Performance Analysis of a Middleware Demultiplexing Pattern

U. Praphamontripong, S. Gokhale
Dept. of CSE
Univ. of Connecticut
Storrs, CT 06269
ssg@engr.uconn.edu

Aniruddha Gokhale
Dept. of EECS
Vanderbilt Univ.
Nashville, TN 37235
a.gokhale@vanderbilt.edu

Jeff Gray
Dept. of CIS
U. of Alabama at Birmingham
Birmingham, AL 35294
gray@cis.uab.edu

Abstract

A key enabler of the recently adopted, assembly-centric development approach for distributed real-time software systems is QoS-enabled middleware, which provides reusable building blocks in the form of design patterns that codify solutions to commonly recurring problems. These patterns can be customized by choosing an appropriate set of configuration parameters. The configuration options of a pattern exert a strong influence on system performance, which especially for real-time systems is of paramount importance. Despite this significant influence, currently there are no techniques available to analyze performance at design time, prior to the use of a pattern in a system.

Many software systems are based on an event-driven paradigm, primarily because it fosters evolvability and composability. The event demultiplexing and dispatching capabilities that are uniform across such systems are encapsulated in the Reactor pattern, which can be used to facilitate their development. Design-time performance analysis of these event-driven systems thus requires a model of the Reactor pattern. In this paper, we present a performance model of the Reactor pattern based on the Stochastic Reward Net (SRN) modeling paradigm. We discuss how the model can be used to obtain performance metrics such as throughput, loss probability and upper and lower bounds on the response time. We illustrate the use of the model to guide the selection of configuration options and for sensitivity analysis using a case study of a handheld mobile device. We also validate the performance estimates obtained from the model using simulation.

1 Introduction

Society today is increasingly reliant on the services provided by distributed real-time software systems. These services have permeated our lives and have become prevalent in many domains including health care, finance, telecommunications and avionics. In many of these domains, the performance of a service is just as important as the functionality provided by the service.

To counter the dual pressures of developing systems which offer a rich menu of services with superior performance, while simultaneously reducing their time to market, service providers are increasingly favoring the assembly-centric approach over the traditional development-centric approach. A key facilitator of this assembly-centric approach has been *QoS-enabled middleware* [1]. Middleware consists of software layers that provide platform-independent execution semantics and reusable services that coordinate how system components are composed and interoperate. Middleware offers a large number of reusable building blocks in the form of design patterns [2, 3], which codify solutions to commonly recurring problems. These patterns can be customized with an appropriate set of configuration parameters as per system requirements.

The choice of configuration parameters have a profound influence on the performance of a pattern and hence a system implemented using the pattern. Despite the influence on system performance, which is crucial for real-time systems, current methods of selecting the patterns and their configuration options are manual, *ad-hoc* and hence error-prone. The problem is further compounded, because there are no techniques available to analyze the impact of different configuration parameters on the performance of a pattern prior to building a system. Performance analysis is thus invariably conducted after a system is assembled, and it is often too late and too expensive to take corrective action if a particular selection of patterns and their configuration parameters cannot satisfy the desired performance expectations. The capability to conduct design-time performance analysis of middleware patterns and the composition of these patterns is thus necessary, especially for systems with stringent performance requirements.

A growing number of software systems are being based

on an event-driven paradigm [4], which constitutes a provider/listener model [5]. In this paradigm, the system listens for service requests or "events" and provides the necessary services in response to these requests. These service requests may be issued by end-users or by other systems. In the latter case, an event-driven system may be viewed as a component of a larger composition of systems, or systems-of-systems. Event-driven systems provide many advantages, the most prominent ones being evolvability and composability. Evolvability is enabled by the separation of event demultiplexing and dispatching from event handling. Composability is enabled by the ability to invoke a service transparently without any knowledge of its underlying implementation. Although event handling is specific to a system, the event demultiplexing and dispatching functionality is uniform across all the systems that follow the event-driven style. The event demultiplexing and dispatching capability is codified into a middleware pattern called the *Reactor* pattern [3]. This pattern can be reused to facilitate the development of event-driven systems.

To enable design-time performance analysis of Reactor-based, event-driven systems, a model which captures the demultiplexing and dispatching functionality encapsulated by the Reactor pattern is essential. This paper describes a performance model of a Reactor-based system based on the Stochastic Reward Net (SRN) modeling paradigm [6]. We discuss how the model can be used to obtain performance metrics such as throughput, loss probability, and upper and lower bounds on the response time. We illustrate the use of the model to guide the selection of configuration options and for sensitivity analysis using a case study of a handheld mobile device. We validate the performance estimates obtained from the model using simulation.

The organization of the paper is as follows: Section 2 provides an overview of the Reactor pattern and Section 3 briefly reviews the SRN modeling paradigm. Section 4 discusses the performance analysis methodology. Section 5 illustrates the use of the methodology using a case study. Section 6 summarizes the related research. Section 7 offers concluding remarks and future research directions.

2 Overview of the Reactor pattern

Figure 1 depicts a typical event demultiplexing and dispatching mechanism documented in the Reactor pattern [3]. The system 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 system-supplied event handler. This is the idea behind the Reactor pattern, which provides synchronous event demultiplexing and dispatching capabilities.

The dynamics of the Reactor pattern can be categorized into two phases [3]:

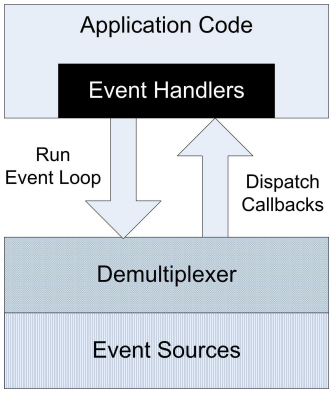


Figure 1. Event demultiplexing pattern

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:** After 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 represents 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.

3 Stochastic Reward Nets (SRNs)

In this section we provide an overview SRNs, which is the modeling paradigm used in the analysis methodology. The details of SRNs can be obtained from elsewhere [6].

A SRN is a directed graph, which contains two types of nodes: *places* and *transitions*. A directed arc connecting a place (transition) to a transition (place) is called an *input (output) arc*. Arcs are associated with a positive integer called the *multiplicity*. Places can contain *tokens* that move from one place to another through transitions. A transition is enabled when each of the places connected to it by its input arc have at least the number of tokens equal to

the multiplicity of those arcs. When an enabled transition fires, a number of tokens equal to the input arc multiplicity is removed from each of the corresponding input places, and a number of tokens equal to the output arc multiplicity is deposited in each of the corresponding output places. A SRN may also include an *inhibitor arc*, which can also have a multiplicity associated with it. An inhibitor arc inhibits the transition it is connected to if the place it is connected to at its other end has a number of tokens equal to at least its multiplicity. The state of a SRN with P places is represented by a vector $(m_1, m_2, \cdots, m_p)$ called the *marking* of the SRN, where m_i is the number of tokens in place i. A SRN marking with at least one immediate transition enabled is called a *vanishing marking*, and a marking with no immediate transitions enabled is called a *tangible marking*. A reward function may be associated with each tangible marking of a SRN. A reward function is a function of a SRN marking that accepts the number of tokens in one or more places of a SRN as input and evaluates to a real number. This function is usually designed to quantify the "value" that the system provides in a state. The tangible markings of a SRN and the rates of transition among them are equivalent to the corresponding states and state transitions of an underlying continuous time Markov chain (CTMC) [7]. Hence, a SRN can be mapped into an equivalent Markov reward model (MRM) [6], automatically using software tools such as the Stochastic Petri Net Package (SPNP) [8]. SRN models allow a concise specification of various reward functions. To extend the power of specification, a SRN may also include the specification of *enabling* (or *guard*) *functions* for each transition. The transition is enabled only if the enabling function returns one.

SRNs substantially extend the modeling power of Generalized Stochastic Petri Nets (GSPNs) [9], which are an extension of Petri nets [10]. SRNs represent a powerful modeling technique with concise specification and form closer to a designer's intuition. As a result, it is also easier to transfer the results obtained from solving the models and interpret them in terms of the entities in the modeled system. SRNs have been extensively used for performance and reliability analysis of a variety of systems including cluster systems, polling systems, and wireless networks [7].

4 Performance analysis methodology

In this section we describe the performance analysis methodology for a Reactor-based system.

4.1 System characteristics

We consider an event-driven system which uses a single-threaded, select-based Reactor with the following characteristics, as shown in Figure 2:

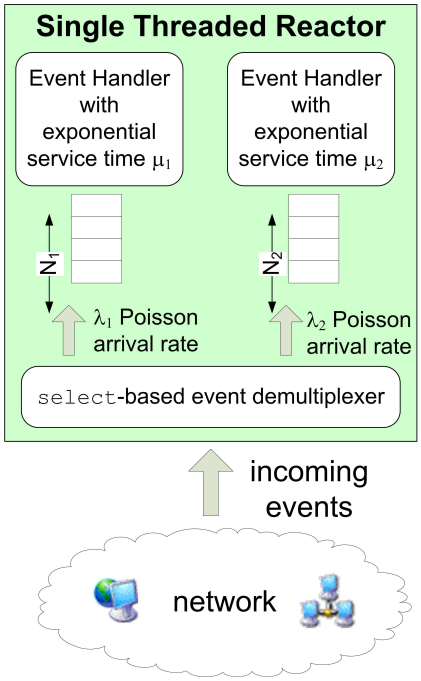


Figure 2. Characteristics of the Reactor-based system

- The system offers two types of services and hence receives two types of requests; one for each. The Reactor pattern used to implement the system thus receives two types of input events[1] with one event handler for each type of event registered with the Reactor.

- In the system, each event type has a separate queue to hold the incoming events of that type. The buffer capacities for the queues of type #1 and type #2 events are denoted N_1 and N_2, respectively. We note that event queues are not a part of the Reactor pattern itself, but are a part of the overall system.

- Event arrivals for both types of events follow a Poisson distribution with rates λ_1 and λ_2.

- 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. Thus, the order in which the events are handled is non-deterministic.

[1] Due to the one-to-one mapping between a service request and an event, we use these two terms interchangeably in the rest of the paper.

4.2 Performance metrics

The following performance metrics are of interest for each service type provided by the Reactor-based system:

- **Throughput** – which provides an estimate of the number of service requests that can be processed by the system. These estimates are important for many systems, such as telecommunications call processing.
- **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 real-time systems.
- **Probability of event loss** – which indicates how many events will have to be discarded due to the 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.
- **Response time** – which indicates the total time taken by the system to complete the service in response to a service request. In addition to obtaining an estimate of the average response time, it is also necessary to estimate the upper and the lower bounds on the response time. The upper bound estimate can be used to determine if the deadlines can be met in the worse case for real-time services.

4.3 Performance model

In this section we introduce the performance model of a Reactor-based system described in Section 4.1 using the SRN paradigm. We first describe the structural aspects of the net, followed by its dynamic evolution and finally a discussion of the assignment of the reward functions.

Description of the net: The net shown in Figure 3 is comprised of two parts. Part (a) models the arrival, queuing, and service of the two types of events. Transitions $A1$ and $A2$ represent the arrivals of the events of type #1 and #2, respectively. Places $B1$ and $B2$ represent the queues for the two types of events. Transitions $Sn1$ and $Sn2$ are immediate transitions that 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 $Sr2$ 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 place $B1$. The presence of $N1$ tokens in place $B1$ indicates that the buffer to hold the input events of type #1 is full, and no additional events can be accepted. The inhibitor arc from place $B2$ to transition $A2$ achieves the same purpose for type #2 events.

Part (b) models the process of taking successive snapshots and non-deterministic service of event handles in each snapshot as follows. 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 $S2$ 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 $S1$. Transition $Sr1$ ($Sr2$) is enabled when there is a token in place $SnpInProg1$ ($SnpInProg2$). Table 1 summarizes the enabling functions for the transitions in the net.

Dynamic evolution of the net: We explain the process of taking a snapshot and servicing the enabled event handles in the snapshot, with a scenario where there is one token each in places $B1$ and $B2$. Because there are tokens in places $B1$, $B2$ and $StSnpSht$, transitions $Sn1$ and $Sn2$ are enabled. Both of these transitions are assigned the same priority, and hence either one of them can 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 the event handle for type #1 event will be executed prior to the event handle for a type #2 event. However, if T_StSnp2 fires before T_StSnp1, then the event handle for a type #2 event will be executed prior to event handle for a type #1 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 transition T_StSnp1 fires depositing a token in place $SnpInProg1$, which enables transition $Sr1$. 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

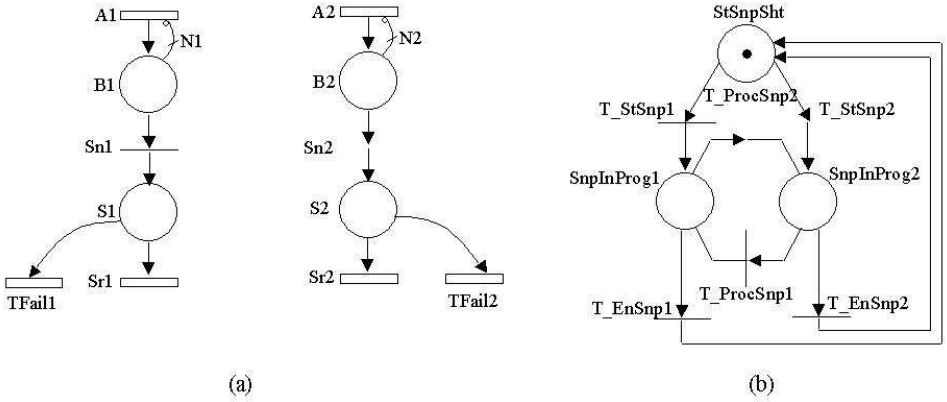


Figure 3. SRN model of the Reactor pattern

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	$((\#S1 == 1)\|\|(\#S2 == 1))?1 : 0$
$T\_ESnpSht1$	$((\#S1 == 0\&\&(\#S2 == 0))?1 : 0$
$T\_ESnpSht2$	$((\#S1 == 0\&\&(\#S2 == 0))?1 : 0$
$T\_ProcSnp1$	$((\#S1 == 1\&\&(\#S2 == 0))?1 : 0$
$T\_ProcSnp2$	$((\#S1 == 0\&\&(\#S2 == 1))?1 : 0$
$Sr1$	$(\#SnpInProg1 == 1)?1 : 0$
$Sr2$	$(\#SnpInProg2 == 1)?1 : 0$

which transition $T\_ProcSnp2$ fires and deposits a token in place $SnpInProg2$. A token in place $SnpInProg2$ enables transition $Sr2$, the firing of which 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 functions: The performance measures for event type #1 can be obtained by assigning the reward functions summarized in Table 2. These rates are obtained using the following reasoning. The throughput T_1 is given by the rate at which transition $Sr1$ fires. The queue length Q_1 is given by the number of tokens in place $B1$. The loss probability L_1 is given by the probability of $N1$ tokens in place $B1$. The reward functions to obtain the lower and the upper bounds of the response time are determined using the tagged customer approach [11], in which the trajectory of an arriving event is followed until its exit. The response time of the tagged event is determined for each possible system state when the event arrives. The expected response time is given by the weighted sum of the response times in each system state, with the weights given by the occurrence probabilities of each state.

To determine the response time of a tagged type #1 event, we define the system state as the number of tokens or markings of places $S1$, $S2$, $B1$ and $B2$. The markings of places $S1$ and $S2$ determine the progress of the ongoing snapshot when the tagged event arrives. Even if the queues are empty, this ongoing snapshot must be completed before the tagged event can be serviced. The time taken to complete the ongoing snapshot is $S1 \times \frac{1}{\mu_1} + S2 \times \frac{1}{\mu_2}$. To determine the contribution of the events in the queues to the response time, we let n_1 and n_2 denote the number of events of type #1 and #2 in the queues, when the tagged type #1 event arrives. Because there are n_1 events in the queue of type #1 events, the tagged event will be serviced after n_1 snapshots. If $n_1 \leq n_2$, the service time of the first n_1 snapshots will be $n_1 \times (1/\mu_1 + 1/\mu_2)$. In the snapshot in which the tagged event is serviced, there is a 50% chance that an event of type #2 will be serviced before the tagged event. Thus, the service time of the snapshot in which the tagged event is serviced is given by $1/\mu_1 + 0.5/\mu_2$. If $n_1 > n_2$, the service

Table 2. Reward functions for performance measures

Performance metric	Reward function
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 ($\#B2 == N2?1 : 0$)

$R_{1,l}$	if ($\#B1 < N1$) {
	if ($\#B1 < \#B2$)
	return($(1/\mu_1 * (\#S1 + \#B1 + 1) + 1/\mu_2 * (\#S2 + \#B1))$)
	else
	return($(1/\mu_1 * (\#S1 + \#B1 + 1) + 1/\mu_2 * (\#S2 + \#B2))$) }
	else return(0.0)

$R_{1,u}$	if ($\#B1 < N1$)
	return($(1/\mu_1 * (\#S1 + \#B1 + 1) + 1/\mu_2 * (\#S2 + \#B1 + 0.5))$)
	else
	return(0.0)

$R_{2,l}$	if ($\#B2 < N2$) {
	if ($\#B2 < \#B1$)
	return($(1/\mu_2 * (\#S2 + \#B2 + 1) + 1/\mu_1 * (\#S1 + \#B2))$)
	else
	return($(1/\mu_2 * (\#S2 + \#B2 + 1) + 1/\mu_1 * (\#S1 + \#B1))$) }
	else return(0.0)

$R_{2,u}$	if ($\#B2 < N2$)
	return($(1/\mu_2 * (\#S2 + \#B2 + 1) + 1/\mu_1 * (\#S1 + \#B2 + 0.5))$)
	else
	return(0.0)

time of the first n_2 snapshots will be $n_2 \times (1/\mu_1 + 1/\mu_2)$. During the first n_2 snapshots, in the best situation, no additional type #2 events will arrive. Then the service time of the $(n_2+1)^{st}$ through ${n_1}^{st}$ snapshot is $(n_1-n_2)\times 1/\mu_1$, and the service time of the snapshot in which the tagged event is serviced is $1/\mu_1$. In the worse case, however, an additional $n_1 - n_2 + 1$ events of type #2 will arrive in the first n_2 snapshots, in which case, the service time of the $(n_2 + 1)^{st}$ through ${n_1}^{st}$ snapshots will be $(n_1 - n_2) \times (1/\mu_1 + 1/\mu_2)$. The service time of the snapshot in which the tagged event is serviced is the same as in the case when $n_1 \leq n_2$. Based on this reasoning, the upper and the lower bounds of the response time of type #1 events, denoted $R_{1,u}$ and $R_{1,l}$, can be obtained using the reward functions in Table 2.

4.4 Model variations

In the SRN model of the Reactor-based system, the arrival and the service time distributions are assumed exponential to facilitate a discussion of the process of building a SRN-based model. This assumption may not hold in certain types of systems. For example, in safety-critical systems, events may occur periodically, hence the arrival process is deterministic. The arrival and service times may also follow any other non-exponential or general distributions, which can be considered in the SRN model using two methods. In the first method, a non-exponential distribution can be approximated using a phase-type approximation [6], and the resulting SRN model can then be solved using SPNP [8]. In the second method, the model can be simulated using the discrete-event simulation in SPNP [8].

5 Case study

In this section we illustrate the use of the performance analysis methodology described in Section 4 with a case study of a handheld mobile device. Figure 4 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 becoming increasingly complex due to the need to support multiple applications, such as email, web browsing, calendar management, multimedia support, and games. Because

these handhelds are used in many critical domains (e.g., patient health care monitoring and emergency response systems), they must provide exceptional performance.

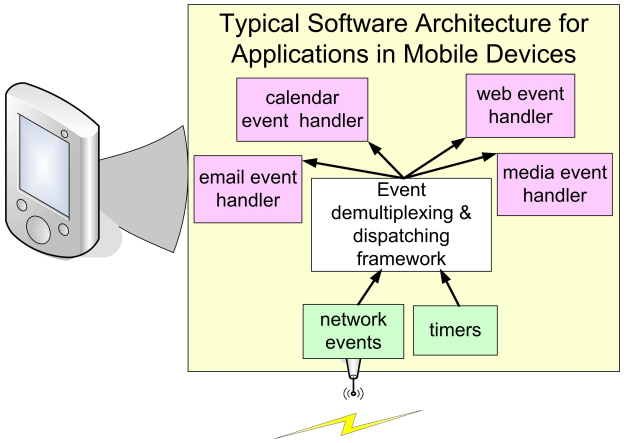


Figure 4. Event demultiplexing in handhelds

The application mix available in the handhelds is aptly suited for event-driven systems, which require 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 web browsing or listening to an MP3 song. 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 of interest is received by the handheld with other notifications, such as Short Message Service (SMS) notifications from command and control.

For the sake of illustration, we consider a handheld system which provides two services; namely, the SMS and email service. To implement this service mix, the Reactor pattern with the characteristics described in Section 4.1 can be used for demultiplexing the requests. Because of the real-time nature of SMS notifications, it is necessary to keep the response time of these notifications below an acceptable threshold. In addition, the probability of rejecting email messages and SMS notifications must be negligible. The SRN model of the Reactor-based system can guide the selection of configuration and provisioning options to achieve the performance objectives outlined above. To use the SRN model, we designate the SMS notification and email message requests as events of type #1 and #2, respectively.

We designed two experiments to demonstrate the use of the SRN model. In each experiment, performance estimates were obtained by solving the SRN model using SPNP [8]. The estimates obtained from the SRN model were also validated using simulation implemented using CSIM [12], a general purpose language used to build simulation models.

Experiment I: Impact of buffer capacity
The first experiment assesses the influence of a system configuration parameter, namely, the buffer capacity on the service performance. The allocation of buffer capacity will have a direct impact on the performance metrics, most notably on the loss probabilities of service requests.

We analyze the impact of two settings of the buffer capacities, $N_1 = N_2 = 1$ and $N_1 = N_2 = 5$ on the performance measures. The arrival rates λ_1 and λ_2 were set to 0.4/s and the service rates μ_1 and μ_2 were set to 2.0/s. The performance metrics for both of these cases are summarized in Table 3. Because the parameters for SMS service (λ_1, μ_1 and N_1) are the same as the parameters of email service (λ_2, μ_2, and N_2), the performance estimates are nearly similar for both types of services. Thus, performance estimates for only one service type are reported in Table 3.

It can be observed that the loss probability is significant when the buffer capacity is 1 and is negligible when the buffer capacity is 5. Also, due to the higher loss probability, the throughput is slightly lower when the buffer capacity is 1 and is almost identical to the arrival rate when the buffer capacity is 5. Thus, for the arrival and service rates considered, the buffer capacity must be at least 5 to ensure very low likelihood of losing service requests.

Table 3. Buffer capacity vs. performance

Measure	Buffer space			
	$N_1 = N_2 = 1$		$N_1 = N_2 = 5$	
	SRN	CSIM	SRN	CSIM
T_1	0.37/sec.	0.37/sec.	0.40/sec.	0.40
Q_1	0.064	0.0596	0.12	0.115
L_1	0.064		0.00024	
$R_{1,l}$	0.63 sec.	0.676 sec. (average)	0.79 sec.	0.830 sec. (average)
$R_{1,u}$	0.86 sec.			1.08 sec.

The results in Table 3 indicate that the throughput, loss probability and queue length obtained from the SRN model are very close to the estimates obtained from simulation. Further, the average response time estimated from simulation lies within the upper and lower bounds of the response times obtained from the SRN model. The SRN model can thus be used to determine an appropriate level of buffer provisioning for specific arrival and service rates, thereby eliminating the need to conduct lengthy simulations.

Experiment II: Impact of request arrival rates
In the early stages of the life cycle, it is rarely the case that the values of the input parameters can be estimated with cer-

tainty, which makes it imperative to analyze the sensitivity of the performance measures to the variations in the input parameters for a given choice of configuration options.

In the second experiment, we analyzed the sensitivity of the performance measures to the input parameters; namely, the arrival rates of the service requests. The parameters λ_1 and λ_2 were varied one at a time in the range 0.5/s to 1.8/s, roughly in steps of 0.025 and the expected performance measures were obtained using the SRN model and simulation. The buffer capacities were set to 5, since the results of the previous experiment indicated that the loss probabilities were negligible for this buffer capacity. The service rates were set to 2.0/s.

Figure 5 shows the performance measures as a function of λ_1. The plots in the first and second column show the measures for SMS and email services respectively. The plots in the first, second, third and fourth rows respectively show throughput, queue length, loss probability, and response time. The top-left plot indicates that the throughput of SMS notifications increases and keeps pace with the arrival rate till it is below 1.1/s. When the arrival rate exceeds 1.1/s, the throughput starts lagging the arrival rate, indicating that the system cannot service the incoming SMS notifications at the rate at which they arrive. The loss probability and the queue length increase after $\lambda_1 = 1.1$/s, indicating that the queue operates near capacity and results in a rejection of SMS notifications leading to reduced throughput. Thus, if the arrival rate of SMS notifications exceeds 1.1/s., the buffer capacity of five is not sufficient to prevent loss of requests. The loss probability increases substantially if λ_1 exceeds 1.9/s, indicating that the system cannot sustain the inflow of SMS notifications with its current event handling/service rate. Finally, referring to the last plot in the first column, the average response time of SMS notifications is closer to the lower bound for low values of λ_1 and moves closer to the upper bound as the value of λ_1 increases. The response time of SMS notifications may thus become unacceptable for the entire range of variation of λ_1 for a given service rate of 2.0/s.

The throughput and the loss probability of email messages is unaffected by an increase in λ_1. The queue length and the response time of email messages shows only a slight increase with λ_1. This slight increase occurs because the probability that a SMS notification will have to be serviced in each snapshot increases with increasing λ_1. As a result, the effective service time of email messages increases causing a rise in the queue length and the response time. The trend in the response time for email messages is similar to the response time trend for SMS notifications; i.e., closer to the lower bound for lower values of λ_1 and to the upper bound for higher values. However, the difference between the highest and the lowest response times for email messages is smaller than the difference for SMS notifications.

Similar trends, with the roles of SMS notifications and email messages reversed were observed when λ_2 was varied. Although the trends are similar, the implications are significantly different. As shown in Figure 5, when the arrival rate of SMS notifications increases, the response time provided by the system for these notifications increases and approaches the pessimistic or upper bound. As indicated earlier, the response time of SMS notifications may become unacceptable beyond a certain threshold, after which it may become necessary to improve the service rate of the event handler. When the arrival rate of email messages (λ_2) increases, the response time of SMS notifications also increases, however, the increase is much smaller compared to the increase when λ_1 increases. Thus, an increase in the response time of SMS notifications occurring due to an increase in λ_2 may be tolerated, whereas, an increase due to an increase in λ_1 may be unacceptable.

6 Related research

Performance and dependability analysis of some middleware services and patterns has been addressed by a few researchers. Aldred *et al.* [13] 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. Kahkipuro [14] propose a multilayer performance modeling framework based on UML and queuing networks for CORBA-based systems. The research reported in this paper is concerned with performance analysis of a specific design pattern used in the development of event-driven systems. The work closest to the research presented in this paper is by Ramani *et al.* [15], where a performance model of the CORBA event service (a pattern for publish/subscribe service) is developed.

To predict the quality attributes of an assembled complex system, it is necessary to compose the attributes of the components comprising the system [16]. Along these lines, our future research is focused on developing techniques to compose models of a collection of patterns to obtain the end-to-end performance of a system.

7 Conclusions and future research

In this paper we presented a performance model of the Reactor-based system, which embodies the event demultiplexing and dispatching capabilities that lie at the heart of

event-driven systems. The model is based on the Stochastic Reward Net (SRN) modeling paradigm and can be used for design-time performance analysis of Reactor-based, event-driven systems. We illustrated the model to guide the selection of configuration parameters and for sensitivity analysis with a case study of a handheld mobile device. We validated the performance estimates obtained from the model using simulation.

State-space explosion, while solving the SRN model may be an issue as the queue sizes increase. Developing model decomposition strategies, which will alleviate this problem, preferably by providing an approximate analytical solution is the topic of future research. Developing performance models of other commonly used middleware patterns, such as the Proactor and Active Object [3] and schemes for the composition of performance models corresponding to the composition of patterns is also a concern of the future.

Acknowledgments
This research was supported by the following grants from the National Science Foundation (NSF): Univ. of Connecticut (CNS-0406376 and CNS-SMA-0509271), Vanderbilt Univ. (CNS-SMA-0509296) and Univ. of Alabama at Birmingham (CNS-SMA-0509342).

References

[1] R. E. Schantz and D. C. Schmidt, "Middleware for distributed systems: Evolving the common structure for network-centric applications," in *Encyclopedia of Software Engineering*, J. Marciniak and G. Telecki, Eds. New York: Wiley & Sons, 2002.

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

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

[4] J. W. S. Liu, *Real-time Systems*. New Jersey: Prentice Hall, 2000.

[5] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River: Prentice Hall, 1996.

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

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

[8] C. Hirel, B. Tuffin, and K. S. Trivedi, "SPNP: Stochastic Petri Nets. Version 6.0," in *Proc. of Computer Performance Evaluation: Modeling Tools and Techniques, 11th Intl. Conference, Lecture Notes in Computer Science 1786*, 2000.

[9] R. A. Sahner, K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Boston: Kluwer Academic Publishers, 1996.

[10] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[11] B. Melamed and M. Yadin, "Randomization procedures in the computation of cumulative-timed distributions over discrete-state markov process," *Operations Research*, vol. 32, no. 4, pp. 926–944, July-August 1984.

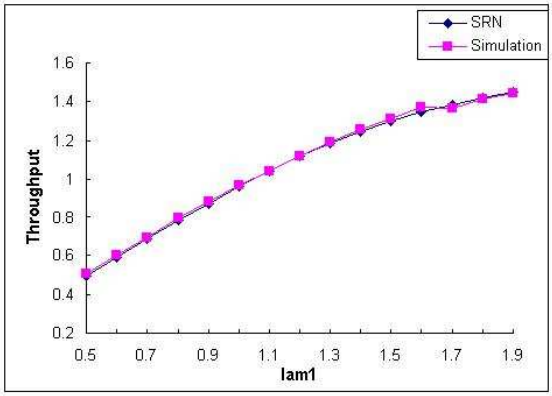
[12] H. Schwetman, "CSIM reference manual (revision 16)," Microelectronics and Computer Technology Corp., Austin, TX, Tech. Rep. ACA-ST-252-87.

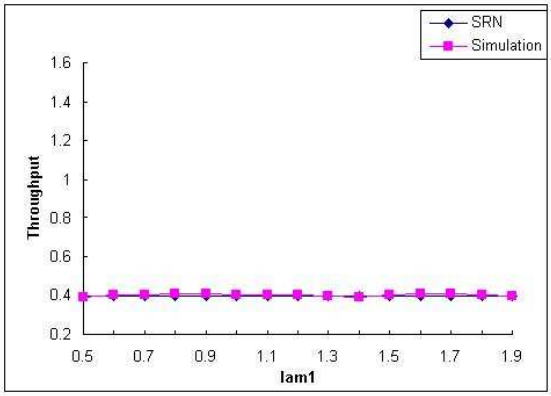
[13] L. Aldred, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "On the notion of coupling in communication middleware," in *Proc. of Intl. Symposium on Distributed Objects and Applications (DOA)*, Agia Napa, Cyprus, 2005, pp. 1015–1033.

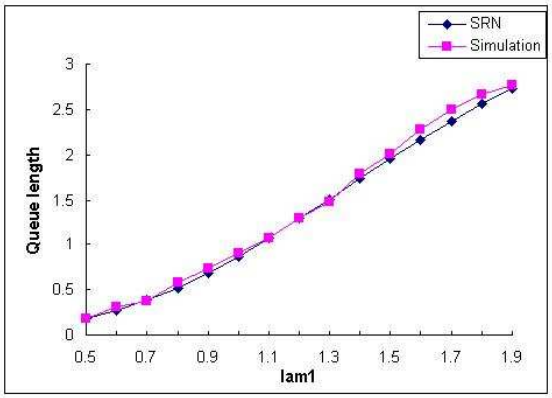
[14] P. Kahkipuro, "Performance modeling framework for CORBA based distrbuted systems," Ph.D. dissertation, Dept. of Computer Science, Univ. of Helsinki, Helsinki, Finland, May 2000.

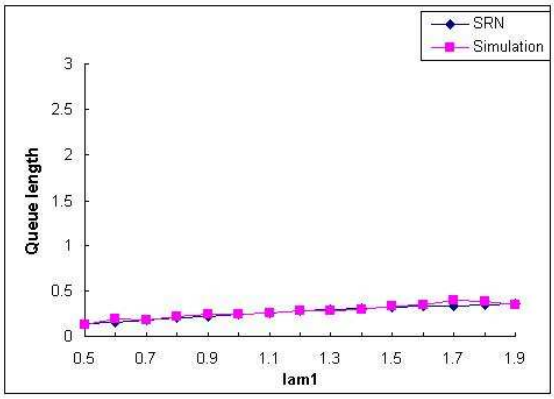
[15] S. Ramani, K. S. Trivedi, and B. Dasarathy, "Performance analysis of the CORBA event service using stochastic reward nets," in *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems*, October 2000, pp. 238–247.

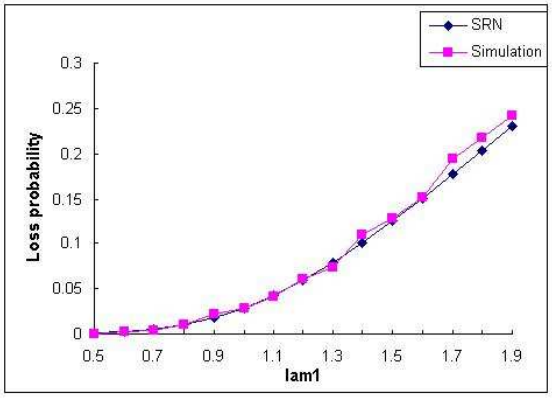
[16] I. Crnkovic, M. Larsson, and O. Preiss, "Concerning predictability in dependable component-based systems: Classification of quality attributes," in *Book on Architecting Dependable Systems III*, R. de Lemos, Ed. Springer-Verlag, 2005, pp. 257–278.

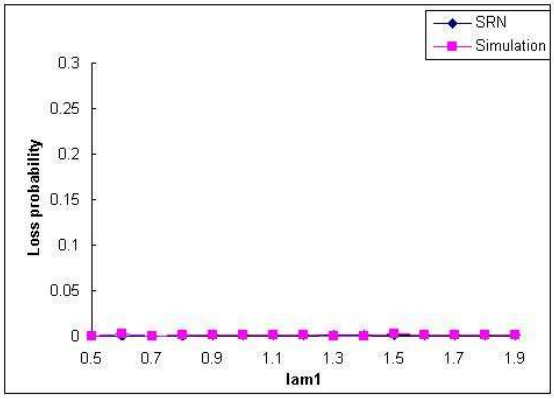


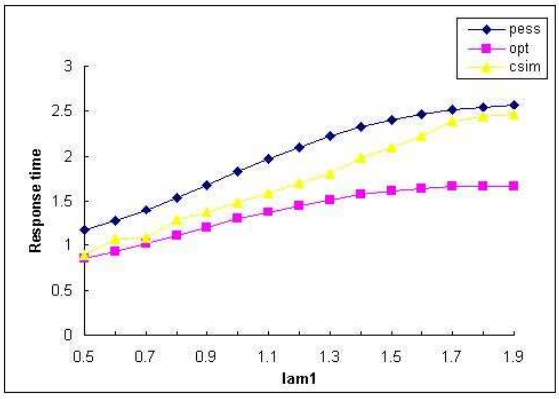












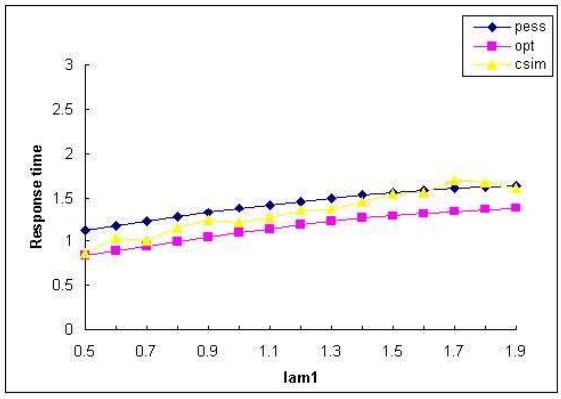


Figure 5. Sensitivity of performance measures to arrival rate λ_1