Software and Systems Modeling
The final publication is available at www.springerlink.com

Performance Modeling and Analysis of Message-oriented Event-driven Systems

Kai Sachs[1], Samuel Kounev[2], Alejandro Buchmann[3]

[1] SAP AG, Germany

[2] Karlsruhe Institute of Technology, Germany

[3] TU Darmstadt, Germany

Abstract Message-oriented event-driven systems are becoming increasingly ubiquitous in many industry domains including telecommunications, transportation and supply chain management. Applications in these areas typically have stringent requirements for performance and scalability. To guarantee adequate quality-of-service, systems must be subjected to a rigorous performance and scalability analysis before they are put into production. In this paper, we present a comprehensive modeling methodology for message-oriented event-driven systems in the context of a case study of a representative application in the supply chain management domain. The methodology, which is based on queueing Petri nets, provides a basis for performance analysis and capacity planning. We study a deployment of the SPECjms2007 standard benchmark on a leading commercial middleware platform. A detailed system model is built in a step-by-step fashion and then used to predict the system performance under various workload and configuration scenarios. After the case study, we present a set of generic performance modeling patterns that can be used as building blocks when modeling message-oriented event-driven systems. The results demonstrate the effectiveness, practicality and accuracy of the proposed modeling and prediction approach.

1 Introduction

Message-Oriented Middleware (MOM) is often used as a communication mechanism for asynchronous data exchange in loosely-coupled event-driven applications such as event-driven supply chain management, transport information monitoring, and ubiquitous sensor-rich applications to name just a few [1]. With their increasing adoption in mission-critical areas, the performance and scalability of such systems are becoming a major concern. To ensure adequate Quality-of-Service (QoS), it is essential that applications are subjected to a rigorous performance and scalability analysis as part of their software engineering lifecycle.

However, the decoupling of communicating parties in event-driven applications makes it difficult to predict their behavior under load and ensure that enough resources are available to meet QoS requirements. Application developers and deployers are often faced with questions such as: What performance will the application exhibit for a given deployment topology, configuration and workload scenario? What will be the expected message delivery latency as well as the utilization of the various system components? What maximum load (number of clients, messaging rates) will the system be able to handle without breaking the service level agreements (SLAs)? Which components will be most utilized as the load increases and are they potential bottlenecks? What influence do transactional and persistent messages have on the system behavior? To answer such questions, techniques for predicting the application performance as a function of its configuration and workload are needed. Common performance metrics of interest are the expected event notification latency as well as the utilization and message throughput of the various system components (e.g., event brokers, network links). Such techniques are essential in order to ensure that systems are designed and sized to provide adequate QoS to applications at a reasonable cost.

While numerous techniques for performance prediction of conventional distributed systems exist in the literature, few techniques specialized for message-oriented event-driven systems have been proposed. Most existing techniques suffer from simplifying assumptions limiting their practical applicability and do not consider important system aspects that occur in realistic applications such as different communication patterns, multiple message types and message persistence. In this paper, we present a comprehensive modeling methodology for message-oriented event-driven systems in the context of a case study of a representative event-driven application deployed on a leading commercial MOM platform. The application we study is the SPECjms2007 standard benchmark[1] which is based on a novel scenario in the supply chain management domain designed to be representative of real-world event-driven applications. The benchmark was developed by SPEC's Java Subcommittee with the participation of IBM, Sun, Oracle, BEA Systems, Sybase, Apache, JBoss and TU Darmstadt. The benchmark workload comprises a set of supply chain interactions between a supermarket company, its stores, its distribution centers and its suppliers. The interactions represent a complex transaction mix exercising both point-to-point and publish/subscribe messaging including one-to-one, one-to-many and many-to-many communication [2]. The benchmark covers the major message types used in practice including messages of different sizes and different delivery modes, i.e., persistent vs. non-persistent, transactional vs. non-transactional. The generated interaction mix can be configured to represent different types of customer workloads.

In this paper, we use SPECjms2007 as a representative application in order to evaluate the effectiveness of our performance modeling technique when applied to a realistic system under different types of event-driven workloads typically used in practice. The reader is introduced to the proposed modeling abstractions showing how the various types of messaging workloads can be modeled. A "learning by example" approach is followed presenting the models in the context of a real-life application to ease understanding. The modeling approach itself is general and, once understood, it can be easily applied to other applications.

The paper starts with a brief introduction to MOM and queueing Petri nets (QPNs) [3] which are used as modeling formalism. Following this, a detailed model of the SPECjms2007 benchmark is built in a step-by-step fashion. QPNs make it possible to accurately model the dissemination of messages in the system which involves forking of asynchronous tasks. The developed model is used to predict the benchmark performance for a number of different workload and configuration scenarios. Model predictions are compared against measurements on the real system and the results are used to evaluate the effectiveness, practicality and accuracy of the proposed modeling and prediction approach. Finally, a set of generic performance modeling patterns are presented that address the various messaging scenarios and workloads that occur in practice.

The contributions of the paper are twofold:

1. *Conceptually*, we present a comprehensive modeling approach and a set of modeling patterns reflecting the needs of realistic applications. Further, we extend the QPN formalism simplifying the abstractions for modeling logical software entities such as message destinations (queues and topics).
   – More specifically, QPNs are extended to support multiple queueing places that share the same physical queue.
   – A flexible mapping of logical to physical resources that makes it easy to customize the model to a specific deployment of the application is introduced.
2. *Practically*, we present a novel case study of a complex and realistic application deployed on a representative MOM platform.
   – Both point-to-point and publish/subscribe messaging are considered as well as multiple message types, different message sizes and different message delivery modes.
   – An extensive evaluation of the accuracy of the modeling approach is presented considering the typical types of workloads used in practice.

Both analytical and simulation techniques for solving QPN models exist including product-form solution techniques and approximation techniques [4–6]. For the scenarios in the paper, we used simulation since we considered very large scenarios. For smaller scenarios analytical techniques can be used. The research value of the proposed modeling approach is that it presents a set of adequate abstractions for messaging applications that have been validated and shown to provide a good balance between modeling effort, analysis overhead and accuracy. Developing a simulation model using a general-purpose simulation language is a time-consuming and error-prone task, and there is no guarantee that the resulting model will provide the required accuracy at reasonable cost (simulation time). The abstractions we propose do not require any programming, they are compact yet expressive, and provide good accuracy at low cost.

To the best of our knowledge, no models of representative event-based systems of the size and complexity of the one considered here exist in the literature. The case study we present in this paper is the first comprehensive validation of our modeling approach. By means of the proposed models we were able to predict the performance of the modeled application accurately for scenarios under realistic load conditions with up to 30,000 messages exchanged per second (up to 4,500 transaction p. sec.). The presented modeling technique can be

[1] SPECjms2007 is a trademark of the Standard Performance Evaluation Corporation (SPEC). The results or findings in this publication have not been reviewed or accepted by SPEC, therefore no comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjms2007 is located at http://www.spec.org/osg/jms2007.

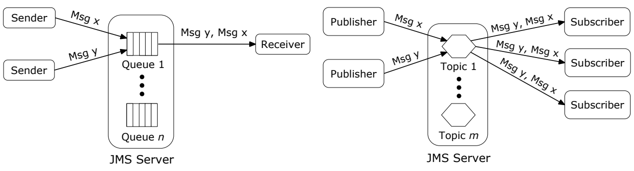


Fig. 1 Point-to-Point vs. Pub/Sub Messaging

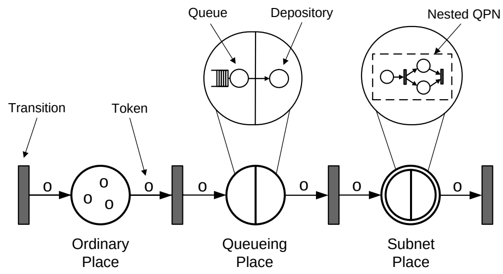


Fig. 2 QPN Notation

exploited as a tool for performance prediction and capacity planning during the software engineering lifecycle of event-driven applications.

The rest of the paper is organized as follows. In Section 2, we provide a brief introduction to MOM and an overview of QPNs. In Section 3, we introduce our modeling technique by showing how it can be used to model the SPECjms2007 application. We then present a detailed experimental evaluation of the accuracy of the proposed technique in Section 4. Following this, in Section 5, we introduce our performance modeling patterns presenting three of them in detail. In Section 6, we survey related work in the area of performance analysis of message-oriented event-driven systems. Finally, the paper is wrapped up with some concluding remarks and a discussion of future work in Section 7. Appendix A provides a detailed introduction to QPNs, while Appendix B provides detailed specifications of the QPN models used in the three selected modeling patterns that are presented in detail.

2 Background

2.1 Message-Oriented Middleware (MOM)

Modern event-driven systems are typically implemented using Message-Oriented Middleware which provides support for loosely-coupled communication among distributed software components by means of asynchronous message-passing as opposed to a request/response metaphor. The MOM acts as an intermediary between communicating parties receiving messages from one or more message producers and delivering them to possibly multiple message consumers.

Most of the MOM platforms currently used in industry (e.g., IBM WebSphere MQ, TIBCO EMS) support the Java Message Service (JMS) [7] standard interface for accessing MOM services. The JMS interface provides two messaging models: *point-to-point (P2P)* and *publish/subscribe (pub/sub)*. Point-to-point messaging is built around the concept of a message *queue* which forms a virtual communication channel. Each message is sent to a specific queue and is retrieved and processed by a single consumer. Pub/sub messaging, on the other hand, is built around the concept of a *topic*. Each message is sent to a specific topic and it may be delivered to multiple consumers interested in the topic. Consumers are required to register by subscribing to the topic before they can receive messages. Consumers can additionally specify filters (selectors) on the messages delivered to the topic. In the pub/sub domain, message producers are referred to as *publishers* and message consumers as *subscribers*. JMS queues and topics are commonly referred to as *destinations*. The two messaging models are depicted in Figure 1. The JMS specification defines several modes of message delivery with different quality-of-service attributes:

Non-Persistent vs. Persistent: In non-persistent mode, pending messages are kept in main memory buffers while they are waiting to be delivered and are not logged to stable storage. In persistent mode, the JMS provider takes extra care to ensure that no messages are lost in case of a server crash. This is achieved by logging messages to persistent storage such as a database or a file system. Most JMS vendors provide their own file storage implementation as well as a JDBC interface.

Non-Durable vs. Durable: JMS supports two types of subscriptions, durable and non-durable. With non-durable subscriptions a subscriber will only receive messages that are published while he is active. In contrast to this, durable subscriptions ensure that a subscriber does not miss any messages during periods of inactivity.

Non-Transactional vs. Transactional: A JMS messaging session can be transactional or non-transactional. A transaction is a set of messaging operations that are executed as an atomic unit of work.

For a detailed introduction to MOM and JMS the reader is referred to [7–9].

2.2 Queueing Petri Nets (QPNs)

Queueing Petri Nets (QPNs) [3] can be seen as an extension of stochastic Petri nets that allow *queues* to be integrated into the places of a Petri net. A place that contains an integrated queue is called a *queueing place* and is normally used to model a system resource, e.g., CPU, disk drive or network link. Tokens in the Petri net are used to model requests or transactions processed by the system. In our case, tokens represent the messages processed by the MOM server. Arriving tokens at a queueing place are first served at the queue and then they become available for firing of output transitions. When a transition fires, it removes tokens from some places and creates tokens at others. Usually, tokens are moved between places representing the flow-of-control during message processing. QPNs also support so-called *subnet places* that contain nested QPNs. Figure 2 shows the notation used for ordinary places, queueing places and subnet places. A detailed introduction to QPNs is included in Appendix A.

As demonstrated in [10], QPNs provide greater modeling power and expressiveness than conventional queueing network models and stochastic Petri nets. Taking advantage of this, our approach provides several important benefits. First of all, QPN models allow the modeling of process synchronization and the integration of hardware and software aspects of system behavior [10, 11]. Second, the use of QPNs makes it possible to accurately model the dissemination of messages in the system which involves forking of asynchronous tasks. Finally, by restricting ourselves to QPN models, we can exploit the knowledge of their structure and behavior for fast and efficient analysis using simulation [6].

3 Modeling Methodology

We now present our modeling methodology based on Queueing Petri Nets (QPNs) by showing how it can be applied to model a deployment of the SPECjms2007 benchmark as a representative example of a realistic message-oriented event-driven system. We follow a "learning by example" approach presenting our methodology in the context of a real-life application to ease understanding. As mentioned earlier, the modeling methodology itself is general and, once understood, it can be easily applied to other applications. To make the paper self-contained, we start by presenting a brief overview of SPECjms2007. A detailed description of the benchmark, including a comprehensive workload characterization showing how the workload can be customized, can be found in [2, 9].

3.1 Scenario - SPECjms2007

The SPECjms2007 benchmark is based on a novel application scenario modeling the supply chain of a supermarket company where RFID technology is used to track the flow of goods. The participants involved can be grouped into the following four roles:

1. *Supermarkets (SMs)* that sell goods to end customers,
2. *Distribution Centers (DCs)* that supply the supermarket stores,
3. *Suppliers (SPs)* that deliver goods to the distribution centers and
4. *Company Headquarters (HQ)* responsible for managing the accounting of the company.

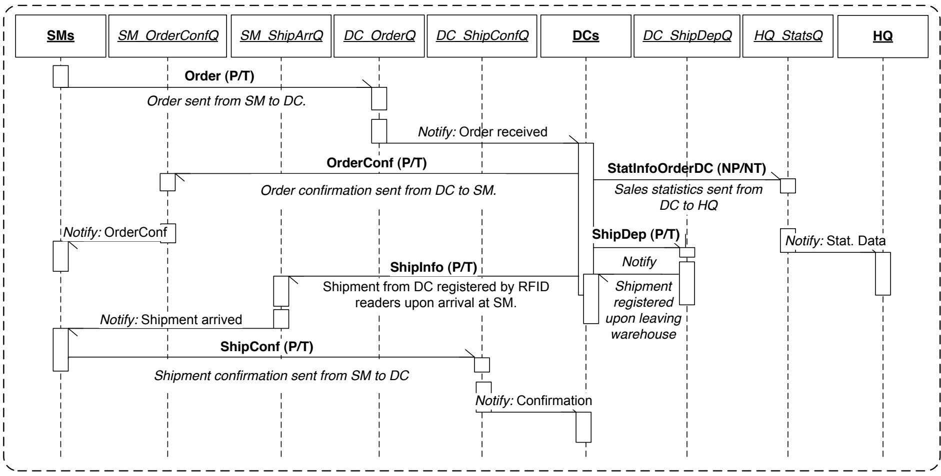
SPECjms2007 implements seven interactions between the participants in the supply chain:

1. Order/shipment handling between SM and DC
2. Order/shipment handling between DC and SP
3. Price updates sent from HQ to SMs
4. Inventory management inside SMs
5. Sales statistics sent from SMs to HQ
6. New product announcements sent from HQ to SMs
7. Credit card hot lists sent from HQ to SMs

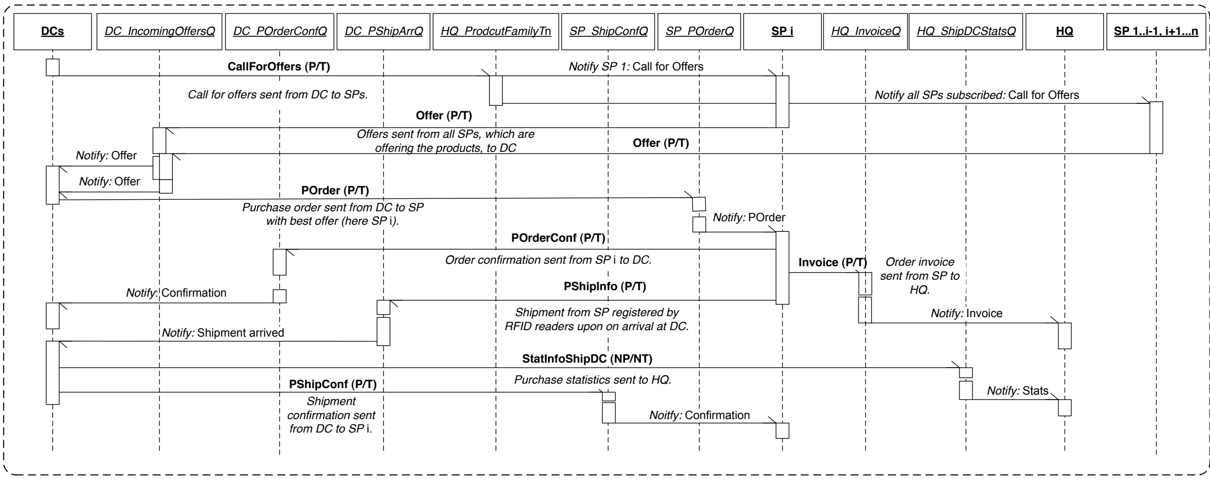
The workflow of the seven interactions is shown in Figure 3. Interactions 1, 4 and 5 exercise point-to-point messaging whereas Interactions 3, 6 and 7 exercise pub/sub messaging. A brief description of Interaction 2, which includes both point-to-point and pub/sub messaging, illustrates the complexity of the workload. The interaction is triggered when goods in a DC are depleted and the DC has to order from a SP to refill stock: i) A DC sends a call for offers to all SPs that supply the required types of goods, ii) SPs send offers to the DC, iii) The DC selects a SP and sends a purchase order to it, iv) The SP ships the ordered goods sending a confirmation and an invoice, v) The shipment is registered by RFID readers upon entering the DC's warehouse, vi) The DC sends a delivery confirmation to the SP, vii) The DC sends transaction statistics to the HQ. The call for offers sent in the beginning is addressed to a topic $HQ\_ProductFamily{<}n{>}T$ where n is the product family.

SPECjms2007 is implemented as a Java application comprising multiple JVMs and threads distributed across a set of *client nodes*. For every destination, there is a separate Java class called *Event Handler (EH)* that encapsulates the application logic executed to process messages sent to that destination. Event handlers register as listeners for queues/topics and receive call backs from the messaging infrastructure as new messages arrive. In addition to the event handlers, for every physical location, a set of threads (referred to as *driver threads*) is launched to drive the benchmark interactions that are logically started at that location.

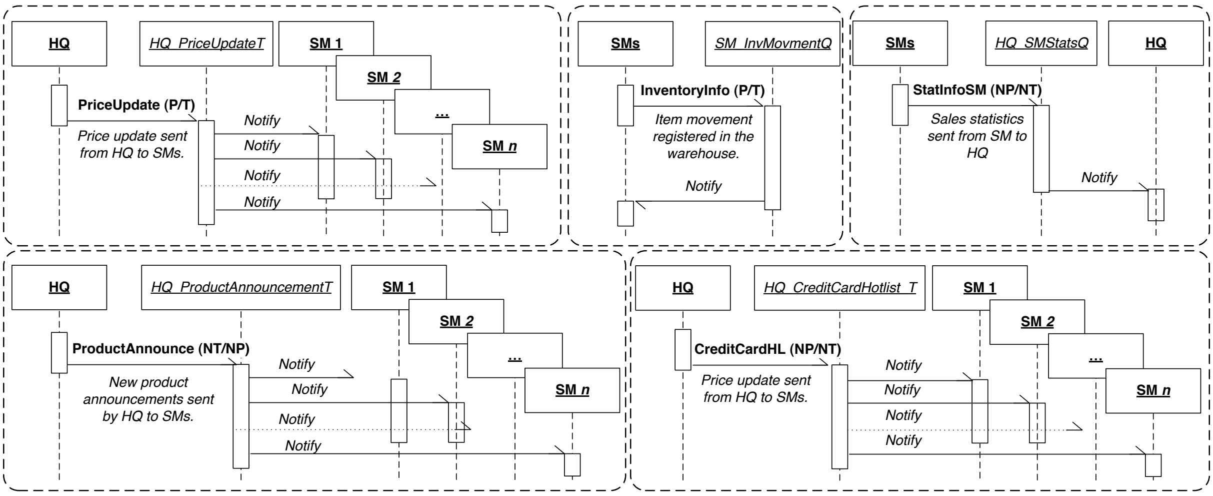
SPECjms2007 offers three different modes of running the benchmark providing different levels of configurability: *horizontal*, *vertical* and *freeform*. The modes are referred to as *workload topologies*. The horizontal topology is meant to exercise the ability of the system to handle increasing message traffic injected through increasing number of destinations. To this end, the workload is scaled by increasing the number of physical locations (SMs, DCs, etc) while keeping the traffic per location constant. The vertical topology, on the other hand, is



(a) Interaction 1



(b) Interaction 2



(c) Interactions 3 to 7

Fig. 3 Workflow of the SPECjms2007 Interactions: (N)P=(Non-)Persistent, (N)T=(Non-)Transactional

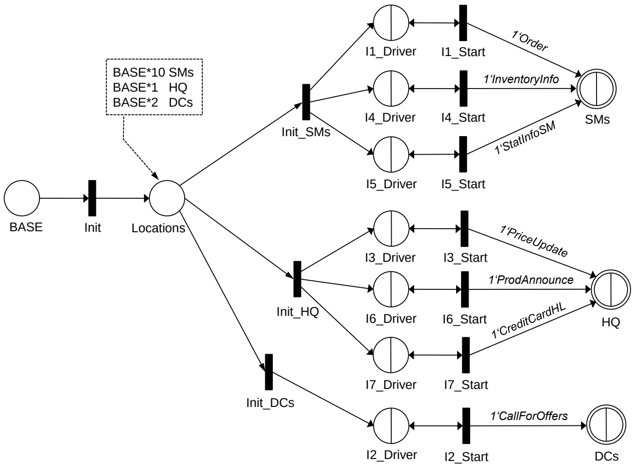


Fig. 4 Model of Interaction Drivers

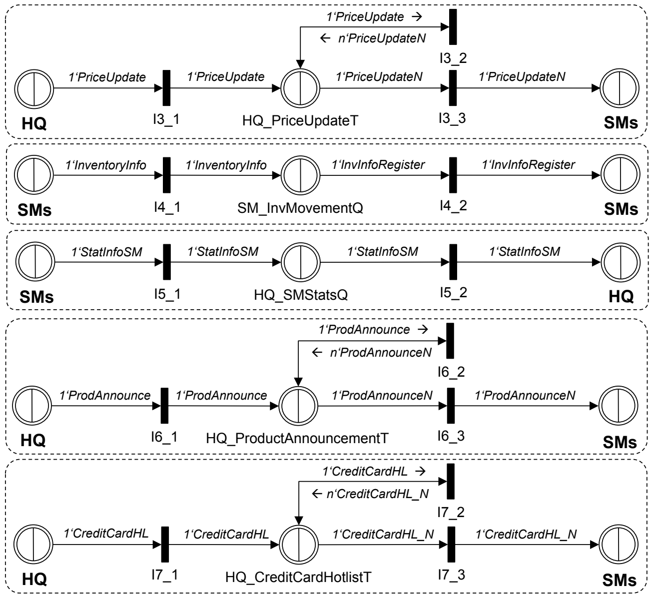


Fig. 5 Models of Interactions 3, 4, 5, 6 and 7

meant to exercise the ability of the system to handle increasing message traffic through a fixed set of destinations. Therefore, a fixed set of physical locations is used and the workload is scaled by increasing the rate at which interactions are run. Both in the horizontal and vertical topology, a single parameter called $BASE$ determines the overall target message traffic and is used as a scaling factor. Finally, the freeform topology allows the user to design his own workload scenario that stresses selected features of the MOM infrastructure in a way that resembles a given target customer workload.

We now show in a step-by-step fashion how the various components of the SPECjms2007 benchmark can be modeled using QPNs. Given the size and complexity of the modeled system, the resulting performance model is much larger and more complex than existing queueing models of message-oriented event-based systems (see Section 6). Overall, the presented model contains a total of 59 queueing places, 76 token colors and 68 transitions with a total of 285 firing modes. Transition and service rates as well as routing probabilities are derived from the workload description published in [2]. For the sake of compactness of the presentation, in the following, we focus on the most important aspects that are relevant to understanding and applying our modeling methodology.

3.2 Modeling Interaction Drivers

We start by building a model of the interaction drivers. For illustration, we assume that the Vertical topology is used. The QPN model we propose is shown in Figure 4. The number of tokens configured in the initial marking of place `BASE` is used to initialize the $BASE$ parameter of the Vertical topology. Transition `Init` fires a single time for each token in place `BASE` destroying the token and creating a respective number of tokens `10'SMs`, `1'HQ` and `2'DCs` in place `Locations`. This results in the creation of the expected number of location drivers specified by the Vertical topology. The location tokens are used to initialize the interaction drivers by means of transitions `Init_SMs`, `Init_HQ` and `Init_DCs`. For each driver, a single token is created in the respective queueing place `Ix_Driver` of the considered interaction. Places `Ix_Driver, x=1..7` each contain a $G/M/\infty/IS$ queue which models the triggering of the respective interaction by the drivers. When a driver token leaves the queue of place `Ix_Driver`, transition `Ix_Start` fires. This triggers the respective interaction by creating a token representing the first message in the interaction flow. The message token is deposited in one of the subnet places SMs, HQ or DCs depending on the type of location at which the interaction is started. Each subnet place contains a nested QPN which may contain multiple queueing places modeling the physical resources at the individual location instances. When an interaction is triggered, the driver token is returned back to the queue of the respective `Ix_Driver` place where it is delayed for the time between two successive triggerings of the interaction. The mean service time of each $G/M/\infty/IS$ queue is set to the reciprocal of the respective target interaction rate as specified by the Vertical topology. Customizing the model for the Horizontal or Freeform topology is straightforward. The number of location driver tokens generated by the `Init` transition and the service time distributions of the $G/M/\infty/IS$ queues have to be adjusted accordingly.

For the sake of compactness of the presentation, the models we present here have a single token color for each message type. In reality, we used three separate token colors for each message type representing the three different message sizes (small, medium and large) modeled by the benchmark, i.e., instead of `InventoryInfo` we have `InventoryInfo_S`, `InventoryInfo_M` and `InventoryInfo_L`.

The only exception is for the `PriceUpdate` messages of Interaction 3 which have a fixed message size. With exception of `I3_Start`, each transition `Ix_Start` on Figure 4 has three firing modes corresponding to the three message sizes. The transition firing weights reflect the target message size distribution.

3.3 Modeling Interaction Workflows

We now model the interaction workflows. We start with Interactions 3 to 7 since they are simpler to model. Figure 5 shows the respective QPN models. For each destination (queue or topic) a subnet place containing a nested QPN (e.g., `SM_InvMovementQ`, `HQ_PriceUpdateT`) is used to model the MOM server hosting the destination. The nested QPN may contain multiple queueing places modeling resources available to the MOM server, e.g., network links, CPUs and I/O subsystems. We briefly discuss the way Interaction 3 is modeled. It starts by sending a `PriceUpdate` message (transition `I3_1`) to the MOM server. This enables transtion `I3_2` which takes as input the `PriceUpdate` message and creates n `PriceUpdateN` messages representing the notification messages delivered to the subscribed SMs (where $n = 10$ for the Vertical topology). Each of these messages is forwarded by transition `I3_3` to place `SMs` representing the machine hosting the SMs. Interactions 4 to 7 are modeled similarly.

We now look at Interactions 1 and 2 whose models are shown in Figures 6 and 7, respectively. The workflow of the interactions can be traced by following the transitions in the order of their suffixes, i.e., `I1_1`, `I1_2`, `I1_3`, etc. In Interaction 2, the `CallForOffers` message is sent to a `HQ_ProductFamily<n>T` topic where n represents the respective product family. The `CallForOffers` message is then transformed to x `CallForOffersN` messages representing the respective notification messages forwarded to the SPs (transition `I2_2_FindSubscribers`). Each SP sends an offer (`Offer` message) to the DC and one of the offers is selected by transition `I2_6` which takes the x offers as input and generates a purchase order (`POrder` message) sent to the `SP_POrderQ` queue. The rest of the workflow is similar to Interaction 1.

3.4 Mapping of Logical to Physical Resources

By using subnet places to represent the MOM server(s) hosting the individual destinations and the clients (HQ, SMs, DCs and SPs) exchanging messages through the MOM infrastructure, we provide flexibility in choosing the level of detail at which the system components are modeled. Each subnet place is bound to a nested QPN that may contain multiple queueing places representing logical system resources available to the respective client or server components, e.g., CPUs, disk subsystems

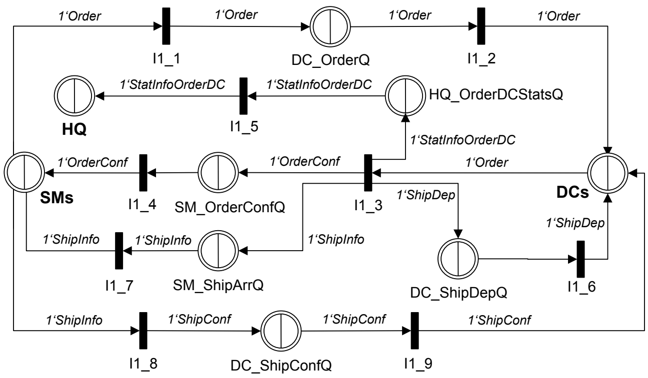


Fig. 6 Model of Interaction 1

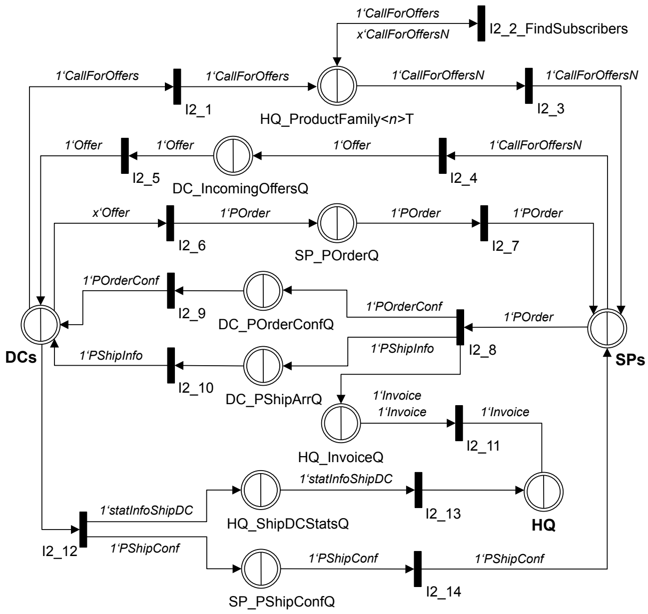


Fig. 7 Model of Interaction 2

and network links. The respective physical system resources are modeled using the queues inside the queueing places. Multiple queueing places can be mapped to the same physical queue. For example, if all destinations are deployed on a single MOM server, their corresponding queueing places should be mapped to a set of central queues representing the physical resources of the MOM server. Similarly, if locations of the same type are deployed on the same client machine, a single set of physical queues modeling the client machine should be shared among the queueing places corresponding to the individual locations. The hierarchical structure of the model not only makes it easy to understand and visualize, but most importantly, it provides flexibility in mapping logical resources to physical resources and thus makes it easy to customize the model to a specific deployment of the benchmark.

3.5 QPN Extensions and Tool Support

We employed the QPME tool (Queueing Petrinet Modeling Environment) [12, 13] to build and analyze the QPN models of the benchmark interactions. QPME is an open-source tool providing a QPN editor for constructing QPN models and an optimized simulation engine SimQPN [6] for model analysis. SimQPN has already been successfully used in multiple modeling studies of significant size and complexity and has been shown to scale well to large realistic systems. For an evaluation of the scalability and efficiency of the tool we refer the reader to [14] and [15]. An essential QPN feature required in order to realize the flexible mapping of logical to physical resources described in the previous section is the ability to have multiple queueing places configured to share the same physical queue. This feature is not supported by standard QPN models and is an extension that we introduced while conducting the case study presented in this paper. While the same effect can be achieved by using multiple subnet places mapped to the same nested QPN containing a single queueing place, this would require expanding tokens that enter the nested QPN with a *tag* to keep track of their origin as explained in [10]. However, currently available QPN modeling tools including QPME do not support this feature which means that the modeler would have to manage the tags manually which is cumbersome and error-prone. Thus, the extension we propose is much simpler and significantly reduces the modeling effort for managing shared queues. In the latest version of QPME, queues are defined centrally (similar to token colors) and can be referenced from inside multiple queueing places. This allows to use queueing places to represent software entities, e.g., software components, which can then be mapped to different hardware resources modeled as queues. The introduced QPN extension, combined with the support for hierarchical QPNs, allows to build multi-layered models of software architectures similar to the way this is done in layered queueing networks, however, with the advantage that QPNs enjoy all the benefits of Petri nets for modeling synchronization aspects.

The case study presented in this paper exploits the ability to share queues in multiple queueing places by decoupling the software and hardware layers of the modeled system allowing the same logical model of the benchmark interactions to be easily customized to different deployment environments. Thus, the results presented in the paper can also be seen as a validation of the introduced extensions to our general-purpose modeling tools and analysis techniques.

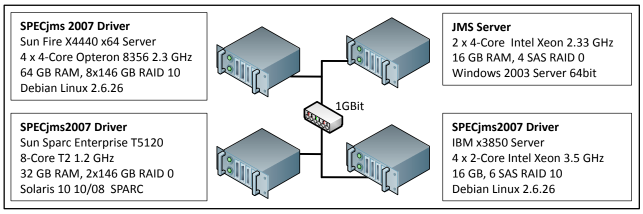


Fig. 8 Experimental Environment

4 Experimental Evaluation

4.1 Experimental Environment

To evaluate the accuracy of the proposed modeling approach, we conducted an experimental analysis of the modeled application in the environment depicted in Figure 8. A leading commercial MOM platform was used as a JMS server installed on a machine with two quad-core Intel Xeon 2.33 GHz CPUs and 16 GB of main memory. The server was run in a 64-bit 1.5 JVM with 8GB of heap space. A RAID 0 disk array comprised of four disk drives was used for maximum performance. The JMS Server was configured to use a file-based store for persistent messages with a 3.8 GB message buffer. The SPECjms2007 drivers were distributed across three machines: i) one Sun Fire X4440 x64 server with four quad-core Opteron 2.3 GHz CPUs and 64 GB of main memory, ii) one Sun Sparc Enterprise T5120 server with one 8-core T2 1.2 GHz CPU and 32 GB of main memory and iii) one IBM x3850 server with four dual-core Intel Xeon 3.5 GHz CPUs and 16 GB of main memory. All machines were connected to a 1 GBit network.

4.2 Model Adjustments

The first step was to customize the model to our deployment environment. The subnet place corresponding to each destination was mapped to a nested QPN containing three queueing places connected in tandem. The latter represent the network link of the MOM server, the MOM server CPUs and the MOM server I/O subsystem, respectively. Given that all destinations are deployed on a single physical server, the three queueing places for each destination were mapped to three central queues representing the respective physical resources of the JMS server. The CPUs were modeled using a $G/M/n/PS$ queue where n is the number of CPU cores (in our case $n = 8$). The network and I/O subsystem were modeled using $G/M/1/FCFS$ queues. The mean message service times at the queues were set according to the message resource demands. The latter were estimated by running the interactions in isolation and measuring the utilization of the respective resources using OS tools. For interactions consisting of multiple messages, the service demands of the individual messages were estimated by con-

Table 1 Service Demands in ms

			CPU			**Disk IO**		
		Place	**Size 1**	**Size 2**	**Size 3**	**Size 1**	**Size 2**	**Size 3**
Intr.	**Message**	*Probability*	*95 %*	*4 %*	*1 %*	*95 %*	*4 %*	*1 %*
1	orderConf	SM_OrderConfQn	0.973	0.987	1.846	0.081	0.067	0.146
	statInfoOrderDC	HQ_StatsQn	0.053	0.112	0.242	*na*		
	shipInfo	SM_ShipArrQn	0.616	1.170	2.501	0.051	0.080	0.198
	shipDep	DC_ShipDepQn	0.539	1.148	2.494	0.045	0.078	0.198
	order	DC_OrderQn	0.838	0.948	1.833	0.065	0.069	0.145
	shipConf	DC_ShipConfQn	0.390	0.365	0.663	0.032	0.025	0.053
2	callForOffers	HQ_ProductFamilyTn	0.343	0.403	0.946	0.045	0.077	0.117
	callForOffers Notification	HQ_ProductFamilyTn	0.130	0.153	0.359	0.017	0.029	0.044
	offer	DC_IncomingOffersQn	0.452	0.831	1.945	0.033	0.056	0.176
	pOrder	SP_POrderQn	0.921	1.097	2.580	0.121	0.209	0.318
	pShipConf	SP_ShipConfQn	0.406	0.500	0.873	0.066	0.078	0.108
	statInfoShipDC	HQ_ShipDCSTatsQn	0.053	0.112	0.242	*na*		
	pOrderConf	DC_POrderConfQn	1.025	1.090	2.504	0.134	0.208	0.309
	invoice	HQ_InvoiceQn	0.842	0.882	2.018	0.110	0.168	0.249
	pShipInfo	DC_PShipArrQn	0.485	0.403	0.872	0.064	0.077	0.108
3	priceUpdate	HQ_PriceUpdateT	0.501			0.118		
	priceUpdate Notification	HQ_PriceUpdateT	0.458			0.027		
4	inventoryInfo	SM_InvMovementQn	0.895	1.447	2.985	0.068	0.140	0.267
5	statInfoSM	HQ_SMStatsQ	0.444			*na*		
6	productAnnouncement	HQ_ProductAnnouncementT	0.164	0.177	0.168	*na*		
	productAnnouncement Notification	HQ_ProductAnnouncementT	0.034	0.024	0.177	*na*		
7	creditCardHL	HQ_CreditCardHotlistT	0.096	0.364	0.430	*na*		
	creditCardHL Notification	HQ_CreditCardHotlistT	0.039	0.144	0.841	*na*		

sidering their relative fraction of the whole interaction. To derive the service demands of notification messages, we repeated the experiments with different numbers of subscribers and used linear regression to estimate the service demands. This resulted in the service demands presented in Table 1. As to the subnet places corresponding to the client locations (SMs, HQ, DCs and SPs), they were each mapped to a nested QPN containing a single queueing place whose queue represents the CPU of the respective client machine. In our setup, all instances of a given location type were deployed on the same client machine and therefore they were all mapped to the same physical queue. Note that this represents the most typical deployment scenario for SPECjms2007. We used the QPME tool to build and analyze the model.

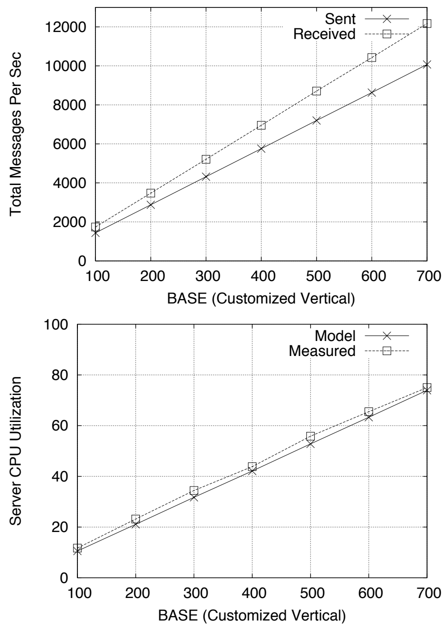
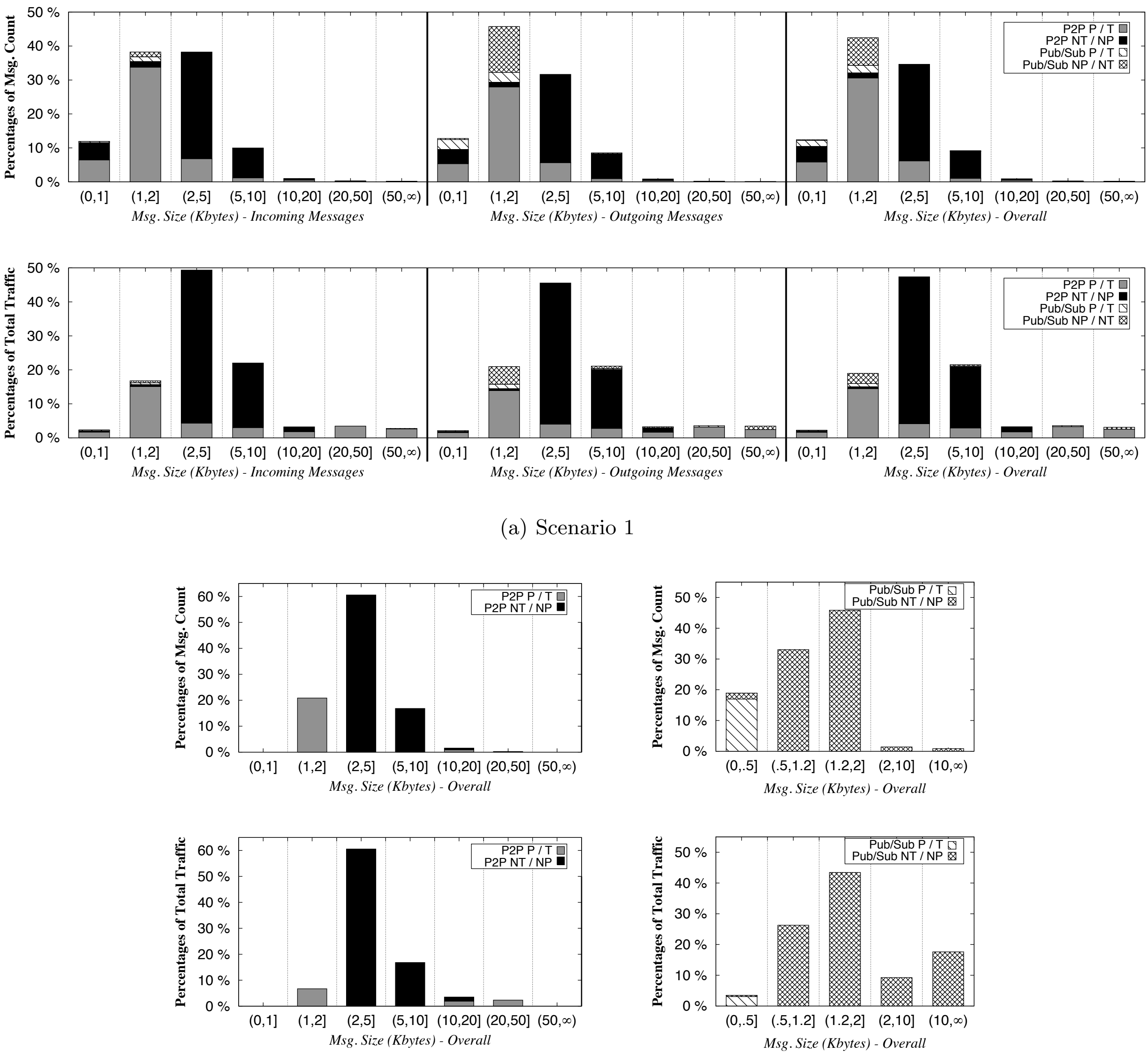


Fig. 10 Server CPU Utilization and Message Traffic for Customized Vertical Topology

4.3 Considered Workload Scenarios

We consider several different scenarios that represent different types of messaging workloads stressing different aspects of the MOM infrastructure including both workloads focused on point-to-point messaging as well as workloads focused on publish/subscribe. In each case, the model was analyzed using SimQPN [6] which took less than 5 minutes. We have intentionally slightly devi-



(a) Scenario 1

(b) Scenario 2 (c) Scenario 3

Fig. 9 Distribution of the Message Size

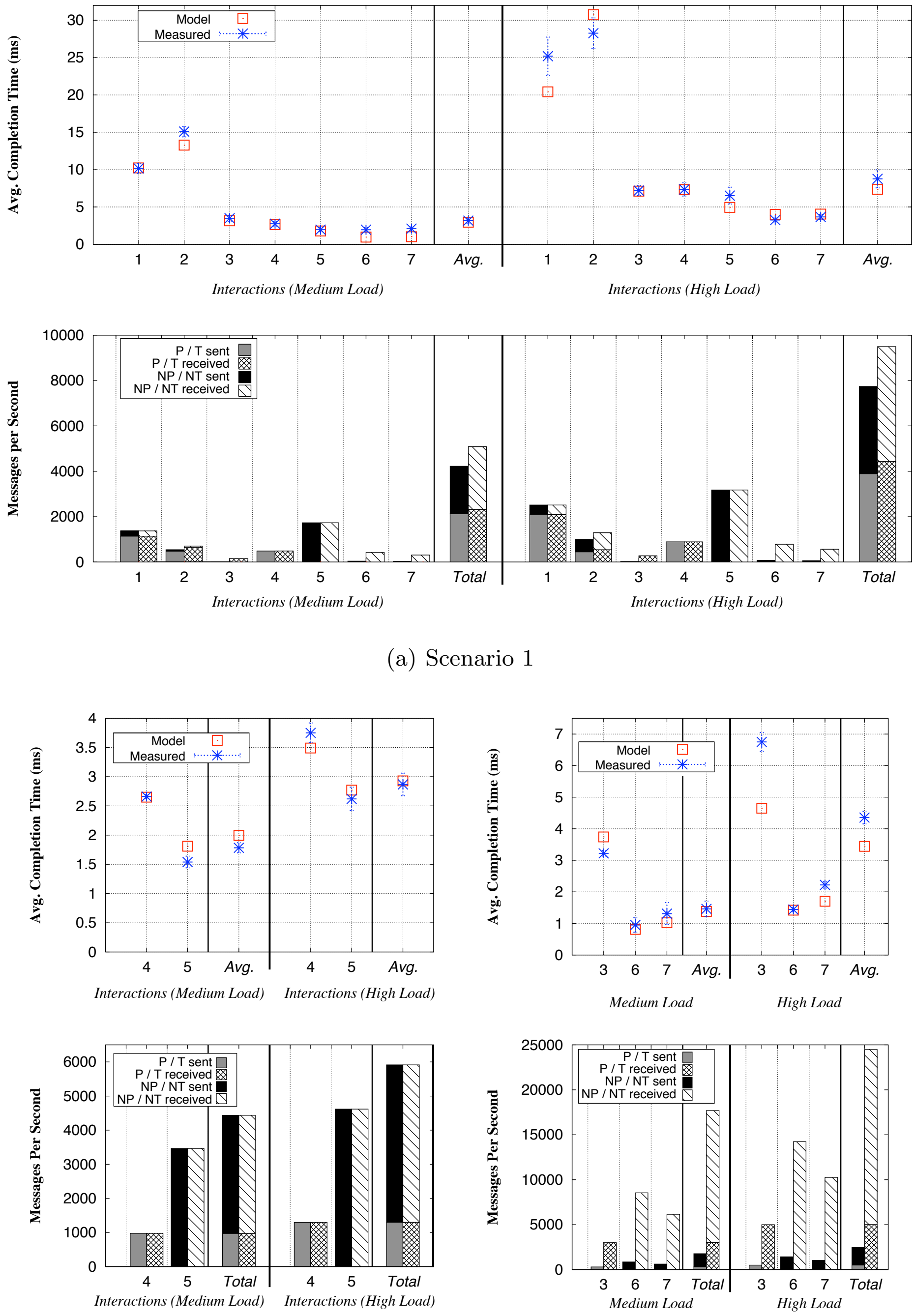
ated from the standard vertical topology to avoid presenting performance results that may be compared against standard SPECjms2007 results. The latter is prohibited by the SPECjms2007 run and reporting rules. To this end, we use freeform topologies based on the vertical topology with the number of DCs and HQ instances each set to 10. We study the following specific workload scenarios:

– *Scenario 1:* A mix of all seven interactions exercising both P2P and pub/sub messaging.
– *Scenario 2:* A mix of Interactions 4 and 5 focused on P2P messaging.
– *Scenario 3:* A mix of Interactions 3, 6 and 7 focused on pub/sub messaging.

In Table 2 and Fig. 9, we provide a detailed workload characterization of the three scenarios to illustrate the differences in terms of transaction mix and message size distribution.

4.4 Experimental Results

Figure 10 shows the predicted and measured CPU utilization of the MOM server for the considered customized vertical topology when varying the $BASE$ between 100



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

Fig. 11 Model Predictions Compared to Measurements for Scenarios 1, 2 and 3

and 700. The total number of messages sent and received per second is shown. As we can see, the model predicts the server CPU utilization very accurately as the workload is scaled. To gain a better understanding of the system behavior, we used the model to breakdown the overall utilization among the seven interactions as shown in Table 4. The bulk of the load both in terms of message traffic and resulting CPU utilization is produced by Interactions 1 and 5 followed by Interactions 2 and 4. Interactions 3, 6 and 7 which exercise only publish/subscribe messaging produce much less traffic which is expected since the standard vertical topology that we used as a basis places the emphasis on point-to-point messaging [2]. In the following, we study in detail the three scenarios under different load intensities consider-

Table 2 Scenario Transaction Mix

	Sc. 1			*Sc. 2*	*Sc. 3*
	In	*Out*	*Overall*		
No. of Msg.					
P2P					
- P/T	49.2%	40.7%	44.6%	21.0%	-
- NP/NT	47.2%	39.0%	42.8%	79.0%	-
Pub/Sub					
- PT	1.8%	6.0%	4.1%	-	17.0%
- NP/NT	1.7%	14.2%	8.5%	-	83.0%
Overall					
- PT	51.1%	46.7%	48.7%	21.0%	17.0%
- NT/NP	48.9%	53.3%	51.3%	79.0%	83.0%
Traffic					
P2P					
- P/T	32.2%	29.5%	30.8%	11.0%	-
- NP/NT	66.6 %	61.0%	63.5%	89.0%	-
Pub/Sub					
- PT	0.5%	2.3%	1.6%	-	3.0%
- NP/NT	0.8%	7.2%	4.1%	-	97.0%
Overall					
- PT	32.7%	31.8%	32.4%	11.0%	3.0%
- NT/NP	67.3%	68.2%	67.6%	89.0%	97.0%
Avg. Size	*(in KBytes)*				
P2P					
- P/T		2.13		2.31	-
- NP/NT		4.59		5.27	-
Pub/Sub					
- PT		1.11		-	0.24
- NP/NT		1.49		-	1.49
Overall					
- PT		2.00		2.31	0.24
- NT/NP		3.76		5.27	1.49

For Scenario 2 &3: In = Out.

Table 3 Detailed Results for Scenario 1,2 and 3

(a) Scenario 1

Input BASE	*Inter-action*	*Rate p. sec*	*Avg. Completion T (ms)* Model	Meas. (95% c.i.)
300 *med. load*	1	228.57	10.24	10.17 +/- 0.68
	2	64	13.28	15.10 +/- 0.71
	3	15	3.16	3.49 +/- 0.41
	4	486.49	2.64	2.76 +/- 0.31
	5	1731.60	1.79	1.97 +/- 0.27
	6	42.69	0.97	1.96 +/- 0.29
	7	30.77	1.02	2.10 +/- 0.24
550 *high load*	1	419.05	20.41	25.19 +/- 2.56
	2	117.33	30.73	28.27 +/- 2.05
	3	27.50	7.12	7.20 +/- 0.67
	4	891.89	7.33	7.35 +/- 0.89
	5	3174.60	4.95	6.52 +/- 1.13
	6	78.27	4.01	3.26 +/- 0.26
	7	56.41	4.05	3.67 +/- 0.34

(b) Scenario 2

Input BASE	*Inter-action*	*Rate p. sec*	*Avg. Completion T (ms)* Model	Meas. (95% c.i.)
600 *med. load*	4	972.97	2.65	2.66 +/- 0.04
	5	3463.20	1.81	1.54 +/- 0.10
800 *high load*	4	1297.30	3.49	3.75 +/- 0.17
	5	4617.60	2.77	2.62 +/- 0.20

(c) Scenario 3

Input BASE	*Inter-action*	*Rate p. sec*	*Avg. Completion T (ms)* Model	Meas. (95% c.i.)
6000 *med. load*	3	300	3.74	3.22 +/- 0.09
	6	853.89	0.81	0.95 +/- 0.23
	7	615.38	1.02	1.31 +/- 0.35
10000 *high load*	3	500	4.65	6.75 +/- 0.30
	6	1423.15	1.42	1.44 +/- 0.07
	7	1025.64	1.70	2.22 +/- 0.10

ing further performance metrics such as the interaction throughput and completion time.

The detailed results for the scenarios are presented in Tables 3(a), 3(b) and 3(c). For each scenario, we consider two workload intensities corresponding to medium and high load conditions configured using the $BASE$ parameter. For each scenario, the interaction rates and the average interaction completion times are shown. The *interaction completion time* is defined as the time between the beginning of the interaction and the time that the last message in the interaction has been processed. The difference between the predicted and measured interaction rates was negligible (with error below 1%) and therefore we only show the predicted interaction rates. For completion times, we show both the predicted and measured mean values where for the latter we provide a 95% confidence interval from 5 repetitions of each experiment. Given that the measured mean values were computed from a large number of observations, their respective confidence intervals were quite narrow. The modeling error does not exceed 20% with exception of the cases where the interaction completion times are below 3 ms, e.g., for Interactions 6 and 7 in the first scenario. In such cases, a small absolute difference of say 1 ms between the measured and predicted values (e.g., due to some synchronization aspects not captured by the model) appears high when considered as a percentage of the respective mean value given that the latter is very

Table 4 Relative Server CPU Load of Interactions

Inter-action	*Relative CPU load*	*No of msgs.* in	out	*Traffic in KByte* in	out
1	31.82%	32.00%	26.48%	17.08%	15.74%
2	15.69%	14.19%	13.60%	9.05%	9.55%
3	2.53%	0.35%	2.90%	0.02%	0.23%
4	17.98%	11.35%	9.39%	8.01%	7.38%
5	30.36%	40.40%	33.44%	65.04%	59.91%
6	0.86%	1.00%	8.25%	0.39%	3.55%
7	0.76%	0.72%	5.94%	0.40%	3.65%

low. However, when considered as an absolute value, the error is still quite small.

Figure 11 depicts the predicted and measured interaction completion times for the three scenarios as well as detailed information on how the total message traffic of each interaction is broken down into sent vs. received messages, on the one hand, and transactional (T) persistent (P) vs. non-transactional (NT) non-persistent (NP) messages, on the other hand. In addition, aggregate data for all of the seven interactions is shown. For example, in Scenario 3, we see that the total number of received messages per second is about 10 times higher than the number of messages sent. This is because each message sent in Interactions 3, 6 and 7 is delivered to 10 subscribers - one for each SM. The results in Figure 11 reveal the accuracy of the model when considering different types of messaging. While for point-to-point messaging, the modeling error is independent of whether (P T) or (NP NT) messages are sent, for the publish/subscribe case under high load (Scenario 3), the modeling error is much higher for the case of (P T) than for the case of (NP NT). In Scenario 1 where all interactions are running at the same time, Interactions 1 and 2 exhibited the highest modeling error (with exception of the interactions with very low completion times). This is due to the fact that these interactions each comprise a complex chain of multiple messages of different types and sizes. Finally, looking at the mean completion time over all interactions, we see that for the most part the model is optimistic in that the predicted completion times are lower than the measured ones. This behavior is typical for performance models in general since no matter how representative they are, they normally cannot capture all factors causing delays in the system.

In summary, the model proved to be very accurate in predicting the system performance, especially considering the size and complexity of the system that was modeled. The proposed modeling methodology can be used as a performance prediction tool in the software engineering lifecycle of event-driven systems. For example at system design time, predictive performance models can be exploited for comparing alternative system designs with different communication and messaging patterns. At system deployment time, models help to detect system bottlenecks and to ensure that sufficient resources are allocated to meet performance and QoS requirements.

5 Performance Modeling Patterns

In this section, we introduce a set of generic *performance modeling patterns (PerfMP)* for message-oriented event-driven systems. The patterns address common workload scenarios and configurations that occur in practice and can be used as building blocks to simplify the modeling process. To the best of our knowledge, no similar patterns have been proposed before for message-oriented event-driven systems.

5.1 Overview of the Patterns

Overall, we define eleven different patterns summarized in Table 5. Several of the patterns can be combined or customized to reflect specific application scenarios. The patterns capture the most common types of interactions in message-oriented event-driven systems. The patterns address the following aspects of MOM-based communication:

– *Asynchronous* communication
– *Pull-based* vs. *push-based* communication
– *Point-to-point* vs. *pub/sub* messaging
– *Resource management*, e.g., the number of messages that can be processed in parallel
– *Time controlled* behavior, e.g., connection times of consumers
– *Load balancing*

Pattern Template

Each pattern definition comprises four parts:

1. *Characteristics:* The main features of the pattern are summarized with keywords.
2. *Example:* A sample scenario for the pattern.
3. *Description:* A detailed description of the pattern, including motivation and high-level implementation.
4. *QPN Definition:* Specification of the respective QPN model presented in four tables:
   (a) *Places:* A list of all places including name, short description and type (Q=queueing place, O=ordinary place, S=subnet place).
   (b) *Colors:* A list of all colors.

Name	Description
Pattern 1: **Standard Queue**	A standard queue implementing point-to-point messaging to a single consumer.
Pattern 2: **Standard Pub/Sub - Fixed Number of Subscribers**	A standard pub/sub scenario in which incoming messages are delivered to a fixed number of subscribers.
Pattern 3: **Standard Pub/Sub - Dynamic No. of Subscribers**	A standard pub/sub scenario in which incoming messages are delivered to a variable number of subscribers.
Pattern 4: **Time-Controlled Pull I**	Implementation of a simple time-controlled pull communication. A consumer connects to the MOM periodically each time processing one message.
Pattern 5: **Time-Controlled Pull II**	A consumer connects to the MOM periodically each time processing all waiting messages before disconnecting.
Pattern 6: **Resource-Controlled Pull I**	A consumer pulls messages sequentially and processes them one at a time.
Pattern 7: **Resource-Controlled Pull II**	Similar to Pattern 6, but with support of parallel message processing.
Pattern 8: **Time Window**	A consumer connects periodically to the MOM and stays online processing messages for a specified time interval before disconnecting.
Pattern 9: **Random Load Balancer**	A load balancer that distributes incoming messages randomly among a set of consumers.
Pattern 10: **Round Robin Load Balancer**	A load balancer that distributes incoming messages round-robin among a set of consumers.
Pattern 11: **Queueing Load Balancer**	A load balancer stores incoming messages which are pulled by consumers asynchronously.

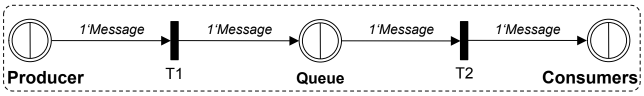
Table 5 Performance Modeling Patterns

(c) *Initial Marking:* The initial number of tokens of each color available in the various places of the QPN.
(d) *Transitions:* A description of all transitions including colors, places and firing weights (FW, mostly 1 or ∞).

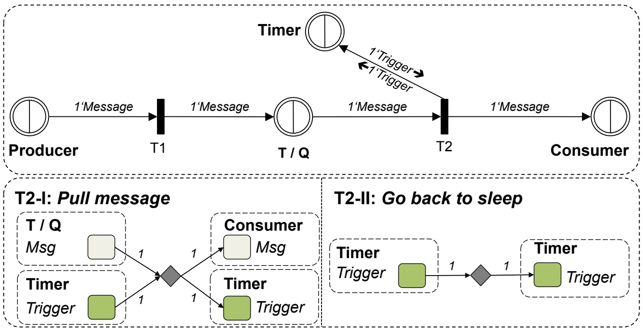
Additionally, a graphical illustration of the underlying QPN is provided for each pattern. Where no cardinality for a transition is specified in the illustration, the cardinality is assumed to be 1.

5.2 Pattern Definitions

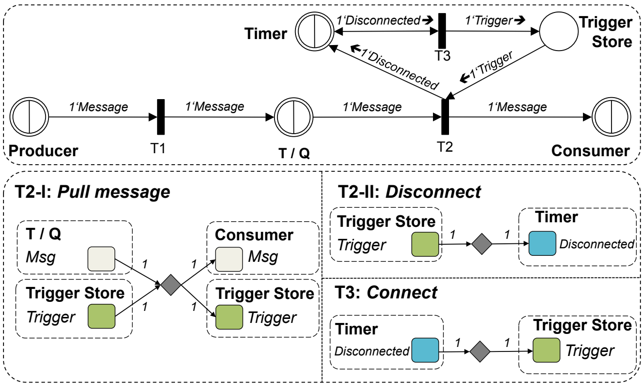
In this section, we describe three of the patterns in detail. An overview of the rest of the patterns is given in Figure 12. For more details, we refer the interested reader to [9] where detailed definitions of all patterns can be found. We selected three patterns each exhibiting different complexity: Patterns 2 & 3 cover specific aspects one-to-many communication and are helpful to model typical pub/sub interactions, while Pattern 6 is focusing on pull-based communication and can also be applied to model a thread pool.



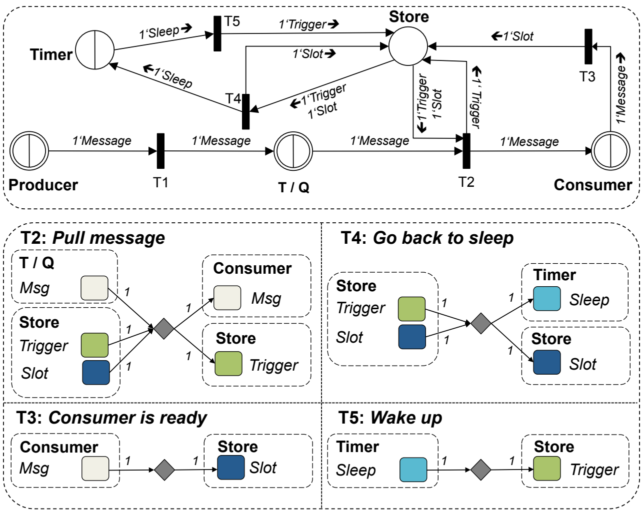
(a) *Pattern 1* - Standard Queue



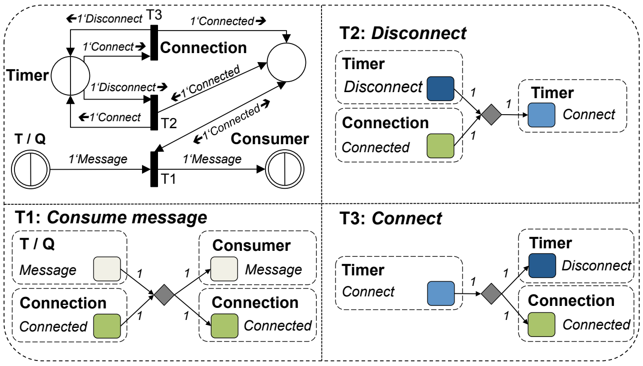
(b) *Pattern 4* - Time controlled Pull I



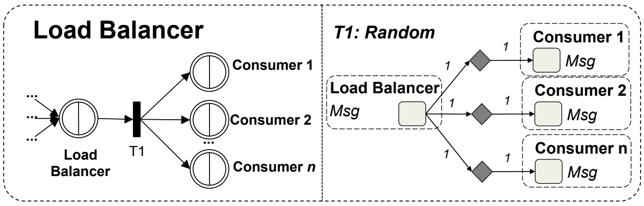
(c) *Pattern 5* - Time controlled Pull II



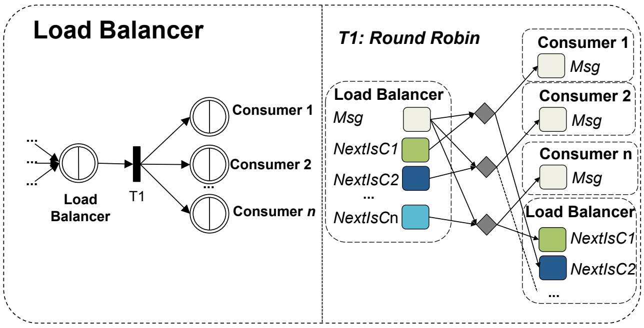
(d) *Pattern 7* - Resource controlled Pull II



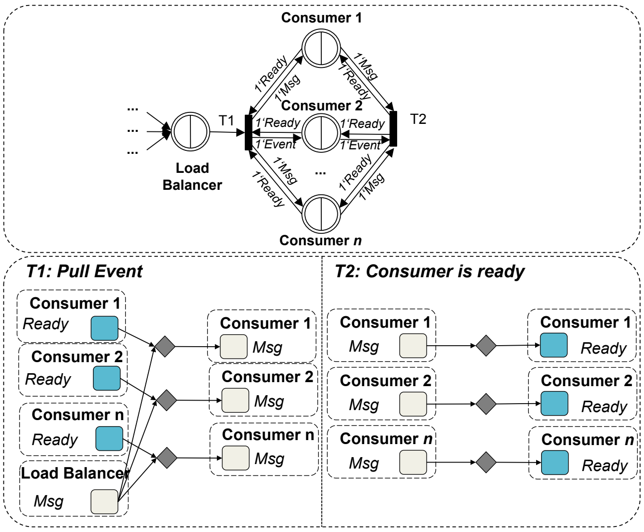
(e) *Pattern 8* - Time Window



(f) *Pattern 9* - Random Load Balancer



(g) *Pattern 10* - Round-Robin Load Balancer



(h) *Pattern 11* - Queueing Load Balancer

Fig. 12 Performance Modeling Patterns - Other Patterns *(T = Topic, Q = Queue)*

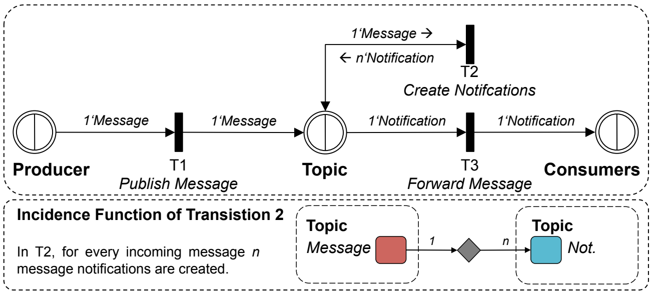


Fig. 13 Standard Pub/Sub Pattern - Fixed Number of Subscribers

Pattern 2: *Standard Pub/Sub - Fixed Number of Subscribers*

Characteristics

– $1:n$ communication (one message is delivered to n consumers)

Description A producer publishes a message to a given topic. The MOM forwards the message to the subscribers of the respective topic by sending a notification to each of them. The main idea of this pattern is based on the presumption that the service demand of the MOM per message delivery is composed of two parts, the service demand incurred for every incoming message and the aggregated service demands for the notification messages to the subscribers:

$$D_{MsgTotal,MOM} = D_{Msg,MOM} + n \cdot D_{Notification,MOM}$$

where

n	No. of message notifications.
$D_{Msg,MOM}$	Service demand of the MOM for receiving and processing an incoming message.
$D_{Notification,MOM}$	Service demand of the MOM for creating, processing and sending notifications.

In this version of the pattern, we implemented a straight-forward approach for a 1:n communication. The number of consumers is specified in the cardinality of the transition (see Figure 13). The downside of this approach is that the number of consumers is fixed in the transition specification and therefore cannot be modified without changing the definition of the model.

Pattern 3: *Standard Pub/Sub - Dynamic No. of Subscribers*

Characteristics

– $1:n$ communication (one message is delivered to a dynamic number of consumers)
– Dynamic number of message notifications

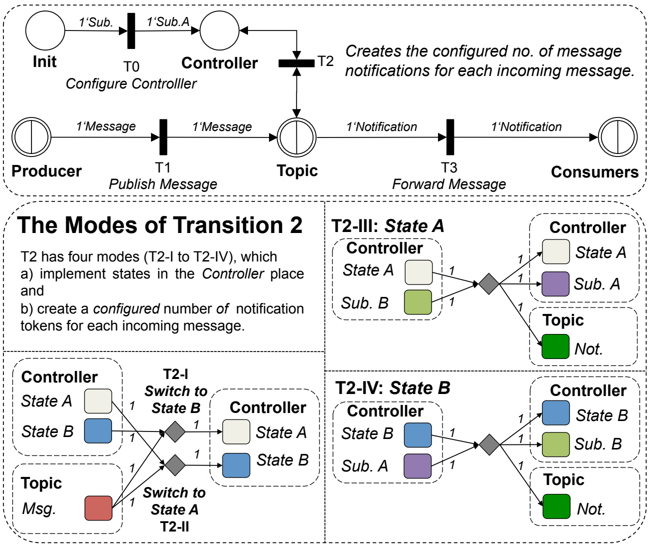


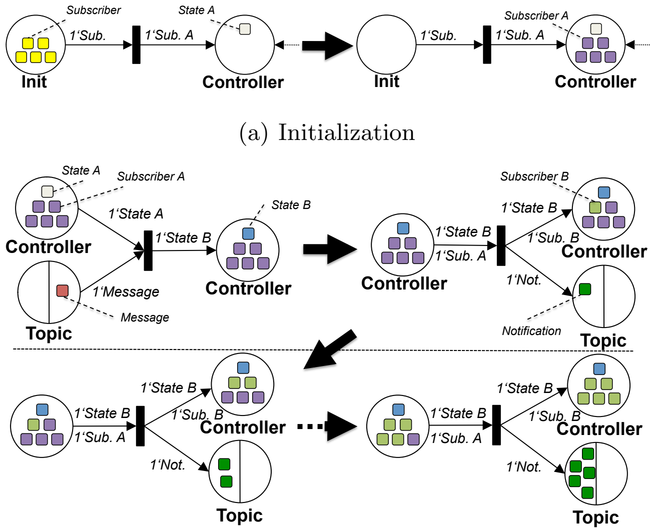
Fig. 14 *Pattern 3* - Standard Pub/Sub - Dynamic No. of Subscribers

Description In many realistic scenarios, the number of subscribers varies over time and it is itself a dynamic parameter of the model. Since the state of a QPN is captured in its marking (i.e., token population), it is desirable to be able to model subscribers using *Subscriber* tokens located in a given place of the QPN. This way the number of subscribers can change by adding or removing tokens from the respective place. Pattern 3 is based on the underlying idea of Pattern 2, however, the number of subscribers is determined dynamically based on the token population of a specified place. For each *Subscriber* token, a message notification is created and forwarded by the MOM.

To implement the above logic, we introduce an ordinary place *Controller* and define two colors, *State A* and *State B*. These colors are used to represent whether the *Controller* is either in state A or B, depending on the token stored in its depository. Further, for each subscriber, a token *Subscriber A* or *Subscriber B* depending on the current state exists. An incoming *Message* triggers a state change from state A to B (or vice versa). In response to an incoming *Message*, the respective number of message notifications are generated. This is implemented by Transition *2-III / 2-IV* (see Figure 14). As a reaction to a state change from A (B) to B (A), all n *Subscriber A (B)* tokens are transformed to n *Subscriber B (A)* tokens stored in the *Controller* and to n *Notification* tokens forwarded to the consumers.

For a better understanding, we provide a detailed description of the different steps and states. The underlying transitions are illustrated in Figure 14.

1. *Initialization*
   First, we define the number of initial subscribers by configuring the initial number of *Subscriber* tokens n. These n tokens are then transformed by transi-



(a) Initialization

(b) Generate Notifications

Fig. 15 Example Behavior of Pattern 3

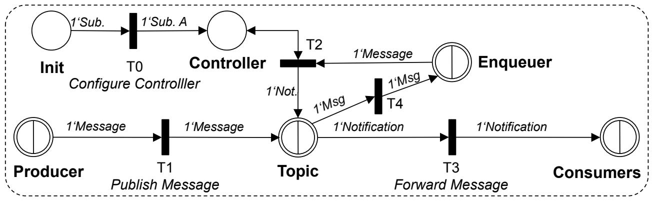


Fig. 16 Pattern 3 using an Enqueuer for Incoming Messages

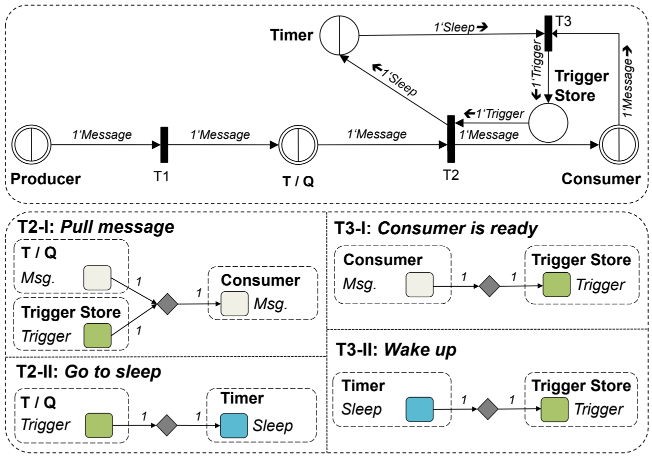


Fig. 17 *Pattern 6* - Resource-Controlled Pull I

tion *T0* to n *Subscriber A* tokens stored in the *Controller* place. This step is illustrated in Figure 15(a). Transition T0 is fired only once in the beginning. The *Controller* place is in state A which is represented by a *State A* token stored in its depository.

2. *Creation of Notifications*
   (a) *Producer Publishes Message*
   The producer publishes a *Message* token, which arrives via *T1* at the Topic. After the MOM receives the *Message* token, transition T2-II is fired and changes the state of the *Controller* from A to B by replacing the *State A* token with a *State B* token.
   (b) *Notification of Subscribers*
   Since the *Controller* place is now in state B, transition *T2-IV* is fired for each of the n *Subscriber A* tokens and transforms them as illustrated in Figure 15(b) into *Notification* tokens (sent to the Topic) and into *Subscriber B* tokens stored in the *Controller*, respectively. These *Notification* tokens will be processed by the MOM and afterwards delivered to the consumers. Therefore, each *Message* token triggers the generation of n *Notifications*.

Transition Priorities The *Controller* is defined as an ordinary place. Since standard QPNs do not support priorities of transition firings this may become an issue: if two messages arrive exactly at the same time, the state of the *Controller* can be changed to the next state without waiting for the creation of the notifications to complete.

Imagine a situation where the *Controller* is in state A and two *Message* tokens arrive at the same time. First, transition *T2-I* is fired and the state of the *Controller* is changed to state B. Second, n notifications should be created by firing transition *T2-IV* n times. However, a major problem arises if the second *Message* token triggers a second state change back to A (via *T2-II*) before all n notifications for the first token are generated.

To address this issue, we propose two approaches using standard QPNs:

1. *Set the firing weight of T2-III and T2-IV to* ∞
   This solution does not completely rule out the incorrect state changes, but their probability converges to zero.
2. *Adding an additional queueing place Enqueuer* (see Figure 16)
   This *Enqueuer* place is a queueing place with a single server and used to form a line of messages. This allows us to process the *Messages* one after another and to avoid incorrect state changes. In addition to the new queueing place, a new transition *T4* has to be added and the existing transitions *T2-I* and *T2-II* have to be adjusted. By defining a service demand close to zero for *Message* tokens in the *Enqueuer* place, a distortion of the results should be avoided.

Note: The above problem does not occur if the Topic place has a single server. In this case, two messages never arrive at the same time.

Pattern 6: *Resource-Controlled Pull I*

Characteristics

– *Pull-based* communication *on demand*
– *Resource modeling (number of service places)*

Description A message consumer connects frequently to the MOM to check whether new messages have arrived. If yes, the consumer receives one of them, closes the connection and processes the message. As soon as the message has been processed, the consumer connects again to the MOM to pull the next message. If no further messages are available, the consumer closes the connection and waits for a specified period of time before pursuing the next pull attempt.

In this scenario the pull attempt of the consumer is not only controlled by time, but also by the availability of the consumer. The consumer tries to pull the next message as soon as he is ready, i.e., after the last message was processed. Only if no further message is available, the consumer disconnects and the next connection is triggered after a specified time interval.

This behavior is reflected in transitions *2 & 3*. When the consumer has processed a message, the *Message* token is transformed by transition *3-I* to a *Trigger* token. Next, transition *2* is fired. Depending on the availability of *Message* tokens in the depository of the T/Q place, either mode *2-I* (Message available) or mode *2-II* (no Message token) is chosen:

1. If a *Message* token exists, the consumer pulls it (transition *3-I*) and disconnects. He will not reconnect before the *Message* token is processed.
2. If no *Message* token exists, the consumer disconnects and waits for a specified time interval. Then, a new *Trigger* token is generated by transition *3-II* and the consumer tries to pull a *Message*.

Number of Service Places (Parallel Messages) The pattern offers a simple way to set the maximum number of *Messages* processed in parallel by defining the initial number of *Trigger* tokens.

However, there is a drawback of this approach: Imagine a scenario where we set the number of parallel messages processed by the consumer to two. For the case that no *Message* token was available at the Q/T, two *Trigger* tokens were transformed to *Sleep* tokens and moved to the Timer. After the specified time interval one of the *Sleep* tokens is processed by the Timer and transformed back to a *Trigger* token by transition *T3-II*: the consumer 'wakes up' and establishes a connection to the MOM. In the meantime, two new *Message* tokens arrived at the Q/T. Since there is one *Trigger* token, only a single *Message* is moved to the consumer. The second *Message* token remains in the depository of the Q/T until the second sleep token has been processed by the timer, even if the consumer has enough resources to process both *Messages*.

Another approach is presented in Pattern 7, where the consumer pulls as many *Messages* at once as free resources are available. This avoids opening several connections and allows processing them as fast as possible.

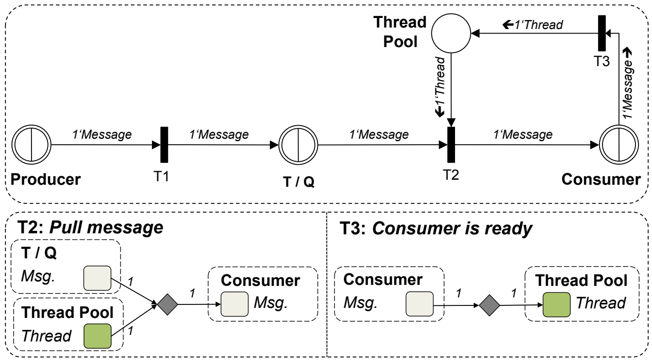


Fig. 18 Modeling a Thread Pool

How to Modify Pattern 6 to Model Thread Pool By removing the Timer place and transitions *T2-II* and *T3-II*, the underlying idea of this approach is made suitable for modeling a thread pool. As illustrated in Figure 18, all we need to implement such a pool are *Thread* tokens and a *Thread Pool* ordinary place (corresponding to *Thread* tokens, respectively *Thread Store*).

6 Related Work

6.1 Performance Evaluation of Message-oriented Event-driven Systems

We present an overview of existing performance modeling and analysis techniques for message-oriented event-driven systems. In [16], an analytical model of the message processing time and throughput of the WebSphereMQ JMS server is presented and validated through measurements. The message throughput in the presence of filters is studied and it is shown that the message replication grade and the number of installed filters have a significant impact on the server throughput. Several similar studies using Sun Java System MQ, FioranoMQ, ActiveMQ and BEA WebLogic JMS server were published. A more in-depth analysis of the message waiting time for the FioranoMQ JMS server is presented in [17]. The authors study the message waiting time based on an $M/G/1-\infty$ queue approximation and perform a sensitivity analysis with respect to the variability of the message replication grade. They derive formulas for the first two moments of the message waiting time based on different distributions (deterministic, Bernoulli and binomial) of the replication grade. These publications, however, only consider the overall message throughput and latency and do not provide any means to model the performance of complex event-driven interactions and message flows.

A method for modeling MOM systems using *performance completions* is presented in [18]. A pattern-based language for configuring the type of message-based communication is proposed and model-to-model transformations are used to integrate low-level details of the MOM system into high-level software architecture models. A case study based on part of the SPECjms2007 workload (more specifically Interaction 4) is presented as a validation of the approach. However, no interactions involving multiple message exchanges or interaction mixes are considered and the studied deployment is unrealistic. In [19], an approach to predicting the performance of messaging applications based on the Java Enterprise Edition is proposed. The prediction is carried out during application design, without access to the application implementation. This is achieved by modeling the interactions among messaging components using queueing network models, calibrating the performance models with architecture attributes, and populating the model parameters using a lightweight application-independent benchmark. However, again the workloads considered are very simple and do not include any complex messaging interactions.

Several performance modeling techniques specifically targeted at distributed publish/subscribe systems [20] exist in the literature. However, such techniques are normally focused on modeling the routing of events through distributed broker topologies from publishers to subscribers as opposed to modeling interactions and message flows between communicating components in event-driven applications. In [21], an analytical model of publish/subscribe systems that use hierarchical identity-based routing is presented. The model is based on continuous time birth-death Markov chains. Closed analytical solutions for the sizes of routing tables, for the overhead required to keep the routing tables up-to-date, and for the leasing overhead required for self-stabilization are presented. The proposed modeling approach, however, does not provide means to predict the event delivery latency and it suffers from a number of restrictive assumptions. Many of these assumptions were relaxed in [22, 23] where a generalization of the model was proposed, however, the generalized model is still limited to systems based on peer-to-peer and hierarchical routing schemes. In [24], a basic approach for workload characterization and performance modeling of distributed event-based systems was proposed and applied to a simple publish/subscribe application. However, many simplifying assumptions were made and important system aspects, that occur in realistic applications, e.g., different communication patterns, multiple message types and message persistence, were not considered. Finally, in [25], probabilistic model checking techniques and stochastic models are used to analyze publish/subscribe systems. The communication infrastructure (i.e., the transmission channels and the publish/subscribe middleware) are modeled by means of probabilistic timed automata. Application components are modeled by using statechart diagrams and then translated into probabilistic timed automata. The analysis considers the probability of message loss, the average time taken to complete a task and the optimal message buffer sizes.

To summarize, while a number of modeling approaches and case studies of event-driven systems exist in the literature, they are mostly based on custom applications and artificial workloads that are not representative of real-life event-driven applications (see also [26]). To the best of our knowledge, no realistic applications of the size and complexity of the one considered in this paper have been studied before.

6.2 Patterns in Performance Modeling

Performance models should reflect real world applications. In this context we face commonly occurring themes. The goal of design patterns is to identify, name, and abstract these themes [27]. Similar to software engineering, where the concept of design patterns is well established, several research results focusing on the usage of patterns in performance engineering and modeling were published. Most of these publications fall in one of the following two categories. The first category focuses on describing knowledge of experienced modelers in a structured way and/or providing reusable building blocks, which can be used by modelers. The goal is to transfer expert knowledge to less experienced modelers, to decrease the time needed for modeling the applications and, by reusing expertise and proven components, to improve the quality of models. In the second category we find research focusing on model-to-model transformation, e.g., UML models to (C)PNs. The ongoing research is closely related to the question how CPNs, QPNs and similar models can be applied in the software development life cycle.

A template for the description of Petri net patterns is introduced in [28]. The authors use a template to describe a number of sample patterns and suggest the introduction of a Petri net pattern repository. In [29] a template is proposed for the systematic description of CPNs. Furthermore, the same authors present a comprehensive and structured collection of 34 design patterns for CPNs in [30]. These patterns have been modeled using CPN Tools. In [31] the authors mention that they created a library of QPN patterns, which contains models of basic constructs appearing repeatedly in the Tomcat architecture such as blocking. An extension to *hierarchical colored Petri nets (HCPN)* named *reusable colored Petri nets (RCPN)* is published and demonstrated in [32]. RCPN support the definition of reusable components.

The authors of [33–35] discuss how to construct an underlying CPN representation based on an UML software architecture model. For this purpose behavioral design patterns (BDP) are specified and mapped to CPN templates. This allows software engineers to focus on the

UML design independent from the CPN model. The generated CPN may be analyzed for performance and functionality. Observed behavioral problems resulting from the CPN analysis can be corrected in the UML software design.

Our work differs from the previous ones in at least two ways. To the best of our knowledge, no patterns for QPNs are published. Existing work focuses mostly on CPNs and PNs. Furthermore, there is no work discussing such patterns for event-based applications.

7 Conclusions and Future Work

We presented a novel modeling methodology for event-based systems in the context of a case study of a representative state-of-the-art event-driven system. The system we studied was a deployment of the SPECjms2007 standard benchmark on a leading commercial middleware platform. A detailed model of the benchmark application was developed in a step-by-step fashion and it was shown how the model can be customized for a particular deployment scenario. The system modeled was much larger and more complex than those considered in existing literature. Overall, the model contains a total of 59 queueing places, 76 token colors and 68 transitions with a total of 285 firing modes. To validate our modeling technique we considered a real-life deployment of the benchmark in a representative environment comparing the model predictions against measurements on the real system. A number of different scenarios varying the workload intensity and interaction mix were considered and the accuracy of the developed models was evaluated. The results demonstrated the effectiveness and practicality of the proposed modeling and prediction approach. The presented case study is the first comprehensive validation of our modeling technique on a representative application. The technique can be exploited as a tool for performance prediction and capacity planning during the software engineering lifecycle of message-oriented event-driven systems. Additionally, we introduced a set of generic performance modeling patterns that can be used as building blocks when modeling message-oriented event-driven systems.

As part of our future work, we will be working on self-adaptive event-based systems based on the presented modeling methodology. Such systems will dynamically adjust their configuration to ensure that QoS requirements are continuously met. The idea is to generate performance models at run-time based on monitoring data and to use them to predict the system performance under forecast workloads. Since performance analysis will be carried out on-the-fly, it is essential that the process of generating and analyzing the models is completely automated. As a first step, we are working on a tool that will allow us to automatically generate system models of jms2009-PS [36], an extended version of SPECjms2007, and a runtime measurement framework. We then plan to integrate our approach into an open-source event-based middleware and provide a prototype implementation.

References

1. A. Hinze, K. Sachs, and A. Buchmann, "Event-based applications and enabling technologies," in *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS 2009)*, 2009.
2. K. Sachs, S. Kounev, J. Bacon, and A. Buchmann, "Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark," *Performance Evaluation*, vol. 66, no. 8, pp. 410–434, 2009.
3. F. Bause, "Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of systems," in *Proc. of the 5th Intl. Workshop on Petri Nets and Performance Models (PNPM 1993), Toulouse (France)*, 1993.
4. F. Bause and P. Buchholz, "Queueing Petri Nets with Product Form Solution," *Performance Evaluation*, vol. 32, no. 4, pp. 265–299, 1998.
5. F. Bause, P. Buchholz, and P. Kemper, "QPN-Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets," in *Quantitative Evaluation of Computing and Communication Systems*, ser. LNCS, H. Beilner and F. Bause, Eds., vol. 977. Springer-Verlag, 1995.
6. S. Kounev and A. Buchmann, "SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation," *Performance Evaluation*, vol. 63, no. 4-5, pp. 364–394, May 2006.
7. Sun Microsystems, "Java Message Service (JMS) Specification - Ver. 1.1," 2002.
8. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.
9. K. Sachs, "Performance Modeling and Benchmarking of Event-Based Systems," Ph.D. dissertation, TU Darmstadt, 2010.
10. F. Bause, P. Buchholz, and P. Kemper, "Integrating Software and Hardware Performance Models Using Hierarchical Queueing Petri Nets," in *Proc. of the 9. ITG /*

GI - Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, (MMB97), Freiberg (Germany), 1997.

11. S. Kounev, "Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets," *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 486–502, July 2006.
12. S. Kounev and C. Dutz, "QPME - A Performance Modeling Tool Based on Queueing Petri Nets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, 2009.
13. Descartes Research Group. (2011) QPME Project Website. [Online]. Available: http://qpme.sourceforge.net
14. P. Meier, S. Kounev, and H. Koziolek, "Automated Transformation of Palladio Component Models to Queueing Petri Nets," in *19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011)*, 2011.
15. S. Kounev, K. Bender, F. Brosig, N. Huber, and R. Okamoto, "Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics," in *4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011)*, 2011.
16. R. Henjes, M. Menth, and C. Zepfel, "Throughput Performance of Java Messaging Services Using WebsphereMQ," in *Proceedings of the 26th IEEE International Conferenceon Distributed Computing Systems Workshops (ICDCSW '06)*, 2006.
17. M. Menth and R. Henjes, "Analysis of the Message Waiting Time for the FioranoMQ JMS Server," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.
18. J. Happe, H. Friedrich, S. Becker, and R. H. Reussner, "A pattern-based Performance Completion for Message-oriented Middleware," in *Proceedings of the 7th international workshop on Software and performance (WOSP 2008)*, 2008.
19. Y. Liu and I. Gorton, "Performance Prediction of J2EE Applications Using Messaging Protocols," in *Proc. of Component-Based Software Engineering, 8th International Symposium (CBSE 2005), St. Louis, MO (USA)*, ser. Lecture Notes in Computer Science, vol. 3489. Springer, 2005.
20. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, 2001.
21. M. A. Jaeger and G. Mühl, "Stochastic Analysis and Comparison of Self-Stabilizing Routing Algorithms for Publish/Subscribe Systems," in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, 2005.
22. G. Mühl, A. Schröter, H. Parzyjegla, S. Kounev, and J. Richling, "Stochastic Analysis of Hierarchical Publish/Subscribe Systems," in *Euro-Par 2009 Parallel Processing*, ser. Lecture Notes in Computer Science, vol. 5704. Springer, 2009.
23. A. Schröter, G. Mühl, S. Kounev, H. Parzyjegla, and J. Richling, "Stochastic Performance Analysis and Capacity Planning of Publish/Subscribe Systems," in *4th ACM International Conference on Distributed Event-Based Systems (DEBS 2010), July 12-15, Cambridge, United Kingdom*. ACM, New York, USA, 2010.
24. S. Kounev, K. Sachs, J. Bacon, and A. P. Buchmann, "A Methodology for Performance Modeling of Distributed Event-Based Systems," in *Proceedings of 11th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC'08)*. IEEE, May 2008.
25. F. He, L. Baresi, C. Ghezzi, and P. Spoletini, "Formal analysis of publish-subscribe systems by probabilistic timed automata," in *Formal Techniques for Networked and Distributed Systems (FORTE 2007)*, ser. Lecture Notes in Computer Science, vol. 4574, 2007.
26. S. Kounev and K. Sachs, "Benchmarking and performance modeling of event-based systems," *it - Information Technology*, vol. 51, no. 5, pp. 262–269, oct 2009.
27. E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *Proceedings of the 7th European Conference on Object-Oriented Programming (ECOOP'93)*,

ser. Lecture Notes in Computer Science, vol. 707. Springer, 1993.

28. M. Naedele and J. W. Janneck, “Design Patterns in Petri Net System Modeling,” in *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS’98)*. IEEE, 1998.
29. N. Mulyar and W. von der Aalst, “Towards a pattern language for colored petri nets,” in *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN ’05)*, 2005.
30. ——, “Patterns in Colored Petri Nets,” BETA Working Paper Series, WP 139, Eindhoven University of Technology, Eindhoven , Tech. Rep., 2005.
31. U. Bellur and A. Shirabate, “Performance Prediction and Physical Design of J2EE based Web applications,” in *World Scientific and Engineering Academy and Society - Circuits, Systems, Communications, Computers (WSEAS CSCC 2004)*, 2004.
32. N. Lee, J. Hong, S. Cha, and D. Bae, “Towards Reusable Colored Petri Nets,” in *Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE’98)*. IEEE, 1998.
33. R. Pettit IV and H. Gomaa, “Modeling Behavioral Patterns of Concurrent Software Architectures Using Petri Nets,” in *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA’04)*. IEEE, 2004.
34. ——, “Modeling Behavioral Patterns of Concurrent Objects Using Petri Nets,” in *Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’06)*, 2006.
35. ——, “Analyzing behavior of concurrent software designs for embedded systems,” in *Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’07)*, 2007.
36. K. Sachs, S. Appel, S. Kounev, and A. Buchmann, “Benchmarking publish/subscribe-based messaging systems,” in *Database Systems for Advanced Applications (DASFAA 2010) International Workshops: BenchmarX10*, ser. Lecture Notes in Computer Science, vol. 6193. Springer, 2010.
37. F. Bause, “QN + PN = QPN - Combining Queueing Networks and Petri Nets,” Department of CS, University of Dortmund, Germany, Technical Report No.461, 1993.
38. S. Kounev, “J2EE Performance and Scalability - From Measuring to Predicting,” in *Proceedings of the SPEC Benchmark Workshop 2006 (SPEC’06)*. SPEC, 2006.
39. ——, “Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction,” Ph.D. dissertation, Technische Universität Darmstadt, 2005.
40. F. Bause and F. Kritzinger, *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg Verlag, 2002.
41. G. Bolch, S. Greiner, H. d. Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*. Wiley-Interscience, 2005.
42. H. J. Genrich and K. Lautenbach, “System Modelling with High-Level Petri Nets,” *Theoretical Computer Science*, vol. 13, pp. 109–136, 1981.
43. K. Jensen and G. Rozenberg, Eds., *High-level Petri nets: theory and application*. Springer, 1991.
44. K. Jensen, “How to find invariants for coloured petri nets,” in *Proceedings on Mathematical Foundations of Computer Science*, ser. Lecture Notes in Computer Science, vol. 118. Springer, 1981.
45. ——, “Coloured petri nets and the invariant-method,” *Theoretical Computer Science*, vol. 14, pp. 317–336, 1981.
46. F. Bause and P. Kemper, “QPN-Tool for qualitative and quantitative analysis of queueing Petri nets,” in *Proceedings of the 7th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, ser. Lecture Notes in Computer Science, vol. 794. Springer, 1994.
47. S. Kounev, C. Dutz, and A. Buchmann, “QPME - Queueing Petri Net Modeling Environment,” in *Proceedings of the Third International Conference on the Quantitative Evaluaiton of Systems (QEST 2006)*. IEEE Computer Society, 2006.

A Introduction to Queueing Petri Nets

In this section we provide a brief introduction to queueing Petri nets (QPNs). QPNs can be considered an extension of stochastic Petri nets that allow *queues* to be integrated into the places of a Petri net [37]. QPNs allow the modeling of process synchronization and the integration of hardware and software aspects of system behavior [10, 11] and provide greater modeling power and expressiveness than conventional queueing network models and stochastic Petri nets [10]. QPNs were applied successfully in several case studies to model system behavior, e.g., [11, 11, 24, 38, 39]. First, we present the formal definition of QPNs. This section is based on [11, 39, 40]. Afterwards we discuss the existing tool support for QPNs.

A.1 Formal Definition

Queueing Petri nets can be seen as a combination of a number of different extensions to conventional Petri nets (PNs) along several dimensions. In this section, we include some basic definitions and briefly discuss how queueing Petri nets have evolved. A more detailed treatment of the subject can be found in [3, 40]. *Petri nets (PNs)* were originally introduced by C.A. Petri in the year 1962. An ordinary Petri net is a bipartite directed graph composed of places P, drawn as circles, and transitions T, drawn as bars, which is defined as follows [11, 40, 41]:

Definition 1 *An ordinary Petri net (PN) is a 5-tuple* $PN = (P, T, I^-, I^+, M_0)$ *where:*

1. $P = \{p_1, p_2, ..., p_n\}$ *is a finite and non-empty set of* places*,*
2. $T = \{t_1, t_2, ..., t_m\}$ *is a finite and non-empty set of* transitions*,* $P \cap T = \emptyset$*,*
3. $I^-, I^+ : P \times T \to \mathbb{N}_0$ *are called backward and forward* incidence functions*, respectively,*
4. $M_0 : P \to \mathbb{N}_0$ *is called* initial marking.

Different extensions to ordinary PNs have been developed in order to increase the modeling convenience and/or the modeling power, e.g., [42, 43]. One of these extensions are *colored PNs (CPNs)* which were introduced by K. Jensen [44, 45] and provide the base for QPNs. In CPNs a type called *color* is attached to a token. A *color function C* assigns a set of colors to each place, specifying the types of tokens that can reside in the place. In addition to introducing token colors, CPNs also allow transitions to fire in different *modes*, so-called *transition colors*. The color function C assigns a set of modes to each transition and incidence functions are defined on a per mode basis. Formally CPNs are defined as follows [40]:

Definition 2 *A colored PN (CPN) is a 6-tuple* $CPN = (P, T, C, I^-, I^+, M_0)$ *where:*

1. $P = \{p_1, p_2, ..., p_n\}$ *is a finite and non-empty set of* places*,*
2. $T = \{t_1, t_2, ..., t_m\}$ *is a finite and non-empty set of* transitions*,* $P \cap T = \emptyset$*,*
3. C *is a* color function *that assigns a finite and non-empty set of* colors *to each place and a finite and non-empty set of* modes *to each transition.*
4. I^- *and* I^+ *are the backward and forward* incidence functions *defined on* $P \times T$*, such that* $I^-(p, t), I^+(p, t) \in [C(t) \to C(p)_{MS}], \forall (p, t) \in P \times T$[2]
5. M_0 *is a function defined on* P *describing the* initial marking *such that* $M_0(p) \in C(p)_{MS}$*.*

Other extensions of ordinary PNs allow timing aspects to be integrated into the net description [40, 41]. In particular, *generalized stochastic PNs (GSPNs)* attach an exponentially distributed *firing delay* (or *firing time*) to each transition, which specifies the time the transition waits after being enabled before it fires. Two types of transitions are defined: *immediate* (no firing delay) and *timed* (exponentially distributed firing delay). If several immediate transitions are enabled at the same time, the next transition to fire is chosen based on *firing weights* (probabilities) assigned to each of the transitions. Timed transitions fire after a random exponentially distributed firing delay. The firing of immediate transitions always has priority over that of timed transitions. GSPNs can be formally defined as [40, 41]:

Definition 3 *A generalized Stochastic PN (GSPN) is a 4-tuple* $GSPN = (PN, T_1, T_2, W)$ *where:*

1. $PN = (P, T, I^-, I^+, M_0)$ *is the underlying ordinary PN,*
2. $T_1 \subseteq T$ *is the set of timed transitions,* $T_1 \neq \emptyset$*,*
3. $T_2 \subset T$ *is the set of immediate transitions,* $T_1 \cap T_2 = \emptyset$*,* $T_1 \cup T_2 = T$*,*

[2] The subscript MS denotes multisets. $C(p)_{MS}$ denotes the set of all finite multisets of $C(p)$.

4. $W = (w_1, ..., w_{|T|})$ is an array whose entry $w_i \in \mathbb{R}^+$ is a rate of a negative exponential distribution specifying the firing delay, if $t_i \in T_1$ <u>or</u> is a firing weight specifying the relative firing frequency, if $t_i \in T_2$.

Combining definitions 2 and 3 leads to *Colored GSPNs (CGSPNs)* [40]:

Definition 4 *A colored GSPN (CGSPN) is a 4-tuple $CGSPN = (CPN, T_1, T_2, W)$ where:*

1. $CPN = (P, T, C, I^-, I^+, M_0)$ is the underlying CPN,
2. $T_1 \subseteq T$ is the set of timed transitions, $T_1 \neq \emptyset$,
3. $T_2 \subset T$ is the set of immediate transitions, $T_1 \cap T_2 = \emptyset$, $T_1 \cup T_2 = T$,
4. $W = (w_1, ..., w_{|T|})$ is an array with $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ such that $\forall c \in C(t_i) : w_i(c) \in \mathbb{R}^+$ is a rate of a negative exponential distribution specifying the firing delay due to color c, if $t_i \in T_1$ <u>or</u> is a firing weight specifying the relative firing frequency due to c, if $t_i \in T_2$.

CGSPNs have proven to be a very powerful modeling formalism. However, they do not provide any means for direct representation of queueing disciplines. To overcome this disadvantage, *queueing Petri nets (QPN)* were introduced based on CGSPNs with so-called *queueing places*. Such a queueing place consists of two components, a *queue* and a *token depository* (see Figure 2). The depository stores tokens which have completed their service at the queue. Only tokens stored in the depository are available for output transitions. QPNs introduce two types of queueing places:

1. *Timed* queueing place:
   The behavior of a *timed queueing place* is as follows:
   (a) A token is fired by an input transition into a queueing place.
   (b) The token is added to the queue according to the scheduling strategy of the queue.
   (c) After the token has completed its service at the queue, it is moved to the depository and available for output transitions.
2. *Immediate* queueing place:
   Immediate queueing places are used to model pure scheduling aspects. Incoming tokens are served immediately and moved to the depository. Scheduling in such places has priority over scheduling/service in timed queueing places and firing of timed transitions.

Apart from this, QPNs behaves similar to CGSPN. Formally QPNs are defined as follows:

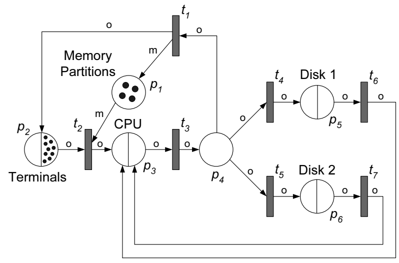


Fig. 19 A QPN Model of a Central Server with Memory Constraints (reprinted from [40]).

Definition 5 *A Queueing PN (QPN) is an 8-tuple $QPN = (P, T, C, I^-, I^+, M_0, Q, W)$ where:*

1. $CPN = (P, T, C, I^-, I^+, M_0)$ is the underlying Colored PN
2. $Q = (\tilde{Q}_1, \tilde{Q}_2, (q_1, ..., q_{|P|}))$ where
   – $\tilde{Q}_1 \subseteq P$ is the set of timed queueing places,
   – $\tilde{Q}_2 \subseteq P$ is the set of immediate queueing places, $\tilde{Q}_1 \cap \tilde{Q}_2 = \emptyset$ and
   – q_i denotes the description of a queue taking all colors of $C(p_i)$ into consideration, if p_i is a queueing place <u>or</u> equals the keyword 'null', if p_i is an ordinary place.
3. $W = (\tilde{W}_1, \tilde{W}_2, (w_1, ..., w_{|T|}))$ where
   – $\tilde{W}_1 \subseteq T$ is the set of timed transitions,
   – $\tilde{W}_2 \subseteq T$ is the set of immediate transitions, $\tilde{W}_1 \cap \tilde{W}_2 = \emptyset$, $\tilde{W}_1 \cup \tilde{W}_2 = T$ and
   – $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ such that $\forall c \in C(t_i) : w_i(c) \in \mathbb{R}^+$ is interpreted as a rate of a negative exponential distribution specifying the firing delay due to color c, if $t_i \in \tilde{W}_1$ <u>or</u> a firing weight specifying the relative firing frequency due to color c, if $t_i \in \tilde{W}_2$.

Example 1 (QPN [40]) Figure 19 shows an example of a QPN model of a central server system with memory constraints based on [40]. Place p_2 represents several terminals, where users start jobs (modeled with tokens of color 'o') after a certain thinking time. These jobs request service at the CPU (represented by a G/C/1/PS queue, where C stands for Coxian distribution) and two disk subsystems (represented by G/C/1/FCFS queues).

To enter the system each job has to allocate a certain amount of memory. The amount of memory needed by each job is assumed to be the same, which is represented by a token of color 'm' on place p_1. According to Definition 5, we have the following:
$QPN = (P, T, C, I^-, I^+, M_0, Q, W)$ where

- $CPN = (P, T, C, I^-, I^+, M_0)$ is the underlying Colored PN as depicted in Figure 19,
- $Q = (\tilde{Q_1}, \tilde{Q_2}, (null, \mathrm{G/C/\infty/IS}, \mathrm{G/C/1/PS}, null, \mathrm{G/C/1/FCFS}, \mathrm{G/C/1/FCFS}))$,
  $\tilde{Q_1} = \{p_2, p_3, p_5, p_6\}, \tilde{Q_2} = \emptyset$,
- $W = (\tilde{W_1}, \tilde{W_2}, (w_1, ..., w_{|T|}))$, where $\tilde{W_1} = \emptyset, \tilde{W_2} = T$ and $\forall c \in C(t_i) : w_i(c) := 1$, so that all transition firings are equally likely.

A.2 Solving of QPNs & Tools for QPNs

For QPNs, the analytic solving approach is well-defined [40] and implemented by several tools, e.g. [5, 46]. However, the analytic approach has limitations regarding the number of possible tokens and places which lead to a state explosion for models of real world applications [39]. Therefore, we decided to use a simulation-based QPN solver for our models. Such a simulation-based approach was presented in [39] which is implemented by the *QPME tool (Queueing Petri net Modeling Environment)* [6, 12, 13, 47]. We employed this tool to build and analyze our QPN models. QPME provides a QPN editor including a graphical user interface, which helps to construct QPN models and the optimized simulation engine SimQPN [6, 39] for model analysis. As a result of our work, several new features were added to QPME and to the SimQPN engine. Further, the performance of the solver was increased significantly.

B QPN Definitons

B.1 QPN Definition of Pattern 2

Places:

Place	**Type**	**Description**
Producer	S	Publish messages.
Topic	Q	Receives all incoming messages and forwards message notifIcations to n consumers.
Consumer	S	Consumes incoming message notifications.

Colors:

Color	**Description**
Message	Represents the sent message.
Message Notification (Not.)	Message notification.

Transitions:

Id	**Input**	**Output**	**Description**
T1	1 Message (*Producer*)	1 Message (*Topic*)	Producer sends messages.
T2	1 Message (*Topic*)	n Not. (*Topic*)	Notifications are created.
T3	1 Not. (*Topic*)	1 Not. (*Consumer*)	Consumer receives message notification.

B.2 QPN Definition of Pattern 3

Places:

Place	**Type**	**Description**
Producer	S	Publishes messages.
Topic	Q	Receives all incoming messages and forwards notifications to the consumers.
Consumer	S	Consumes incoming messages.
Controller	O	Controls the creation of notification token.
Init	O	Central place for the configuration.

Colors:

Color	**Description**
Message	Represents the published message.
Notification (Not.)	Message notification.
State A	Exists only if Controller is in state A.
State B	Exists only if Controller is in state B.
Subscriber A (Sub. A)	Each Sub. A stands for a notification, which will be generated after the state of the *Controller* place changes to state B.
Subscriber B (Sub. B)	Each Sub. B stands for a notification, which will be generated after the state of the *Controller* place changes to state A.
Subscriber	Is used to initialize the number of subscribers. Each token represents one subscriber.

Init No. of Colors:

Color	**Place**	**No.**	**Description**
State A	*Controller*	1	At the beginning the *Controller* place is in state A.
Subscriber	*Init*	n	One token for each consumer.

Transitions

Id	**Input**	**Output**	FW	**Description**
T0	1 Conf. Not. (*Init*)	n Not. B (*Controller*)	1	Initialization of Controller place.
T1	1 Message (*Producer*)	1 Message (*Topic*)	1	Producer publishes message.
T2-I	1 State A (*Controller*) 1 Message (*Topic*)	1 State B(*Controller*)	1	Switch state of *Controller* to B.
T2-II	1 State B (*Controller*) 1 Message (*Topic*)	1 State A(*Controller*)	1	Switch state of *Controller* to A.
T2-III	1 State A (*Controller*) 1 Sub. B (*Controller*)	1 State A (Controller) 1 Not. (*Topic*) 1 Sub. A (*Controller*)	∞	If in state A, all *Not. A* are converted to *Notications*
T2-IV	1 State B (*Controller*) 1 Sub. A (*Controller*)	1 State B (*Controller*) 1 Not. (*Topic*) 1 Sub. B (*Controller*)	∞	If in state B, all *Not. B* are converted to *Notications*
T3	1 Notification (*Topic*)	1 Not. (*Consumer*)	1	Consumer receives messages.

B.2.1 QPN Definition of Pattern 6

Places:

Place	**Type**	**Description**
Producer	S	Publishes messages.
T / Q	S	Stores all incoming messages.
Timer	Q	Timer queue (scheduling strategy: infinite server).
Trigger Store	O	Stores trigger tokens.
Consumer	S	Consumes incoming messages.

Colors:

Color	**Description**
Message	Represents the published messages.
Trigger	Triggers pull commands.
Sleep	Exists for time between an unsuccessful pull attempt and a reconnect.

Init No. of Colors:

Color	**Place**	**Count**	**Description**
Trigger	*Trigger Store*	j	j is equal to the number of messages the consumer can process in parallel.

Transitions:

Id	**Input**	**Output**	**FW**	**Description**
T1	1 Message (*Producer*)	1 Message (T/Q)	1	Producer publishes a message.
T2-I	1 Message (T/Q) 1 Trigger (*Trigger Store*)	1 Message (*Consumer*)	∞	Consumer pulls a message and processes it.
T2-II	1 Trigger (*Trigger Store*)	1 Sleep (*Timer*)	1	If no message is stored at the $T/Q \rightarrow$ go to sleep.
T3-I	1 Message (*Consumer*)	1 Trigger (*Trigger Store*)	1	After a message is processed, the consumer creates a trigger for a pull attempt.
T3-II	1 Sleep (*Timer*)	1 Trigger (*Trigger Store*)	1	After a specified time interval, the consumer wakes up to pull a message.