this phase all the event handlers register with the reactor associating themselves with a particular event type they are interested in. Event types usually supported by a reactor are input,

output, timeout and exceptions. The reactor will maintain a set of handles corresponding to each handler registered with it.

2. **Snapshot phase:** Once the event handlers have completed their registration, the main thread of control is passed to the reactor, which in turn listens for events to occur. A snapshot is then an instance in time wherein a reactor determines all the event handles that are enabled at that instant. For all the event handles that are enabled in a given snapshot, the reactor proceeds to service each event by invoking the associated event handler. There could be different strategies to handle these events. For example, a reactor could handle all the enabled events sequentially in a single thread or could hand it over to worker threads in a thread pool. After all the events are processed, the reactor proceeds to take the next snapshot of the system.

2.2 Characteristics of the Reactor Performance Model

In our performance model of the reactor, we consider a single-threaded, select-based implementation of the reactor pattern with the following characteristics, as shown in Figure 3:

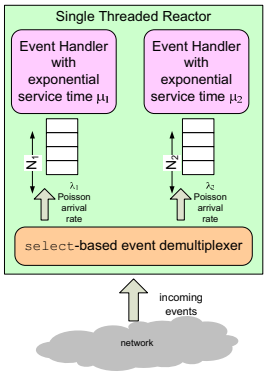

Figure 3. Characteristics of the Reactor

- The reactor receives two types of input events with one event handler for each type of event registered with 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 performance metrics are of interest for each one of the event types in the reactor pattern described in Section 2.2:

- **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.

2.3 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 [17]. Stochastic reward nets have been extensively used for performance, reliability and performability analysis of a variety of systems [9–11, 15, 18, 22]. The work closest to our work is reported by Ramani *et al.* [18], where SRNs are used for the performance analysis of the CORBA event service. The CORBA event service is yet another pattern that provides publish/subscribe services.

Description of the net: Figure 4 shows the SRN model for the Reactor pattern described in Section 2.1. The net comprises 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.

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 $SnpInProg2$. Table 1 summarizes the enabling functions for the transitions in the net shown in Figure 4.

Dynamic evolution of the net: We explain the process of taking a snapshot and servicing the enabled event handles in the snapshot with the help of an example. We consider a scenario where there is one token each in places $B1$ and $B2$. Due to the presence of tokens in places $B1$ and $B2$, and a token in place $StSnpSht$, transitions $Sn1$ and $Sn2$ are enabled. Both these transitions are assigned the same priority, and hence either one of them can

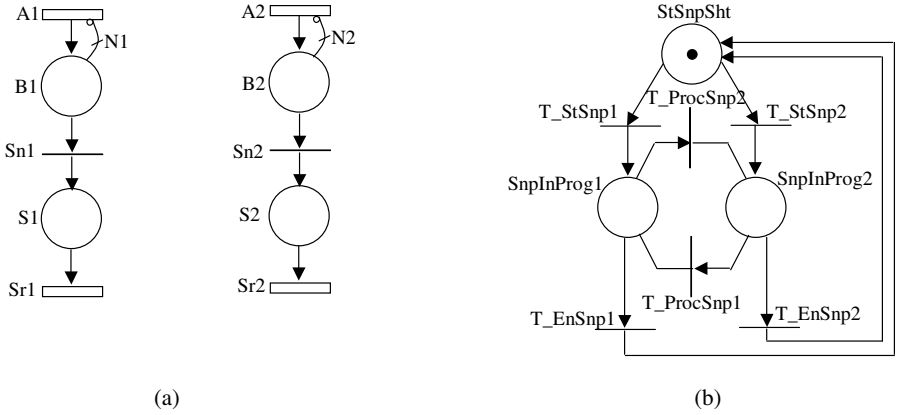

Figure 4. SRN model of the Reactor pattern

Table 1. Guard functions

Transition	Guard function
$Sn1$	$((\#StSnpShot == 1)\&\&(\#B1 >= 1)\&\&(\#S1 == 0))?1 : 0$
$Sn2$	$((\#StSnpShot == 1)\&\&(\#B2 >= 1)\&\&(\#S2 == 0))?1 : 0$
T_StSnp1	$((\#S1 == 1)\|\|(\#S2 == 1))?1 : 0$
T_StSnp2	$((\#S1 == 1)\|\|(\#S2 == 1))?1 : 0$
$T\_ESnpSht1$	$((\#S1 == 0\&\&(\#S2 == 0))?1 : 0$
$T\_ESnpSht2$	$((\#S1 == 0\&\&(\#S2 == 0))?1 : 0$
$T\_ProcSnp1$	$((\#S1 == 1\&\&(\#S2 == 0))?1 : 0$
$T\_ProcSnp2$	$((\#S1 == 0\&\&(\#S2 == 1))?1 : 0$
$Sr1$	$(\#SnpInProg1 == 1)?1 : 0$
$Sr2$	$(\#SnpInProg2 == 1)?1 : 0$

fire. Without loss of generality, we assume that transition $Sn1$ fires first, which deposits a token in place $S1$. The presence of a token in place $S1$ and place $StSnpSht$ enables transition T_StSnp1. Also, transition $Sn2$ is already enabled. If transition T_StSnp1 were to fire before transition $Sn2$, the firing of transition $Sn2$ would be precluded. In order to prevent this from happening, transition $Sn2$ is assigned a higher priority than transition T_StSnp1, so that transition $Sn2$ fires before T_StSnp1 when both are enabled. Firing of transition $Sn2$ deposits a token in place $S2$ which enables transition T_StSnp2. Transitions T_StSnp1 and T_StSnp2 are both enabled, corresponding to the event handles of both types of events. If transition T_StSnp1 fires before T_StSnp2, then event handle for the first type of event will be executed prior to event handle for the second type of event. On the other hand, T_StSnp2 fires before T_StSnp1, then event handle for the second type of event will be executed prior to event handle for the first type of event. Both the transitions T_StSnp1 and T_StSnp2 have an equal chance of firing, and this represents the non-determinism in the execution of the enabled event handles. Without loss of generality, we assume transitions T_StSnp1 fires depositing a token in place $SnpInProg1$, which enables transitions $Sr1$. Additionally, firing of transition T_StSnp1 precludes the firing of transition T_StSnp2 and vice versa. Once transition $Sr1$ fires, a token is removed from place $S1$, after which transition $T\_ProcSnp2$ fires and

deposits a token in place $SnpInProg2$. A token in place $SnpInProg2$ enables transition $Sr2$. Firing of $Sr2$ removes the token from place $S2$. Once $Sr2$ fires, there are no tokens in places $SnpInProg1$ and $SnpInProg2$, which enables transition T_EnSnp2. The firing of T_EnSnp2 marks the completion of the present snapshot, and the beginning of the next one.

Assignment of reward rates: The performance measures described in Section 2.2 can be computed by assigning reward rates at the net level as summarized in Table 2. 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 2. Reward assignments to obtain performance measures

Performance 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($\#B2 + \#S2$)

2.4 Scalability Challenges

The above sections describe the manual process of constructing a SRN model for the reactor pattern and assigning reward rates to obtain the different performance 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 performance 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 performance models for different variants based on the performance 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 focusses on manually developing these models and the associated input scripts used by the model solvers, such as SPNP. As the application complexity grows, however, it becomes tedious and error prone to develop these models manually since the accidental complexities involved in modeling the system performance characteristics and the input script sizes for the model solver grow substantially. As explained earlier there could be several different event types that an event demultiplexing and handling framework may have to handle. Moreover, it should be possible for a software architect, who is a non expert in the tools and techniques of performance analysis to be able to use the model. To enable these dual objectives, it is necessary to encapsulate the performance modeling approach described in the earlier sections into user-friendly tools that enable model scalability to large systems. In this section, we describe the model-driven development (MDD) [16, 23] approach, which allows the user to scale the smaller base model and automate the process of performance analysis.

3.1 Modeling Languages for Performance Analysis

In this section we describe the ideas based on a model-driven [16] 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) [12]. 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 performance models.[2] The first language is called POSAML (Patterns-Oriented Software Architecture Modeling Language), which models the patterns described in the POSA [19] 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.

Figures 5 and 6 illustrate two different views of the metamodel for the POSAML language. Although the formal description of this language is beyond the scope of this paper, we describe the important features supported by this language. POSAML allows a user to describe the middleware as a collection of patterns. Progress to date on POSAML enables users to model the Reactor, Acceptor and Connector patterns. POSAML decouples the middleware structural composition i.e, the way individual patterns are associated with each other from the variability modeling in each pattern. The variability in each pattern stems from the allowable configurations of each pattern, which can be modeled using the configuration capabilities of the language. For example, an Acceptor pattern can be configured with the transport protocol and the end point it will use to listen for incoming requests. Yet another dimension of modeling permitted by POSAML is related to benchmarking, a capability enabled by the benchmarking view of the language. This view of the system 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.

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

[2] Formal descriptions of these modeling languages is beyond the scope of this paper.

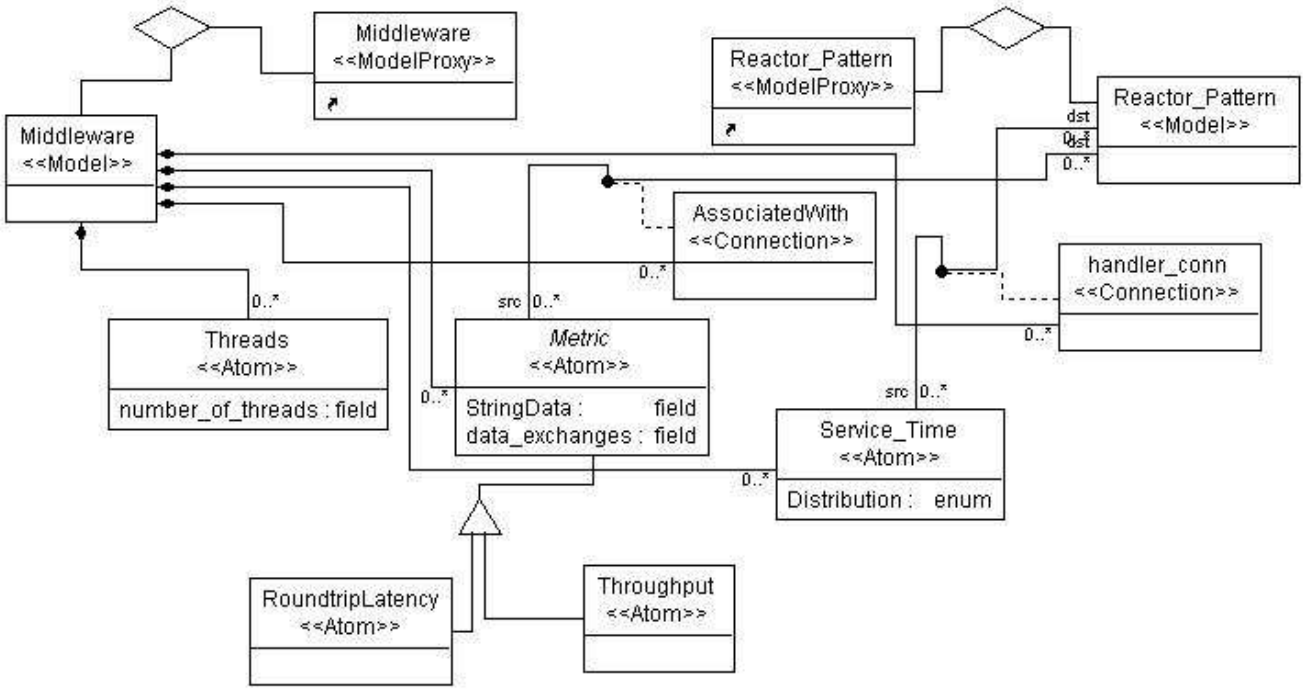

Figure 5. POSAML Metamodel: Benchmarking View

example, if a modeler choses 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 7 illustrates an example middleware composition comprising three patterns of the POSAML language. The figure illustrates the configuration view of the system. Such a configuration will be useful in generating the artifacts for the empirical benchmarking tests as well as simulations.

A second language we have developed is the Stochastic Reward Net Modeling Language (SRNML). Parts of SRNML are shown in Figure 8. SRNML provides the artifacts necessary to model a system as a SRN.

Figure 9 illustrates the SRN model we have used for our case study and evaluation described in Section 4.

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:

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).

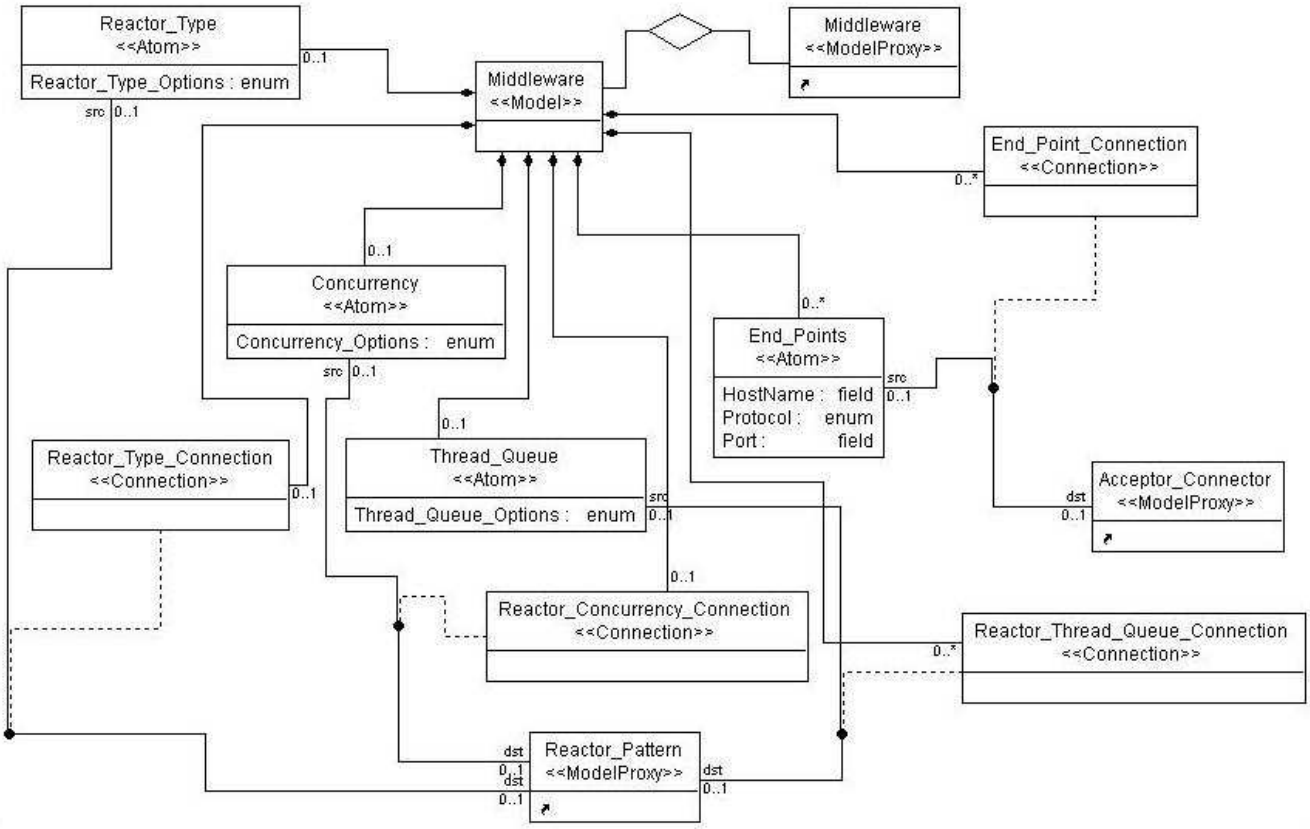

Figure 6. POSAML Metamodel: Configuration View

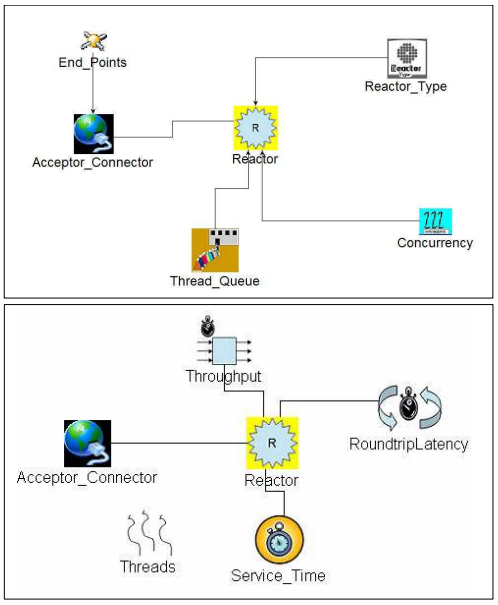

Figure 7. POSAML GME Model for the Reactor Pattern

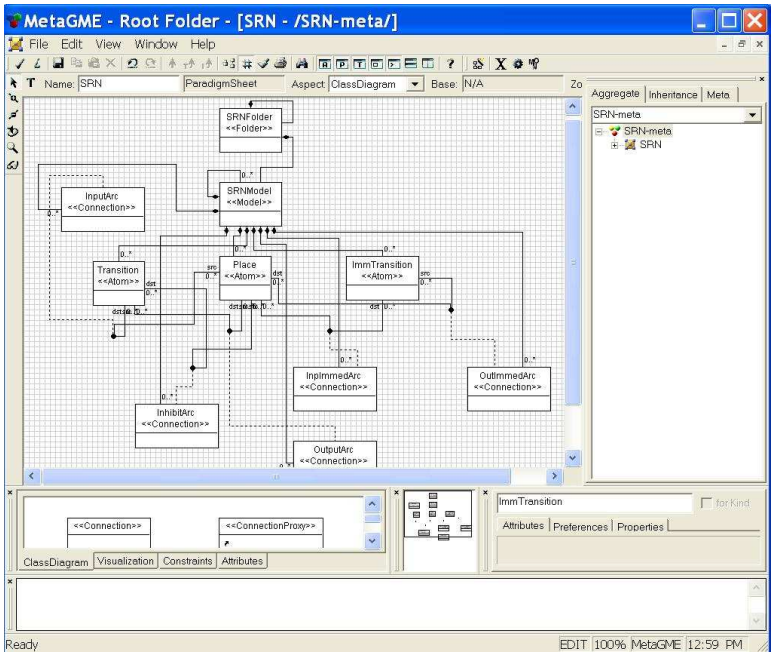

Figure 8. SRN Metamodel

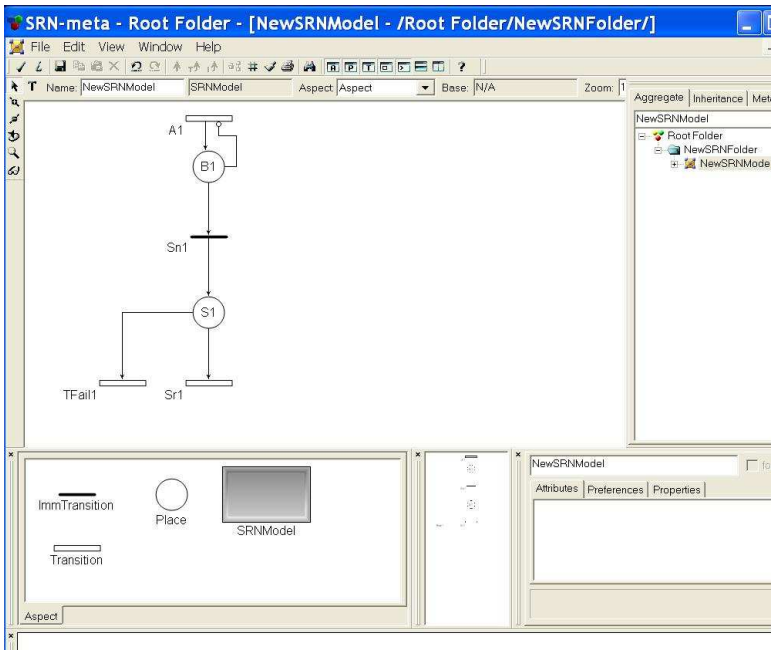

Figure 9. SRN GME Model of the reactor pattern

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>
```

The other interpreter in POSAML is the configuration interpreter, which synthesizes the configuration parameters for each pattern modeled in the system.

Model scalability is addressed using a model replicator tool we have developed previously called C-SAW (Constraint-Specification Aspect Weaver) [7]. The model replication and scalability was discussed in an earlier work of ours [8]. 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. Thus, the benefits of C-SAW are applicable across a whole spectrum of metamodels. C-SAW is thus useful to scale the models such as those modeled by POSAML and SRNML.

4 Model-driven Performance Analysis

Section 3 described the modeling languages and the generative tools we have developed to create performance models of the composed systems and to synthesize the metadata artifacts required to run the analytical model solvers, simulators and benchmarking tools. In this section we illustrate how the SRN model of the reactor pattern described in Section 2 could be applied to a case study and used for its performance analysis.

4.1 Case Study: Event-Driven Software Architecture in Mobile Handhelds

Figure 10 illustrates a typical software architecture for event demultiplexing and dispatching in mobile handhelds. Handhelds, such as PDAs, have been at the forefront of ubiquitous computing and are getting increasingly complex due to the need to support multiple applications, such as email, web browsing, calendar management, multimedia support, and games. Since these handhelds are used in many critical domains such as patient healthcare monitoring and emergency response systems they must provide exceptional performance. Further, since the software residing in these handhelds is primarily responsible for providing the desired functionality, it is necessary to analyze the software applications for their performance. In particular, in this paper we describe a methodology for the performance evaluation of such event-driven software applications, which are commonly used in ubiquitous computing.

The application mix available in the handhelds is aptly suited for event-driven systems, which requires demultiplexing and dispatching events to the correct event handlers in the handheld. For example, a user of the handheld could have set appointments in his/her calendar, which might raise an event while the user may be in the midst of web browsing. Similarly it is conceivable that email notification arrives while the user is in the midst of other applications like web browsing or listening to audio. In the context of handhelds that are tailored to provide service to emergency response personnel, one could conceive of scenarios where sensor data from a phenomenon

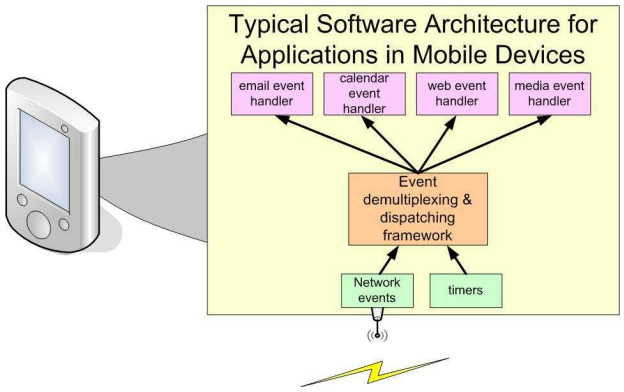

Figure 10. Event Demultiplexing in Handhelds

of interest is received by the handheld with other notifications, such as SMS notifications from the command and control, received by the handheld.

4.2 Performance Evaluation of Event-handling Software Architecture in Handhelds

This section illustrates how we used the model-generated artifacts for performance evaluation of an application mix consisting of two services, namely, calendar service and email service in handhelds. The user of the handheld device expects that he/she be notified of the incoming emails and the calendar events in a reasonable amount of time. Further, the probability of denying the service requests should be minimal. From the point of view of the service provider designing the system, the rate at which the service requests are processed or the throughput is of interest, since this will determine the quality of the product and ultimately the revenue generated for the provider.

In order to implement the mix of these two services, the reactor pattern with the characteristics described in Section 2.2 can be used to demultiplex the events. The SRN model of the reactor pattern can thus be used for the performance analysis of the services. In order to use the SRN model, we designate the email notification requests as events of type #1, and calendar update events as events of type #2.

The following listing illustrates a snippet of the model generated artifacts that drive the SPNP analytical model solver. In this case it is a C program used by the SPNP solver. Similar artifacts can be generated by our modeling tools that drive simulations and empirical benchmarks.

```
void net () {
/* Places */

place("B1");
place("B2");
place("S1");
place("S2");
place("StSnp"); init("StSnp",1);
place("SnpInProg");

/* Timed transitions */
trans("A1"); rateval("A1",lam1);
trans("A2"); rateval("A2",lam2);
trans("Sr1"); rateval("Sr1",mu1);
trans("Sr2"); rateval("Sr2",mu2);
```

```
/* Immediate transitions */
imm("Sn1"); probval("Sn1",1.0); priority("Sn1",100);
guard("Sn1",sn1);
imm("Sn2"); probval("Sn2",1.0); priority("Sn2",100);
guard("Sn2",sn2);
imm("TStSnp"); probval("TStSnp",1.0); priority("TStSnp",60);
guard("TStSnp",gd);
imm("TESnp"); probval("TESnp",1.000);
guard("TESnp",gd1); priority("TESnp",120);
```

In the design stage, it is important to assess the impact of configuration options and parameters on the performance metrics. One of the configuration options is the buffer space available to hold the incoming events of each type. This choice will have a direct impact on all the performance metrics. Most importantly, from the user's perspective, the buffer space will influence the probability of denying the service requests.

We analyze the impact of the buffer capacities on the performance measures. The values of the remaining parameters (except for the buffer capacities) are reported in Table 3. We consider two values of buffer capacities N_1 and N_2. In the first case, the buffer capacity is set to 1 for both types of events, whereas in the second case the buffer capacity of both types of events is set to 5. The performance metrics for both these cases are summarized in Table 4. Because the values of the parameters of the service requests from organization #1 (λ_1, μ_1 and N_1) are the same as the values of the parameters for the service requests from organization #2 (λ_2, μ_2, and N_2), the throughputs, queue lengths, and the loss probabilities are the same for both types of service requests for each one of the buffer capacities as indicated in Table 4. It can be observed that the loss probabilities are significantly higher when the buffer capacity is 1 compared to the case when the buffer space is 5. Also, due to the higher loss probability, the throughput is slightly lower when the buffer capacity is 1 than when the buffer capacity is 5.

For both buffer capacities, the total number of service requests from organization #2, denoted E_2, is slightly higher than the total number of requests from organization #1, denoted E_1. Because the requests from organization #1 are provided prioritized service over the requests from organization #2, on an average it takes longer to service a request from organization #2 than it takes to service a request from organization #1. This results in a higher total number of requests from organization #2 than the total number of requests from organization #1. This may result in a higher response time for service requests from organization #2 as compared with the service requests from organization #1. The reward rates to obtain the response times can be obtained using the tagged customer approach [14] and is a topic of our future research.

The performance metrics obtained from the SRN model were validated by the metrics obtained from simulation using CSIM [20]. The metrics obtained using simulation are also summarized in Table 4.

Table 3. Parameter values

Parameter	Event #1	Event #2
Arrival rate	λ_1 = 0.400/sec.	λ_2 = 0.400/sec.
Service rate	μ_1 = 2.000/sec.	μ_2 = 2.000/sec.
Failure rate	γ_1 = 0.02/sec.	γ_2 = 0.02/sec.

It is rarely the case that the values of the arrival and service rates can be estimated with accuracy in the design phase. The SRN model can also be used to analyze the sensitivity of the performance measures to the parameter values. The results of sensitivity analysis are not included in this paper due to space limitations.

Table 4. Impact of buffer capacity on performance measures

Measure	Buffer space			
	$N_1 = N_2 = 1$		$N_1 = N_2 = 5$	
	SRN	Simulation	SRN	Simulation
T_1	0.37/sec.	0.36/sec.	0.39/sec.	0.39/sec.
T_2	0.37/sec.	0.38/sec.	0.39/sec.	0.40/sec.
Q_1	0.064	0.060	0.12	0.12
Q_2	0.064	0.065	0.12	0.13
E_1	0.26	0.25	0.33	0.34
E_2	0.26	0.26	0.33	0.34
L_1	0.064	0.068	0.00024	0.00024
L_2	0.064	0.0059	0.00024	0.00024

5 Related Work

Performance and dependability analysis of some middleware services and patterns has been addressed by some researchers. Ramani *et al.* [18] develop a SRN model for the performance analysis of 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 performance 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 [21] investigated the use of aspect-oriented modeling techniques to address performance concerns that are weaved into a primary UML model of functional behavior. It has been observed that an improved separation of the performance description from the core behavior enables various design alternatives to be considerd more readily (i.e., after separation, a specific peformance concern can be represented as a variability measure that can be modified to examine the overall systemic effect). The performance 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 profiles also forces performance concerns to be specified in a manner than limits the ability to be tailored to a specific performance 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 performance modeling process. Our work falls in the category of developing domain-specific models.

6 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 performance analysis and validation of services that are composed, configured and deployment using patterns-based middleware building blocks. The performance analysis models 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 performance models for large systems. We then present a model-driven generative framework that can be used to automatically synthesize complex SRN models, simulations and empirical benchmarks for the composed systems. We demonstrate the results of solving a SRN model for the Reactor pattern and validate these results using simulations generated by our MDD approach.

Current shortcoming in our work will be addressed as we continue enhancing our framework. In particular, in future work we will extend POSAML to include the additional patterns used in network systems. This will include the ability to compose large-scale systems and addressing their correctness issues of their compositions. Moreover, we will explore the integration of the separate languages we have developed i.e., POSAML and SRNML. The goal is to use this integrated framework to drive performance 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 performance dimension addressed in this paper.

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] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[5] D. Garlan and M. Shaw. *Advances in Software Engineering and Knowledge Engineering, Volume 1, edited by V. Ambriola and G. Torotora*, chapter An Introduction to Software Architecture. World Scientific Publishing Company, New Jersey, 1993.

[6] 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.

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

[8] 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.

[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. 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.

[13] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, New Jersey, 2000.

[14] B. Melamed and M. Yadin. Randomization procedures in the computation of cumulative-timed distributions over discrete-state markov process. *Operations Research*, 32(4):926–944, July-August 1984.

[15] 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.

[16] Object Management Group. *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.
[17] 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.
[18] 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.
[19] 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.
[20] H. Schwetman. "CSIM reference manual (revision 16)". Technical Report ACA-ST-252-87, Microelectronics and Computer Technology Corp., Austin, TX.
[21] 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.
[22] 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.
[23] J. Sztipanovits and G. Karsai. Model-Integrated Computing. *IEEE Computer*, 30(4):110–112, Apr. 1997.
[24] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley, 2001.
[25] L. G. Williams and C. U. Smith. "PASA: A method for the performance assessment of software architectures". In *Proc. of the Third Intl. Workshop on Software and Performance (WOSP)*, pages 179–189, 2002.