Performance Modeling and Benchmarking of Event-Based Systems

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von

Kai Sachs
aus Bad Homburg

Referenten:
Prof. Alejandro P. Buchmann, PhD, TU Darmstadt
Prof. Jean Bacon, PhD, University of Cambridge

Darmstadt 2010, Darmstädter Dissertation D17

Abstract

Event-based systems (EBS) are increasingly used as underlying technology in many mission critical areas and large-scale environments, such as environmental monitoring and location-based services. Moreover, novel event-based applications are typically highly distributed and data intensive with stringent requirements for performance and scalability. Common approaches to address these requirements are benchmarking and performance modeling. However, there was a lack of general performance modeling methodologies for EBS as well as test harnesses and benchmarks using representative workloads for EBS. Therefore, this thesis focused on approaches to benchmark EBS as well as the development of a performance modeling methodology. In this context, novel extensions for queueing Petri nets (QPNs) were proposed. The motivation was to support the development and maintenance of EBS that meet certain Quality-of-Service (QoS) requirements.

To address the lack of representative workloads we developed the first industry standard benchmark for EBS jointly with the Standard Performance Evaluation Corporation (SPEC) in whose development and specification the author was involved as a chief benchmark architect and lead developer. Our efforts resulted in the *SPECjms2007* standard benchmark. Its main contributions were twofold: based on the feedback of industrial partners, we specified a comprehensive standardized workload with different scaling options and implemented the benchmark using a newly developed complex and flexible framework. Using the SPECjms2007 benchmark we introduced a methodology for performance evaluation of message-oriented middleware platforms and showed how the workload can be tailored to evaluate selected performance aspects. The standardized workload can be applied to other EBS. E.g., we developed an innovative research benchmark for publish/subscribe-based communication named *jms2009-PS* based on the SPECjms2007 workload. The proposed benchmarks are now the de facto standard benchmarks for evaluating messaging platforms and have already been used successfully by several industrial and research organizations as a basis for further research on performance analysis of EBS.

To describe workload properties and routing behavior we introduced a novel formal definition of EBS and their performance aspects. Furthermore, we proposed an innovative approach to characterize the workload and to model the performance aspects of EBS. We used operational analysis techniques to describe the system traffic and derived an approximation for the mean event delivery latency. We showed how more detailed performance models based on QPNs could be built and used to provide more accurate performance prediction. It is the first general performance modeling methodology for EBS and can be used for an in-depth performance analysis as well as to identify potential bottlenecks. A further contribution is a novel terminology for performance modeling patterns targeting common aspects of event-based applications using QPNs.

To improve the modeling power of QPNs, we defined several extensions of the standard QPNs. They allow us to build models in a more flexible and general way and address several limitations of QPNs. By introducing an additional level of abstraction, it is possible to distinguish between logical and physical layers in models. This enables to flexibly map logical to physical resources and thus makes it easy to customize the model to a specific deployment. Furthermore, we addressed two limiting aspects of standard QPNs: constant cardinalities and lack of transition

priorities.

Finally, we validated our modeling methodology to model EBS in two case studies and predicted system behavior and performance under load successfully. As part of the first case study we extended SIENA, a well-known distributed EBS, with a runtime measurement framework and predicted the runtime behavior including delivery latency for a basic workload. In the second case study, we developed a comprehensive model of the complete SPECjms2007 workload. To model the workload we applied our performance modeling patterns as well as our QPN extensions. We considered a number of different scenarios with varying workload intensity (up to 4,500 transaction / 30,000 messages per second) and compared the model predictions against measurements. The results demonstrated the effectiveness and practicality of the proposed modeling and prediction methodology in the context of a real-world scenario.

This thesis opens up new avenues of frontier research in the area of event-based systems. Our performance modeling methodology can be used to build self-adaptive EBS using automatic model extraction techniques. Such systems could dynamically adjust their configuration to ensure that QoS requirements are continuously met.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 1
- 1.3 Approach and Contributions of This Thesis . . . 2
  - 1.3.1 Contributions in Performance Engineering . . . 3
  - 1.3.2 Contributions in the Area of Benchmarking . . . 4
  - 1.3.3 Summary . . . 6
  - 1.3.4 Related Activities and Publications . . . 7
- 1.4 Thesis Organization . . . 7

2 Background **9**
- 2.1 Event-Based Systems . . . 9
- 2.2 Technology Platforms of Event-based Systems . . . 12
  - 2.2.1 Active Databases . . . 12
  - 2.2.2 Continuous Queries, Stream Processing . . . 13
  - 2.2.3 Materialized Views . . . 13
  - 2.2.4 Message-Oriented Middleware (MOM) . . . 13
  - 2.2.5 Distributed Event-Based Systems (DEBS) . . . 18
  - 2.2.6 Reactive Middleware . . . 20
- 2.3 Introduction to Queueing Petri Nets . . . 21
- 2.4 Concluding Remarks . . . 24

3 Related Work **25**
- 3.1 Performance Modeling of Event-Based Systems . . . 25
- 3.2 Benchmarking of Event-Based Systems . . . 27
- 3.3 Patterns in Performance Modeling . . . 30
- 3.4 Concluding Remarks . . . 31

4 Performance Engineering of Event-Based Systems **33**
- 4.1 Modeling Methodology for EBS . . . 35
  - 4.1.1 Formal Definition . . . 35
  - 4.1.2 Analysis of the Event Routing Behavior . . . 36
  - 4.1.3 Estimation of Event Service Times . . . 37
  - 4.1.4 System Operational Analysis . . . 38
  - 4.1.5 Performance Model Construction and Evaluation . . . 40
- 4.2 Performance Modeling Pattern . . . 43
- 4.3 Extensions of QPNs . . . 73
  - 4.3.1 Mapping of Logical to Physical Resources . . . 73
  - 4.3.2 Non-Constant Cardinalities of Transitions . . . 74
  - 4.3.3 Priority of Transitions . . . 74

4.3.4 Tool Extension . . . 75
4.4 Concluding Remarks . . . 75

5 Benchmarking of Event-Based Systems 77
5.1 SPECjms2007 - A Standard Benchmark . . . 77
5.1.1 Workload Requirements and Goals of the SPECjms2007 Benchmark . . . 77
5.1.2 Workload Scenario . . . 80
5.1.3 Modeled Interactions . . . 81
5.1.4 SPECjms2007 Workload Characterization . . . 85
5.1.5 Benchmark Implementation . . . 95
5.2 Case Study I: SPECjms2007 . . . 101
5.2.1 Experimental Setting . . . 101
5.2.2 Horizontal and Vertical Scaling . . . 102
5.2.3 Customized Vertical Workloads . . . 102
5.2.4 Publish/Subscribe Messaging . . . 104
5.2.5 P2P Messaging . . . 107
5.2.6 Conclusions of the SPECjms2007 Case Study . . . 110
5.3 jms2009-PS - A Publish /Subscribe Benchmark . . . 110
5.3.1 Configuration Parameters . . . 113
5.4 Case Study II: jms2009-PS . . . 115
5.4.1 Introduction . . . 115
5.4.2 Test Scenarios . . . 115
5.4.3 Experimental Results . . . 117
5.5 Concluding Remarks . . . 118

6 Performance Modeling of EBS - Case Studies 119
6.1 DEBS Case Study . . . 119
6.1.1 Scenario . . . 119
6.1.2 Setup . . . 120
6.1.3 Experimental Results . . . 120
6.1.4 Conclusions . . . 121
6.2 Modeling SPECjms2007 . . . 121
6.2.1 Introduction . . . 121
6.2.2 Modeling SPECjms2007 . . . 122
6.2.3 Experimental Evaluation . . . 125
6.2.4 Conclusion . . . 133
6.3 Concluding Remarks . . . 133

7 Conclusions and Outlook 135
7.1 Ongoing and Future Work . . . 137

List of Figures

1.1 A (Distributed) Event-based System . . . 3

2.1 Event Notification in an EBS . . . 10
2.2 Point-to-point messaging. . . . 13
2.3 Example for Pub/Sub Messaging. . . . 13
2.4 Subscription Models . . . 15
2.5 Subscription Models (cont.) . . . 16
2.6 Router Network of REBECA [155] . . . 20
2.7 QPN Notation . . . 23
2.8 A QPN Model of a Central Server with Memory Constraints (reprinted from [26]). 24

4.1 Response Time (RT) in Traditional Request / Reply and in EBS . . . 34
4.2 System Topology . . . 35
4.3 High-Level System Model. . . . 41
4.4 Modeling Non-Poisson Event Publications. . . . 42
4.5 Firing of Transition φ_i in Mode μ_i^σ . . . 43
4.6 Modeling Network Connections. . . . 43
4.7 Standard Queue Pattern . . . 45
4.8 Standard Pub/Sub Pattern - Fixed Number of Subscribers . . . 46
4.9 Standard Pub/Sub Pattern - Configurable No. of Subscribers . . . 48
4.10 Example for Pattern 3 . . . 49
4.11 Pattern 3 using an Enqueuer for Incoming Events . . . 53
4.12 Time-Controlled Pull Pattern . . . 54
4.13 Time-controlled Pull Pattern . . . 57
4.14 Standard Pull Patterns . . . 59
4.15 Modeling a Thread Pool . . . 61
4.16 Pull Pattern - Request controlled II . . . 63
4.17 Standard Queue Pattern . . . 66
4.18 Load Balancer - Random . . . 68
4.19 Load Balancer - Round Robin . . . 69
4.20 Load Balancer - Pull . . . 71
4.21 Physical and Logical Layers . . . 73
4.22 Example for Priority of Transitions . . . 74

5.1 Overview of the Workload Scenario and its Roles . . . 80
5.2 Interaction 1 - Communication between SM and DC . . . 82
5.3 Interaction 2 - Communication between SP and DC . . . 82
5.4 Workflow of the SPECjms2007 Interactions (N)P=(Non-)Persistent; (N)T= (Non-)Transactional . . . 83
5.5 # Locations for Horiz. Topology . . . 91
5.6 Horiz. Topology Message Mix . . . 91

5.7 Proportions of the Interactions based on Msg. Throughput . . . 91
5.8 Proportions of the Interactions based on Msg. Traffic in KBytes . . . 91
5.9 Horizontal Topology: # msg. sent . . . 92
5.10 Message Traffic in Kbytes . . . 92
5.11 Vert. Topology Message Mix . . . 93
5.12 Vertical Topology: # msg. sent . . . 94
5.13 Message Traffic in Kbytes . . . 94
5.14 Proportions of the Interactions based on Msg. Throughput . . . 94
5.15 Proportions of the Interactions based on Msg. Traffic in KBytes . . . 94
5.16 Driver Framework . . . 96
5.17 Formal Measurement Points during SPECjms2007 Run [214] . . . 98
5.18 Output of Run Time Reporter . . . 98
5.19 Experimental Environment . . . 101
5.20 Measurement Results for Horizontal Experiments . . . 102
5.21 Measurement Results for Vertical Experiments . . . 103
5.22 Measurement Results for Customized Vertical Experiments with P2P Messaging 103
5.23 Measurement Results for Customized Vertical Experiments with Pub/Sub Messaging . . . 104
5.24 Scenario 1: NPNTND Pub/Sub Messaging with Increasing Number of Consumers 105
5.25 Scenario 2: NPNTND Pub/Sub Messaging with Increasing Message Size . . . 106
5.26 Scenario 3: NPNTND Pub/Sub Messaging with Varying Number of Producers and Consumers . . . 107
5.27 Scenario 4: PTD Pub/Sub Messaging with Increasing Number of Consumers . . . 108
5.28 Scenario 5: NPNTND vs. PTD Pub/Sub Messaging . . . 109
5.29 Scenarios 1 and 2: NPNT vs. PT P2P Messaging with Increasing Number of Queues . . . 110
5.30 Scenario 3: PT P2P Messaging with Increasing Message Size . . . 112
5.31 Experimental Environment . . . 115
5.32 Considered Scenarios . . . 116
5.33 Experimental Results . . . 117

6.1 Broker Topology . . . 120
6.2 Model of Interaction Drivers . . . 123
6.3 Models of Interactions 3, 4, 5, 6 and 7 . . . 124
6.4 Model of Interaction 1 . . . 125
6.5 Model of Interaction 2 . . . 126
6.6 Experimental Environment . . . 128
6.7 Distribution of the Message Size . . . 129
6.8 Server CPU Utilization and Message Traffic for Customized Vertical Topology . . . 131
6.9 Model Predictions Compared to Measurements for Scenarios 1, 2 and 3 . . . 132

7.1 Open Research Issues . . . 137
7.2 Benchmark Aspects . . . 138

List of Tables

2.1 Standards for MOMs 17
2.2 Message-Oriented Middleware (OS = Open Source, P = Planned), State: March 2010 19

4.1 Performance Modeling Patterns 44

5.1 Message Types Used in The Interactions - (N)P=(Non-)Persistent; (N)T=(Non-)Transactional; (N)D=(Non-)Durable 86
5.2 Parameters for Message Size Calculation 87
5.3 Message Groups 87
5.4 Interaction Rates for the Horizontal Topology 91
5.5 Message Sizes in KByte 93
5.6 Topology Message Mix 93
5.7 Interaction Rate Scaling Factors for the Vertical Topology 94
5.8 Audit Tests 99
5.9 Configuration for Pub/Sub Scenarios 105
5.10 Configuration for P2P Scenarios 108
5.11 Configuration parameters supported for each message type. 111
5.12 Target destination options. 114

6.1 Broker Throughput (msg. / sec) 121
6.2 Delivery Latency (ms) 121
6.3 Scenario Transaction Mix 127
6.4 Detailed Results for Scenarios 1, 2 and 3 130
6.5 Relative Server CPU Load of Interactions 130

List of Acronyms

Acronym	Meaning
$\lambda_j^{t,k}$	Arrival Rate at which events of type t published by publisher k arrive at node j.
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AS	Application Server/s
CB	Component-Based
CEP	Complex Event Processing
CGSPN	Colored Generalized Stochastic Petri Net
CPN	Colored Petri Net
D	Durable
DBMS	Database Management System
DBS	Database Server
DC	Distribution Center
DDS	Data Distribution Service for Real-time Systems
DEBS	Distributed Event-Based Systems
EBS	Event-Based Systems
ECA	Event-Condition-Action
EDA	Event-Driven Architecture
EJB	Enterprise Java Bean
EPTS	Event-Processing Technical Society
ESB	Enterprise Service Bus
FCFS	First-Come-First-Serve (scheduling strategy)
FIFO	First-In-First-Out
FW	Fire Weight
GB	Giga Byte
GC	Garbage Collection
GSPN	Generalized Stochastic Petri Net
HLQPN	High-Level Queueing Petri Net
HQ	Headquarters
HQPN	Hierarchical Queueing Petri Net
HTTP	Hypertext Transfer Protocol
IID	Independent and Identically Distributed (random variables)
IR	Injection Rate
IS	Infinite Server (scheduling strategy)
IT	Information Technology
Java EE	Java Enterprise Edition Platform
Java SE	Java Standard Edition Platform

Acronym	Meaning
JCP	Java Community Process
JDBC	Java Database Connectivity
JMS	Java Messaging Service
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KByte	Kilo Byte
LAN	Local Area Network
LB	Load balancer
LBC	Load balancing component
LCFS	Last-Come-First-Served (scheduling strategy)
LLQPN	Low-Level Queueing Petri Net
LQN	Layered Queueing Network
MB	Mega Byte
MDB	Message-Driven Bean
MOM	Message-Oriented Middleware
MQ	Message Queue
ms	Millisecond
msg	Message
ND	Non-Durable
NP	Non-Persistent
NT	Non-Transactional
OLTP	Online Transaction Processing
OMG	Object Management Group
OS	Operating System
P	Persistent
P2P	Point-To-Point
PE	Performance Engineering
PerfMP	Performance Modeling Pattern
PN	(Ordinary) Petri Net
PO	Purchase Order
PS	Processor-Sharing (scheduling strategy)
P& S	Performance and Scalability
Pub/Sub	Publish / Subscribe
Q	Queue
QN	Queueing Network
QoS	Quality-of-Service
QPN	Queueing Petri Net
$R_{t,j}^{res}$	Response Time of resource res of events of type t at node / connection j.
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RR	Round-Robin (scheduling strategy)
$S_{t,j}^{res}$	Mean Service Time of resource res of an event of type t at node / connection j.
sec	Second
SLAs	Service Level Agreements
SM	Supermarket
SP	Supplier
SPEC	Standard Performance Evaluation Corporation

Acronym	**Meaning**
SPEC-OSG	SPEC's Open Systems Group
SPE	Software Performance Engineering
SPN	Stochastic Petri Net
SQL	Structured Query Language
ST	Subscription Type
SUT	System Under Test
T	Transactional
TD	Target Destination
TPC	Transaction Processing Performance Council
UML	Unified Modeling Language
U_j^{res}	Utilization of resource res at node j
$V_j^{t,k}$	Relative Arrival Rate(Visit Ratio) of an event of type t published by publisher k visits at node j.
WAN	Wide Area Network
X_j^t	Throughput

Chapter 1

Introduction

1.1 Motivation

Event-based systems (EBS) have been gaining attention in many domains of industry. With the advent of ambient intelligence and ubiquitous computing, many new applications of EBS have been proposed [101], for example, in the areas of transport information monitoring [15, 206], event-driven supply chain management [33, 1, 196], ubiquitous (wireless) sensor environments [2, 30, 179], environmental monitoring, ambient assisted living, and location-based services [59, 35, 102, 100]. Many of these novel event-based applications are highly distributed and data intensive and hence pose some serious performance and scalability challenges. With the increasing popularity of EBS and their gradual adoption in mission critical areas, performance issues are becoming a major concern. The performance and scalability of event-based middleware (used to process real-time event data) are of crucial importance for the successful adoption of such applications in the industry, and methodologies are needed to guarantee an adequate quality-of-service (QoS) level.

As a consequence, EBS have to be subjected to a rigorous performance analysis at all stages of an application's life cycle. To meet QoS requirements, techniques for predicting system performance as a function of configuration and workload are needed. Common performance metrics of interest are, for example, expected event notification latency as well as utilization and message throughput of the various system components (e.g., event brokers, network links). Since the components of EBS are loosely coupled and communicate asynchronously, the understanding of these metrics may differ from traditional performance engineering. However, obtaining such information is essential in order to determine the optimal system topology, configuration, and capacity for providing adequate QoS to applications at a reasonable cost. Moreover, given the dynamics of most EBS applications, it is important that the performance of the system is continuously monitored and analyzed during operation to help anticipate changes in the workload and take corrective actions to ensure that QoS requirements are satisfied.

The goal of this thesis is to develop novel approaches to analyze and predict the behavior of EBS and their performance and scalability under load. To achieve this, we focus on workload characterization, benchmarking and performance modeling.

1.2 Problem Statement

EBS are often used in business critical environments and thus their reliability is crucial for the whole IT infrastructure. A certain QoS level has to be ensured. Since EBS are loosely coupled, highly distributed, data intensive and often heterogeneous systems, this is a very challenging task. The dynamics of most EBS applications and their underlying middleware makes

it difficult to monitor and analyze system performance during operation. Closely related to EBS is the publish-subscribe paradigm that is nowadays used as a building block in major new software architectures and technology domains such as enterprise service bus (ESB), enterprise application integration (EAI), service-oriented architecture (SOA) and event-driven architecture (EDA) [50, 51, 101]. Modern EBS are implemented using several technology platforms such as centralized systems based on message-oriented middleware (MOM), e.g., IBM WebSphere and TIBCO Rendezvous, or large-scale distributed event-based systems (DEBS), e.g., SIENA [40], Hermes[179] or REBECA [154]. Several standards such as Java Message Service (JMS) [221], Advanced Message Queuing Protocol (AMQP), and Data Distribution Service (DDS) have been established and are supported by middleware vendors.

A major task of system architects, developers and deployment managers is to choose adequate technologies and deploy the EBS in such a way that the QoS requirements are fulfilled. While developing, deploying, and maintaining event-based applications (and their underlying middleware), the following questions are often raised:

- What performance would the system exhibit for a given deployment topology, configuration, and workload scenario?
- What is a typical workload scenario?
- What would be the expected notification and subscription delays as well as the utilization of the various system components (e.g., brokers, network links)?
- What maximum load (number of publishers and subscribers, event publication rates) would the system be able to handle without breaking the service level agreements?
- What would be the optimal number of brokers and the optimal system topology?
- Which components would be most utilized as the load increases and when are they potential bottlenecks?
- Will the event-based middleware scale for future loads?
- Which product offers the best performance for a certain workload scenario?

Benchmarks and performance modeling techniques help answering these questions. However, there is a lack of test harnesses and benchmarks using representative workloads for EBS. The same applies to performance models. Only a few approaches for modeling EBS have been published and they have limitations such as unrealistic assumptions or scalability issues. Furthermore, most traditional performance modeling techniques have not yet been evaluated using EBS. As far as the author of this thesis knows, neither an EBS benchmark implementing a real-world workload nor a performance modeling methodology focusing on EBS was available when this thesis effort was started.

1.3 Approach and Contributions of This Thesis

In this section, we summarize the results of this thesis. First, we provide a brief overview of these results in the area of performance engineering. Second, we discuss those that lie within the area of benchmarking of EBS. We summarize our contributions and conclude this section with an overview of related activities and publications.

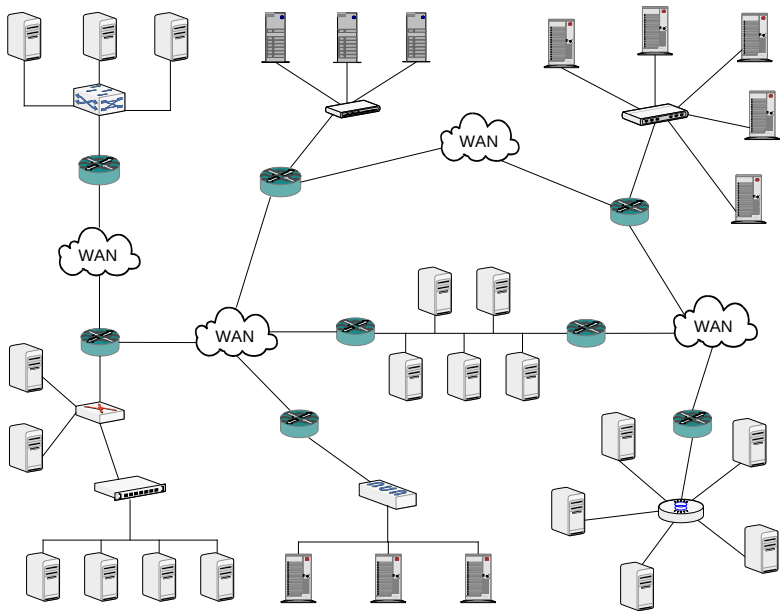


Figure 1.1: A (Distributed) Event-based System

1.3.1 Contributions in Performance Engineering

Modeling EBS is challenging because of the decoupling of the communicating parties, on the one hand, and the dynamic changes in the system structure and behavior, on the other hand. If a request is sent in a traditional request/reply-based distributed system, it is sent directly to a given destination. This makes it easy to identify the system components and resources involved in its processing. In contrast, when an event is published in an EBS, as illustrated in Figure 1.1, it is not addressed to a particular destination, but rather routed along all those paths that lead to event consumers. The event might have to be processed by multiple system nodes on its way to consumers and it is difficult to predict in advance which nodes will be involved in delivering the event. Moreover, individual events published by a given producer might be routed along completely different paths, visiting various sets of system nodes. Another difficulty stems from the fact that every time a new subscription is created or an existing one is modified, this might lead to significant changes in the system behavior. Thus, the dynamics of EBS require that workload characterization be done on a regular basis in order to reflect changes in the system configuration and workload.

We proposed a novel approach to workload characterization and performance modeling of EBS, aiming to address the above challenges. We developed a workload model based on monitoring data that captures the system routing behavior and resource consumption at a level that allows us to use this information as an input to performance models. The workload model we proposed does not make any assumptions about the algorithms used at the event routing and overlay network layers of the system. Using the workload model and applying operational analysis techniques, we were able to characterize the message traffic and determine the utilization of system components. This in turn enabled us to derive an approximation of the mean event delivery latency. For more accurate performance prediction, we proposed detailed performance models based on *queueing Petri nets (QPNs)*. Furthermore, we discussed how different features of EBS can be reflected in performance models and introduced eleven *performance modeling patterns* for the most common settings and features of EBS. We used QPNs as a modeling technique to illustrate the patterns. However, several of our patterns are not (or only with high effort) convertible with standard QPNs.

To solve these shortcomings and limitations, and to increase modeling flexibility without increasing its complexity, we developed several new features for QPNs:

1. *The ability to have multiple queueing places share the same physical queue*
   We used this feature to implement the concept of *mapping logical to physical resources.*

2. *Support of Non-Constant Cardinalities in Transitions*

3. *Priority Support for Transitions*

Our extensions allow building QPNs in a more flexible and general way. The concept of mapping logical to physical resources is implemented in the QPME / SimQPN software tools.

Our modeling approach is the first to provide a comprehensive methodology for workload characterization and performance modeling of EBS that is applicable to a wide range of systems. It allows the modeling of individual message flows and interactions in an EBS. This methodology helps identify and eliminate bottlenecks and ensure that systems are designed and sized to meet their QoS requirements. We demonstrated our approach in two case studies:

Case Study I: A case study using the SIENA publish/subscribe system with a basic workload comprising a single message type was carried out. The SIENA publish/subscribe system was enhanced with self-monitoring functionality. We instrumented the system to monitor and collect the event publication rates and routing probabilities needed for characterizing the workload. The workload model thus generated was used as input for a QPN model of the system, which was analyzed by means of simulation. The model predictions were compared against measurements on the real system and the modeling error was below 5% for all metrics considered. This case study served as a proof-of-concept, confirming the effectiveness of the proposed methodology.

Case Study II: While the results of the first case study are promising, they do not reveal whether the proposed modeling methodology scales to realistic systems providing performance predictions with reasonable accuracy. Therefore, we presented a second case study of a representative state-of-the-art event-driven application. We applied our modeling approach and extended it to address the issues that arise when considering a complex and realistic application. The application we chose is the SPECjms2007 standard benchmark, which is designed to be representative of real-world event-driven applications. We developed a comprehensive model of the complete workload including the persistent layer, point-to-point, and publish/subscribe communication, and evaluated its accuracy in a commercial middleware environment —the Oracle WebLogic Server Enterprise Edition. 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 4500 transaction per second). To the best of our knowledge, no models of representative systems of this size and complexity exist in the literature.

The modeling technique presented can be exploited as a powerful tool for performance prediction and capacity planning during the software engineering lifecycle of event-driven applications. We published our performance modeling methodology in [132] including case study I. In [131] we published a survey of currently available modeling techniques for EBS.

1.3.2 Contributions in the Area of Benchmarking

Over the last decade several proprietary and open-source benchmarks for evaluating EBS platforms have been developed and used in the industry (e.g., [213, 106, 8, 118]). Benchmarks not only help to compare alternative platforms and validate them, but can also be exploited to study

the effect of different platform configuration parameters on overall system performance. However, for a benchmark to be useful and reliable, it must fulfill several fundamental requirements. First of all, the benchmark workload must be designed to stress platforms in a manner representative of real-world messaging applications. It has to exercise all critical services offered by the platforms and must provide a basis for performance comparisons. Finally, the benchmark must generate reproducible results without having any inherent scalability limitations. While previous benchmarks for EBS have been employed extensively for performance testing and product comparisons, they do not meet the above requirements. This lack can be attributed to the fact that these benchmarks use artificial workloads not reflecting any real-world application scenario. Furthermore, they typically concentrate on stressing individual MOM features in isolation and do not provide a comprehensive and representative workload for evaluating the overall MOM server performance.

To address these concerns, in September 2005, we launched a project at the Standard Performance Evaluation Corporation with the goal of developing a standard benchmark for evaluating the performance and scalability of MOM products. The effort continued over a period of two years and the new benchmark was released at the end of 2007. The benchmark was called SPECjms2007 and is the first industry standard benchmark for message-oriented middleware. It was developed under the lead of TU Darmstadt with the participation of IBM, Sun, BEA, Sybase, Apache, Oracle, and JBoss. SPECjms2007 exercises messaging products through the JMS standard interface that is supported by all major MOM vendors. The contributions to the SPECjms2007 benchmark that are also part of this thesis are in the area of the specification, implementation, and detailed analysis of the SPECjms2007 benchmark.

Based on the feedback of our industrial partners, we specified a standard workload for message-driven EBS and implemented it using a newly developed flexible framework. The aim of the SPECjms2007 benchmark is to provide a standard workload and metrics for measuring and evaluating the performance and scalability of MOM platforms. From the beginning the workload was designed in a parameterized manner with default settings for the industry standard benchmark and other settings to support research. Based on our analysis of the demands of benchmarks we presented a list of requirements a benchmark and its workload has to fulfill: First of all, it must be based on a representative workload scenario that reflects the way platform services are exercised in real-life systems. The communication style and the types of messages sent or received by the different parties in the benchmark scenario should represent a typical transaction mix. The goal is to allow users to relate the observed behavior to their own applications and environments. Second, the workload should be comprehensive in that it should exercise all platform features typically used in MOM applications including both point-to-point (P2P) and publish/subscribe (pub/sub) messaging. The features and services stressed should be weighted according to their usage in real-life systems. The third requirement is that the workload should be focused on measuring the performance and scalability of the MOM infrastructure. It should minimize the impact of other components and services that are typically used in the chosen application scenario. For example, if a database was used to store business data and manage the application state, it could easily become the limiting factor of the benchmark—as experience with other benchmarks has shown [126]. Finally, the SPECjms2007 workload must not have any inherent scalability limitations. The user should be able to scale the workload both by increasing the number of destinations (queues and topics) as well as the message traffic pushed through a destination.

SPECjms2007 provides numerous configuration options to build customized workload scenarios. In an extensive analysis of the workload, we discussed the different interactions and traffic produced by the benchmark and described how to specify a customized transaction mix by configuring, e.g., message properties such as size, delivery mode, and arrival rate. We illustrated correlations between the configuration options and presented a methodology for using a standard benchmark to evaluate the performance of a MOM. We demonstrated our approach in a comprehensive case study of leading commercial JMS platforms, the Oracle Weblogic Enterprise

Server, conducting an in-depth performance analysis of the platform under a number of different workload and configuration scenarios and illustrated how the workload may be customized to exercise and evaluate selected aspects of MOM performance.

The SPECjms2007 workload scenario was designed in a general fashion and can be easily applied to specific types of EBS. For this purpose, we explained how the workload can be implemented for a pure publish/subscribe environment considering *jms2009-PS*—a benchmark for publish/subscribe-based communication—as an example. jms2009-PS provides a flexible framework for performance analysis with a strong focus on research and implements the workload using topic-based pub/sub. It allows the user to define complex scenarios in an easy and flexible way. In a case study we demonstrated how the benchmark can be used as test harness to compare different topic deployments and analyzed the influence of message filtering.

Both benchmarks are actively used by industry and academia. Since they exercise MOMs in a realistic way, they are used as benchmarks, as test harnesses, and as reference applications. Resulting from our efforts on benchmarking and performance analysis, we have established and maintained several successful collaborations with middleware vendors and academic institutions: e.g., we cooperated with JBoss and the Apache Foundation to publish official SPECjms2007 results (reviewed by OSG Java Subcommittee of SPEC[1]) and Karlsruhe Institute of Technology [215].

The results of our work focusing on benchmark development and SPECjms2007 (including workload characterization and framework) were published in [203, 202, 198, 130]. Our methodology for performance evaluation using standard benchmarks was introduced in [201]. The jms2009-PS benchmark is within the focus of [199, 197]. An overview of our activities in the area of MOM benchmarking is provided in [200], and in [12] we demonstrated an approach to benchmark AMQP-based middleware using SPECjms2007 and jms2009-PS.

1.3.3 Summary

The contributions of this thesis are in the areas of *benchmarking* and *performance engineering* of *event-based systems*. The main contributions can be outlined as follows:

1. ***Conceptually:***

   (a) **Analysis and Classification of Benchmark Requirements**
   An analysis of the requirements that a benchmark and its workload must meet in order to be meaningful. We deduce a classification for these requirements.

   (b) **Performance Evaluation Methodology Based on Benchmarks**
   A new methodology to analyze the performance of messaging middleware using standard benchmarks.

   (c) **Performance Modeling Methodology for EBS**
   A novel approach to model EBS, using queueing Petri nets for predicting system and performance behavior.

   (d) **Queueing Petri Net (QPN) Extensions**
   An extension of the QPN formalism in several ways, e.g., by simplifying the abstractions for modeling logical software entities, such as, message destinations.

2. ***Practically:***

   (a) **A Novel Representative Workload and Benchmark for EBS**
   The first standard benchmark for MOM—*SPECjms2007*:

[1]Standard Performance Evaluation Corporation (SPEC) is a non-profit organization whose mission is to establish, maintain, and endorse standardized benchmarks to evaluate performance for the newest generation of computing systems. With more than 80 members and associates from industry and academia, SPEC is one of the world's largest and most successful performance standardization organizations.

- Standardized by the Standard Performance Evaluation Corporation (SPEC).
- A comprehensive and representative workload for message-oriented EBS.
- Additionally, a flexible benchmark framework for in-depth analysis.

(b) **Performance Evaluation of a State-of-the-Art MOM**
A case study of a leading commercial MOM conducting an in-depth performance analysis of the platform under a number of different workload and configuration scenarios using the SPECjms2007 standard benchmark.

(c) **Performance Modeling Case Studies of EBS**

Case Study I: A simple application deployed on a representative DEBS platform.

Case Study II: A complex and realistic application deployed on a representative MOM.

1.3.4 Related Activities and Publications

Event-Based Applications and Enabling Technologies

In [101], we introduced the basic notions of event processing to create a common understanding, presented the enabling technologies that are used for the implementation of event-based systems, surveyed a wide range of applications identifying their main features, and discussed open research issues.

ECA Rule Engines

We demonstrated in [83] an implementation of an ECA rule engine of an embedded system. Such ECA rule engines provide a flexible environment for supporting the management, reconfiguration, and execution of business rules. However, modeling the performance of a rule engine is challenging because of its reactive nature. In [84], we presented an analytical performance model for ECA rule engines. We discussed the difficulties in building a performance model of an ECA rule engine and introduced a novel methodology for the performance evaluation of ECA rule engines. We introduced the concept of event paths and showed how ECA rules can be mapped to queueing networks.

QoS of Event-Based Systems

In [11], we provided a general overview of QoS in event-based systems. We introduced an architecture that supports different types of QoS in an EBS and discussed the QoS in the context of MOM and complex event processing (CEP).

Statistical Inference and Performance Models

Statistical inference is the process of drawing conclusions by applying statistics to observations or hypotheses based on quantitative data. The goal is to determine the relationship between input and output parameters. In [88], we proposed an approach to the statistical inference of performance models based on observation data. In our case study, we used *multivariate adaptive regression splines* (MARS) and genetic optimization to estimate the influence of message sizes and arrival rates on the system performance of a MOM. We considered message delivery time, throughput, and resource utilization as part of our analysis.

1.4 Thesis Organization

This thesis is organized as follows. In Chapter 2, we provide an overview of the state-of-the-art and related work in the areas of event-based systems (EBS), performance modeling, and

benchmarking. We discuss the meaning of events and typical applications as well as different middlewares for EBS. We introduce queueing Petri nets (QPN).

Chapter 3 reviews the related work in the areas of performance modeling of EBS including the use of patterns in performance modeling. We discuss the current state of EBS benchmarking, especially in the area of message-oriented middleware (MOM), and provide an overview of previous work and performance studies.

In Chapter 4, we present a methodology for modeling performance aspects of event-based systems. In the second part of this chapter, we introduce the *performance modeling patterns* (PerfMP) for EBS. The PerfMP describe how common interaction patterns and communication behaviors of EBS can be represented in a performance model. As an example of these representations, we use QPNs. At the end of the chapter we introduce a set of extensions for QPNs. These extensions are developed to support the modeling of EBS and component-oriented software.

In Chapter 5 we start with an analysis of the different requirements a benchmark has to fulfill and then provide a classification for these requirements. Taking these requirements into account, we developed the *SPECjms2007* benchmark, which is the focus of this chapter. The complex workload of this benchmark was implemented using a flexible framework with numerous configuration options. Furthermore, we illustrate the different aspects of SPECjms2007 in detail including a comprehensive workload characterization and a description of the framework. A case study showing how to apply the SPECjms2007 benchmark to a MOM for analyzing different performance aspects is provided. The workload is defined based on the experience of the industrial members of SPEC. Since it is mainly focused on point-to-point communication, we extended the workload of SPECjms2007 and used the benchmark framework to develop a performance test harness for publish / subscribe based communication, the *jms2009-PS* performance test harness. An introduction to the different features of jms2009-PS and a case study using jms2009-PS complete the chapter.

In Chapter 6, we combine the results of the previous chapters (area performance modeling, event-based systems, and benchmarking) and present the results of two case studies where we apply our modeling methodology to two different scenarios. In the first case study, we apply the method to a distributed event-based system (DEBS). As workload, we use messages of the SPECjms2007 scenario. As underlying DEBS, we use SIENA. In the second case study, we model the complete workload of SPECjms2007 (including different message sizes etc.) using our PerfMP and extended QPNs.

Finally, the summary and conclusions of this thesis including an outlook for future research are presented in Chapter 7.

Chapter 2

Background

In this chapter we provide an overview of the state-of-the-art in the area of event-based systems (EBS) and performance modeling. We discuss the meaning of events and typical applications as well as different middlewares for EBS and introduce performance models called queueing Petri nets (QPN).

2.1 Event-Based Systems

For a better understanding of event-based systems we first introduce our understanding of events and related terms. The following definitions are based on our work presented in [101]. However, slightly different definitions are used by [144, 143, 223].

An *event* is defined as a significant change in the state of the universe [48, 50]. By referring to significant changes, the infinite number of events is limited to those that are relevant to an application. Since time is an inherent dimension of the universe, two observations of the universe at different points in time constitute two distinct events, even if no other properties have changed.

Further, we distinguish between *change events* and *status events*:

- A *change event* is an observed state change in comparison to the previous state.
  Example: An object has changed its position by a few meters.

- A *status event* describes a current state.
  Example: Two readings of a temperature sensor at different points in time. Even the observation that both yielded the same temperature constitutes an event.

By considering time an integral part of the state of the universe, both change and status events can be modeled in a uniform manner: a status event is a change event, in which the time has changed.

Events must be observed to be reported and processed. An *observation* captures a discrete instance of a (possibly continuous) signal. An observation of an event carries a timestamp and descriptive parameters and is typically represented as a tuple of values. Depending on the type of event and application system, the timestamp may be just one point (point semantics of time, *instantaneous event*) or an interval (interval semantics of time, *interval event*). Parameters may be absolute values or deltas relative to older reference values ($\Rightarrow$ change event).

Events, or more precisely, their representation, must be reported to *event consumers (event sink, event handler, event listener, event subscriber)* using an *event notification*. It is generally accepted that event notifications are routed from *event producers (event source, publisher)* to event consumers by a *notification service*. The notification service decouples producers and consumers, and provides the routing from source to sink [155]. In the simplest form, this

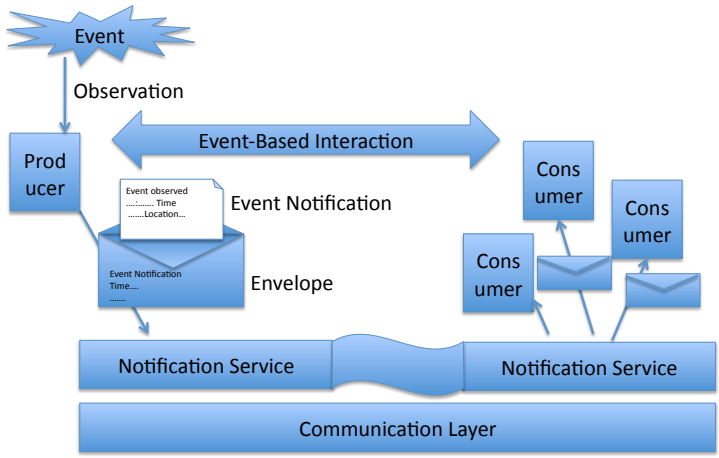


Figure 2.1: Event Notification in an EBS

may be a low-level channel into which event notifications are placed and from where they are retrieved. In this case, the envelope of the notification is minimal and streams of tuples are delivered over a fixed channel. However, the notification service may be a more sophisticated network of brokers routing the notifications based on type or content. Notifications consist of one or more event representations packaged in an *envelope*. Routing may occur on the content of the envelope data or the content of the notification. The notification process is illustrated in Figure 2.1.

Events may be *simple events* or compositions of simple and/or other *composite events*. Simple events may be individual sensor observations, method invocations or absolute temporal events. Composite events are aggregations or collections of other events. Composite events are produced from event representations and the operators of an event algebra. Two commonly used approaches to event composition exist:

1. *Event trees* consisting of events at the leaves and the operators of an event algebra in the inner nodes [46] pioneered by active database systems, and

2. *Continuous or streaming queries* based on the operators of relational algebra applied to subsets of streams of tuples (sliding windows) [47].

Derived events (synthesized events*)* are caused by other events and often are at a different level of abstraction. For example, five failed logins with the wrong password may cause an intrusion attempt event to be signaled. Derived events involve semantic knowledge. They may be detected automatically, e.g., from a combination of sensor readings as the action part of an *event-condition-action (ECA)* [54, 84] rule or be raised explicitly, e.g., based on direct observation of an event by a user. Derived events are often enriched with data from external sources.

An *event-based system* is a software system in which observed events cause reactions in the system. Event-based systems consist of three essential parts:

- *Monitoring component*,
- *Transmission mechanism*, and
- a *Reactive component*.

The monitoring component is responsible for event observation, representation, and composition as described above. The transmission mechanism is responsible for event notification. It

is generally accepted that event notification is push-based. In push-based systems, producers disseminate information to consumers; in pull-based systems the consumer must request the information. Some authors go as far as requiring a complete decoupling of event producers and event consumers through a publish/subscribe notification service [155]. For generality, we also accept point-to-point notification of events, e.g., in the context of messaging systems [103]. This implies a tighter coupling between producers and consumers, since the producers must be aware of the consumers to notify them without the help of a broker.

The reactive component of an event-based system expresses the application logic in form of rules (or other code) triggered by the corresponding events. This behavior is also called *event-driven*: an entity is event-driven, if it consumes event notifications and, if appropriate, reacts on them.

Rules may have different formats that result in different execution models. Procedural ECA rules are fired whenever the corresponding event (simple, composite or derived) is raised. The condition acts as a guard that can be used to express more complex application logic. Only if the condition is met, the action is executed. Missing conditions are considered to be true and result in event-action rules. Much debate has occurred in the past concerning the best separation of functionality between events and conditions. More powerful event expressions decrease the need for explicit conditions but require more powerful event algebras. This also makes the event detection mechanism heavier and more difficult for users to use properly. On the other hand, a lightweight event mechanism can be more responsive and is less error prone but requires an explicit condition to express more powerful application logic. The decision on the trade-off between expressiveness of the event language and the lightweight nature of the event system is domain-dependent.

While the logical distinction is clear, specific implementations of event-based systems may partition the functionality differently. In particular the event composition may be implemented at the monitoring component, in the notification service, or as part of the reactive component. The decision of where to realize event composition depends on many application and environment specific factors, such as capabilities of the sensing devices, bandwidth of the communication channels, complexity of the composite events, and source of the events that are to be composed.

2.2 Technology Platforms of Event-based Systems

Many different technologies have contributed to the field of event-based systems. We review the contributions of the following platforms in this section:

- Active databases
- Continuous queries, stream processing
- Distributed event-based systems (DEBS)
- Materialized views
- Message-oriented middleware (MOM)
- Reactive middleware

Since our work presented in this thesis is focusing on MOMs and DEBS, we discuss them in more detail than the others.

2.2.1 Active Databases

Active databases were developed in the mid to late 1980s [174, 233]. Two distinct strands can be identified:

- Relational and
- Object-oriented active databases.

Relational active databases were limited mainly to basic database events, such as update, insert and delete. They could express conditions on either the old or new state of a relation, and the action was always some SQL statement. Today's triggers in SQL are the relational incarnation of simple ECA rules. Relational active databases introduced the notions of before, after or instead execution, meaning that the rule should be executed accordingly before, after or instead of the triggering statement.

Object-oriented active databases had a richer event type system. It included any method invocation, state changes effected through generic accessor functions, temporal events, control flow events, arbitrary user defined events, and composition of events through event algebras of varying expressiveness. The first generation of active ooDBMSs assumed a central clock, point semantics for the events, and a complete ordering of events. Active ooDBMSs introduced coupling modes to define when a rule should be executed (immediately or deferred) and whether it should be executed within the scope of the triggering transaction or as a separate transaction. If rules execute in separate transactions this can occur independently or be causally dependent, in which case the triggered transaction may only begin or end execution depending on the fate of the triggering transaction. The various causal dependence modes take care of violations of ACID properties that occur when uncommitted data is made visible to independent transactions. Another major contribution was the notion of event consumption, referring to the way in which events are consumed during event composition. Four consumption modes (originally termed contexts) were defined: chronicle, recent, continuous, and cumulative. These determine that the events be consumed either in chronological order (typical in workflow-like applications), always using the most recent occurrence (typical in control applications), in form of windows (typical in financial applications) or accumulating the effects of incoming events until another event occurs (typical in inventory control situations).

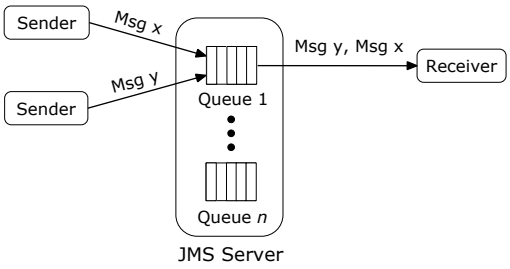


Figure 2.2: Point-to-point messaging.

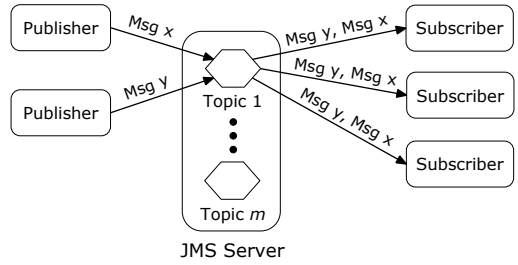


Figure 2.3: Example for Pub/Sub Messaging.

2.2.2 Continuous Queries, Stream Processing

Continuous queries [52] can be seen as an attempt to change the processing paradigm from issuing a single non-persisting query against stored, persistent data to storing the query persistently in the database and applying it to streams of incoming data. Continuous/continual queries were expressed in variants of the SQL language modified to operate on windows [140]. These can be defined either through temporal events or through a count of incoming events. While early work on continuous queries assumed that the continuous queries would operate on the stored data and would be executed by the traditional query engine, work on streaming queries has changed the processing paradigm: queries defined in a SQL dialect, such as StreamSQL, process streams of data or events before they are placed in the database and results of this processing step are only selectively stored in the database. Many of the extensions to the relational operators and how to process for example joins on windows in continuous queries, have carried over to current products for stream processing and have been extended there for high volume applications [45].

2.2.3 Materialized Views

Materialized views can be seen as a particular application of active database principles. Views are typically subsets of a database that are defined in the database schema. They are computed on the fly from the stored base tables. As an optimization, views were materialized, i.e., stored, resulting in the need for maintaining them whenever the base data changed. Propagation of base-table updates to the materialized views was accomplished using the mechanisms developed for active relational databases [86].

2.2.4 Message-Oriented Middleware (MOM)

Message-oriented middleware (MOM) is a specific class of middleware that supports loosely coupled communication between distributed software components by means of asynchronous message-passing as opposed to a request/response metaphor. This allows a producer to send a message (event notification) and then continue working while the message is being delivered and processed. Optionally the message producer can be notified later when the message is completely processed. The decoupling of communicating parties has several important advantages:

- Message producers and consumers do not need to know about each other.
- They do not need to be active at the same time to exchange information.
- They are not blocked while sending or receiving messages [67].

Message-oriented Middleware contains notification services supporting two message models:

1. *Point-to-point (P2P)*[1] and
2. *Publish/subscribe (pub/sub)*

Both are depicted in Figures 2.2 and 2.3. P2P 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 is a one to many communication model. Each message is sent (*published*) by a message producer (*publisher*) to the notification service and it may be delivered to multiple consumers (*subscriber*) interested in the notification. Consumers are required to register (*subscribe*) for notifications they are interested in before being able to receive messages. Publish/subscribe systems come in many different flavours, both centralized and distributed. A common distinction is based on the information carried by the notification, and whether the content of the notification is used for routing or only the information on the envelope of a message [67]:

Channel-based: Producers place their notifications into a channel and consumers subscribe to a channel of interest (see Figure 2.4(a)).

Subject-based: Pioneered in the 1990's by TIBCO [49], defines subject hierarchies according to which messages are classified. A combination of subject hierarchy levels with the use of wild cards allows for reasonably powerful subscriptions. One disadvantage, though, is the relative inflexibility of subject hierarchies (see Figure 2.4(b)).

Topic-based: A variant of channel based publish/subscribe with additional predicates definable on the envelope data used by JMS [221] (see Figure 2.4(c)).

Content-based: The content of a message is used to route the message from producer to subscriber [194]. Filters are placed as close as possible to the source to minimize traffic. Predicates of different degrees of expressiveness can be specified. However, the more powerful the predicate language and the more fine grained the filters are, the more critical it becomes to control the size of the routing tables.

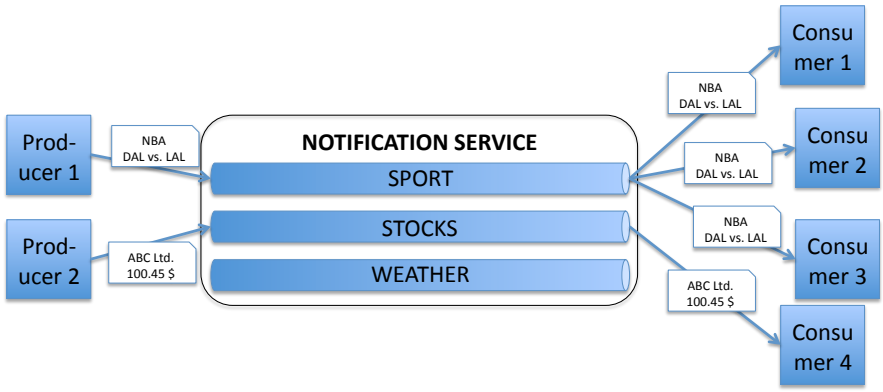
Type-based: Type-based is comparable to subject-based. As illustrated in Figure 2.5(a), consumers subscribe for a certain event type of event publications whereby object inheritance is considered [155].

Type- & attribute-based: Type-based is extended by content-based filtering on event attributes [66, 155]. Type-& attribute-based pub/sub is illustrated in Figure 2.5(b))

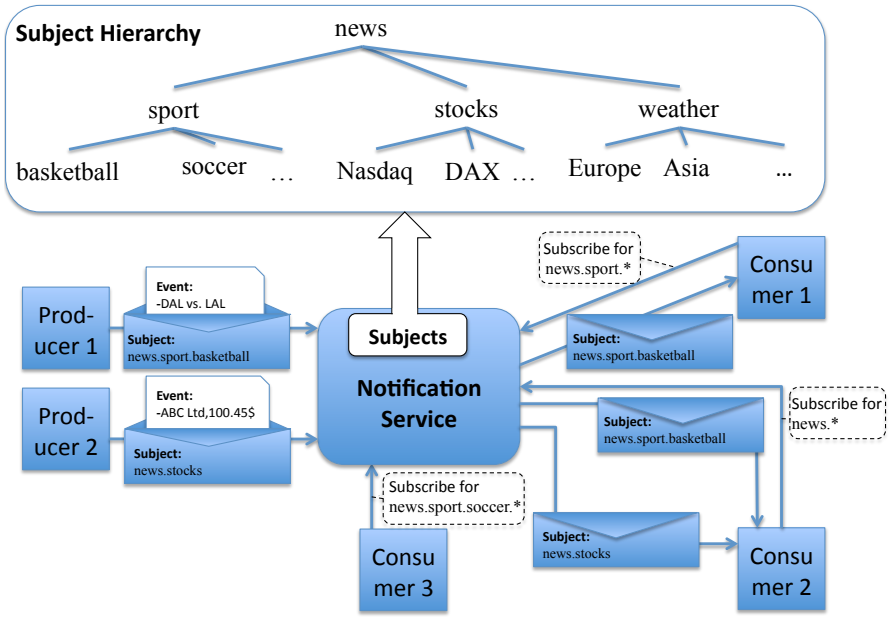
XML-based: A variant of content-based where notifications are XML messages with a flexible document structures [224]. Subscriptions should be expressed by a powerful language based on XPath [235] or XQuery [234].

Concept-based: Originally addresses the problem of heterogeneity [55]. All the previous approaches to publish/subscribe assume a common understanding of the name space used. If this is not the case, an additional layer of mediation that resolves semantic conflicts based on an ontology service can be used. Concept-based publish/subscribe uses predefined contexts. If notifications are to be routed within a common context, i.e., publisher and subscriber use the same context, no additional mediation is needed. If publisher and subscriber use different contexts, an additional mediation step is needed. Concept-based publish/subscribe can be implemented on top of any of the other publish/subscribe methods. This in turn requires merging of filters. An example for concept-based pub/sub is shown in Figure 2.5(c).

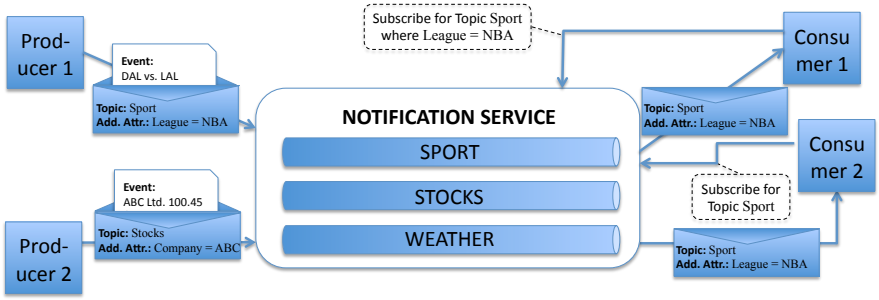
[1]The usage of P2P in this thesis is distinct from the acronym referring to peer-to-peer systems.



(a) Channel-based

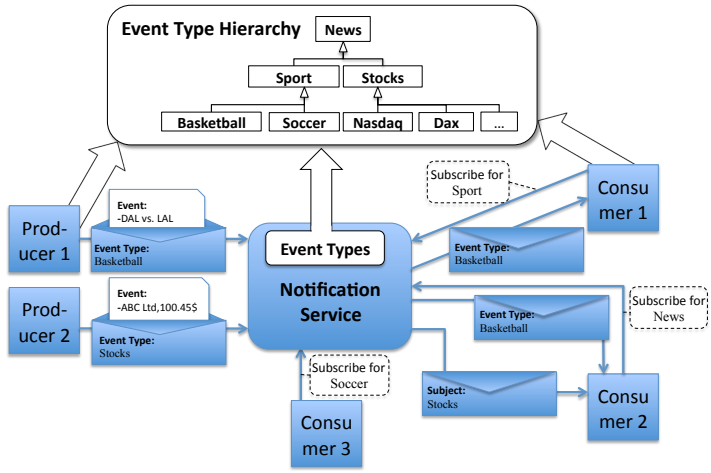


(b) Subject-based

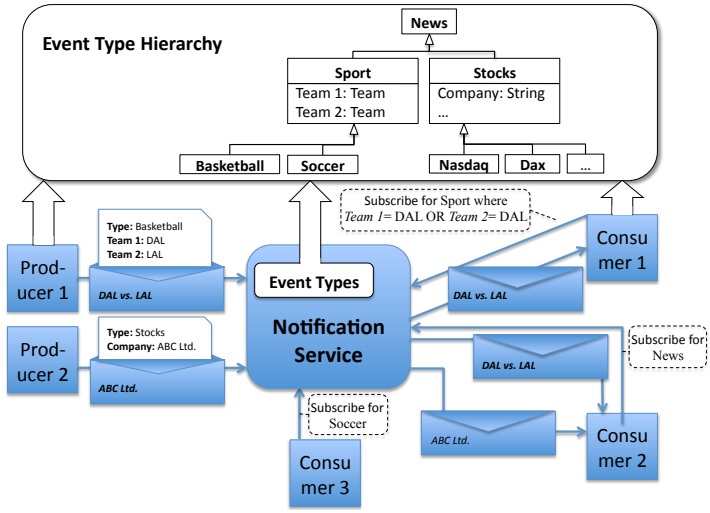


(c) Topic-based

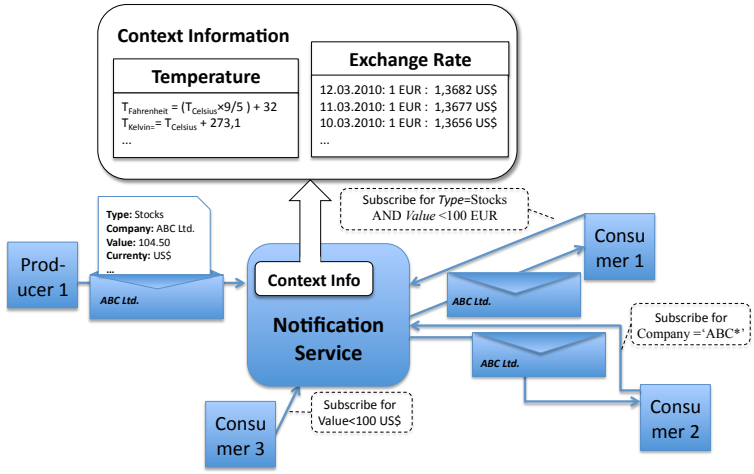
Figure 2.4: Subscription Models



(a) Type-based



(b) Type- & attribute-based



(c) Concept-based

Figure 2.5: Subscription Models (cont.)

Organization: ***Standard***	**Description**
Sun: *Java Message Service (JMS)*	The quasi-standard is Java Message Service (JMS) [221], a standard Java API, which is widely being adopted by almost all MOM products.
OMG: *CORBA Event Service*	Specifies how Event Supplier und Event Consumer communicate via an Event Channel using asynchronous message exchange [163]. Push and pull communication models are defined.
OMG: *CORBA Notification Service*	The Notification Service is an extended version of the Event Service [164], e.g., in the area of event filtering. A Notification/JMS Interworking Service allows to manage Notification Service interworking with Java Message Service [165].
OMG: *Data Distribution Service for Real-time Systems (DDS)*	Provides an API specification as well as a wire level protocol for publish-subscribe middleware[166].
Apache: *OpenWire*	An open binary wire level protocol [7]. To the best of our knowledge OpenWire is only supported by ActiveMQ.
Apache: *Streaming Text Orientated Messaging Protocol (Stomp)*	A very simple text based-protocol formerly known as TTMP with limited features [216]. Native support for Stomp is implemented by ActiveMQ and announced for the next release of HornetQ. With StompConnect [217] all Stomp clients can communicate with JMS providers. Since not all features on JMS are supported by Stomp it is not possible to use a JMS to connect to a Stomp provider.
Digistan & RestMS: *RestMS (RESTful Messaging Service)*	RestMS works over plain HTTP/HTTPS [98]. Three implementations exist: Ahkera[74], Zyre (part of OpenAMQ [99]), and a client stack written in Perl. Additionally an AMQP/0.9.1 profile for RestMS was implemented.
AMQP Working Group: *Advanced Message Queuing Protocol (AMQP)*	AMQP is a wire-level protocol specification with the goal to provide interoperability between MOM products.

Table 2.1: Standards for MOMs

There exist several important standards, which cover different aspects such as transport protocols or APIs. Some of these standards are popular in certain application domains, e.g., DDS is widely used in military applications and is part of the Navy Open Architecture. In Table 2.2 we provide a comprehensive overview of MOM products and which standards are supported by these products. The most important standard is JMS, which is supported by most products. Our work in the area of MOM focuses on the JMS standard and the knowledge of its features is required for an understanding of our work. Therefore, we discuss the Java Message Service in detail in this section. Further, we summarize the latest state in the development of the Advanced Message Queuing Protocol (AMQP), an emerging wire level protocol. Other standards are briefly listed in Table 2.1.

Java Message Service (JMS)

The Java Message Service (JMS) [221] is a standard Java-based interface for accessing the facilities of enterprise MOM servers and part of the Java EE standard [191]. In the terminology of JMS, a MOM server that supports the JMS API is referred to as *JMS provider* (or *JMS server*) and applications that use the JMS provider to exchange messages are referred to as *JMS clients*. JMS supports P2P as well as topic-based publish/subscribe communication models. JMS queues and topics are commonly referred to as *destinations*. The JMS specification defines several modes of message delivery with different QoS attributes:

Non-Persistent/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.

This provides low messaging overhead at the cost of losing undelivered messages in case of a server crash. 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. In case of a server crash, undelivered messages are recovered from stable storage on system restart. In non-persistent mode, each message is guaranteed to be delivered *at-most-once*, whereas in persistent mode it is guaranteed to be delivered *once-and-only-once*.

Non-Durable/Durable: JMS supports two types of subscriptions, durable and non-durable. Non-durable subscriptions last for the lifetime of their subscriber, i.e., a subscriber will only receive messages that are published while it is active (connected). Messages published while the subscriber is inactive, will be missed by the latter. In contrast to this, durable subscriptions ensure that a subscriber does not miss any messages during periods of inactivity.

Non-Transactional/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. JMS supports two types of transactions: local and distributed. Local transactions are limited to messaging operations executed on a JMS server. Distributed transactions allow other transactional operations such as database updates to be executed with JMS messaging operations as part of a single atomic transaction.

In addition to the above described delivery modes, JMS allows the specification of *selectors* to enable message filtering. When publishing messages, producers can specify property-value pairs (e.g., *"color=red"*) which are stored in the message headers. When subscribing, consumers can specify a selector to receive only messages with certain property values (e.g., *"color=blue AND size=42"*). Selectors are specified using a subset of the SQL92 conditional expression syntax. For a more detailed introduction to MOM and JMS the reader is referred to [103, 221].

Advanced Message Queuing Protocol (AMQP)

Advanced Message Queuing Protocol (AMQP) is an increasingly important protocol for MOMs with its origin in the financial services industry. The motivation behind AMQP is the need for an open standard which enables complete interoperability between MOM providers [230, 133]. AMQP provides a wire-level protocol specification and not an API as JMS. Due to the popularity of JMS, it was decided to design AMQP to encompass JMS semantics [167, 4]. This allows building JMS clients for AMQP products. Therefore, JMS and AMQP complement each other by defining interoperability on the application level (JMS) as well as on the wire level (AMQP). To achieve this interoperability, AMQP specifies the exact semantic of services in its queueing model; the specification covers messaging models (P2P, pub/sub, request/response), transaction management, distribution, security and clustering [5]. AMQP offers several features which are not supported by JMS.

Even if the AMQP specification is not finalized yet, several products [3] supporting different drafts of AMQP exist today (see Table 2.2). They are already used in mission critical deployments, e.g., JPMorgan reported an AMQP environment supporting 2,000 users on five continents processing 300 million messages per day [167]. There were some discussions in the AMQP community because Red Hat submitted a patent application closely related to AMQP [192]. Since the main target was to establish an open standard, this patent application was criticized as being counterproductive [232].

2.2.5 Distributed Event-Based Systems (DEBS)

A generic distributed event-based system (DEBS) is normally composed of nodes deployed in a distributed environment and exchanging information through a set of communication networks

Name	Vendor	JMS	AMQP	DDS	STOMP	OS	Ref	Note
ActiveMQ	Apache	✓	P [6]	-	✓	✓	[9]	
AMQP Infrastructure	Fedora	✓	✓	-	-	✓	[69]	Based on Qpid
CoreDX DDS Middleware	Twin Oaks Computing	-	-	✓	-	-	[229]	
FUSE Message Broker	Progress Software	✓	P	-	✓	✓	[56]	Based on ActiveMQ
FioranoMQ	Fiorano	✓	-	-	-	✓	[70]	
HornetQ	JBoss / Red Hat	✓	P [73]	-	✓[1]	✓	[117]	Prev. known as JBoss Msg.
InterCOM DDS	Gallium	-	-	✓	-	-	[75]	
JORAM	OW2 Consortium	✓	✓	-	✓[1]	✓	[172]	
Message Queuing	Microsoft	✓[2]	-	-	-	-	[152]	
MilSOFT DDS	MilSOFT	-	-	✓	-	-	[153]	
Mule MQ	Mulesoft	✓	-	-	✓[1]	-	[157]	Based on Nirvana Ent. Msg.
Nirvana Enterprise Messaging	my-Channels	✓	-	-	✓[1]	-	[160]	
OpenAMQ	iMatix Corporation	✓	✓	-	-	✓	[110]	
OpenDDS	OCI	✓	-	✓	✓[1]	✓	[162]	
Open Message Queue	Oracle	✓	-	-	✓[1]	✓	[168]	
OpenSplice DDS	PrismTech	✓	-	✓	✓[1]	(✓)	[181]	Not all layers are OS
Oracle 11g Streams AQ	Oracle	✓	-	-	✓[1]	-	[169]	Integrated in Oracle DB
Qpid	Apache	✓	✓	-	✓[1]	✓	[10]	
RabbitMQ	Rabbit Technologies	✓	✓	-	✓[1]	✓	[184]	JMS via Qpid JMS Client
Red Hat Enterprise MRG	Red Hat	✓	✓	-	✓[1]	-	[188]	Based on Qpid
RTI Data Distribution Service	Real-Time Innovations	✓	-	✓	✓[1]	-	[185]	
RTI Message Service	Real-Time Innovations	✓	-	✓	✓[1]	-	[186]	DDS supported via RTI DDS
SAP NetWeaver WebAS	SAP AG	✓	-	-	✓[1]	-	[204]	
SonicMQ	Progress Software	✓	-	-	✓[1]	-	[183]	
Sun GlassFish Message Queue	Oracle	✓	-	-	✓[1]	-	[170]	Based on OpenMQ
SwiftMQ	IIT Software	✓	-	-	✓[1]	-	[108]	
TIBCO Ent. Message Service	TIBCO Software	✓	-	-	✓[1]	-	[225]	
TIBCO Rendezvous	TIBCO Software	✓[2]	-		-	-	[226]	
WebLogic	Oracle	✓	-	-	✓[1]	-	[171]	
webMethods Broker Server	Software AG	✓	-	-	✓[1]	-	[209]	
WebSphere Application Server	IBM	✓	-	-	✓[1]	-	[105]	
WebSphere MQ	IBM	✓	-	-	✓[1]	-	[105]	Formerly known as MQSeries
ZeroMQ	iMatix Corporation	-	-[3]	-	-	✓	[111]	Acquired by iMatix [109]

[1] Via JMS adapter. [2] Via third-party software. [3] Support was dropped *'to protect ZeroMQ users from infringement on AMQP-related patents'* [112].

Table 2.2: Message-Oriented Middleware (OS = Open Source, P = Planned), State: March 2010

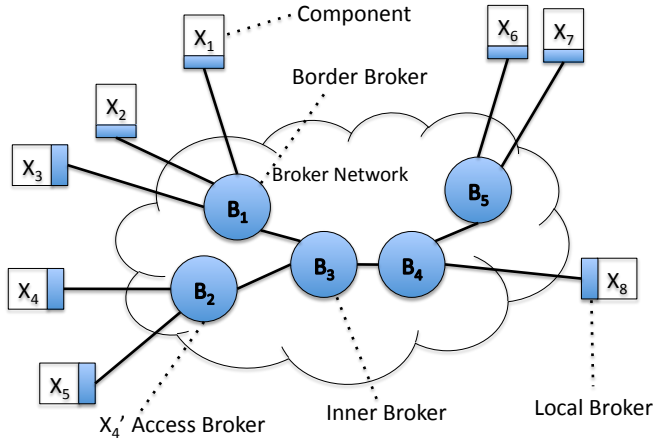


Figure 2.6: Router Network of REBECA [155]

(see Figure 2.6). Clients of the system are either publishers or subscribers depending on whether they act as producers or consumers of information. Publishers publish information in the form of *events* which are commonly structured as a set of attribute-value pairs. Subscribers express their interest in specific events through *subscriptions*. Most generally, subscriptions are defined as a set of constraints on the content of events. The constraints are specified using a subscription language. A published event is said to *match* a subscription if it satisfies all constraints of the subscription on the event attributes. The main task of the system is to deliver published events to all subscribers that have issued matching subscriptions.

Depending on the subscription model, DEBS can be classified, e.g., as topic-based or content-based. As discussed in Section 2.2.4 the various subscription models have different expressive power. Highly expressive models enable subscribers to precisely specify the events they are interested in. However, the more expressiveness, the higher is the system's overhead for matching events. The typical architecture of DEBS can be decomposed into four logical layers: network layer, overlay layer, event routing layer and event matching layer. A detailed overview of these layers as well as the techniques used for implementation can be found in [19]. Many prototypes of DEBS exist, such as CEA (Cambridge Event Architecture) [14, 16], Cobra [193], Echo [65], Elvin [205], GREEN [207], Hermes [179], IBM Gryphon [218, 107], IndiQoS [39], JEDI (*Java Event-Based Distributed Infrastructure*) [60], Le Subscribe [68], Narada Brokering [173], ONYX [63], PADRES [113, 114], REBECA [154], READY [81, 82], REDS [61], SCRIBE [195, 43], SIENA (*Scalable internet event notification architecture*) [40], ToPSS [139], WebFilter [175] and XMessages [208]. An overview and discussion of different DEBSs and their features are provided, e.g., in [180, 155, 142, 113].

2.2.6 Reactive Middleware

Reactive middleware can be traced back to the CORBA platform and the event service defined therein [89]. Modern versions of basic reactive capability can be found in the form of the J2EE message driven beans, which consume event notifications and allow the asynchronous processing of messages in the J2EE platform [222].

Reactive middleware benefited to some extent from work on active databases and the attempts to unbundle active functionality from active databases. Major insights gained while unbundling were the need for interval semantics instead of point semantics for many distributed environments, the impossibility of using a central clock and the fact that notification delays cause uncertainty. This resulted in the 2g–precedence model used in networks with bounded

delay and the imprecision interval model that distinguishes between the stable past, the unstable past and the present for networks without an upper bound on delay [138].

2.3 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 [20]. QPNs allow the modeling of process synchronization and the integration of hardware and software aspects of system behavior [24, 125] and provide greater modeling power and expressiveness than conventional queueing network models and stochastic Petri nets [24]. QPNs were applied successfully in several case studies to model system behavior, e.g., [125, 124, 125, 123, 132]. First, we present the formal definition of QPNs. This section is based on [26, 123, 125, 125]. Afterwards we discuss the existing tool support for QPNs.

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 [26, 21]. *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 [26, 32, 125]:

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., [77, 121]. One of these extensions are *colored PNs (CPNs)* which were introduced by K. Jensen [120, 119] 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 [26]:

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* $M_0(p) \in C(p)_{MS}$

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

Other extensions of ordinary PNs allow timing aspects to be integrated into the net description [26, 32]. 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 [26, 32]:

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,* $T_1 \cap T_2 = \emptyset$*,* $T_1 \cup T_2 = T$*,*
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* $t_i \in T_1$ *or 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)* [26]:

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* $w_i \in [C(t_i) \mapsto \mathbb{R}^+]$ *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* $t_i \in T_1$ or *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 2.7). 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.

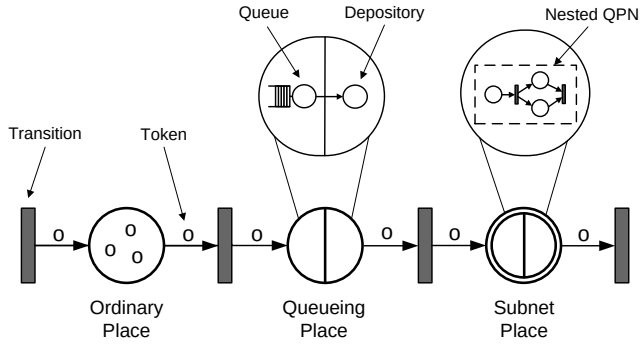


Figure 2.7: QPN Notation

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

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 or 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* $t_i \in \tilde{W_1}$ *or a firing weight specifying the relative firing frequency due to color c, if* $t_i \in \tilde{W_2}$.

Example 1 (QPN [26]) *Figure 2.8 shows an example of a QPN model of a central server system with memory constraints based on [26]. 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 2.8,*
- $Q = (\tilde{Q_1}, \tilde{Q_2}, (null, G/C/\infty/IS, G/C/1/PS, null, G/C/1/FCFS, G/C/1/FCFS))$,
  $\tilde{Q_1} = \{p_2, p_3, p_5, p_6\}, \tilde{Q_2} = \emptyset$,
- $W = (\tilde{W_1}, \tilde{W_2}, (w_1, ..., w_{|T|}))$, *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.*

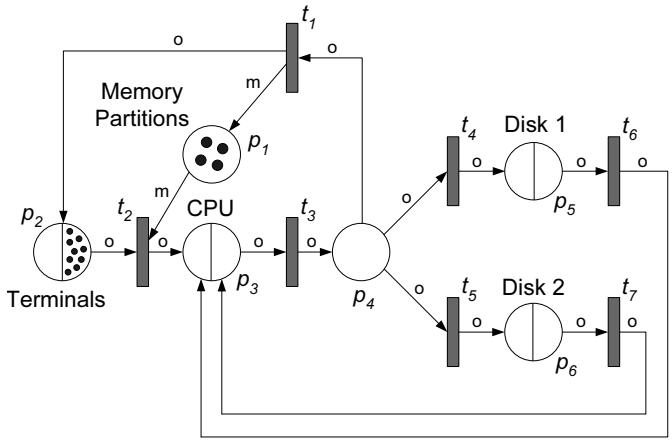


Figure 2.8: A QPN Model of a Central Server with Memory Constraints (reprinted from [26]).

Solving of QPNs & Tools for QPNs

For QPNs, the analytic solving approach is well-defined [26] and implemented by several tools, e.g. [25, 23]. 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 [123]. Therefore, we decided to use a simulation-based QPN solver for our models. Such a simulation-based approach was presented in [123] which is implemented by the *QPME tool (Queueing Petri net Modeling Environment)* [127, 129, 128]. 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 [127, 123] 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.

2.4 Concluding Remarks

In this chapter we provided the background needed for the understanding of this thesis. We started with the concept of events and event-based systems in general. As part of this discussion, we reviewed related concepts such as publish/subscribe communication in detail and described underlying technologies such as Active Databases and their properties. Focusing on MOMs and DEBS, established standards were specified in detail. Additionally, we composed a comprehensive list of existing middleware products and supported standards. At the end of this chapter, we introduced the queueing Petri net paradigm and gave an overview of existing tools and solvers. QPNs provide several benefits over conventional modeling paradigmas by combining the ideas of QNs and SPNs. As a result, system aspects such as blocking, software and hardware contention and synchronization can be easily reflected in a QPN model.

Chapter 3

Related Work

In this chapter, we provide an overview of the current state in performance modeling and benchmarking of event-based systems. We will focus on distributed event-based systems as well as MOMs. A comprehensive overview can also be found in [131].

3.1 Performance Modeling of Event-Based Systems

In this section we present an overview of existing performance modeling techniques for event-driven systems considering both centralized and distributed environments.

Modeling of MOM

A method for modeling MOM systems using *performance completions* is presented in [87]. Performance completions provide a general mechanism for including low-level details of execution environments into abstract performance models. 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. In [141], an approach for predicting the performance of messaging applications based on the Java Enterprise Edition (JavaEE) is proposed. The forecast 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 associated with these components, 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. In [88], the dependency between the MOMs usage and its performance was analyzed using statistical inference. For the validation of the approach, parts of SPECjms2007 and jms2009-PS were used.

Modeling of Distributed Publish/Subscribe Systems

Several performance modeling techniques specifically targeted at distributed publish/subscribe systems [40] 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 [115], 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. For example, the broker topology is assumed to be a complete n-ary tree and publishers are only allowed to be connected to leaf brokers. Furthermore, subscriptions are assumed to be equally distributed among filter classes and brokers. Finally, the considered metrics are limited to routing table sizes and the message bandwidth which do not directly characterize the system performance. Many of these assumptions were relaxed in [156] where a generalization of the model was proposed, however, the generalized model is still limited to systems based on hierarchical identity-based routing. In [42], an analytical model of pub/sub systems based on subscription forwarding is presented. The authors provide closed form analytical expressions for the overall network traffic required to disseminate subscriptions and propagate notifications, as well as for the message forwarding load on individual system nodes. However, the same restrictive assumptions as in [115] are made about system topology and the distribution of publishers and subscribers among brokers. Thus, the proposed model is not applicable in most practical scenarios. Finally, in [90], 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.

In [17, 231] a computational model of a publish/subscribe notification service is proposed, where the latter is abstracted as a black box connecting all participants in the computation. Based on the computational model, a probabilistic model for measuring the effectiveness of the notification service in delivering publications to the set of interested subscribers is developed. The effectiveness of the notification service is studied as a function of the subscription and diffusion delays. While some interesting results are presented, the proposed model is too coarsely grained and it is based on the assumption that the subscription and diffusion delays are known which is not realistic to expect. In [18], the authors present an attempt to formally model a publish/subscribe communication system as a classical distributed computation. The authors formalize the concept of information availability and model a few properties of the computation, namely completeness and minimality, that capture the expected behavior of a publish/subscribe system from an application viewpoint. The protocol-level requirements for managing availability and providing basic QoS properties under very simplified conditions are discussed. In [115], a stochastic analysis of self-stabilizing routing algorithms for publish/subscribe systems is presented. The analysis is based on continuous time birth-death Markov Chains and investigates the characteristics of systems in equilibrium. 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 is rather limited in terms of generality. In [34], Bricconi et al. present a simple model of the Jedi publish/subscribe system. The model is mainly used to calculate the number of notifications received by each broker for uniformly distributed subscriptions. To model the multicast communication, the authors introduce a spreading coefficient between 0 and 1 which models the probability that a broker at a given distance (in hops) from the publishing broker receives a published notification.

A general overview of relevant QoS metrics in the context of distributed and decentralized publish/subscribe systems can be found in [27]. In [13], it is advocated that QoS attributes in publish/subscribe systems should be managed in a uniform way with regard to other event attributes such as type or content. The authors propose a model for QoS-aware publications and subscriptions in which QoS-related properties are decoupled from event type and content. In a following paper [39], the authors present an architecture of a distributed QoS-aware pub-

lish/subscribe broker. The broker, called IndiQoS, leverages existing network-level QoS reservation mechanisms to automatically select QoS-capable paths. The approach, however, concentrates on QoS at the network level and does not consider contention for processing resources at the broker level. In [57] an overview of the QoS aspects of publish/subscribe middleware is given. Two industrial standards for publish/subscribe middleware, the Java Message Service (JMS) [221] and the Data Distribution Service (DDS) [166] are described and their QoS-related features are discussed.

Modeling of ECA Rule Engines

In [84], a new approach named *event paths* for modeling ECA rule engines was introduced. It simplifies the modeling of the different components of a rule engine. The idea of the model is to consider all possible paths that events may cause. A path is defined as the sequence of ECA components an event goes through, possibly of different rules. The simplest paths to be identified are those initiated by events (whether they are simple or composite) that directly trigger a single rule and then exit the system. These paths then must be distinguished if, depending on the event values, the execution may conclude at the condition component or it may proceed until the action service. Moreover, the path must be split if it involves condition or action statements that differ in their service time under certain situations (first-time invocations, warm-up, caching, etc.). Finally, if a statement may generate another event which in turn triggers another rule, an extra path is included with the additional services. This avoids having loops in the model (i.e., services are never visited twice). In a case study, the performance of an ECA rule engine [83, 33] was evaluated on an embedded device using the *Performance Evaluation Tool Set*.

3.2 Benchmarking of Event-Based Systems

In this section, we review related work in the area of benchmark development and benchmarking of event-based systems in particular.

Benchmark Development

Benchmark development has turned into a complex team effort. While historical benchmarks were only some hundreds lines long, modern benchmarks are composed of hundert thoundsands or millions of lines of code. Compared to traditional software, the development process has different goals and challenges. Unfortunately, even if an enormous number of benchmarks exist, only a few contributions focusing on the benchmark development process were published.

The best known publication is Gray's *The Benchmark Handbook* [80]. Besides a detailed description of several benchmarks, the author discusses the need for domain specific benchmarks and defined four important criteria, which a domain-specific benchmark has to fulfill:

- *Relevance:* the benchmark result has to measure the performance of the typical operation within the problem domain.
- *Portability:* it should be easy to implement on many different systems and architectures.
- *Scalability:* it should be scaleable to cover small and large systems.
- *Simplicity:* the benchmark should be understandable to avoid lack of credibility.

Another newer work dealing with the question, which criteria a benchmark should fulfill is [104]. The questions, what a 'good' benchmark should look like and which aspects should be kept in mind from the beginning of the development process, are discussed in detail and five key criteria are presented:

- *Relevance:* the benchmark has to reflect something important.
- *Repeatable:* the benchmark result can be reproduced by rerunning the benchmark under similar conditions with the same result.
- *Fair & Portable:* All systems compared can participate equally (e.g., portability, 'fair' design).
- *Verifiable:* There has to be confidence that documented results are real. This can, e.g., be achieved by reviewing results by external auditors.
- *Economical:* The cost of running the benchmark should be affordable.

The author believes that it is impossible to be perfect in all criteria and good benchmarks have clear strengths in one or two areas, and accommodate the others.

Standardization organizations such as the SPEC (Standard Performance Evaluation Corporation) or the TPC (Transaction Processing Performance Council) use internal guidelines covering the development process. A short summary of the keypoints of the SPEC Benchmark Development Process is provided in [136]. However, these guidelines mostly cover formal requirements, e.g., design of run rules and result submission guidelines, not the benchmark development process itself.

Benchmarking of MOM & Publish/Subscribe:

A number of proprietary and open-source benchmarks for MOM supporting JMS have been developed and used in the industry, for example, the SonicMQ Test Harness [213], IBM's Performance Harness for Java Message Service [106], Apache's ActiveMQ JMeter Performance Test [8] and JBoss' Messaging Performance Framework [118]. Using these and other similar benchmarks, numerous comparative performance studies of competitive products have been conducted and published by MOM product vendors over the last years, see for example [58, 134, 212, 210, 38, 71, 190, 72, 97].

Even though these works mostly focus on pure JMS environments, for other MOM standards such as DDS, benchmarks and test harnesses exist as well. The DDS Benchmarking Environment (DBE) [236] is an open-source framework for automated performance testing of DDS environments. It comes with a repository storing scripts, configuration files, test data and provides several Perl scripts for test setup. Further, it contains a shared library for collecting results and calculating statistics. Another DDS benchmark is the DDS TouchStone Benchmark Suite [182]. It was originally published by PrismTech and is available as open source since version 4.1. It allows the measurement of roundtrip latencies and throughput numbers and supports the creation of workload scenarios. Further, a test harness named Middleware Evaluation Test Tool (METT) was used in [146] to compare the performance of two different products and some simple test environments for DDS-based systems were used in different performance studies, e.g., [189, 187]. A first benchmark for AMQP was presented in [219] including several simple test cases. A primitive benchmark was used in [36] to measure the performance of two implementations.

Some general guidelines for designing a benchmark suite for evaluating distributed publish/subscribe systems are presented in [41], however, no specific implementation or measurement results are provided.

As evident from the above, numerous approaches to benchmark MOM performance have been developed and used in industry and academia. However, almost all of them are based on artificial workloads that do not reflect real-world application scenarios. Furthermore, they typically concentrate on exercising individual MOM features in isolation and do not stress the server in a manner representative of real-life applications. In most cases, performance studies conducted using these workloads have been biased in favor of particular products leading to contradictory claims made by MOM vendors [116, 211, 210, 58, 134, 220].

Benchmarking of CEPs

The Java-based framework FINCoS [29] is focusing on performance aspects of complex event processing systems [148]. FINCoS was developed as part of the BiCEP project [31] and provides a set of benchmarking tools for load generation and performance measuring of event processing systems. It follows a flexible and neutral approach, which allows to attach load generators, datasets, queries, and adapters for diverse CEP systems. In [149], a case study evaluating three different CEPs using the FINCoS framework is presented. In this case study several micro-benchmarks, e.g., for aggregation and window policies, were defined.

Benchmarking of Active Databases

Several research benchmarks were published for active databases. The first benchmark for object-oriented active databases was the BEAST Benchmark [79, 44]. Using the database and schema of the OO7 Benchmark [37], the BEAST Benchmark runs a series of tests targeting different aspects of an active database such as event detection and rule management. As performance metric, system response time is reported. The workload can be scaled by modifying the number of defined events (simple and composite) and the number of rules. In [78], performance results using the BEAST Benchmark for four active database systems were presented. Another benchmark for active databases is the ACT-1 benchmark. The ACT-1 benchmark concentrates on the minimal features of an active database [237]. Using a simple underlying database, ACT-1 models the operation of a power plant to address four basic issues: overhead of method wrapping, rule firing cost, event consumption costs and serialization of rules. To overcome the limitations of the ACT-1 Benchmark and the BEAST Benchmark the OBJECTIVE Benchmark was developed. Design goals were to provide a comprehensive set of operations which stress all critical functions using (compared to the schema of the OO7 Benchmark used in the BEAST) a simple database, and consider both hot and cold execution times [44]. An extended version of the OBJECTIVE Benchmark targeting additional features of component-based active rule systems is presented in [122].

Performance Studies and Analyses of Message-oriented Middleware

In [228], an evaluation of IBM's MQSeries V5.2 platform is presented. The authors study the performance of four different styles of messaging: non-persistent non-transactional, persistent non-transactional, persistent local transactional and persistent global transactional. The server's *maximum sustainable throughput* is introduced as a metric for characterizing the server performance. The results show the impact of various factors including the message length, the server log buffer space and the number of receiver threads. In [227], the authors evaluate three leading JMS providers, IBM WebSphere MQ/MQIntegrator, TIBCO Rendezvous/MessageBroker V4.0 and Mercator Integration Manager V6.0. A synthetic transactional workload is used and the maximum sustainable throughput for persistent and non-persistent messages is measured. Similarly, in [53] an empirical methodology for evaluating the QoS of JMS products is presented. In addition to the maximum sustainable throughput, several further evaluation criteria are considered, such as the message delivery latency, the elapsed time for batch messaging and the effectiveness of persistent message recovery after a server crash. Two leading JMS servers are evaluated. Unfortunately, the study only considers point-to-point messaging and the authors do not disclose the names of the tested products.

Another performance study comparing TIBCO Rendezvous (TIB/RV) with SonicMQ was published in [145]. This study considers both point-to-point and publish/subscribe messaging. For point-to-point messaging, the effects of increasing the number of sender and receiver pairs is analyzed. For publish/subscribe messaging, the effect of increasing the number of publishers and subscribers is analyzed. Furthermore, the authors consider the duration of the time taken for a batch of messages to be delivered, the connection time for new subscribers, as well as the

server memory and CPU utilization. Some general guidelines for designing a benchmark suite for distributed publish/subscribe systems are presented in [41], however, no specific implementation or measurement results are provided. In [64], the performance of the individual elements used in message broker applications is evaluated highlighting the cost of using each element rather than the cost of running complete applications.

In [96], the capacity of the WebsphereMQ JMS server is evaluated in terms of its throughput performance. 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. An analytical model of the message processing time and the server throughput is presented and validated through measurements. Several similar studies using Sun Java System Message Queue, FioranoMQ, ActiveMQ and BEA WebLogic JMS server were published in [95], [92], [93] and [94], respectively. The study in [93] considers complex AND-, OR-. and IN-filters of different length. In [151], the results from the evaluation of the different products are compared and summarized. A more in-depth analysis of the message waiting time for the FioranoMQ JMS server is presented in [150]. The authors study the message waiting time based on an $M/G/1-\infty$ approximation and perform a sensitivity analysis with respect to the variability of the message replication grade. The analysis shows that the message waiting time is low as long as the server throughput is sufficiently high. The authors derive formulas for the first two moments of the message waiting time based on different distributions (deterministic, Bernoulli and binomial) of the replication grade. Finally, two simple distributed architectures based on conventional JMS servers that increase the JMS capacity beyond the capacity provided by a single server are proposed.

In [135], a simple test harness for testing of JMS providers for correctness and performance is presented. The authors develop a formal model for JMS behavior based on the I/O automata used in other group communication systems. The focus here is on verifying the correctness of JMS implementations and only basic support for performance analysis is provided. In [85], an efficient strategy for reliable messaging using different persistence methods with various kinds of messages is developed. The strategy utilizes daemon threads to reduce its influence on the system and has been implemented as part of a JMS server.

3.3 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 [76]. 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 [161]. The authors use a template to describe a number of sample patterns and suggest the introduction of a Petri net pattern repository. In [159] 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 [158]. These patterns have been modeled using CPN Tools. In [28] 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 [137]. RCPN support the definition of reusable components.

The authors of [176, 177, 178] 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.

3.4 Concluding Remarks

This chapter provided an overview of related work in the areas of performance modeling and benchmarking of EBS and discussed the usage of performance modeling patterns in previously published work. In our review of current research we considered all kinds of EBS including DEBS, MOMs and ECA rule engines. While several benchmarks for EBS exist, these benchmarks do not fulfill the requirements we defined. We identified a lack of test harnesses and benchmarks using representative workloads for EBS. The same applies to performance models. While performance and QoS issues in EBS have been discussed in several publications, no previous work exists that provides a general methodology for performance modeling with the goal of performance prediction.

In previous work, performance modeling pattern collections were introduced and approaches to use them for system modeling were proposed. However, none of these publications used QPNs for their patterns nor did they focus on EBS.

Chapter 4

Performance Engineering of Event-Based Systems

As already discussed in Section 2.1, event-based systems differ from traditional software in several aspects. Since they are used in business critical environments, there is a need for performance models [101] which allow the user to predict system behavior or analyze certain performance aspects and to identify possible bottlenecks.

The fact that event consumer and producer are completely decoupled from each other and communicate using asynchronous patterns, influences the modeling approach for the performance aspects of event-based systems. Further, dynamic changes and implementations of EBS in very large scale and distributed environments make it even harder to find the right model representation for a given scenario.

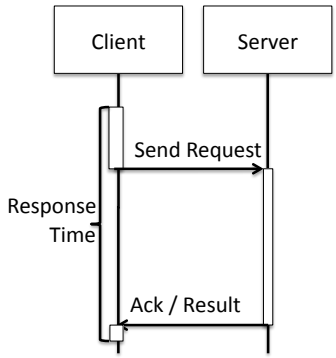
As a consequence, we have to investigate whether and how traditional performance modeling approaches can be applied on EBS. Our first step is to redefine the performance metrics to reflect particular properties of EBS.

For example, when a request is sent in a traditional request/reply-based distributed system, it is sent directly to a given destination which makes it easy to identify the system components and resources involved in its processing. The response time in such an environment can be defined as time needed by the client to trigger the request and by the server to reply to the request (see Figure 4.1(a)). The client is blocked while waiting for the answer of the server. This understanding of response time is not applicable to EBS systems. When an event is published in an EBS, it is not addressed to a particular destination, but rather routed along all paths that lead to subscribers with matching subscriptions. Since event producer and consumers are decoupled, the event producer does not wait for an acknowledgement by the event consumer. As illustrated in Figure 4.1(b), from the perspective of the event producer the response time does include the execution time of the event consumer nor the time needed to forward the event notifications to the n consumers by the transport layer. Therefore, it is questionable whether metrics such as response time are still appropriate in a decoupled and asynchronous environment and we see a need for new metrics.

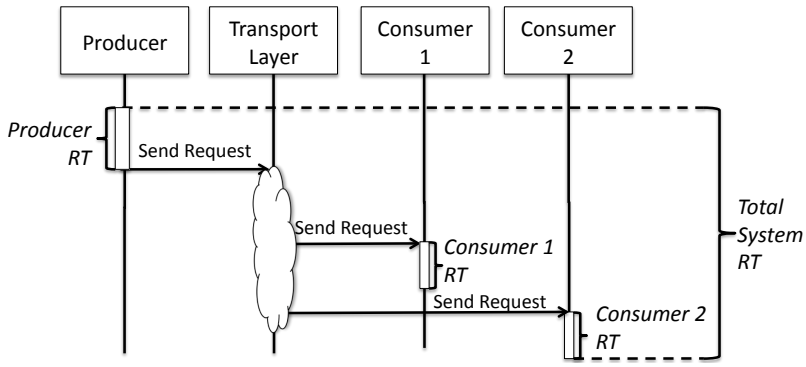
The high concurrency in EBS, their flexibility and the large number of events pose high requirements to the efficiency and features of modeling technologies. Challenges are, e.g., the correct representation of event forking, changing or durable subscriptions as well as mobile consumers and, especially in real world scenarios with thousands or millions of events, a reasonable solving time for the models is mandatory.

In this chapter, we present our contributions to the area of performance engineering of event-based system.

Formal Definition of EBS:



(a) Request - Reply



(b) Event-based System

Figure 4.1: Response Time (RT) in Traditional Request / Reply and in EBS

We provide a formal definition of EBS and their performance aspects which allows among other to characterize workload properties. Our definitions reflect a distributed event-based system, but can easily be applied to a non-distributed environment.

Modeling Methodology:
Based on these definitions, we present a modeling methodology for EBS. Our approach allows to predict system behavior including the utilization of different components and can be used in a wide range of applications, e.g., as decision base for self-adaptive systems or for capacity management.

Performance Modeling Pattern:
We discuss how different features of EBS can be reflected in performance models and introduce in this context performance modeling patterns for the most common settings and features of EBS. We use QPNs as modeling technique to illustrate the patterns.

Extensions of QPNs:
Several of our patterns are not, or only with high effort, convertible with standard QPNs. Therefore, we developed several new features for QPNs and conclude this chapter with a list of extensions for standard QPNs.

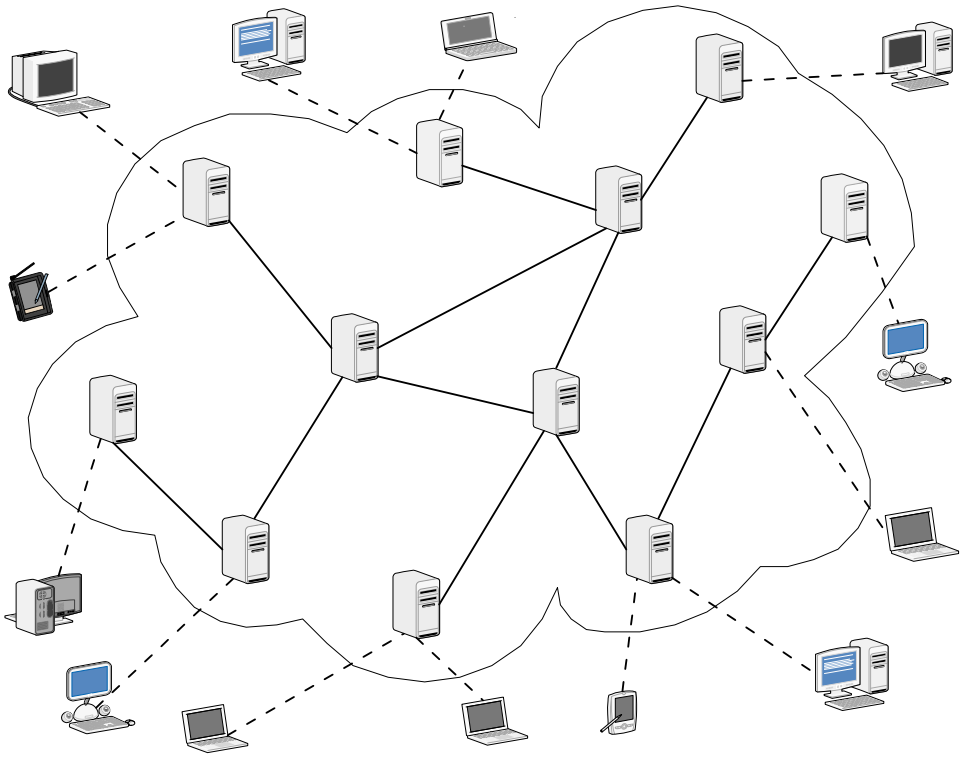


Figure 4.2: System Topology

4.1 Modeling Methodology for EBS

Modeling EBS is challenging because of the complete decoupling of communicating parties on the one hand, and, on the other hand, the dynamic changes in system structure and behavior. The event might have to be processed by multiple system nodes on its way to subscribers. It is hard to know in advance which system nodes will be involved in delivering the event. There are numerous event routing algorithms and they all have different implementation variants leading to different routing behavior. Moreover, depending on existing subscriptions, individual events published by a given publisher might be routed along completely different paths visiting different sets of system nodes. Therefore, it is hard to partition events into workload classes that have similar resource usage. Another difficulty stems from the fact that every time a new subscription is created or an existing one is modified, or when nodes join or leave the system, this might lead to significant changes in the workload. Thus, the dynamics of EBS necessitate that workload characterization be done more frequently in order to reflect the changes in the system configuration and workload.

4.1.1 Formal Definition

Let us consider a (distributed) event-based system implemented as a network of event brokers arranged in the topology depicted in Figure 4.2. Formally, the system can be represented as a 5-tuple $G = (N, C, P, S, E)$ where:

$\mathbf{N} = \{n_1, n_2, ..., n_{|N|}\}$ is the set of system nodes (in our example event brokers).

$\mathbf{C} = \{c_1, c_2, ..., c_{|C|}\}$ is the set of connections between nodes.

$\mathbf{P} = \{p_1, p_2, ..., p_{|P|}\}$ is the set of publishers.

$\mathbf{S} = \{s_1, s_2, ..., s_{|S|}\}$ is the set of subscribers.

$\mathbf{E} = \{e_1, e_2, ..., e_{|E|}\}$ is the set of event types.

We will use the following additional notation:

$H_P(r)$ is the id of the system node that publisher p_r is connected to.

$H_S(l)$ is the id of the system node that subscriber s_l is connected to.

$H_C^L(q)$ is the id of the system node on the "left side" of connection c_q. The left side is defined as the side of the node with lower id.

$H_C^R(q)$ is the id of the system node on the "right side" of connection c_q.

B_q is the bandwidth of the underlying network corresponding to connection c_q.

M_q^t is the size of the message that has to be transferred (taking protocol overhead into account) when an event of type e_t is sent over the network corresponding to connection c_q.

$\nu_{i,j}^{t,k}$ is the probability that an event of type e_t, published by publisher p_k, is forwarded to system node n_j after visiting system node n_i. If $i = j$, $\nu_{i,j}^{t,k} \stackrel{def}{=} 0$.

$\lambda^{t,k}$ is the rate at which events of type e_t are published by publisher p_k.

$\lambda_j^{t,k}$ is the rate at which events of type e_t, published by publisher p_k, arrive at node n_j.

λ_j^t is the total rate at which events of type e_t (published by any publisher) arrive at node n_j.

τ_q^t is the rate at which events of type e_t (published by any publisher) are sent over connection c_q.

$S_{t,j}^{CPU}$ is the mean CPU service time of an event of type e_t at node n_j.

$S_{t,j}^{I/O}$ is the mean disk I/O service time of an event of type e_t at node n_j.

$S_{t,q}^{NET}$ is the mean network service time when an event of type e_t is sent over the network link corresponding to connection c_q.

$\delta_{i,j}$ is the Kronecker function, i.e., $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ if $i \neq j$.

We can consider the events published in the system as basic components of the workload. Events can be partitioned into workload classes based on their type. However, events of the same type published by different publishers could have completely different routing behavior and resource consumption. Therefore, to make the workload classes more homogeneous in terms of resource consumption, we further partition them based on the publisher.

4.1.2 Analysis of the Event Routing Behavior

To determine the routing behavior of events in the system, we suggest conducting some experiments in a small testing environment. Brokers are configured according to the desired topology (Figure 4.2), however, instead of being deployed on separate servers distributed over a WAN, they are deployed on *a few* locally hosted servers connected to a LAN. Ideally, all brokers should be deployed on a single machine, however, this might be impossible due to technical reasons. Note that co-located brokers still use TCP/IP to exchange messages and event routing at the overlay and event routing layers is done in exactly the same way that it would be done in the

target environment. Subscriptions are set up according to the target workload and experiments are conducted to estimate the routing probabilities $\nu_{i,j}^{t,k}$ for $1 \leq i \leq |N|$, $1 \leq j \leq |N|$ and $i \neq j$. We assume that the system has been instrumented to monitor the event traffic and extract the routing probabilities. In each experiment, a subset of the overall set of publishers are emulated by generating events of the respective types. The event publication rates need not be equal to the target publication rates $\lambda^{t,k}$ and they should be chosen in such a way that the load injected does not exceed the capacity of the testing environment. If content-based routing is used, it must be ensured that events with content representative of the target workload are generated when conducting the experiments, i.e., the distributions of the event attributes must match their distributions in the target workload.

The following relationship between the routing probabilities $\nu_{i,j}^{t,k}$ and the arrival rates $\lambda_j^{t,k}$ holds for $j = 1, 2, ..., |N|$:

$$\lambda_j^{t,k} = \lambda^{t,k}\delta_{j,H_P(k)} + \sum_{i=1}^{|N|} \lambda_i^{t,k}\nu_{i,j}^{t,k} \tag{4.1}$$

Dividing both sides of Eq. (4.1) by $\lambda^{t,k}$ we obtain:

$$\frac{\lambda_j^{t,k}}{\lambda^{t,k}} = \delta_{j,H_P(k)} + \sum_{i=1}^{|N|} \frac{\lambda_i^{t,k}\nu_{i,j}^{t,k}}{\lambda^{t,k}} \tag{4.2}$$

The ratio $\frac{\lambda_j^{t,k}}{\lambda^{t,k}}$ is equal to the mean number of visits to node n_j of an event of type e_t published by publisher p_k. This ratio is known as *visit ratio* or *relative arrival rate* and we will denote it as $V_j^{t,k}$. Thus, from Eq. (4.2) the following relationship between the visit ratios and routing probabilities follows:

$$V_j^{t,k} = \delta_{j,H_P(k)} + \sum_{i=1}^{|N|} V_j^{t,k}\nu_{i,j}^{t,k} \tag{4.3}$$

Solving the above simultaneous equations enables us to derive the visit ratios based on the measured routing probabilities.

4.1.3 Estimation of Event Service Times

The next step is to determine the service times of events at the system resources. This includes all resources used by the system to deliver published events. The two types of resources that we have to consider are the CPUs of the system nodes and the networks used for communication between nodes. In addition, secondary storage devices at the system nodes (e.g., disk drives) might have to be considered if they are used by the event-based middleware, e.g., for reliable delivery.

There are different approaches to estimate the service times of events at the CPU of a system node. First, the system node can be instrumented to directly measure the CPU usage when processing events. Another approach which does not require instrumentation is to estimate the service times based on measured CPU utilization and event throughput. Assume that the considered system node is deployed on a separate dedicated machine running on a given reference hardware configuration if possible similar to the node's configuration in the target environment. By injecting events of the respective type and measuring the throughput X_j^t of events at the node and the machine CPU utilization U_j^{CPU}, we can derive the mean service time $S_{t,j}^{CPU} = U_j^{CPU}/X_j^t$.

This obvious relationship follows from the Utilization Law [62]. Once the CPU service times of events at the considered system node on the reference hardware configuration have been estimated, they can be used as reference values to extrapolate the service times to the node's configuration in the target environment. Benchmark results (e.g., SPECcpu2006 or SPECjms2007) for the respective hardware configurations can be exploited in such extrapolations. If the chosen reference configuration is similar to the target configuration, no extrapolation is necessary. If, in addition to the CPUs, further resources at the system node are used when delivering events (e.g., secondary storage devices), the mean service times at these resources can be estimated based on the measured resource utilization in exactly the same way as shown above for the CPUs. In certain cases, techniques can be employed that help to estimate the service times without the need to do any measurements on the system [147]. Such techniques are based on analyzing how the system components are implemented at the code level.

The mean network service time $S_{t,q}^{NET}$ can be calculated based on the available bandwidth and the size of the message that has to be transferred (taking protocol overhead into account) when an event of type e_t is sent over the network corresponding to connection c_q: $S_{t,q}^{NET} = M_q^t / B_q$.

4.1.4 System Operational Analysis

If we look at the CPUs of the nodes in our system as M/M/1 queues, we can use the following relationship from queueing theory to obtain an *approximation* for the mean response time $R_{t,j}^{CPU}$ of events of type e_t at the CPU of node n_j:

$$R_{t,j}^{CPU} = \frac{S_{t,j}^{CPU}}{1 - U_j^{CPU}} \tag{4.4}$$

From the Utilization Law it follows that:

$$U_j^{CPU} = \sum_{t=1}^{|E|} \lambda_j^t S_{t,j}^{CPU} = \sum_{t=1}^{|E|} \left(\sum_{k=1}^{|P|} \lambda_j^{t,k} \right) S_{t,j}^{CPU} \tag{4.5}$$

Similarly, it follows that the disk I/O utilization of node n_j can be computed as

$$U_j^{I/O} = \sum_{t=1}^{|E|} \left(\sum_{k=1}^{|P|} \lambda_j^{t,k} \right) S_{t,j}^{I/O} \tag{4.6}$$

and an *approximation* for the mean response time $R_{t,j}^{I/O}$ of events of type e_t at the disk I/O subsystem of node n_j is given by:

$$R_{t,j}^{I/O} = \frac{S_{t,j}^{I/O}}{1 - U_j^{I/O}} \tag{4.7}$$

Analogously, an *approximation* for the mean response time $R_{t,j}^{I/O}$ of events of type e_t at the disk I/O subsystem of node n_j can be obtained. The arrival rates $\lambda_j^{t,k}$ can be derived by solving the system of simultaneous equations (4.1). Using the same approach, we can estimate the utilization of the network links in the system and the network response times. The rate at which events of type e_t (published by any publisher) are sent over connection c_q can be computed as follows:

$$\tau_q^t = \sum_{k=1}^{|P|} \left(\lambda_l^{t,k} \nu_{l,r}^{t,k} \right) + \sum_{k=1}^{|P|} \left(\lambda_r^{t,k} \nu_{r,l}^{t,k} \right) \tag{4.8}$$

where $l = H_C^L(q)$ and $r = H_C^R(q)$. Assuming that connections use dedicated network links, the utilization U_q^{NET} of the network link corresponding to connection c_q is:

$$U_q^{NET} = \sum_{t=1}^{|E|} \tau_q^t S_{t,q}^{NET} \tag{4.9}$$

An *approximation* for the mean response time $R_{t,q}^{NET}$ of events of type e_t at connection c_q can be computed according to:

$$R_{t,q}^{NET} = \frac{S_{t,q}^{NET}}{1 - U_q^{NET}} \tag{4.10}$$

If multiple connections are sharing a network link, the utilization of the network link due to each of these connections must be taken into account when computing the mean response times at the connections. Assume for example that connections $c_{q_1}, c_{q_2}, ..., c_{q_m}$ all share a single physical network link. The relative utilization of the link due to connection c_{q_i} is given by:

$$U_{q_i}^{NET} = \sum_{t=1}^{|E|} \tau_{q_i}^t S_{t,q_i}^{NET} \tag{4.11}$$

The total utilization of the network link can be computed by summing up the relative utilizations due to the connections that share it:

$$U_{q_1,...,q_m}^{NET} = \sum_{i=1}^{m} U_{q_i}^{NET} \tag{4.12}$$

An *approximation* for the mean response time R_{t,q_i}^{NET} of events of type e_t at connection c_{q_i} can then be obtained by substituting $U_{q_1,...,q_m}^{NET}$ for U_q^{NET} in Eq. 4.10. Now that we have approximations for the mean response times of events at the system nodes and network links, we can use this information to derive an approximation for the mean event delivery latency. In order to do that we need to capture the paths that events follow on their way from publishers to subscribers.

Definition 6 (Delivery Path) *A delivery path of an event is every ordered sequence of nodes $(n_{i_1}, n_{i_2}, ..., n_{i_m})$ without repetitions that is followed by the event upon its delivery to a subscriber (the event is published at node n_{i_1} and delivered to a subscriber at node n_{i_m}).*

Event delivery paths can be determined by monitoring the system during the experiments conducted to measure the routing probabilities $\nu_{i,j}^{t,k}$ (Section 4.1.2). Every delivery path can be seen as a vector $\vec{w} = (n_{i_1}, n_{i_2}, ..., n_{i_m})$ whose elements are system nodes.

Definition 7 (Dissemination Tree) *The set W of all delivery paths of an event will be referred to as the dissemination tree of the event.*

Let $W^{t,k}$ be the union of the dissemination trees of all events of type e_t published by publisher p_k. By definition, $W^{t,k} = \emptyset$, if publisher p_k does not publish any events of type e_t. Let W^t be the union of the dissemination trees of all events of type e_t irrespective of the publisher, i.e., $W^t = \bigcup_{k=1}^{|P|} W^{t,k}$ is the set of all delivery paths of events of type e_t. Let $Q(i,j)$ for $1 \leq i < j \leq |N|$ be the id of the connection between nodes n_i and n_j assuming that such a connection exists.

Definition 8 (Mean Delivery Latency) *If $\tilde{W} \subseteq W^t$, the mean delivery latency $L_t(\tilde{W})$ of an event of type e_t over the set of delivery paths $\tilde{W}$ is defined as the average time it takes to deliver an event of type e_t over a randomly chosen path from $\tilde{W}$.*

If $\tilde{W}$ includes a single delivery path $\vec{w} = (n_{i_1}, n_{i_2}, ..., n_{i_m})$, an approximation for $L_t(\tilde{W})$ can be computed as follows:

$$L_t(\{\vec{w}\}) = \left(\sum_{r=1}^{m} R_{t,i_r}^{CPU} + \sum_{r=1}^{m} R_{t,i_r}^{I/O} \right) + \sum_{r=1}^{m-1} R_{t,Q(i_r,i_{r+1})}^{NET} \tag{4.13}$$

If $\tilde{W}$ includes multiple delivery paths $\{\vec{w}_1, \vec{w}_2, ..., \vec{w}_h\}$, we have:

$$L_t(\{\vec{w}_1, \vec{w}_2, ..., \vec{w}_h\}) = \frac{\sum_{k=1}^{h} L_t(\{\vec{w}_k\})}{h} \tag{4.14}$$

Definition 9 (Max Mean Delivery Latency) *If $\tilde{W} \subseteq W^t$, the max mean delivery latency $L_{t,MAX}(\tilde{W})$ of an event of type e_t over the set of delivery paths $\tilde{W}$ is defined as max delivery latency time it takes to deliver an event of type e_t over a randomly chosen path from $\tilde{W}$.*

If $\tilde{W}$ includes a single delivery path $\vec{w} = (n_{i_1}, n_{i_2}, ..., n_{i_m})$, than $L_{t,MAX}(\tilde{W}) = L_t(\tilde{W})$
If $\tilde{W}$ includes multiple delivery paths $\{\vec{w}_1, \vec{w}_2, ..., \vec{w}_h\}$, we have:

$$L_{t,MAX}(\{\vec{w}_1, \vec{w}_2, ..., \vec{w}_h\}) = \max(L_t(\{\vec{w}_1\}), L_t(\{\vec{w}_2\}), ...L_t(\{\vec{w}_n\})) \tag{4.15}$$

4.1.5 Performance Model Construction and Evaluation

If approximate results are not enough and accurate performance prediction is required, a more detailed performance model must be built. One possibility is to model the system using a queueing network, where system nodes and networks are represented as queues and events are represented as jobs served at the queues. Modeling the system in this way would result in a non-product form queueing network. This is because every time an event arrives at a system node, it might be forwarded to multiple other nodes, resulting in forking of multiple asynchronous tasks. Even though extended queueing networks make it possible to model the forking of asynchronous tasks, existing analysis techniques for this type of model (for example [91]) are rather restrictive and only provide approximate results.

An alternative approach is to model the system using a QPN, where system nodes and networks are represented as queueing places and events are represented as tokens. The forking of asynchronous tasks is much easier to model in this case. Whenever an event is forwarded to multiple system nodes, a transition can be used to create an instance of the event, i.e., an event token, at each of the queueing places corresponding to the target nodes. Modeling the system using QPNs provides a number of important benefits. QPN models provide excellent expressiveness and allow the integration of hardware and software aspects of system behavior into the same model [125]. This can be exploited to model the individual system nodes at a higher level of detail, capturing both hardware and software contention aspects. Furthermore, the knowledge of the structure and behavior of QPNs can be exploited for fast and efficient simulation [127]. This, on the one hand, ensures that models of realistically sized systems can be analyzed. On the other hand, it allows us to have service times with non-exponential distributions, thus improving the model's representativeness.

Figure 4.3 shows a QPN model of the system topology in Figure 4.2. In this model, we ignore the network, assuming that network delays are negligible. We will later show how the model can be extended to include contention for network resources. Each system node is modeled using a nested QPN (represented as a subnet place). The latter can be made as detailed as required to accurately capture the internal behavior of the node. Events are modeled using tokens and transitions are used to move events among nodes as they are routed in the system. Every system node has a single output transition. Event publications are modeled using timed transitions. We will use the following notation:

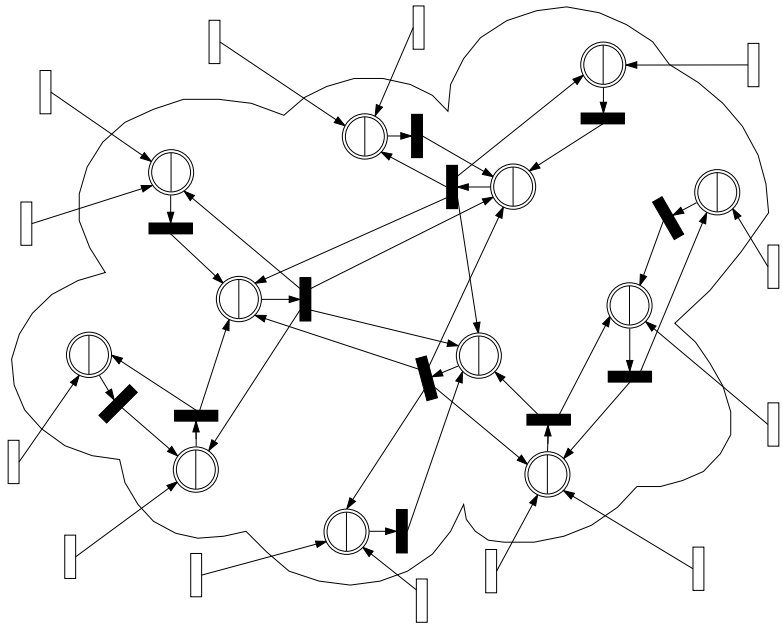


Figure 4.3: High-Level System Model.

π_i is the subnet place corresponding to node n_i.

φ_i is the output transition of place π_i.

ϕ_k is the timed transition used to model event publications by publisher p_k.

To distinguish between events with different resource consumption and routing behavior, a separate token color $x^{t,k}$ is defined for every combination of event type e_t and publisher p_k that publishes events of this type ($\lambda^{t,k} > 0$). The token color $x^{t,k}$ is defined for place π_i, if and only if events of type e_t, published by publisher p_k, visit the place, i.e., $\lambda_i^{t,k} > 0$.

Every timed transition ϕ_k has a separate firing mode η_k^t for each event type e_t published by the publisher it represents:

$$\eta_k^t : \emptyset \rightarrow \pi_{H_P(k)}\{1'x^{t,k}\} \tag{4.16}$$

This definition specifies that whenever the transition fires in mode η_k^t, no tokens are removed from any place and only one $x^{t,k}$ token is deposited in place $\pi_{H_P(k)}$. The firing delay $\rho(\eta_k^t)$ of this mode is set to the reciprocal of the rate at which events of type e_t are published by publisher p_k: $\rho(\eta_k^t) = 1/\lambda^{t,k}$.

Since the firing delays of timed transitions in QPNs are assumed to be exponentially distributed, the above approach to modeling event publications results in Poisson event arrivals. If, however, the distribution of the time between successive event publications is not exponential, a different approach can be used. Instead of a timed transition, a queueing place and an immediate transition are used to model event publications as shown in Figure 4.4. The queueing place has an integrated queue with an Infinite Server (IS) scheduling strategy. The queue service time distribution is equal to the distribution of the time between successive event publications. For every event type e_t published by the publisher, there is a single event token $x^{t,k}$ in the initial marking of the place. After an event token is served at the IS queue, the immediate transition fires, moving the token back to the queue and depositing a copy of it in the system node the publisher is connected to. The latter corresponds to an event publication.

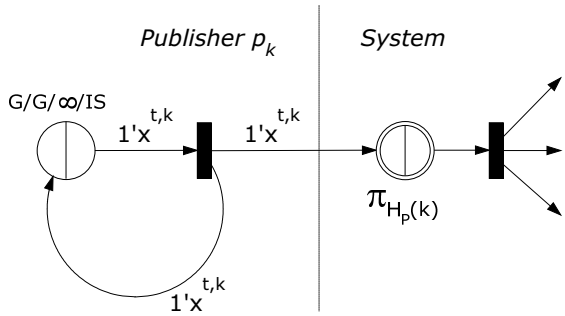


Figure 4.4: Modeling Non-Poisson Event Publications.

We now show how the firing weights of the immediate transitions φ_i for $i = 1..N$ must be set in order to achieve the desired routing behavior. We use the notation

$\mu : A_1\{c_1{}'x_1\} \wedge A_2\{c_2{}'x_2\} \wedge ... \wedge A_n\{c_n{}'x_n\} \rightarrow B_1\{d_1{}'y_1\} \wedge B_2\{d_2{}'y_2\} \wedge ... \wedge B_m\{d_m{}'y_m\}$

to denote a transition mode μ in which $c_i \times x_i$ tokens are taken from place A_i for $i = 1..n$ and $d_j \times y_j$ tokens are deposited in place B_j for $j = 1..m$. If an event modeled by token color $x^{t,k}$ visits the system node corresponding to place π_i, from there it can possibly be forwarded to every node n_j such that $\nu_{i,j}^{t,k} > 0$. Assuming that there are m such nodes, let us denote the set of their id's as $\zeta = \{j_1, j_2, ..., j_m\} = \{j : \nu_{i,j}^{t,k} > 0\}$. For every subset of these nodes $\sigma = \{l_1, l_2, ..., l_r\} \subseteq \{j_1, j_2, ..., j_m\}$, we define a firing mode μ_i^σ of transition φ_i as follows:

$$\mu_i^\sigma : \pi_i\{1'x^{t,k}\} \rightarrow \pi_{l_1}\{1'x^{t,k}\} \wedge \pi_{l_2}\{1'x^{t,k}\} \wedge ... \wedge \pi_{l_r}\{1'x^{t,k}\} \tag{4.17}$$

which means that a $x^{t,k}$ token is taken from place π_i and a $x^{t,k}$ token is deposited in each of the places $\pi_{l_1}, \pi_{l_2}, ..., \pi_{l_r}$, as shown in Figure 4.5. This corresponds to node n_i forwarding an arriving event of type e_t, published by publisher p_k, to nodes $n_{l_1}, n_{l_2}, ..., n_{l_r}$. In order to achieve the desired routing behavior, the firing weight $\psi(\mu_i^\sigma)$ of the mode is set as follows:

$$\psi(\mu_i^\sigma) = \prod_{g \in \sigma} \nu_{i,g}^{t,k} \prod_{g \in \zeta \setminus \sigma} (1 - \nu_{i,g}^{t,k}) \tag{4.18}$$

To explain this, let us consider the action of node n_i forwarding an arriving event to node n_{l_h} as an "event" in terms of probability theory. For $h = 1..r$, we have r events and their probabilities are given by $\nu_{i,l_h}^{t,k}$. At the same time for each $g \in \zeta \setminus \sigma$ we can consider the action of node n_i *not* forwarding an event to node n_g as an event. We have $m - r$ such events in total and their probabilities are given by $(1 - \nu_{i,g}^{t,k})$. If we assume that all these events are independent, the probability of all of them occurring at the same time would be equal to the product of their probabilities. Thus, Eq. (4.18) can be interpreted as the probability of forwarding an arriving event of type e_t, published by publisher p_k, to nodes n_{l_h} for $h = 1..r$ and no other nodes. Even though in reality the independence assumption might not hold, it is easy to see that by setting the firing weights as indicated above, routing behavior with equivalent resource consumption is enforced.

The model we presented is focused on capturing resource contention inside the system nodes, however, it can easily be extended to also capture contention for network resources. Network links can be modeled using queueing places that event tokens visit when they are sent from one place to another. Figure 4.6 shows an example of how networks can be modeled. Nodes n_{i_1} and n_{i_2} (represented with places π_{i_1} and π_{i_2}) communicate with nodes n_{j_1} and n_{j_2} over a network (e.g., a LAN) modeled using queueing place χ_1. Node n_{i_2} communicates with node n_{j_3} through another network modeled using queueing place χ_2. Depending on the size of QPN models, different methods can be used for their analysis, from product-form solution methods [22] to optimized simulation techniques [127].

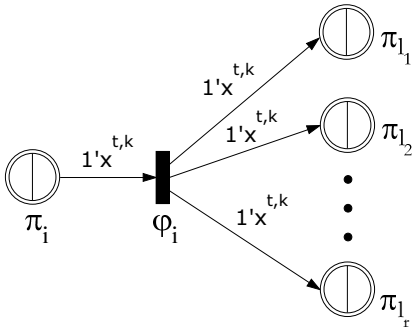


Figure 4.5: Firing of Transition φ_i in Mode μ_i^σ

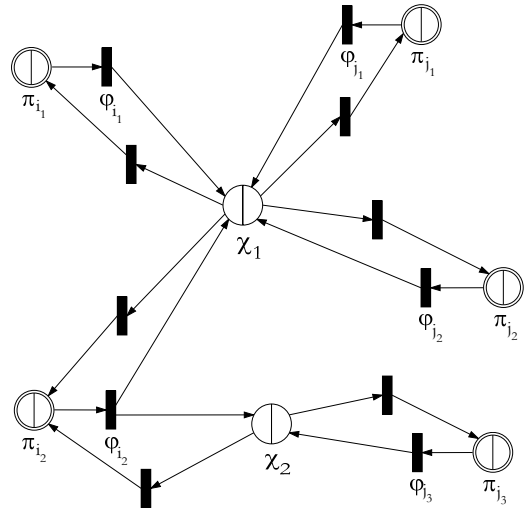


Figure 4.6: Modeling Network Connections.

4.2 Performance Modeling Pattern

In this Section we introduce *performance modeling patterns (PerfMP)* for QPNs. To the best of our knowledge, no such patterns have been published yet for QPNs or for EBS. Overall, we define eleven patterns, which we illustrate in Table 4.1. Our goal is to provide solutions for common application scenarios in EBS. Some of the patterns target specific scenarios of EBS while others provide solutions to general modeling issues, e.g., thread pooling.

In detail we reflect the following aspects in our QPN patterns:

- *Asynchronous communication*
- *Pull-based* vs. *push-based* communication
- *Point-to-point* vs. *one-to-many* communication
- *Resource management*, e.g., the number of events a consumer can process in parallel
- *Time controlled* behavior, e.g., connection times
- *Load balancing*

Several of our patterns can be combined or modified to reflect a certain application behavior in our QPN models. In the following we have a closer look at our patterns. However, before we discuss them in detail, we introduce our template that we use to describe the patterns in a structured way.

Pattern Template

Our pattern is composed of four parts:

1. *Characteristics:* In this part the main aspects of a pattern are summarized in keywords.
2. *Example:* Providing a sample scenario for the pattern.
3. *Description:* A detailed description of the pattern including motivation and underlying ideas.

Name	Description
Pattern 1: **Standard Queue**	Models a queue, which pushes notifications to the consumer.
Pattern 2: **Standard Pub / Sub - Fixed Number of Subscribers**	Models a standard publish / subscribe scenario, in which incoming notifications are pushed to a constant number of subscribers.
Pattern 3: **Standard Pub / Sub - Configurable Number of Subscribers**	A standard publish / subscribe scenario, in which incoming notifications are pushed to a variable number of subscribers.
Pattern 4: **Time controlled Pull I**	Implementation of simple time-controlled pull communication. An event consumer connects frequently to pull one event.
Pattern 5: **Time controlled Pull II**	An event consumer connects frequently to the broker to pull all waiting event notifications.
Pattern 6: **Resource controlled Pull I**	An event consumer pulls an event and processes it. When the event is processed, the consumer pulls the next event.
Pattern 7: **Resource controlled Pull II**	Similar to Pattern 6, but a different implementation for parallel event processing.
Pattern 8: **Time Window**	A consumer connects frequently to a broker and stays online for a specified time interval before disconnecting.
Pattern 9: **Random Load Balancer**	A load balancer which distributes incoming event notifications randomly among the consumers.
Pattern 10: **Round-Robin Load Balancer**	A load balancer which distributes incoming event notifications round-robin among the consumers.
Pattern 11: **Queuing Load Balancer**	A load balancer stores incoming event notifications and consumers pull events for processing them.

Table 4.1: Performance Modeling Patterns

4. *QPN Definition:* The definition of the QPN is presented in four tables:
   (a) *Places:* A list of all places including the name, type (Q=queueing place, O=ordinary place, S=subnet place) and a short description.
   (b) *Colors:* A list of all colors.
   (c) *Initial Number of Colors:* Specifies, how many tokens of which color in which place are generated in the initial phase of a QPN.
   (d) *Transitions:* A description of all transitions including colors, places and firing weight (FW, mostly 1 or ∞).

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

Pattern 1: Standard Queue

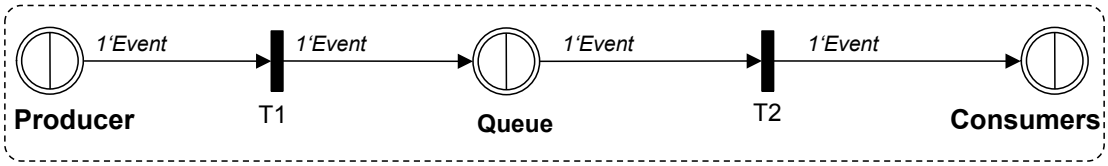


Figure 4.7: Standard Queue Pattern

Characteristics

- 1 : 1 communication
- Push-based

Example

A customer sends an order message to a component for "incoming orders". All incoming orders are first stored by a queue of a message-oriented middleware and afterwards delivered to the order-processing application.

Description

The *standard queue pattern* is a simple example for modeling *asynchronous point-to-point messaging*: An event is sent by its producer to the queue. After the queue processed the event, the event is forwarded (*pushed*[1]) to exactly one consumer.

QPN Definition

Places:

Place	Type	Description
Producer	S	Publishes events.
Queue	Q	Queue for incoming events.
Consumer	S	Consumes events.

Colors:

Color	Description
Event	Represents the event notification.

Transitions:

Id	Input	Output	FW	Description
T1	1 Event (*Producer*)	1 Event (*Queue*)	1	Prod. publishes event.
T2	1 Event (*Queue*)	1 Event (*Consumer*)	1	Forwarding of events of the Consumer.

[1] A support for push-based communication using queues is, e.g., part of the JMS specification.

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

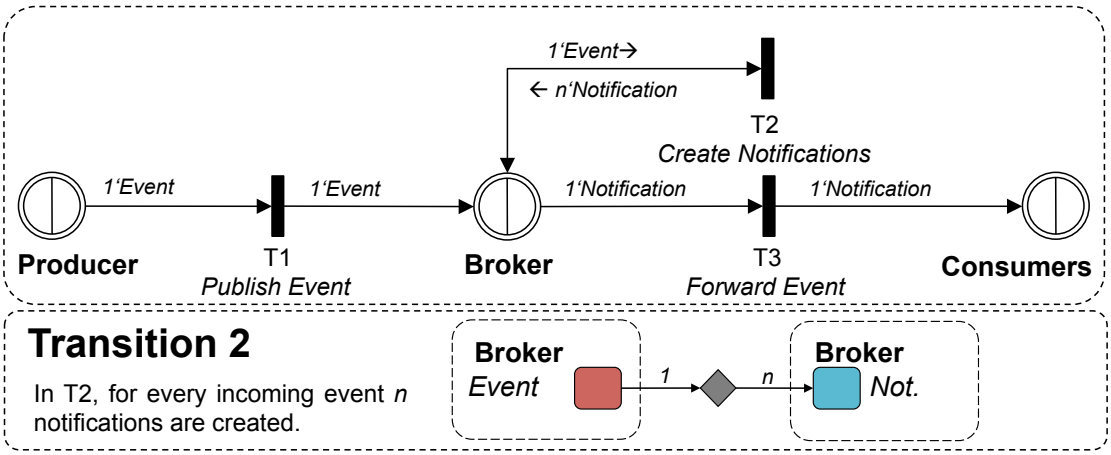


Figure 4.8: Standard Pub/Sub Pattern - Fixed Number of Subscribers

Characteristics

- $1 : n$ communication (one event is consumed by n consumers)

Example

A fixed number of event consumers subscribe to a topic.

Description

A producer publishes an event, which is received by a broker. The broker notifies the event consumers (e.g. subscribers of a topic) by sending a notification event to them.

The idea of this pattern is based on the presumption that the service demand per event on the broker is composed of two parts, the service demand needed for every incoming event and the aggregated service demands for the notification of the subscribers:

$$S_{EventTotal,Broker} = S_{Event,Broker} + n \cdot S_{Notification,Broker} \tag{4.19}$$

where

n	:	No. of event notifications.
$S_{Event,Broker}$	:	Service demand for receiving and processing incoming message.
$S_{Notification,Broker}$	:	Service demand of broker to create, process and send notifications.

We implement the service demand in our patterns by using two different token colors, one for the general service costs (*event*) and one for the notifications (*notification*) created by the broker. Therefore each incoming event is represented by one token *event* and n tokens of the color *notification*, where n is the number of consumers.

In this version of the pattern, we implemented a straight-forward approach for a $1{:}n$ communication. The number of consumers is directly set in the cardinality of the transition (see Figure 4.8). The downside of this approach is, that the number of consumers is fixed in the transition and therefore cannot be modified without changing the transition (and therefore the structure of the model). The reason is that cardinality of transitions in standard QPNs is defined as constant number. To add more flexibility we propose to allow not only constants but also, e.g., distribution functions (see Section 4.3).

QPN Definition

Places:

Place	**Type**	**Description**
Producer	S	Publish events.
Broker	Q	Receives all incoming events and forwards notifIcation events to n consumers.
Consumer	S	Consumes incoming events.

Colors:

Color	**Description**
Event	Represents the published event.
Notification (Not.)	Event notification.

Transitions:

Id	**Input**	**Output**	**Description**
T1	1 Event (*Producer*)	1 Event (*Broker*)	Producer publishes event.
T2	1 Event (*Broker*)	n Notifications (*Broker*)	Notifications are created.
T3	1 Notification (*Broker*)	1 Not. (*Consumer*)	Consumer receives event.

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

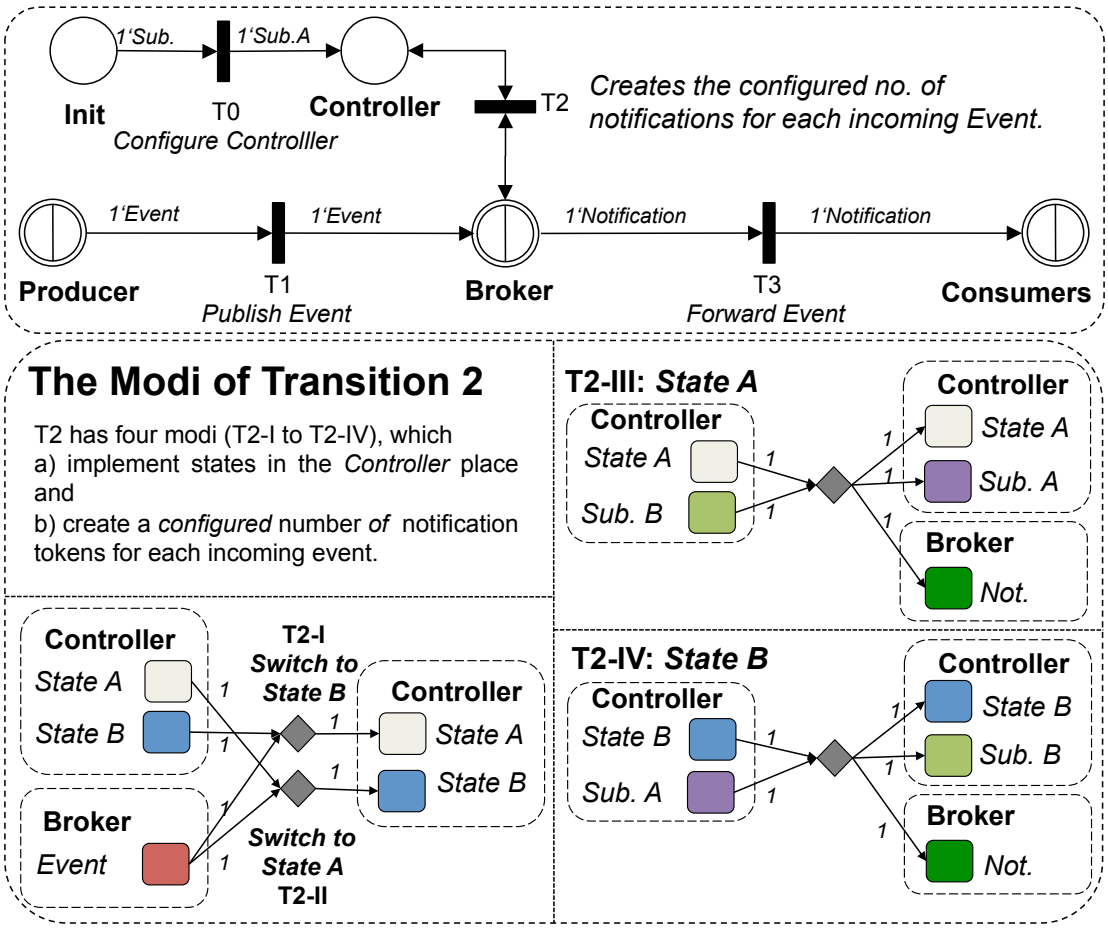


Figure 4.9: Standard Pub/Sub Pattern - Configurable No. of Subscribers

Characteristics

- $1 : n$ communication (one event is consumed by a configurable number of consumers)
- Dynamic number of notifications

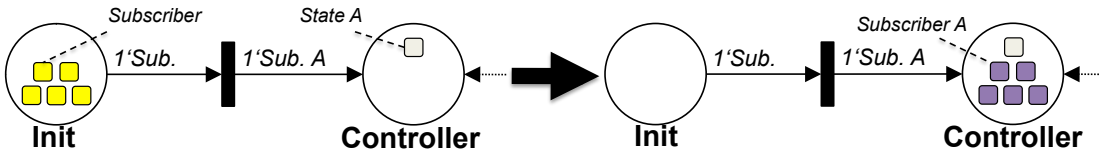
Example

A dynamic number of event consumers subscribe to a topic.

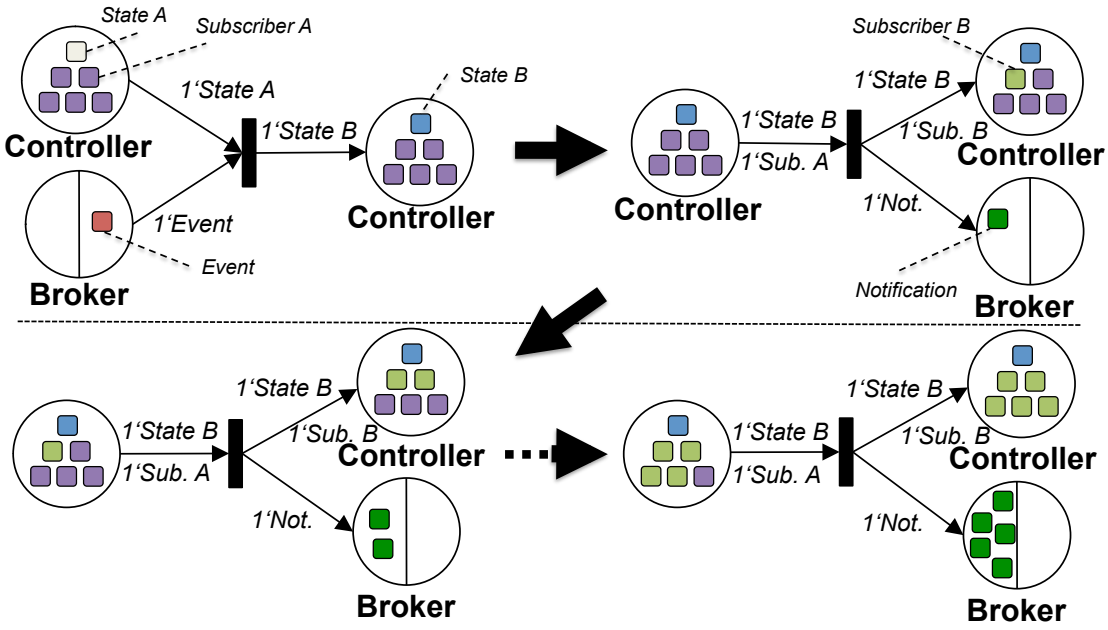
Description

In complex and flexible models we need the possibility to scale the number of notifications easily. In these cases Pattern 2 is not applicable. As a consequence, we define Pattern 3 where the number of notifications per event can be configured by the token count.

Pattern 3 is based on the underlying idea of Pattern 2 regarding the service demand. In contrast to Pattern 2, the user can configure the number of notifications for each event simply by adjusting the number of tokens named *Subscriber* in the system without modifying the



(a) Initialization



(b) Generate Notifications

Figure 4.10: Example for Pattern 3

cardinality of the transitions. The number of *Subscriber* tokens can be set either by modifying the number of initial tokens in the system or by adjusting them dynamically at simulation time. The idea is that for an incoming event a notification is created for each *Subscriber* token and forwarded to the consumers.

For this purpose we introduce an ordinary place *Controller* and define two colors, *State A* and *State B*, in QPNs. 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 state exists.

An incoming *Event* triggers a state change from state A to B (or vice versa). As a response to an incoming *Event* the configured number of event notifications is generated. This is implemented by Transition *2-III / 2-IV* (see Figure 4.9). 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 consumer.

For a better understanding, we provide a detailed description of the different steps and states. The underlying transitions are illustrated in Figure 4.9.

1. *System Initialization*
   First, we configure the number of subscribers by setting the initialization number of *Subscriber* tokens n. These n tokens are then transformed by transition *T0* to n *Subscriber A* tokens stored in the *Controller* place. This step is illustrated in Figure 4.10(a). Transition T0 is only fired once at system initialization time.

The *Controller* place is in state A which is represented by a *State A* token stored in the depository.

After the system initialization, the following tokens exist in the system:

Place	**Color**	**Count**	**Note**
Controller	State A	1	Init value.
Controller	Subscriber A	n	Received via T0.

2. *Creation of Notifications*

   (a) *Producer Publishes Event*
   The producer publishes an *Event* token, which arrives via *T1* at the broker. After the broker received the *Event* token, the 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.

Place	**Color**	**Count**	**Note**
Controller	State B	1	Produced by T2.
Controller	Subscriber A	n	

   (b) *Notification of Subscribers*
   Since the *Controller* place is now state B, transition *T2-IV* is fired for each of the n *Subscriber A* tokens and transforms them as illustrated in Figure 4.10(b) into *Notification* tokens (sent to the Broker) and into *Subscriber B* tokens stored in the *Controller*, respectively. These *Notification* tokens will be processed by the *Broker* and afterwards delivered to the consumers. Therefore, each *Event* token triggers the generation of n *Notifications*.

Place	**Color**	**Count**	**Note**
Controller	State B	1	
Controller	Subscriber B	n	
Broker	Notification	n	To be forwarded to the consumer.

QPN Definition

Places:

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

Note The *Init* place can be used to configure the number of subscribers of different interactions.

Colors:

Color	Description
Event	Represents the published event.
Notification (Not.)	Event 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 (see Figure 4.9):

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

Missing Priorities of Transitions

The Controller is defined as an ordinary place. Since standard QPNs do not support priorities of transitions this may become an issue: if two events 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 notification.

Imagine a situation where the *Controller* is in state A and two *Event* 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 arrises if the second *Event* token triggers a second state change back to A (via *T2-II*) before all n notifications for the first token are generated.

To solve this issue we developed three approaches:

1. *Using transition priorities*
   We suggest to extend standard QPNs by adding transition priorities. This allows to define rules such as: "If possible, fire transitions *T2-III / T2-IV* always before *T2-I / T2-II*."

2. *Set the firing weight of T2-III and T2-IV to* ∞.
   This solution does not completely rule out the incorrect state changes, but the probability converges to zero.

3. *Adding an additional queueing place* (see Figure 4.11)
   This *Enqueuer* place is a queueing place with a single server and used to form a line of events. This allows us to process the *Events* 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 modified as described below. By defining a service demand close to zero for *Event* tokens on the *Enqueuer*, a distortion of the results should be avoided.

Id	Input	Output	FW	Description
T2-I	1 State A (*Controller*) 1 Event (*Enqueuer*)	1 State B(*Controller*)	1	Switch state of *Controller* to B.
T2-II	1 State B (*Controller*) 1 Event (*Enqueuer*)	1 State A(*Controller*)	1	Switch state of *Controller* to A.
T4	1 Event (*Broker*)	1 Event (*Enqueuer*)	1	

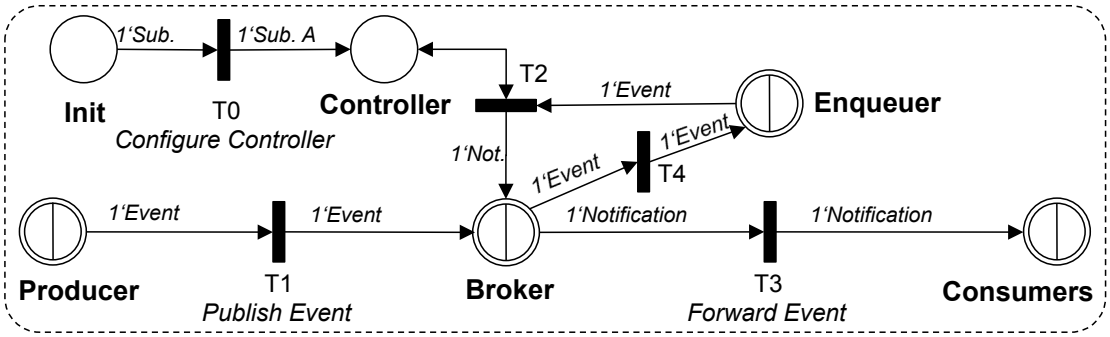


Figure 4.11: Pattern 3 using an Enqueuer for Incoming Events

Note: This problem does not occur if the broker place has a single server. In this case, two events never arrive at the same time.

Pattern 4: Time Controlled Pull I

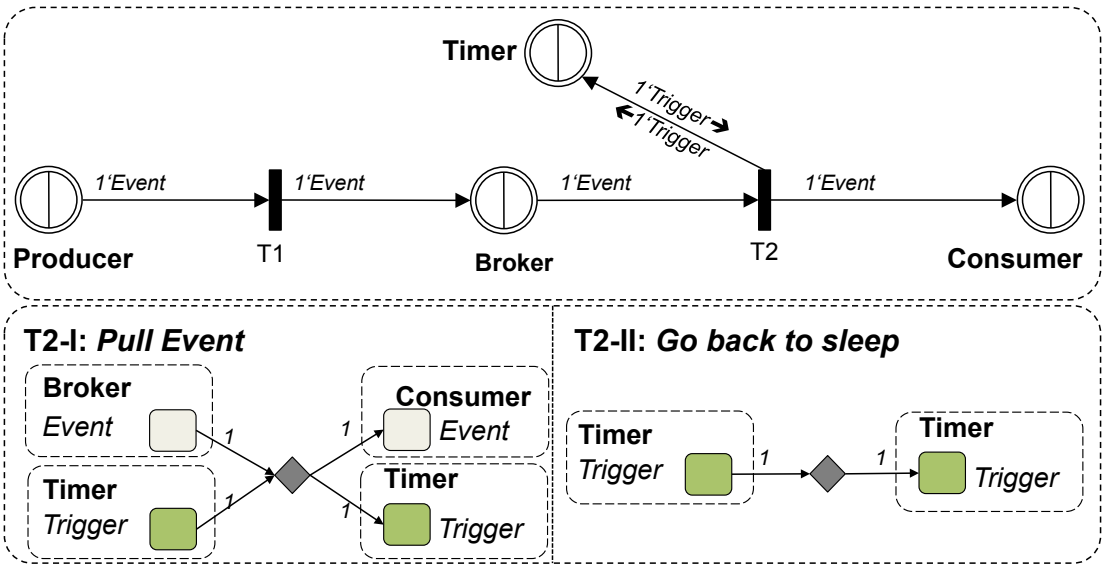


Figure 4.12: Time-Controlled Pull Pattern

Characteristics

- *Pull-based* communication
- *Time controlled behavior*: the consumer connects frequently to the broker to pull *one* event after a certain time interval.

Example

An event consumer connects frequently to a broker to check whether notifications are waiting. If yes, he downloads exactly *one* of them to process it. Afterwards, the consumer closes the connection and waits a specified period of time before reconnecting.

A real world scenario represented by this pattern are Wireless-Sensor Networks (WSN). To save battery, the nodes are mostly disconnected. However, they connect frequently for a short moment to other nodes to exchange events and data.

Description

In modeling techniques such as QPNs, events are *pushed* by transitions from one place to another, e.g., after an event was served at one place, it is pushed to the next place. However, in scenarios like our example we are facing pull-based communication behavior and an approach is needed to reflect this in our models. Therefore we introduce a methodology how *pull-based* communication can be modeled using, e.g., QPNs. To model all aspects of our scenario, we presents additionally a way to implement *time-controlled* behavior.

In our scenario the *Consumer* connects frequently to the *Broker*, tries to pull, if available, exactly one event and disconnect. To model this behavior, we need a timer which is responsible for establishing the connection. As shown in Figure 4.12 we implement the timer by adding a queuing place *Timer* (scheduling strategy: infinite server) and a new token color named *Trigger*, which triggers the establishment of the connections and the pull attempt after a certain time

interval. The interval between two connections is controlled by the service demand of the *Trigger* token on the *Timer* place ($S_{Trigger,Timer}$). Since the consumer disconnects immediately after the pull attempt there is no need to model connection times. We model the scenario as an endless loop composed of four phases. In the first phase, the consumer is disconnected and the *Timer* is processing the *Trigger* token. When the *Timer* has processed the Trigger token, the second phase starts and the connection is established. In the next phase, the consumer tries to pull an *Event*. Finally, the consumer closes the connection and we re-enter the first phase.

In the following we illustrate our modeling approach in more detail:

1. *System Initialization*
   In the beginning, the connections is closed and one *Trigger* token is stored in the depository of the *Timer* place (end of the first phase).

2. *Open and Close Connections*
   When the *Trigger* token is available in the depository of the *Timer*, transition *T2* is fired.

   Transition *T2* implements the establishment of a connection (II. phase), the pull attempt (III. phase) and the disconnection (IV. phase). Two possibilities exist (see Figure 4.12):

   (a) The connection is established and the consumer pulls one *Event* from the *Broker* (*T2-I*).

   (b) The connection is established, but no *Event* is available (*T2-II*).

   In both cases, the connection has to be closed afterwards. This is implemented by adding the *Trigger* token to the *Timer* queue. When the connection is closed, we are back in the first phase. After t=$S_{Trigger,Timer}$[2], the *Timer* has processed the *Trigger* token and moves it to its depository. The next step is to enter the second phase and to open a connection to the broker again.

Note *T2-II* should only be fired if no *Event* is available in the depository of the *Broker*. If an *Event* is available, *T2-I* has to be fired. Again, we have to face the limitation of standard QPNs that transition priorities are not supported. In this scenario, the easiest way is to define the firing weights of the transitions so that $\frac{\text{Firing Weight of T2-II}}{\text{Firing Weight of T2-I}}$ is close to zero, i.e. the possibility that T2-II is fired although a *Event* token exists is close to zero.

QPN Definition

Places:

Place	Type	Description
Producer	S	Publishes events.
Broker	Q	Broker for all incoming events.
Timer	Q	Timer place (scheduling strategy: Infinite Server).
Consumer	S	Pulls events from broker.

Colors:

Color	Description
Event	Represents the published event.
Trigger	Triggers pull commands.

Init No. of Colors:

Color	Place	Count
Trigger	*Trigger Store*	1

[2]Since the *Timer* is using $InfiniteServer$ as scheduling strategy, the response time of the queue for a token is equal to its service demand (and waiting time is zero).

Transitions:

Id	Input	Output	FW	Description
T1	1 Event ($Producer$)	1 Event ($Broker$)	1	Producer publishes event.
T2-I	1 Event ($Broker$) 1 Trigger ($Timer$)	1 Event ($Consumer$) 1 Trigger ($Timer$)	∞	Pull exactly one event and go back to sleep.
T2-II	1 Trigger ($Timer$)	1 Sleep ($Timer$)	1	If no event is available $\rightarrow$ go back to sleep.

Pattern 5: Time Controlled Pull II

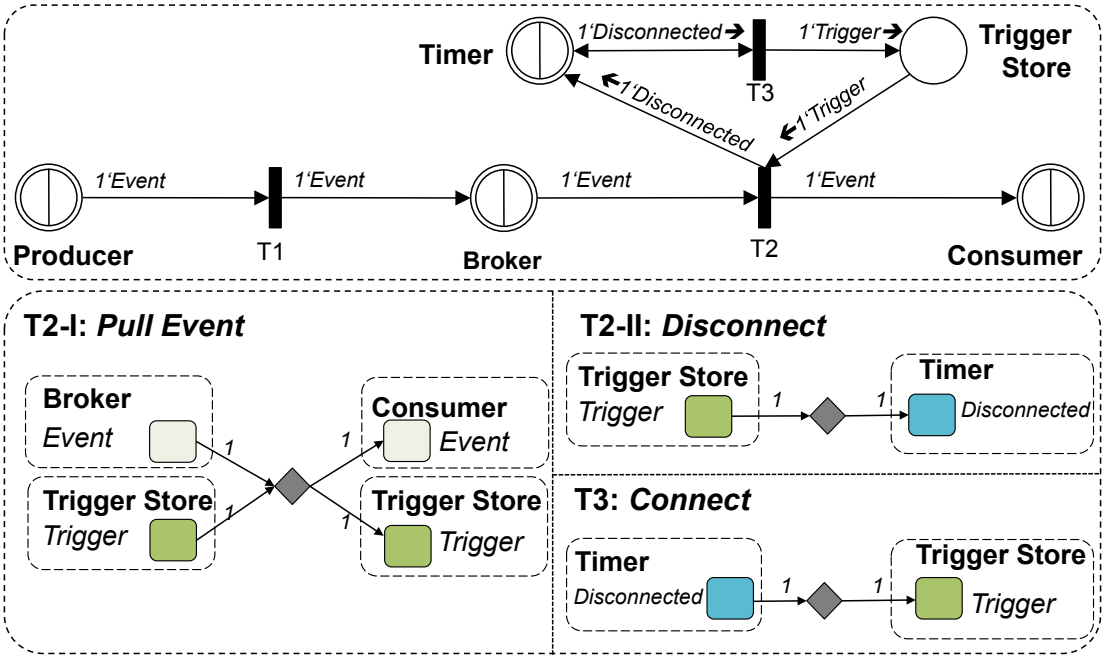


Figure 4.13: Time-controlled Pull Pattern

Characteristics

- *Dynamic pull-based* communication
- *Time controlled behavior*: the consumer connects frequently to the broker to pull *all* waiting events after a certain time interval.

Example

An event consumer connects frequently to a broker to check whether notifications are waiting. If yes, the consumer downloads all of them, closes the connection afterwards and waits for a specified period of time before reconnecting.

Description

In this scenario we face again a time controlled pull-based communication behavior. In contrast to Pattern 4, the consumer does not only pull *one* event per connection, but *all* event notifications available at the broker. To reflect this, we introduce this pattern.

Similar to Pattern 4, the scenario can be described as a loop composed of four phases:

Phase I:	Consumer is disconnected
Phase II:	Establish connection
Phase III:	Pull *all* available events
Phase IV:	Consumer disconnects

Phase I, II and *IV* are comparable to the one of the previous pattern. However, *Phase III* differs from Pattern 4, where the consumer tries to pull a fixed number (1) of *Events*. In the

underlying scenario of this scenario, the consumer wants to pull *all* existing *Event* tokens. Since the number of existing *Event* tokens is changing, our model has to dynamically adjust itself to pull all tokens.

As illustrated in Figure 4.13 we model *Phase I* similar to Pattern 4 by defining a *Timer* place and a token named *Disconnected*. While the *Timer* is processing the *Disconnected* token (and the *Disconnected* exists), the consumer is disconnected. In *Phase II* the connection is established: the *Disconnected* token is transformed by transition *T3* to a *Trigger* token. This token is stored in an ordinary place named *Trigger Store*. While the *Trigger* token exists, the consumer is connected to the broker. When the connection is established, the consumer tries to pull all waiting event notifications (*Phase III*). The number of notifications are waiting for the consumer is unknown. Therefore, we pull them one by one by firing transition *T2-I* for each *Event* notification. If no further *Event* exists in the depository of the broker, the consumer closes the connection. In our model this is performed by transition *T2-II* which transforms the *Trigger* token back to a *Disconnected* token. It has to be guaranteed that *T2-II* is only fired if no further *Event* tokens exist ($\rightarrow$ transition priorities).

QPN Definition

Places:

Place	**Type**	**Description**
Producer	S	Publishes events.
Broker	Q	Stores all incoming events.
Timer	Q	Timer queue (scheduling strategy: Infinite Server). If the *Timer* is empty, the consumer is connected to the broker.
Trigger Store	O	Stores Trigger token. If the consumer is disconnected, it is empty.
Consumer	S	Pulls events from broker.

Colors:

Color	**Description**
Event	Represents the published event.
Trigger	Triggers pull commands.
Disconnected	Represents disconnected state.

Init No. of Colors:

Color	**Place**	**Count**
Trigger	*Trigger Store*	1

Transitions:

Id	**Input**	**Output**	**FW**	**Description**
T1	1 Event (*Producer*)	1 Event (*Broker*)	1	Producer publishes event.
T2-I	1 Event (*Broker*) 1 Trigger (*Trigger Store*)	1 Event (*Consumer*) 1 Trigger (*Trigger Store*)	∞	Pull one event.
T2-II	1 Trigger (*Trigger Store*)	1 Disconnected (*Timer*)	1	If no event is available $\rightarrow$ go back to sleep.
T3	1 Disconnected (*Trigger Store*)	1 Event (*Broker*)	1	Establishes a connection.

Pattern 6: Request Controlled Pull I

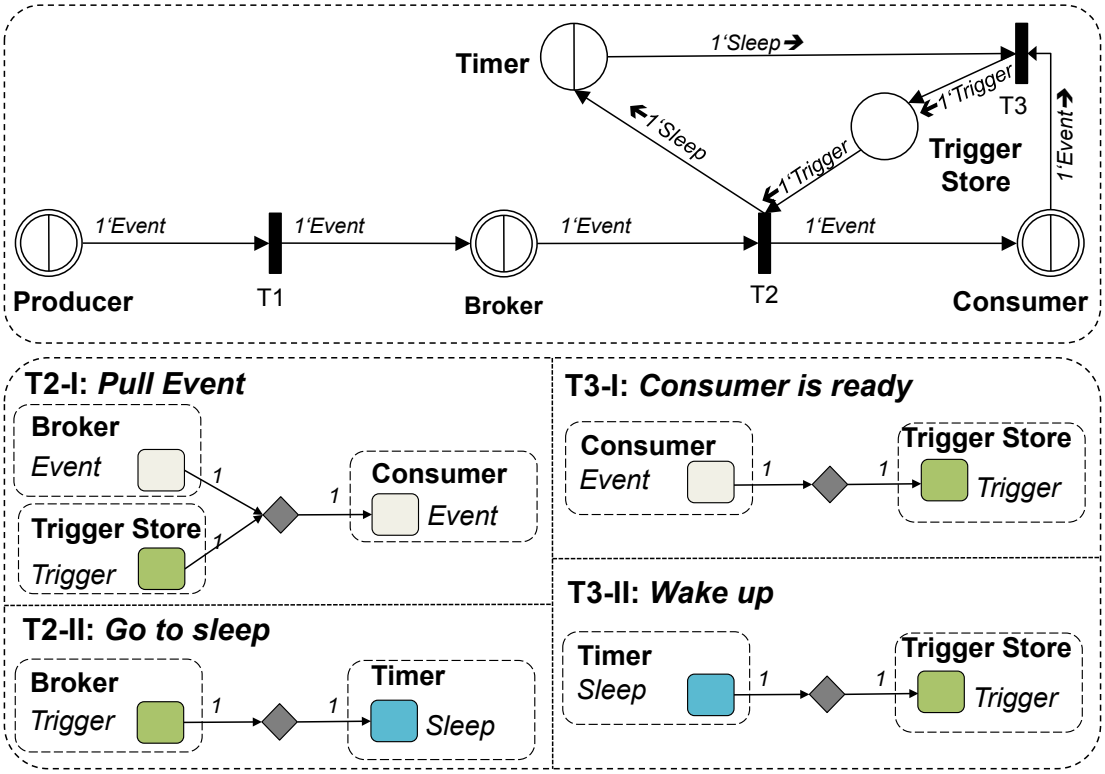


Figure 4.14: Standard Pull Patterns

Characteristics

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

Example

An event consumer connects frequently to a broker to check whether notifications are waiting. If yes, the consumer downloads one of them, closes the connection and processes the event notification. As soon as the event has been processed the consumer connects again to the broker to pull the next event. If no event is available, the consumer closes the connection and waits for a specified period of time before pursuing the next pull attempt.

Description

This scenario differs mainly from the previous ones in that 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 event as soon as he is ready, i.e. after the last event was processed. Only if no further event is available at the depository of the broker, 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 an event, the *Event* token is transformed by transition *3-I* to a Trigger token. Next, transition *2* is fired.

Depending on the availability of *Event* tokens in the depository of the Broker, either mode *2-I* (Event available) or mode *2-II* (no Event token) is chosen:

1. If an *Event* token exists the consumer pulls it (transition *3-I*) and disconnects. He will not reconnect before the *Event* token is processed.

2. If no *Event* 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 an *Event*.

Number of Service Places (Parellel Events) and Issues The pattern offers a simple way to set the maximum number of *Events* 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 events processed by the consumer to two. For the case that no *Event* token was available at the broker, 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 broker. In the meantime two new *Event* tokens arrived in the depository of the Broker. Since there is one *Trigger* token, only a single *Event* is moved to the consumer. The second *Event* token remains in the depository of the broker until the second sleep token has been processed by the timer, even if the Consumer has enough resources to process both *Events*.

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

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 4.15, all we need to implement such a pool are *Thread* tokens and a *Thread Pool* ordinary place (corresponding to *Thread* tokens, respectively *Thread Store*).

QPN Definition

Places:

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

Colors:

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

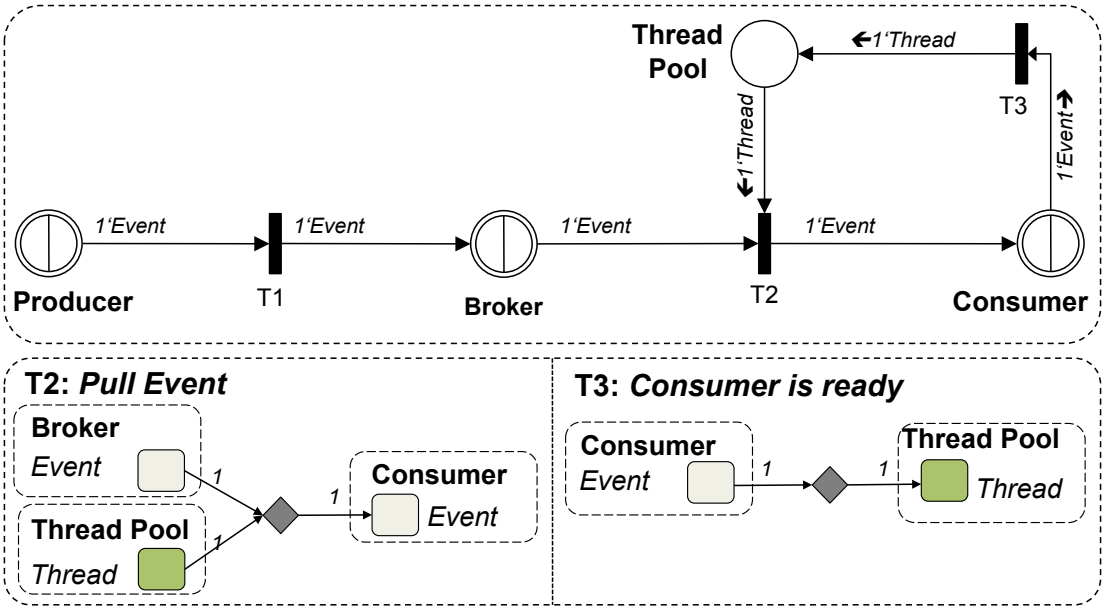


Figure 4.15: Modeling a Thread Pool

Init No. of Colors:

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

Transitions:

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

Pattern 7: Request Controlled Pull II

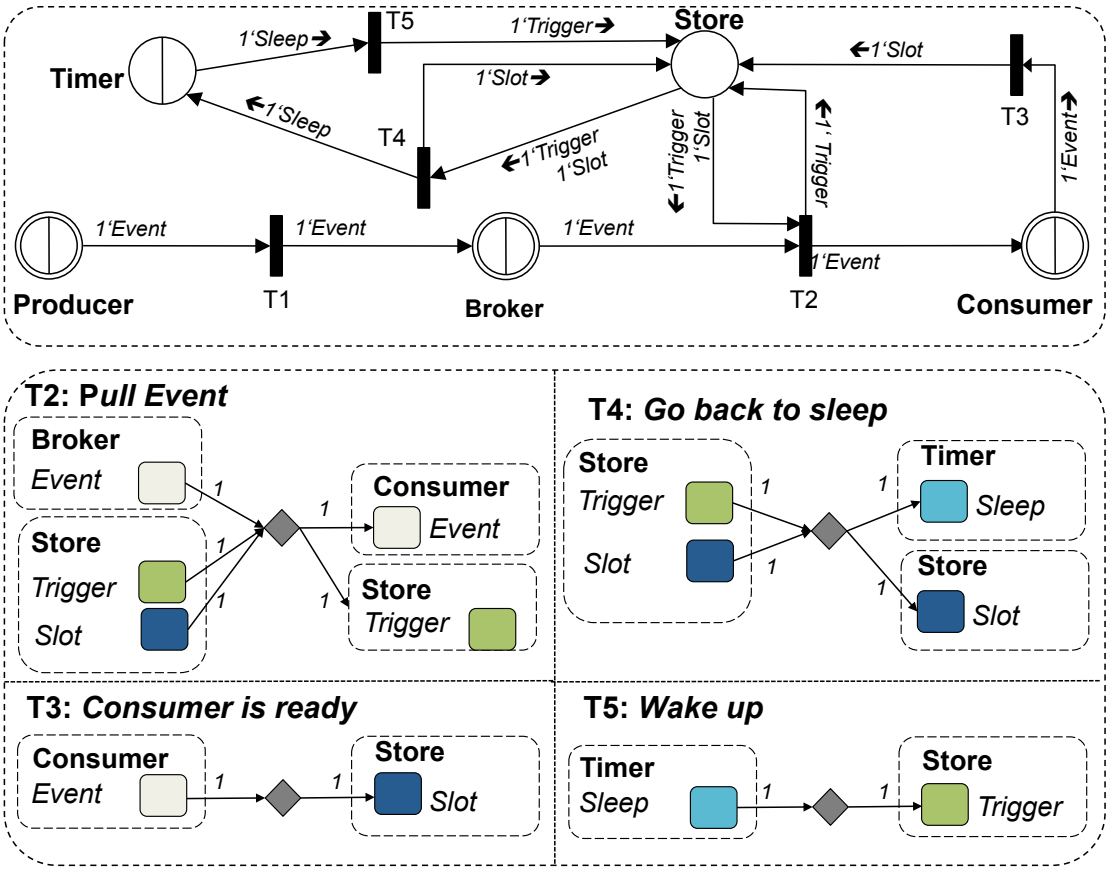


Figure 4.16: Pull Pattern - Request controlled II

Characteristics

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

Example *(similar to Pattern 6)*

An event consumer connects frequently to a broker to check whether notifications are waiting. If yes, the consumer downloads one of them, closes the connection and processes the event notification. As soon as the event has been processed the consumer connects again to the broker to pull the next event. If no event is available, the consumer closes the connection and waits for a specified period of time before pursuing the next pull attempt.

Description

This pattern addresses the same scenario as Pattern 6. It solves the issues of Pattern 6 discussed in the previous section for the costs of higher model complexity. In the previous pattern we

used a single color (*Trigger*) to model the pull attempt of the consumer and its resources. In this pattern, we use two colors to model these aspects:

1. *Trigger*: responsible to trigger pull attempt.
2. *Slot*: represents available resources of consumer.

The *Trigger* token controls if and when the consumer establishes a connection to the broker. The pull attempt of the consumer is modeled by transition *T2*: if an *Event*, a *Trigger* and a *Slot* (standing for free resources) exist, the consumer pulls an *Event*. The consumer tries to pull *Events* (by firing transition *T2*) until no further resources (*Slots*) or *Events* are available in the depository of the Store respectively the Broker:

- If no further resources are available, the consumer disconnects and waits for resources to be released (see transition *T3*) before establishing a new connection.
- If there are free resources (*Slot* tokens) available but no further *Event* token available, the consumer closes the connection and waits for ceratin time interval before he reconnects (see transitions *T4* & *T5*).

All places and transitions are illustrated in Figure 4.16. The number of initial *Slot* tokens in the ordinary place Store defines, how many events the consumer is able to handle in parallel, e.g., the number of service places.

QPN Definition

Places:

Place	**Type**	**Description**
Producer	S	Publishes an event.
Broker	S	Stores incoming events.
Timer	Q	Timer queue (scheduling strategy: infinite server).
Store	O	Stores trigger and slot tokens. Used to model free resources.
Consumer	S	Consumes events.

Colors:

Color	**Description**
Event	Represents the published event.
Trigger	Triggers pull commands.
Slot	Represents resources of consumer.
Sleep	Exists for time interval the consumer waits after an unsuccessful pull attempt before reconnecting.

Init No. of Colors:

Color	**Place**	**Count**	**Description**
Trigger	*Store*	1	
Slot	*Store*	j	j is equal to the number of events the consumer can process in parallel.

Transitions:

Id	**Input**	**Output**	**FW**	**Description**
T1	1 Event (*Producer*)	1 Event (*Broker*)	1	Producer publishes an event.
T2	1 Event (*Broker*) 1 Trigger (*Store*) 1 Slot (*Store*)	1 Event (*Consumer*) 1 Trigger (*Store*)	∞	Consumer pulls an event.
T3	1 Event (*Consumer*)	1 Slot (*Store*)	1	After an event is processed the consumer releases the resources.
T4	1 Trigger (*Store*) 1 Slot (*Store*)	1 Slot (*Store*) 1 Sleep (*Timer*)	1	If no event is available, disconnect and wait for a specified time interval.
T5	1 Sleep (*Timer*)	1 Trigger (*Store*)	1	After a specified time interval, the consumer wakes up to pull events.

Pattern 8: Time Window

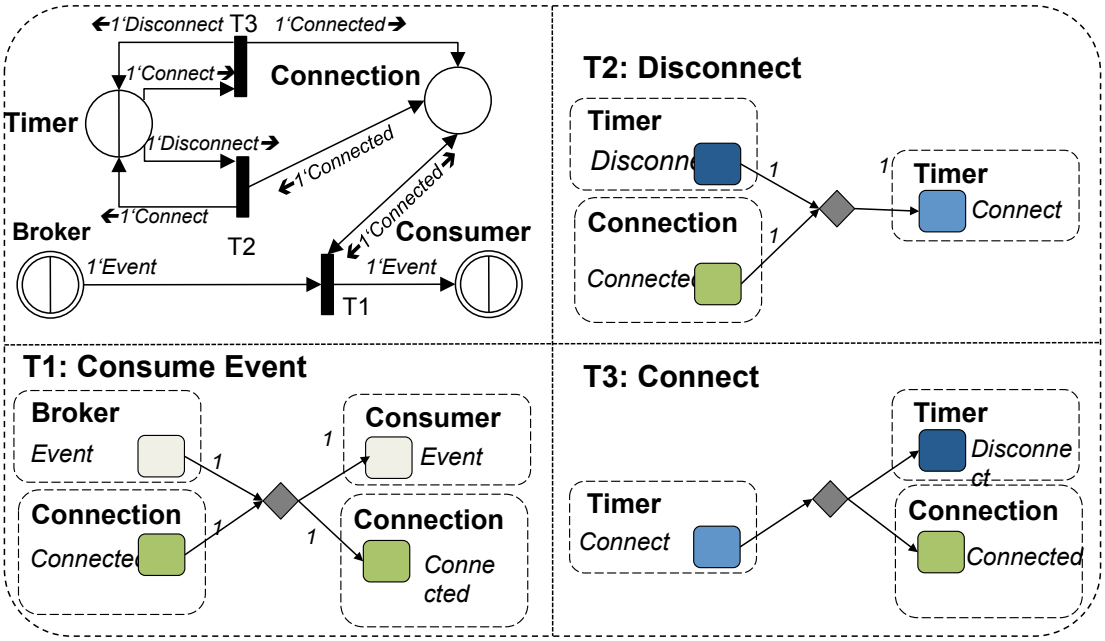


Figure 4.17: Standard Queue Pattern

Characteristics

- *Time-controlled*
- Modeling of a *time window*
- Modeling of connections

Example

An event consumer connects to a broker to receive events, stays online for a specified interval (time window) and disconnects afterwards, e.g., a consumer connects every hour for five minutes to a broker to receive events.

Description

In the previous pattern, we implemented a consumer who connects to the broker, pulls events and immediately disconnects. In this pattern, we handle another situation in which the consumer connects and stays online for specified a period. Therefore, we present a way to model a connection with two states (connected and disconnected) in this pattern.

The state change is triggered by two colors named *Connect* respectively *Disconnect*. The Connection itself is modeled by an ordinary place. If the Consumer has established a connection (transition *T3*), a token *Connected* is stored in the Connection place and a *Disconnect* token is sent to the Timer. The Timer controls the time between the state changes: the time a consumer stays connected (*disconnected*) is equal to the service demand of the *Disconnect* token (*Connect token*) of the Timer place. As soon as the Consumer disconnects (triggered by an *Disconnect* token), the *Connected* token is removed from the Connection place by transition

T2. As illustrated in Figure 4.17, *Events* are only pushed to the Consumer by the transition *T1* if a *Connected* token exists.

QPN Definition

Places:

Place	Type	Description
Broker	S	Responsible for forwarding events to the consumer.
Connection	O	Represents the state of connection. If a Connected token is stored inside, the consumer is connected.
Consumer	S	Consumes incoming events.

Colors:

Color	Description
Event	Represents the published event.
Connected	Only exists if connection is active.
Connect	Triggers to establish a connection.
Disconnect	Triggers the consumer to disconnect.

Init No. of Colors:
It is possible to configure the model in two ways: either the consumer is connected and disconnects immediately or the consumer establishes a connections as first action:

The Consumer is connected and disconnects:

Color	Place	Count
Connected	*Trigger*	1
Disconnect	*Timer*	1

The Consumer establishes a connection:

Color	Place	Count
Connect	*Timer*	1

Transitions:

Id	Input	Output	FW	Description
T1	1 Event (*Broker*) 1 Connect (*Conn.*)	1 Event (*Consumer*) 1 Connected (*Conn.*)	1 1	An event is forwarded to the consumer.
T2	1 Disconnect (*Timer*) 1 Connected (*Conn.*)	1 Connect (*Conn.*)	1 1	Closes connection and triggers reconnection.
T3	1 Connect (*Timer*)	1 Disconnect (*Conn.*) 1 Connected (*Conn.*)	1 1	Opens connection and triggers disconnection.

Pattern 9: Random Load Balancer

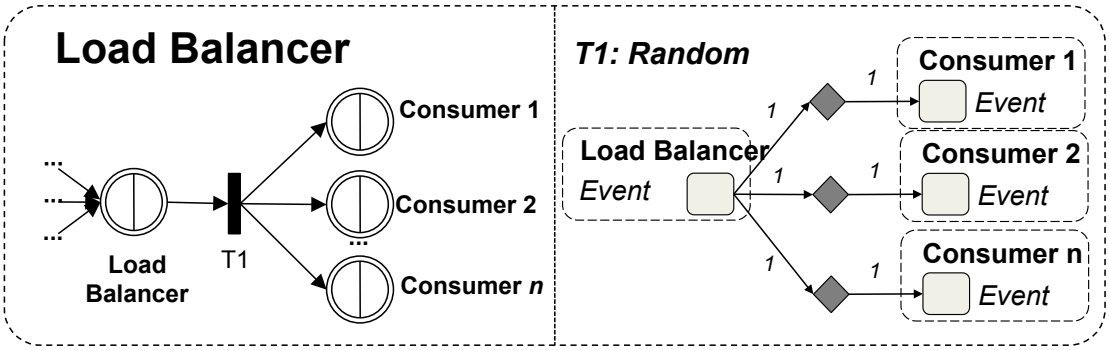


Figure 4.18: Load Balancer - Random

Characteristics

- Load balancing
- Random load distribution (uniformly distributed)

Example

Incoming events are distributed uniformly among consumers by a load balancer. Each event is forwarded to exactly one consumer.

Description

Incoming events are arriving at the place *Load Balancer* and are forwarded (pushed) to the Consumer by transition *T1*. For each Consumer, an own mode in transition *T1* is implemented. All these modes have the same fire weight. The events are uniformly distributed among consumers.

QPN Definition

Places:

Place	Type	Description
Load Balancer	S	Receives inc. events and forwards them to the Consumer.
Consumer i	S	Consumes incoming events. One place for each Consumer.

Colors:

Color	Description
Event	Represents a published event.

Transitions:

Id	Input	Output	FW	Description
T1	1 Event (*Load Balancer*)	1 Event (*Consumer*)	1	Forwards event to Consumer.

Pattern 10: Round-robin Load Balancer

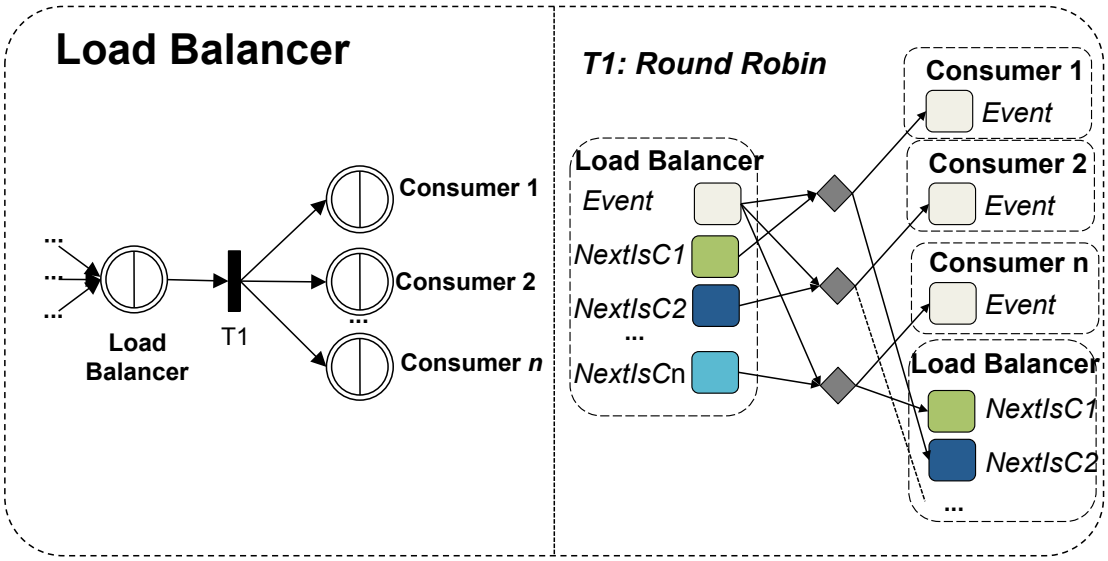


Figure 4.19: Load Balancer - Round Robin

Characteristics

- *Load balancing*
- *Round-robin*

Example

Incoming events are processed by a load balancer and afterwards distributed among event consumers in a round robin manner. Each event is forwarded to exactly one consumer.

Description

Incoming events are arriving at the place *Load Balancer* and forwarded to the Consumer by transition *T1*. For each Consumer exists an own mode in transition *T1*. Further, a token named *NextIsC*i is defined. *i* is the unique id of the consumer, e.g. for Consumer 1 the color name is *NextIsC1*. To implement a round robin behavior the load balancer needs to keep track which Consumer is next. We model this knowledge by storing a *NextIsC*i token in the Load Balancer. This token identifies the Consumer *i* who should receive the next *Event*. As illustrated in Figure 4.19, the modes of transition *T1* takes into account, which consumer is next and update the state of the Load Balancer by replacing the *NextIsC*i token with a *NextIsC*i+1 token.

For example, if an *Event* token and a *NextIsC1* token are stored in the depository place of the load balancer, the corresponding mode of transition *T1* is fired. The *Event* token is forwarded to Consumer 1 and the *NextIsC1* token is replaced by a *NextIsC2* token.

To initialize the model we have to specify the consumer that should receive the first *Event* token. This is done by initializing the corresponding *NextIsC*i token.

QPN Definition

Places:

Place	**Type**	**Description**
Load Balancer	S	Receives incoming events and forward an event.
Consumer i	S	Consumes incoming events. One place for each of the n Consumers.

Colors:

Color	**Description**
Event	Represents the published event.
NextIsCi	If exists, the next Event will be forwarded to Consumer i. One color for each of the n Consumers.

Init No. of Colors:

Color	**Place**	**Count**	**Description**
NextIsCi	*LoadBal.*	1	Defines that Consumer i will receive the first *Event* token.

Transitions:

Id	**Input**	**Output**	**FW**	**Description**
If $i < n$**:** T1-i	1 Event (*Load Balancer*) 1 NextIsCi (*Load Balancer*)	1 Event (*Consumer* i) 1 NextIsC$i+1$ (*Load Balancer*)	1 1	Forwards event to consumer.
If $i = n$**:** T1-n	1 Event (*Load Balancer*) 1 NextIsCn (*Load Balancer*)	1 Event *Consumer 1* 1 NextIsC1 (*Load Balancer*)	1 1	Forwards event to consumer.

Note: n is the count of Consumers.

Pattern 11: Queuing Load Balancer

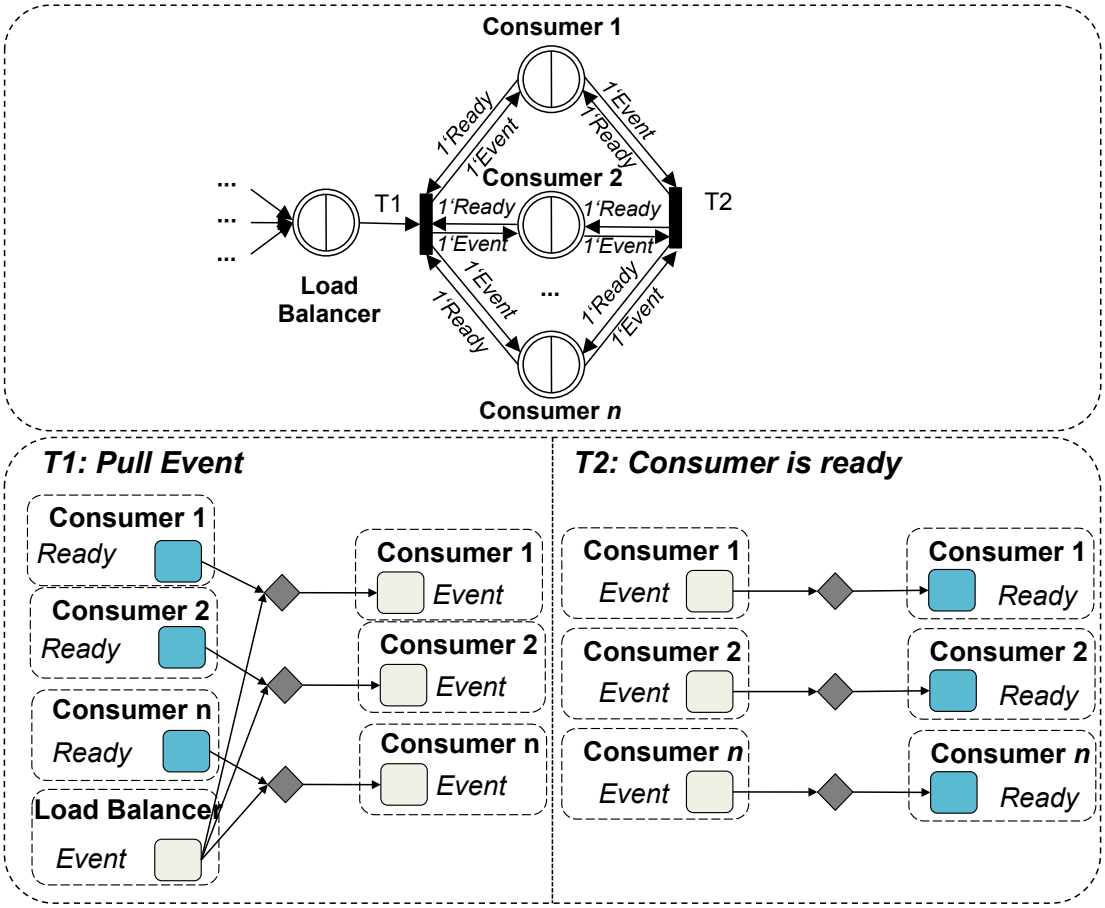


Figure 4.20: Load Balancer - Pull

Example

Incoming events are stored at the load balancer, e.g., in a queue. If a consumer has free resources he pulls an event and processes it.

Characteristics

- *Load Balancing*
- *Resource controlled*
- *Pull-based*

Description

Patterns 9 & 10 model a Load Balancer, who pushes *Events* directly after processing them to the Consumer. In this scenario the Load Balancer stores them after processing and the Consumer pulls *Event* tokens based on the availability of resources.

Each Consumer is reflected by an own place. Available resources of the consumer are represented by *Ready* tokens which are stored at the consumer place. The pull attempt of the Consumer is implemented by transistion *T1*: if an *Event* is available at the Load Balancer and the Consumer is ready (symbolized by a *Ready* token) to process the next *Event* token, the *Event* is forwarded to the Consumer. If multiple Consumers are ready, an *Event* is randomly assigned to one Consumer. After an *Event* is processed by the Consumer, the resources are released by creating a *Ready* token, which is done by transition *T2*. The number of available resources of a Consumer can be modeled by the initial number of *Ready* tokens at a Consumer place.

QPN Definition

Places:

Place	**Type**	**Description**
Load Balancer	S	Receives incoming events.
Consumer i	S	Consumes incoming events. One place for each of the n consumers.

Colors:

Color	**Description**
Event	Represents the published event.
Ready	Represents that the Consumer is ready to process an *Event*.

Init No. of Colors:

Color	**Place**	**Count**	**Description**
Ready	*Consumer i*	*j*	Specifies for each Consumer i the number of *Events*, which can be processed in parallel.

Transitions:

Id	**Input**	**Output**	**FW**	**Description**
T1	1 Event (*Load Balancer*) 1 Ready (*Consumer* i)	1 Event (*Consumer* i)	1	Forwards event
T2	1 Event (*Consumer* i)	1 Ready (*Consumer* i)	1	After an event is processed, the resources are released.

Note: n is the count of Consumers.

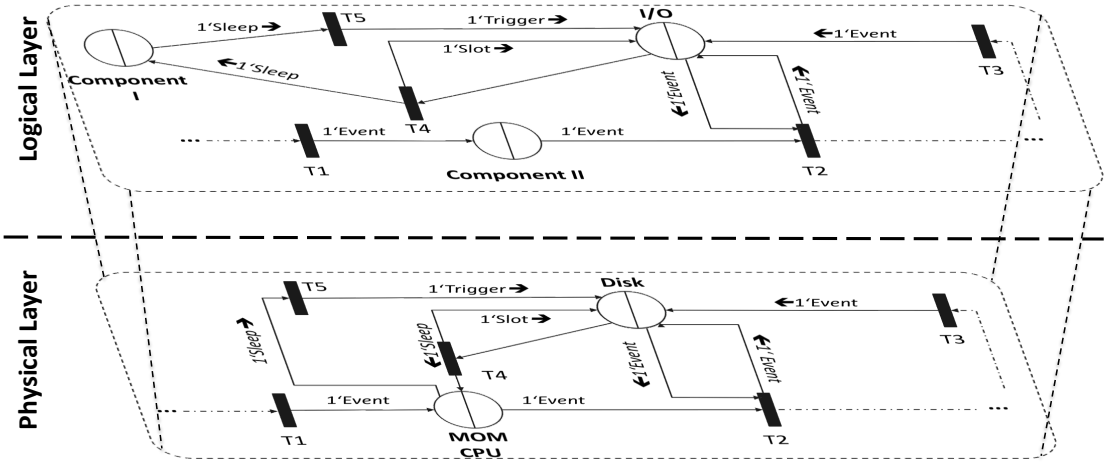


Figure 4.21: Physical and Logical Layers

4.3 Extensions of QPNs

In this section we discuss how QPNs can be conceptually extended to solve the mentioned shortcomings and limitations and to increase modeling simplicity and flexibility without increasing complexity. We propose three extensions for standard QPNs:

1. *Mapping of logical to physical resources*
   - 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. *Non-constant cardinalities of transitions*
3. *Priority support for transitions*
   - With transition priorities we introduce a firing hierarchy and can control the firing order of transitions effectively.

Our target implementation platform is the SimQPN simulation engine of QPME [127, 129, 128]. The concept of *mappings of logical to physical resources* is already available in the current version of QPME while both the others are planned for future release.

4.3.1 Mapping of Logical to Physical Resources

In traditional QPNs, the physical resources of an application in a specific setup are modeled. In our approach, we introduce a new level of indirection to increase flexibility and reusability of the models by distinguishing between *logical layer* and *physical layer* (see Figure 4.21).

In our approach the first step is to model the logical relations of an application and to focus on the interactions of logic entities such as components instead on physical resources such as CPU. By using subnet places to represent these logical entities, 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 and

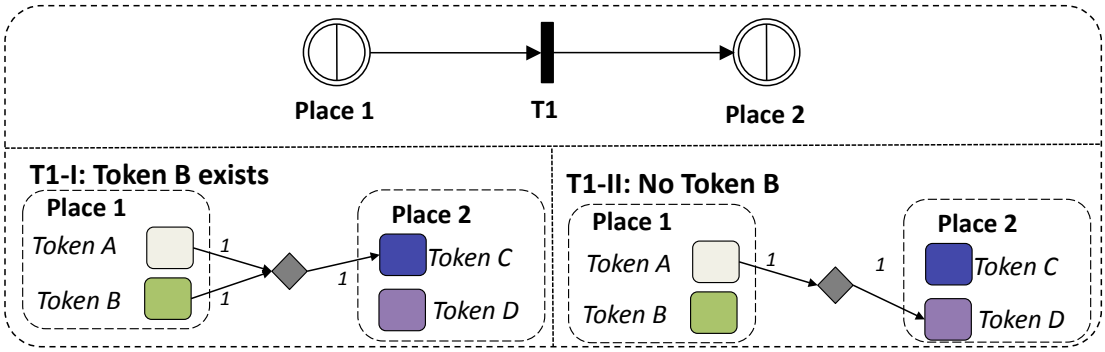


Figure 4.22: Example for Priority of Transitions

network links. The respective physical system resources are modeled using the queues inside the queueing places. After completing our logical model we can use it to generate different physical representations by defining mappings of logical resources to physical resources.

In Figure 4.21, an example application composed of two different components is illustrated. In the logical model, we define how they interact with each other. Multiple queueing places can be mapped to the same physical queue. For example, if all entities are deployed on a single server, their corresponding queueing places should be mapped to a set of central queues representing the physical resources of the server.

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 and to reuse the logical model by mapping into different physical scenarios. Further, it is possible to map several logic models in one physical model, e.g., to verify if a server has enough resources to run two applications.

4.3.2 Non-Constant Cardinalities of Transitions

In standard QPNs, the cardinalities of transitions are specified by a constant value. This has several limitations. In our approach it is possible to specify the cardinality of a transition by using a distribution function. This increases the flexibility and at the same time minimizes the number of transition modes. Furthermore, models are easier to maintain.

For example, in Pattern 2 (1:n communication) we define the number of subscribers directly by setting a constant integer value for the cardinality of the transition. For each different sized subscriber set we have to define an extra transition mode (e.g. one for 10 subscribers, one for 15, etc.). By allowing distribution functions, such scenarios are easier to model with less complexity. The modeler can concentrate on the logical relations instead of creating and maintaining a high number for transition modes.

4.3.3 Priority of Transitions

Standard QPNs do not support priorities of transitions. This approach provides only limited control of transition firing order. To illustrate this limitation we have a look at the example in Figure 4.22. We have two modes defined for transition *T1* and would like to model the following behavior:

- If *Token A* and *Token B* exist, the first mode *T1-I* should be fired.
- If only *Token A* and no *Token B* exist, *T1-II* should be fired.

Without priorities of transitions it is not possible to implement this correctly. However, while designing our patterns we faced frequently situations where we need a way to model such behavior. Therefore, there is a strong need for a solution, which we provide by extending QPNs with transition priority.

There are several possibilities to implement priorities. We propose the following one:

- Each transition is extended with a property *Priority* that stores an integer value
- Larger value means higher priority
- All transitions with the same priority x are element of a set *Priority Group* $PG_x = t_1, ...t_m$
- If $x < y$, all transitions $\in PG_x$ can only be fired, if no transition $t \in PG_y$ can be fired.
- Each transition is member of a priority group, therefore $T = PG_1 \cap PG_2 \cap ...PG_m$, m is the highest defined priority

To decide whether a transition (and if yes which transition) should be fired, we do not consider all transitions in T in the beginning. Instead, we start by checking PG_m (containing the transitions with the highest priority). If no transition $\in PG_m$ can be fired, the next transition set PG_{m-1} are inspected and so on. If two or more transitions of the same priority group could be fired, we choose a transition by taking the firing weight into account.

A more detailed discussion of the limitations and issues of missing priorities and possible modeling tricks are available in the section describing Pattern 3.

4.3.4 Tool Extension

We extended the QPME tool [128] to build and analyze the QPN. In the new version of QPME a first implementation of our approach to model logical and physical relations separately is available. Queues are now defined centrally (similar to token colors) and can be referenced from inside multiple queueing places. The flexible mapping of logical to physical resources is the ability to have multiple queueing places configured to share the same physical queue. This allows using 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 allows building 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. To the best of our knowledge, none of the currently available QPN modeling tools supports a comparable feature which means that the modeler would have to manage the tags manually which is cumbersome and error-prone

The support of priorities of transitions and non-constant cardinalities is announced for future QPME releases. In addition to the conceptual changes we contributed to QPME by improving performance, stability and usability.

4.4 Concluding Remarks

Performance engineering of EBS has it own specific challenges and differs from modeling traditional systems. Therefore, new methodologies are needed. In this chapter, we introduced a comprehensive methodology for workload characterization and performance modeling of EBS. We developed a workload model of a generic EBS and used operational analysis techniques to characterize the system traffic and derive an approximation for the mean event delivery latency. QPNs were chosen as a modeling paradigm because of their modeling power and expressiveness. Our methodology is general and can be applied to model and analyze different types of EBS, not only MOMs or DEBS.

As a further contribution of this chapter, we introduced a novel terminology for Performance Modeling Patterns, which we used to describe eleven performance modeling patterns for EBS. Our goal is to support modelers in building good performance models of EBS in a structured and efficient way by providing building blocks, which reflect common aspects of EBSs and capture their performance relevant aspects. These performance modeling patterns are the first published patterns using QPNs as modeling paradigm. Furthermore, they are the first patterns targeting EBS applications.

Additionally to the modeling methodology and the performance modeling patterns, we present several newly developed extensions for QPNs to improve their flexibility and efficiency. Our extensions are already implemented in the QPME tool or planned for future releases.

Chapter 5

Benchmarking of Event-Based Systems

In this chapter we discuss how to benchmark EBS using the example of MOMs and present a methodology for performance evaluation of MOM platforms using the SPECjms2007 standard benchmark. SPECjms2007 is the first industry standard benchmark for message-oriented middleware and was developed by SPEC member organizations under the lead of TU Darmstadt. The main contributions of SPECjms2007 are a comprehensive and standardized workload describing use cases of a real world application and a flexible benchmark framework. The workload itself is not specific to MOMs and can easily be adopted for other systems. We show by means of the example of the *jms2009-PS* benchmark we show how the workload can be adopted for pub/sub-based platforms. In two comprehensive case studies we show how both benchmarks, SPECjms2007 and jms2009-PS, can be applied to analyze specific features of the underlying middleware.

5.1 SPECjms2007 - A Standard Benchmark

SPECjms2007 is the first industry standard benchmark for MOM servers based on the Java Message Service standard interface. It was developed by the Java subcommittee of the Standard Performance Evaluation Corporation with the participation of TU Darmstadt, IBM, Sun, BEA, Sybase, Apache, Oracle and JBoss [130, 198, 200]. One of the major benefits of SPECjms2007 is that, in addition to providing a standard workload and metrics for MOM performance, the benchmark provides a flexible and robust framework for in-depth performance evaluation of messaging infrastructures [202, 201]. It allows us to create custom workload scenarios and interactions to stress selected aspects of the MOM infrastructure. Several officially reviewed results for different MOM products were published by SPEC [215].

In this section we discuss the requirements and provide an in-depth analysis of the workload. We describe the workload interactions, the way they are interrelated and how they can be customized. In a detailed case study we present how to apply this knowledge and evaluate different performance aspects of a MOM using SPECjms2007.

5.1.1 Workload Requirements and Goals of the SPECjms2007 Benchmark

In order to guarantee that applications meet their QoS requirements, it is essential that the platforms on which they are built are tested using benchmarks to measure and validate their performance and scalability. Benchmarks not only help to compare alternative platforms and

validate them, but can also be exploited to study the effect of different platform configuration parameters on the overall system performance. However, if a benchmark is to be useful and reliable, it must fulfill the following fundamental requirements [123]:

- It must be based on a workload *representative* of real-world applications.
- It must *exercise all critical services* provided by platforms.
- It must *not be tuned/optimized for a specific product.*
- It must generate *reproducible* results.
- It must not have any inherent *scalability* limitations

The major goal of the SPECjms2007 benchmark is to provide a standard workload and metrics for measuring and evaluating the performance and scalability of JMS-based MOM platforms[1]. In addition, the benchmark should provide a flexible framework for JMS performance analysis. To achieve this goal, the workload must be designed to meet a number of requirements that can be grouped according the following five categories:

1. *Representativeness*
2. *Comprehensiveness*
3. *Focus*
4. *Configurability*
5. *Scalability*

We now briefly discuss each of these requirements.

Representativeness

No matter how well a benchmark is designed, it would be of little value if the workload it is based on does not reflect the way platform services are exercised in real-life systems. Therefore, the most important requirement for the SPECjms2007 benchmark is that it is based on a representative workload scenario including a representative set of interactions, message types, message sizes and message delivery modes. The communication style and the types of messages sent and received by the different parties in the scenario should represent a typical transaction mix. The goal is to allow users to relate the observed behavior to their own applications and environments.

Comprehensiveness

Another important requirement is that the workload is comprehensive in that it exercises all platform features typically used in the major classes of JMS applications. Both the point-to-point and publish/subscribe messaging domains should be covered. The features and services stressed should be weighted according to their usage in real-life systems. There is no need to cover features of MOM platforms that are used very rarely in practice.

The following dimensions have to be considered when defining the workload transaction mix:

- *Transactional* vs. *non-transactional* messages.
- *Persistent* vs. *non-persistent* messages.

[1]For an introduction to MOM and JMS see Section 2.2.4

- Usage of *different message types*, e.g. TextMessages, ObjectMessages, StreamMessages or MapMessages.
- Usage of *messages of different sizes* (small, medium, large).
- *Publish/subscribe* vs. *point-to-point messages* (queues vs. topics).
- One-to-one vs. one-to-many vs. many-to-many interactions.
- *Durable* vs. *non-durable* subscriptions.
- Ratio of message producers over message consumers.

In the case of SPECjms2007 the definition of the subset of JMS functionality and weighting of the different functions was done with respect to the observations in practice of different vendors.

Focus

The workload should be focused on measuring the performance and scalability of the JMS server's software and hardware components. It should minimize the impact of other components and services that are typically used in the chosen application scenario. For example, if a database would be used to store business data and manage the application state, it could easily become the limiting factor of the benchmark, as experience with previous benchmarks shows [126]. This is especially true in the case of SPECjms2007, since JMS servers, in their role as mediators in interactions, are typically less loaded than database or application servers. Another potential concern is the client side of the benchmark where messages are sent and received. The impact of client-side operations, such as XML parsing, on the overall performance of the benchmark should be minimized.

Configurability

As mentioned earlier, in addition to providing standard workload and metrics for JMS performance, SPECjms2007 aims to provide a flexible performance analysis framework which allows users to configure and customize the workload according to their requirements. Producing and publishing standard results for marketing purposes will be just one usage scenario for SPECjms2007. Many users will be interested in using the benchmark to tune and optimize their platforms or to analyze the performance of certain specific MOM features. Others could use the benchmark for research purposes in academic environments where, for example, one might be interested in evaluating the performance and scalability of novel methods and techniques for building high-performance MOM servers. All these usage scenarios require that the benchmark framework allows the user to precisely configure the workload and transaction mix to be generated. Providing this configurability is a challenge because it requires that interactions are designed and implemented in such a way that one could run them in different combinations depending on the desired transaction mix. The ability to switch interactions off implies that interactions should be decoupled from one another. On the other hand, it should be ensured that the benchmark, when run in its standard mode, behaves as if the interactions were interrelated according to their dependencies in the real-life application scenario.

Scalability

The SPECjms2007 application scenario must provide a way to scale the workload along the following two dimensions:

1. *Horizontal* scaling
2. *Vertical* scaling

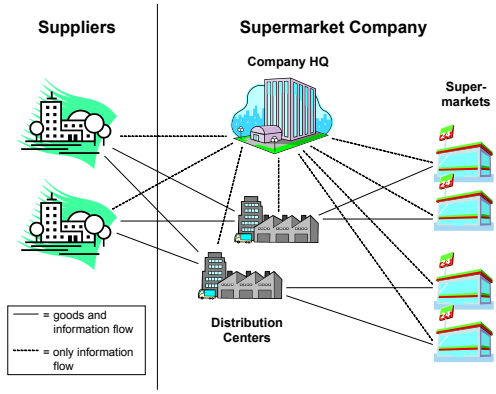


Figure 5.1: Overview of the Workload Scenario and its Roles

In horizontal scaling, the workload is scaled by increasing the number of destinations (queues and topics) while keeping the traffic per destination constant. In vertical scaling, the traffic (in terms of message count) pushed through a destination is increased while keeping the number of destinations fixed. Both types of scaling should be supported in a manner that preserves the relation to the real-life business scenario modeled. In addition, the user should be offered the possibility to scale the workload in an arbitrary manner by defining an own set of scaling points.

5.1.2 Workload Scenario

The workload scenario chosen for SPECjms2007 models the supply chain of a supermarket company. The participants involved are the supermarket company, its stores, its distribution centers and its suppliers. The scenario offers an excellent basis for defining interactions that stress different subsets of the functionality offered by MOM servers. Moreover, it offers a natural way to scale the workload. The participants involved in the scenario can be grouped into the following four roles:

1. *Company Headquarters*
2. *Distribution Centers*
3. *Supermarkets*
4. *Suppliers*

The first three roles are owned by the supermarket company and therefore all communication among them is intra-company. The suppliers are external companies and therefore their communication with the roles of the supermarket company is inter-company. The interactions among the different roles are illustrated in Figure 5.1.

Company Headquarters (HQ)

The company's corporate headquarters are responsible for managing the accounting of the company, managing information about the goods and products offered in the supermarket stores, managing selling prices and monitoring the flow of goods and money in the supply chain.

Distribution Centers (DCs)

The distribution centers supply the supermarket stores. Every distribution center is responsible for a set of stores in a given area. The distribution centers in turn are supplied by external

suppliers. The distribution centers are involved in the following activities: taking orders from supermarkets, ordering goods from suppliers, delivering goods to supermarkets and providing sales statistics to the HQ (e.g. for data mining).

Supermarkets (SMs)

The supermarkets sell goods to end customers. The scenario focuses on the management of the inventory of supermarkets including their warehouses. Some supermarkets are smaller than others, so that they do not have enough room for all products, others may be specialized for some product groups like certain types of food. We assume that every supermarket is supplied by exactly one of the distribution centers.

Suppliers (SPs)

The suppliers deliver goods to the distribution centers of the supermarket company. Different suppliers are specialized for different sets of products and they deliver goods on demand, i.e. they must receive an order from the supermarket company to send a shipment. Not every supplier offers the same products. Instead, the suppliers have their own product catalogues. They deliver goods on demand, i.e. they must receive an order from the supermarket company to send a shipment. As a simplification, it is assumed that each SP offers either all products of a given product family or none of them.

5.1.3 Modeled Interactions

The following seven interactions between the participants in the supermarket supply chain are modeled in SPECjms2007:

1. *Order / Shipment Handling* between SM and its assigned DC
2. *(Purchase) Order / Shipment Handling* between a DC and the SPs
3. *Price Updates*
4. *Inventory Management*
5. *Sales Statistics Collection*
6. *Product Announcements*
7. *Credit Card Hotlists*

Inter-company communication, i.e. communication between the suppliers and the supermarket company, is implemented using TextMessages containing XML documents. For intra-company communication (between supermarkets, distribution centers and the company headquarters) the whole set of possible message types supported by JMS is used. For a better understanding of the data flow we provide detailed message schemas of the exchanged messages in Appendix **??**.

Note: In the following, unless otherwise noted, all messages exchanged are assumed to be persistent and transactional.

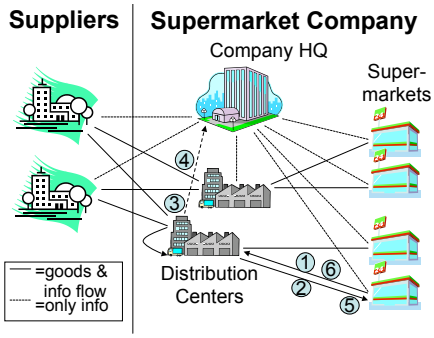


Figure 5.2: Interaction 1 - Communication between SM and DC

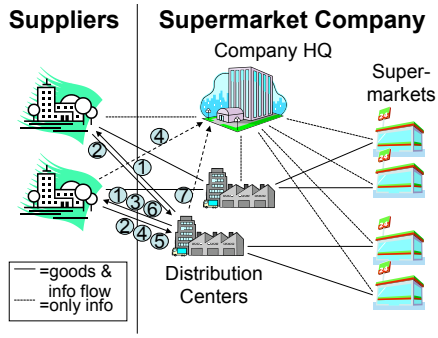


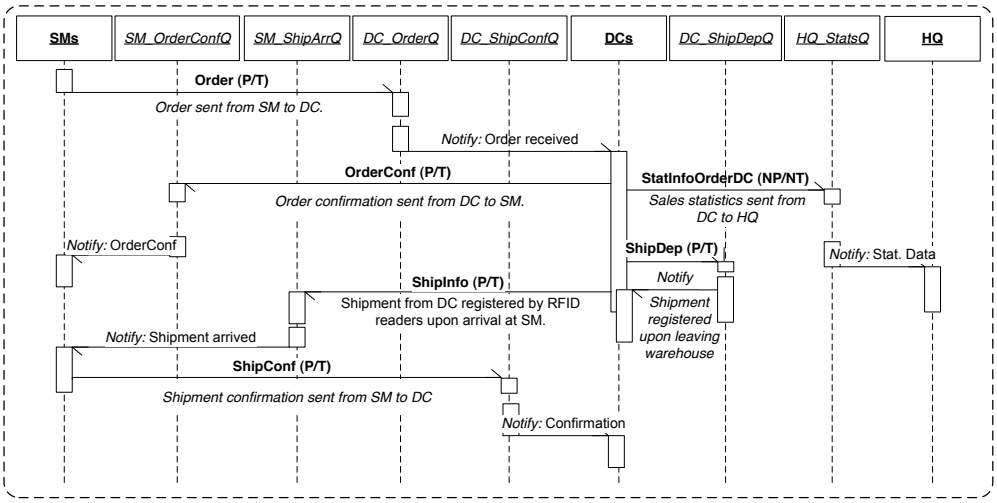
Figure 5.3: Interaction 2 - Communication between SP and DC

Interaction 1: Order/Shipment Handling between SM and DC This interaction exercises persistent P2P messaging between the SMs and DCs. The interaction is triggered when goods in the warehouse of a SM are depleted and the SM has to order from its DC to refill stock. The following steps are followed as illustrated in Figures 5.2 and 5.4(a) :

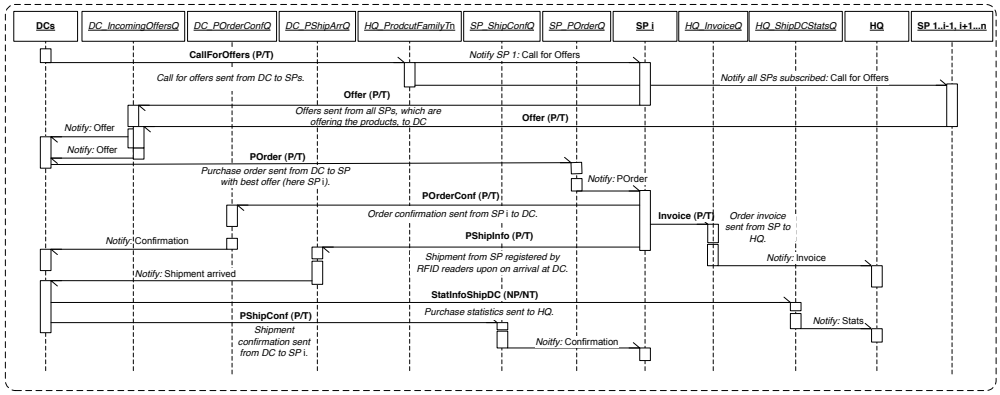
1. A SM sends an order to its DC.
2. The DC sends a confirmation to the SM and ships the ordered goods.
3. Goods are registered by RFID readers upon leaving the DC warehouse.
4. The DC sends information about the transaction to the HQ (sales statistics).
5. The shipment arrives at the SM and is registered by RFID readers upon entering the SM warehouse.
6. A confirmation is sent to the DC.

Interaction 2: Order/Shipment Handling between DC and SP This interaction exercises persistent P2P and pub/sub (durable) messaging between the DCs and SPs. The interaction is triggered when goods in a DC are depleted and the DC has to order from a SP to refill stock. The following steps are followed as illustrated in Figures 5.3 and 5.4(b) :

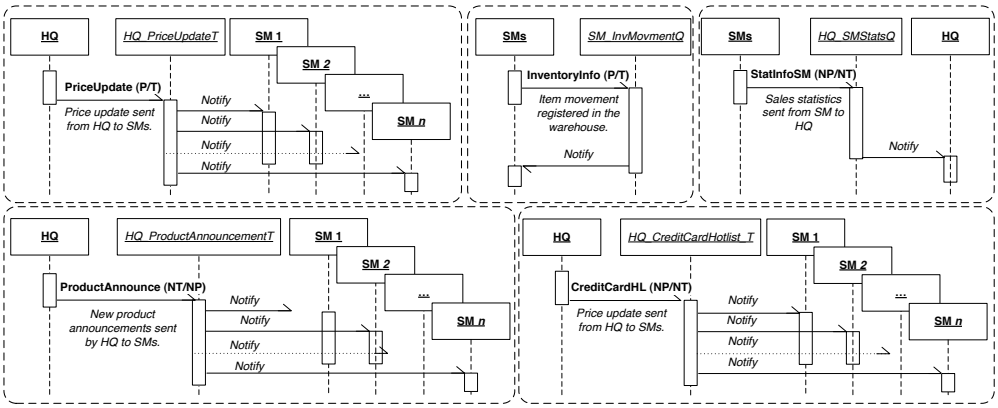
1. A DC sends a call for offers to all SPs that supply the types of goods that need to be ordered.
2. SPs that can deliver the goods send offers to the DC.
3. Based on the offers, the DC selects a SP and sends a purchase order to it.
4. The SP sends a confirmation to the DC and an invoice to the HQ. It then ships the ordered goods.
5. The shipment arrives at the DC and is registered by RFID readers upon entering the DC's warehouse.
6. The DC sends a delivery confirmation to the SP.
7. The DC sends transaction statistics to the HQ.



(a) Interaction 1



(b) Interaction 2



(c) Interactions 3 to 7

Figure 5.4: Workflow of the SPECjms2007 Interactions
(N)P=(Non-)Persistent; (N)T= (Non-)Transactional

Interaction 3: Price Updates This interaction exercises persistent, durable pub/sub messaging between the HQ and the SMs. The interaction is triggered when selling prices are changed by the company administration. To communicate this, the company HQ sends messages with pricing information to the SMs. The communication here is one-to-many and is based on pub/sub messaging.

SMs subscribe to all messages related to products they sell.

1. HQ sends a price update to SMs.
2. Affected SMs update their information systems.

Interaction 4: SM Inventory Management This interaction exercises persistent P2P messaging inside the SMs. The interaction is triggered when goods leave the warehouse of a SM (to refill a shelf). Goods are registered by RFID readers and the local warehouse application is notified so that inventory can be updated.[2]

1. As goods leave a SM's warehouse, they get registered by RFID-readers.
2. RFID-readers send observations to the local warehouse application.
3. The local warehouse inventory is updated.

Interaction 5: Sales Statistics Collection This interaction exercises non-persistent P2P messaging between the SMs and the HQ. The interaction is triggered when a SM sends sales statistics to the HQ. HQ can use this data as a basis for data mining in order to study customer behavior and provide useful information to marketing. For example, based on such information, special offers or product discounts could be made.

1. SM sends a non-transactional, non-persistent message to HQ containing sales statistics.
2. HQ update their data warehouse (OLAP).

Interaction 6: New Product Announcements This interaction exercises non-persistent, non-durable pub/sub messaging between the HQ and the SMs. The interaction is triggered when new products are announced by the company administration. To communicate this, the HQ sends messages with product information to the SMs selling the respective product types.

SMs subscribe to announcement messages related to the product classes they sell.

1. HQ sends a new product announcement to SMs.
2. Subscribed SMs update their information systems.

Interaction 7: Credit Card Hot Lists This interaction exercises non-persistent, non-durable pub/sub messaging between the HQ and the SMs. The interaction is triggered when the HQ sends credit card hot lists to the SMs (complete list once every hour and incremental updates as required).

1. HQ sends a credit card hot list to SMs.
2. Subscribed SMs receive the list and store it locally.

Interaction 3 to 7 are illustrated in 5.4(c).

[2] *Note:* Since incoming goods are part of another interaction (Interaction 1), they are not considered here.

5.1.4 SPECjms2007 Workload Characterization

In this section we discuss different design issues and decisions regarding the workload. Amongst others the focus of this section is on scaling of the benchmark, the message sizes and the transaction mixes of the workload. Especially the first is a very important issue for us. A workload with scalability limitations is somehow useless and does not provide a proper base for a benchmark. Therefore we put our main focus on this aspect and its validation. From our point of view, in a scalable benchmark, the relative proportions of different aspects of the workload (like ratio between Publish / Subscribe and P2P communication, proportion of the different interactions, etc.) should be independent from the scaling factor. This is needed to allow comparability of benchmarks results.

Message Traffic Analysis

We start with a detailed analysis of the message traffic produced by the benchmark workload in terms of the number and type of messages generated and their sizes. We consider the workload parameters that can be configured in the most general freeform topology and show how they affect the resulting message traffic. The different types of messages and destinations used in the various interactions are detailed in Table 5.1.

Messages Sizes The sizes of the messages generated as part of each interaction can be configured by setting an interaction-specific message sizing parameter (for example, "number of order lines sent to DC" for Interaction 1). Each sizing parameter can be assigned three possible values with respective probabilities (discrete probability distribution). The message sizing parameters used for the different interactions are listed in Table 5.2, along with some data that can be used to compute the resulting message sizes in KBytes. This data is based on measurements we took using a deployment of SPECjms2007 on a major JMS server platform[3]. The exact message sizes may be slightly different on different platforms, as MOM servers add their own platform-specific message headers. The measurements provided here were compared against measurements on a second popular JMS server and the differences were negligible. Based on the data in Table 5.2, the message sizes in KBytes for Interactions 1, 2, 4, 6 and 7 can be computed as $\vartheta = m_1 \cdot x + b$ where x is the interaction's message sizing parameter and m_1 and b are set to their respective values from Table 5.2. The `priceUpdate` messages of Interaction 3 have constant size that cannot be changed by the user. The size of the `statInfoSM` messages used in Interaction 5 is configured using two sizing parameters as follows $\vartheta = x \cdot (m_1 + m_2 \cdot y) + b$ where x and y are the two sizing parameters (i.e. "number of SM cash desks" and "number of sales lines") and m_1, m_2 and b are set to their respective values from Table 5.2. Based on the above two formulas and the data in Table 5.2, the user can configure the benchmark to use message sizes that match the user's own target workload.

Message Throughput We now characterize the message throughput first on a per interaction basis and then on a per location basis. The two most important sets of workload parameters that determine the message throughput are the number of locations of each type and the interaction rates. We denote the sets of physical locations as follows:

$$\begin{aligned}
\Psi_{SM} &= \{SM_1, SM_2, SM_3, \ldots, SM_{|\Psi_{SM}|}\} \\
\Psi_{DC} &= \{DC_1, DC_2, DC_3, \ldots, DC_{|\Psi_{DC}|}\} \\
\Psi_{SP} &= \{SP_1, SP_2, SP_3, \ldots, SP_{|\Psi_{SP}|}\} \\
\Psi_{HQ} &= \{HQ_1, HQ_2, HQ_3, \ldots, HQ_{|\Psi_{HQ}|}\}
\end{aligned}$$

[3]Due to product license restrictions, the specific configuration used cannot be disclosed.

Intr.	Message	Destination	Type	Prop.	Description
1	order	Queue (DC)	ObjectMsg	P, T	Order sent from SM to DC.
	orderConf	Queue (SM)	ObjectMsg	P, T	Order confirmation sent from DC to SM.
	shipDep	Queue (DC)	TextMsg	P, T	Shipment registered by RFID readers upon leaving DC.
	statInfo-OrderDC	Queue (HQ)	StreamMsg	NP, NT	Sales statistics sent from DC to HQ.
	shipInfo	Queue (SM)	TextMsg	P, T	Shipment from DC registered by RFID readers upon arrival at SM.
	shipConf	Queue (DC)	ObjectMsg	P, T	Shipment confirmation sent from SM to DC.
2	callForOffers	Topic (HQ)	TextMsg	P, T, D	Call for offers sent from DC to SPs (XML).
	offer	Queue (DC)	TextMsg	P, T	Offer sent from SP to DC (XML).
	pOrder	Queue (SP)	TextMsg	P, T	Order sent from DC to SP (XML).
	pOrderConf	Queue (DC)	TextMsg	P, T	Order confirmation sent from SP to DC (XML).
	invoice	Queue (HQ)	TextMsg	P, T	Order invoice sent from SP to HQ (XML).
	pShipInfo	Queue (DC)	TextMsg	P, T	Shipment from SP registered by RFID readers upon arrival at DC.
	pShipConf	Queue (SP)	TextMsg	P, T	Shipment confirmation sent from DC to SP (XML).
	statInfo-ShipDC	Queue (HQ)	StreamMsg	NP, NT	Purchase statistics sent from DC to HQ.
3	priceUpdate	Topic (HQ)	MapMsg	P, T, D	Price update sent from HQ to SMs.
4	inventoryInfo	Queue (SM)	TextMsg	P, T	Item movement registered by RFID readers in the warehouse of SM.
5	statInfoSM	Queue (HQ)	ObjectMsg	NP, NT	Sales statistics sent from SM to HQ.
6	product-Announcement	Topic (HQ)	StreamMsg	NP, NT, ND	New product announcements sent from HQ to SMs.
7	creditCardHL	Topic (HQ)	StreamMsg	NP, NT, ND	Credit card hotlist sent from HQ to SMs.

Table 5.1: Message Types Used in The Interactions - (N)P=(Non-)Persistent; (N)T=(Non-)Transactional; (N)D=(Non-)Durable

Intr.	Message Sizing Parameters	Message	m_1	m_2	b
1	No of order lines sent to DC	orderConf	0.0565	*na*	1.7374
		statInfoOrderDC	0.0153	*na*	0.1463
		shipInfo	0.0787	*na*	0.8912
		shipDep	0.0787	*na*	0.7222
		order	0.0565	*na*	1.4534
		shipConf	0.0202	*na*	0.7140
2	No of purchase order lines sent to SP	callForOffers	0.1785	*na*	0.8094
		offer	0.2489	*na*	0.9414
		pOrder	0.2498	*na*	1.1076
		pShipConf	0.0827	*na*	0.7612
		statInfoShipDC	0.0831	*na*	0.7681
		pOrderConf	0.2410	*na*	1.3494
		invoice	0.1942	*na*	1.1211
		pShipInfo	0.0827	*na*	0.7279
3	*Message has fixed size*	priceUpdate	*na*	*na*	0.2310
4	No of registered items leaving warehouse	inventoryInfo	0.0970	*na*	0.5137
5	No of cash desks & sales lines	statInfoSM	0.0139	0.3650	0.9813
6	No of new products announced	productAnnouncement	0.0103	*na*	0.1754
7	No of credit cards in hot list	creditCardHL	0.0166	*na*	0.1846

Table 5.2: Parameters for Message Size Calculation

Group	a	b	c	d
Type	Pub/Sub	Pub/Sub	P2P	P2P
Properties	NP NT ND	P T D	NP NT	P T

Table 5.3: Message Groups

Note that although the modeled scenario has a single physical HQ location, the benchmark allows multiple HQ instances to exist each with its own set of queues. The goal is to avoid the HQ queues becoming a bottleneck when scaling the number of SMs, DCs and SPs. It is assumed that messages sent to the HQ are distributed evenly among the HQ instances. Multiple HQ instances are considered as separate servers within the same physical location.

For each interaction, the *interaction rate* specifies the rate at which the interaction is initiated by every physical instance of its initiating location, SM for Interaction 1, DC for Interaction 2, etc. We denote the interaction rates as $\lambda_i, 1 \leq i \leq 7$. Since multiple HQ instances are not considered as separate physical locations, it follows that the rates of Interactions 3, 6 and 7 which are initiated by the HQ are interpreted as rates over all HQ instances as opposed to rates per HQ instance. Interaction 2 uses a set of topics representing the different product families offered by suppliers. These topics help to distribute the `callForOffers` messages sent by DCs. Suppliers subscribe to all topics corresponding to groups of products they offer so that they receive all relevant `callForOffers` messages. We denote the set of product families as $\Pi = \{PF_1, PF_2, PF_3, \ldots, PF_{|\Pi|}\}$.

The probability that a SP offers products from a given product family $PF_i \in \Pi$ is a configurable workload parameter and will be denoted as ρ. Every SP subscribes to $\rho \cdot |\Pi|$ product families and thus $|\Psi_{SP}| \cdot \rho \cdot |\Pi|$ subscriptions exist overall. The number of subscribers that subscribe to a given product family is denoted as $\zeta = |\Psi_{SP}| \cdot \rho$.

In the following, we show how the message throughput, in terms of the number of messages sent and received per unit of time, can be broken down according to the type of messaging (P2P vs. pub/sub) and the message delivery mode (persistent vs. non-persistent, transactional vs. non-transactional, durable vs. non-durable). To this end, we group messages as shown in

Table 5.3. Further, we define the following sets:

$$
\begin{aligned}
\Gamma &= \{a, b, c, d\} \\
&\quad \text{Message groups as defined in Table 5.3.} \\
\Omega &= \{se, re\} \\
&\quad \text{Messages sent vs. messages received.} \\
\Lambda &= \{SM, SP, DC, HQ\} \\
&\quad \text{Types of physical locations.}
\end{aligned}
$$

Message Throughput per Interaction We first analyze the message throughput on a per interaction basis. We will use the following notation:

$\xi^j_{i,k}$ for $j \in \Omega, 1 \leq i \leq 7$ and $k \in \Gamma$
No of messages of group k sent/received per sec as part of Interaction i.

$\xi^j_i = \sum_{k \in \Gamma} \xi^j_{i,k}$ for $1 \leq i \leq 7, j \in \Omega$
Total no of messages sent/received per sec as part of Interaction i.

$\xi^j = \sum_{i=1}^{7} \xi^j_i$ for $j \in \Omega$
Total no of messages sent/received per sec over all interactions.

Based on the information provided in the previous sections and analysis of the benchmark design, the following equations are derived characterizing the message throughput of each interaction:

Interaction 1:

$$
\begin{aligned}
\xi^{se}_{1,c} &= \xi^{re}_{1,c} = \lambda_1 \cdot |\Psi_{SM}| \\
\xi^{se}_{1,d} &= \xi^{re}_{1,d} = 5 \cdot \lambda_1 \cdot |\Psi_{SM}| \\
\xi^{j}_{1,k} &= 0, \ \ \forall k \in \{a, b\} \wedge j \in \Omega
\end{aligned}
$$

Interaction 2:

$$
\begin{aligned}
\xi^{j}_{2,a} &= 0, \ \forall j \in \Omega \\
\xi^{se}_{2,b} &= \lambda_2 \cdot |\Psi_{DC}| \\
\xi^{re}_{2,b} &= \zeta \cdot \lambda_2 \cdot |\Psi_{DC}| \\
\xi^{se}_{2,c} &= \xi^{re}_{2,c} = \lambda_2 \cdot |\Psi_{DC}| \\
\xi^{se}_{2,d} &= \xi^{re}_{2,d} = (\zeta + 5) \cdot \lambda_2 \cdot |\Psi_{DC}|
\end{aligned}
$$

Interaction 3:

$$
\begin{aligned}
\xi^{se}_{3,b} &= \lambda_3 \\
\xi^{re}_{3,b} &= \lambda_3 \cdot |\Psi_{SM}| \\
\xi^{j}_{3,k} &= 0, \ \ \forall k \in \Gamma, k \neq b \wedge j \in \Omega
\end{aligned}
$$

Interaction 4:

$$
\begin{aligned}
\xi^{se}_{4,d} &= \xi^{re}_{4,d} = \lambda_4 \cdot |\Psi_{SM}| \\
\xi^{j}_{4,k} &= 0, \ \ \forall k \in \Gamma, k \neq d \wedge j \in \Omega
\end{aligned}
$$

Interaction 5:

$$
\begin{aligned}
\xi^{se}_{5,d} &= \xi^{re}_{5,d} = \lambda_5 \cdot |\Psi_{SM}| \\
\xi^{j}_{5,k} &= 0, \ \ \forall k \in \Gamma, k \neq d \wedge j \in \Omega
\end{aligned}
$$

Interaction 6:

$$\begin{aligned}\xi^{se}_{6,a} &= \lambda_6\\ \xi^{re}_{6,a} &= \lambda_6 \cdot |\Psi_{SM}|\\ \xi^{j}_{6,k} &= 0,\ \ \forall k \in \Gamma, k \neq a \wedge j \in \Omega\end{aligned}$$

Interaction 7:

$$\begin{aligned}\xi^{se}_{7,a} &= \lambda_7\\ \xi^{re}_{7,a} &= \lambda_7 \cdot |\Psi_{SM}|\\ \xi^{j}_{7,k} &= 0,\ \ \forall k \in \Gamma, k \neq a \wedge j \in \Omega\end{aligned}$$

Message Throughput per Location We now analyze the message throughput on a per location basis. The following notation will be used:

$\chi^{j}_{l,k}$ for $j \in \Omega, l \in \Lambda, k \in \Gamma$
No of messages of group k sent/received per sec by a location of type l.

$\chi^{j}_{l} = \sum_{k \in \Gamma} \xi^{j}_{l,k}$ for $j \in \Omega, l \in \Lambda$
Total no of messages sent/received per sec by a location of type l.

SMs participate in all interactions apart from Interaction 2. The following equations characterize the message throughput of each SM:

$$\begin{aligned}\chi^{se}_{SM,a} &= \chi^{se}_{SM,b} = \chi^{re}_{SM,c} = 0\\ \chi^{re}_{SM,a} &= \lambda_6 + \lambda_7\\ \chi^{re}_{SM,b} &= \lambda_3\\ \chi^{se}_{SM,c} &= \lambda_5\\ \chi^{se}_{SM,d} &= 2\lambda_1 + \lambda_4\\ \chi^{re}_{SM,d} &= 2\lambda_1 + \lambda_4\end{aligned}$$

SPs participate only in Interaction 2. Overall $\lambda_2 \cdot |\Psi_{DC}|$ `callForOffers` messages are sent by the DCs per sec. Therefore, every SP receives $\rho \cdot \lambda_2 \cdot |\Psi_{DC}|$ messages and for each of them it sends an offer to the respective DC. The probability that an offer is accepted is $\frac{1}{\zeta}$ and hence the number of SP offers accepted per sec is given by:

$$\frac{\rho \cdot \lambda_2 \cdot |\Psi_{DC}|}{\zeta} = \frac{\lambda_2 \cdot |\Psi_{DC}|}{|\Psi_{SP}|}$$

The following equations characterize the message throughput of each SP:

$$\begin{aligned}\chi^{se}_{SP,a} &= \chi^{re}_{SP,a} = \chi^{se}_{SP,b} = \chi^{se}_{SP,c} = \chi^{re}_{SP,c} = 0\\ \chi^{re}_{SP,b} &= \rho \cdot \lambda_2 \cdot |\Psi_{DC}|\\ \chi^{se}_{SP,d} &= \rho \cdot \lambda_2 \cdot |\Psi_{DC}| + \frac{3\lambda_2 \cdot |\Psi_{DC}|}{|\Psi_{SP}|}\\ \chi^{re}_{SP,d} &= \frac{2\lambda_2 \cdot |\Psi_{DC}|}{|\Psi_{SP}|}\end{aligned}$$

DCs participate in Interactions 1 and 2 both as producers and consumers of messages. The number of SMs supplied by each DC is given by $\delta = \frac{|\Psi_{SM}|}{|\Psi_{DC}|}$.

The following equations characterize the message throughput of each DC:

$$\begin{aligned}
\chi^{se}_{DC,a} &= \chi^{re}_{DC,a} = \chi^{re}_{DC,b} = \chi^{re}_{DC,c} = 0 \\
\chi^{se}_{DC,b} &= \lambda_2 \\
\chi^{se}_{DC,c} &= \delta \cdot \lambda_1 + \lambda_2 \\
\chi^{se}_{DC,d} &= 3\lambda_1 \cdot \delta + 2\lambda_2 \\
\chi^{re}_{DC,d} &= 3\lambda_1 \cdot \delta + \lambda_2(\zeta + 2)
\end{aligned}$$

The HQ participate in Interactions 1, 2, and 5 as message consumer and in Interactions 3, 6, and 7 as message producer. The following equations characterize the message throughput of the HQ:

$$\begin{aligned}
\chi^{re}_{HQ,a} &= \chi^{re}_{HQ,b} = \chi^{se}_{HQ,c} = \chi^{se}_{HQ,d} = 0 \\
\chi^{se}_{HQ,a} &= \lambda_6 + \lambda_7 \\
\chi^{se}_{HQ,b} &= \lambda_3 \\
\chi^{re}_{HQ,c} &= \lambda_1 \cdot |\Psi_{SM}| + \lambda_2 \cdot |\Psi_{DC}| + \lambda_5 \cdot |\Psi_{SM}| \\
\chi^{re}_{HQ,d} &= \lambda_2 \cdot |\Psi_{DC}|
\end{aligned}$$

The detailed message throughput analysis presented above serves two main purposes. First, using the throughput equations, the user can assemble a workload configuration (in terms of number of locations and interaction rates) that stresses specific types of messaging under given scaling conditions. As a very basic example, the user might be interested in evaluating the performance and scalability of non-persistent pub/sub messaging under increasing number of subscribers. In this case, a mix of Interactions 6 and 7 can be used with increasing number of SMs. Second, the characterization of the message traffic on a per location basis can help users to find optimal deployment topology of the agents representing the different locations such that the load is evenly distributed among client nodes and there are no client-side bottlenecks. This is especially important for a messaging benchmark where the server acts as mediator in interactions and a significant amount of processing is executed on the client side.

Horizontal Topology

As mentioned earlier, the goal of the horizontal topology is to exercise the ability of the system to handle increasing number of destinations. To achieve this, the workload is scaled by increasing the number of physical locations (SMs, DCs, etc) while keeping the traffic per location constant. A scaling parameter `BASE` is introduced and the following rules are enforced:

1. $|\Psi_{SM}| = \texttt{BASE}$
2. $|\Psi_{DC}| = \lceil \frac{|\Psi_{SM}|}{5} \rceil$
3. $|\Psi_{SP}| = [0.4 \cdot |\Psi_{SM}|]$
4. $|\Psi_{HQ}| = \lceil \frac{|\Psi_{SM}|}{10} \rceil$
5. $|\Pi| = |\Psi_{SM}|$
6. $\rho = \frac{5}{|\Pi|}$
7. λ_i, $1 \leq i \leq 7$ are set as shown on Table 5.4

Figure 5.5 shows how the number of locations of each type is scaled as the `BASE` parameter is increased. The rates λ_i at which interactions are initiated by participants are fixed so that the

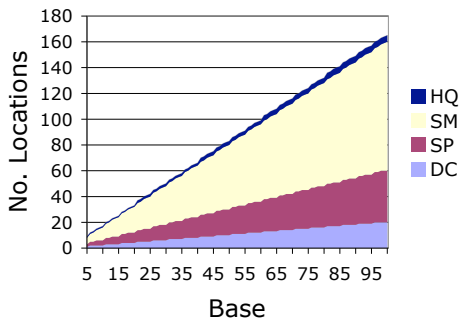


Figure 5.5: # Locations for Horiz. Topology

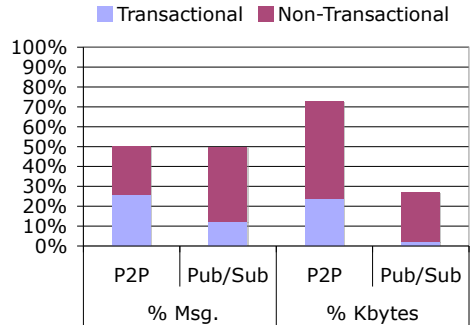


Figure 5.6: Horiz. Topology Message Mix

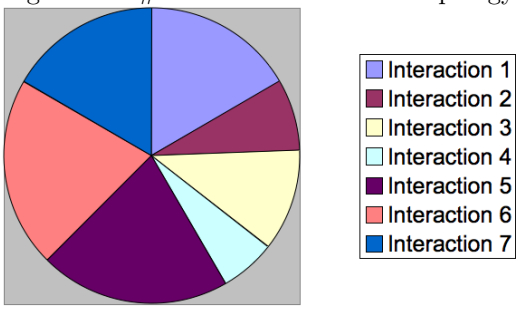


Figure 5.7: Proportions of the Interactions based on Msg. Throughput

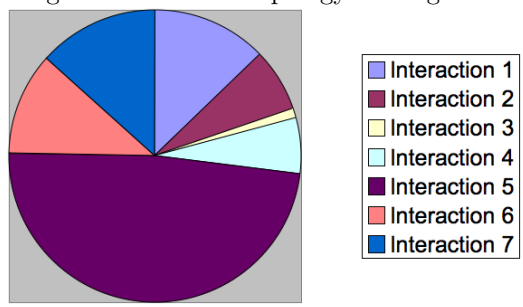


Figure 5.8: Proportions of the Interactions based on Msg. Traffic in KBytes

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7
1.53920154	2.13333333	6.00000000	3.37837837	11.54401154	11.38519924	9.23076923

Table 5.4: Interaction Rates for the Horizontal Topology

traffic per location (and therefore also per destination) remains constant. In the Figures 5.7 and 5.8 the relative weights of the interactions is illustrated. They are set based on a detailed business model of the supermarket supply chain which captures the interaction interdependencies. This model has several input parameters (e.g. total number of product types, size of supermarkets, average number of items sold per week) whose values are chosen in such a way that the following overall target messaging mix is achieved as close as possible:

- 50% P2P messages and 50% pub/sub
- 50% of P2P messages persistent, 50% non-persistent
- 25% of pub/sub messages persistent, 75% non-persistent

The goal is to put equal weight on P2P and pub/sub messaging. Within each group the target relative weights of persistent vs. non-persistent messaging have been set according to the relative usage of these messaging styles in real-life applications. The criteria for what is a typical MOM application were defined based on input provided by the various participating vendors in the SPECjms working group including IBM, Sun, Oracle, BEA, Sybase and Apache. A comprehensive survey was conducted considering real-life customer applications and analyzing their workloads.

Table 5.5(a) shows the resulting message mix in the horizontal topology. Figure 5.6 presents the same data in graphical form. Figures 5.9 and 5.10 show how the number of messages of each

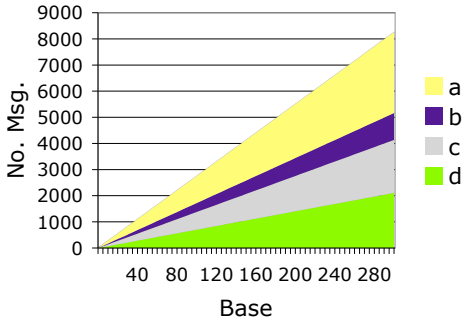


Figure 5.9: Horizontal Topology: # msg. sent

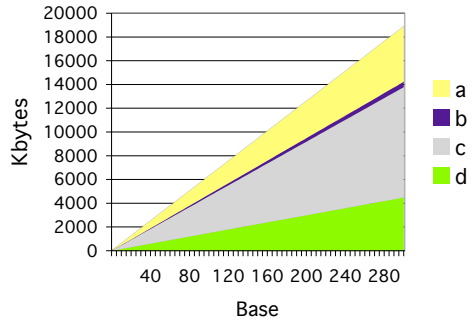


Figure 5.10: Message Traffic in Kbytes

type and the bandwidth they use are scaled as a function of the `BASE` parameter. As evident from the figure, when scaling the workload the proportions of the different types of messages remain constant. This is expected since the relative weights of the various messaging styles used by the workload should not depend on the scaling factor.

The sizes of the messages used in the various interactions have been chosen to reflect typical message sizes in real-life MOM applications. Pub/sub messages are generally much smaller than P2P messages due to the decoupled nature of the delivery mechanism. For every type of message, SPECjms2007 generates messages with sizes chosen from a discrete distribution with three possible values as shown in Table 5.5. There are two exceptions, the `priceUpdate` message used in Interaction 4 and the `statInfoSM` message used in Interaction 5. The former has a fixed size, while the latter has size between 4.7 and 24.78 KBytes with an average of 5.27 KBytes. Since `statInfoSM` messages contain sales statistics, their size is determined by the rate at which items are sold in supermarkets which depends on the number of customers visiting a supermarket per day and the average number of items sold per customer.

P2P transactional messages tend to be medium in size and P2P non-transactional range from very small ($\leq$ 1KBytes) to big (50 KBytes). Large messages are very rarely used in practice.

We decided to model three different messages sizes per message type based on a certain probability:

- *Small messages:* up to 2 KBytes, probability 95 %
- *Medium messages:* up to 10 KBytes, probability of 4 %
- *Large messages:* up to 55 KBytes, probability of 1 %

Vertical Topology

The goal of the vertical topology is 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 executed. Similar to the horizontal case, a single parameter `BASE` is used as a scaling factor. The following rules are enforced:

1. $|\Psi_{SM}| = 10$
2. $|\Psi_{DC}| = 2$
3. $|\Psi_{SP}| = 5$

Intr.	Message *Probability*	Size 1 *95 %*	Size 2 *4 %*	Size 3 *1 %*	Avg. Size
1	orderConf	2.02	7.39	41.29	2.63
	statInfoOrderDC	0.22	1.67	10.83	0.39
	shipInfo	1.28	8.76	55.95	2.13
	shipDep	1.12	8.59	55.79	1.96
	order	1.74	7.10	41.01	2.34
	shipConf	0.81	2.73	14.83	1.03
2	callForOffers	1.35	7.06	36.52	1.93
	offer	1.69	9.65	50.71	2.50
	pOrder	1.86	9.85	51.07	2.67
	pShipConf	1.01	3.65	17.29	1.28
	statInfoShipDC	1.02	3.68	17.38	1.29
	pOrderConf	2.07	9.79	49.56	2.86
	invoice	1.70	7.92	39.95	2.33
	pShipInfo	0.98	3.62	17.26	1.24
3	priceUpdate	0.24	0.24	0.24	0.24
4	inventoryInfo	1.48	10.22	49.03	2.31
5	statInfoSM	*na*			5.27
6	productAnnouncement	1.21	0.28	10.51	1.26
7	creditCardHL	1.01	8.49	50.00	1.80

Table 5.5: Message Sizes in KByte

(a) Horizontal

Message Group	Message Count Target	Achieved	Bandwidth Used
a	37.50%	37.46%	24.66%
b	12.50%	12.45%	2.41%
c	25.00%	24.55%	49.19%
d	25.00%	25.55%	23.74%

(b) Vertical

Message Group	Message Count Target	Achieved	Bandwidth Used
a	15.00%	14.19%	7.19%
b	5.00%	5.99%	2.25%
c	40.00%	39.09%	61.03%
d	40.00%	40.74%	29.52%

Table 5.6: Topology Message Mix

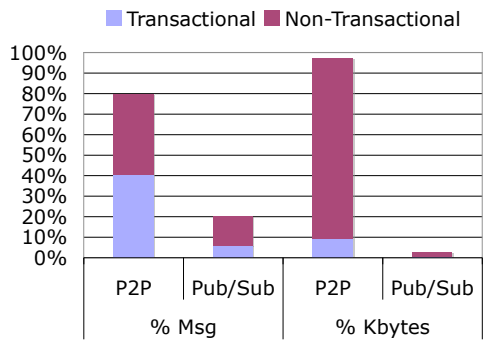


Figure 5.11: Vert. Topology Message Mix

c_1	c_2	c_3	c_4	c_5	c_6	c_7
0.076190476	0.106666667	0.050000000	0.162162162	0.577200577	0.142314991	0.102564103

Table 5.7: Interaction Rate Scaling Factors for the Vertical Topology

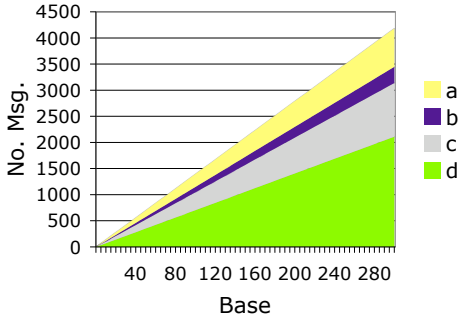


Figure 5.12: Vertical Topology: # msg. sent

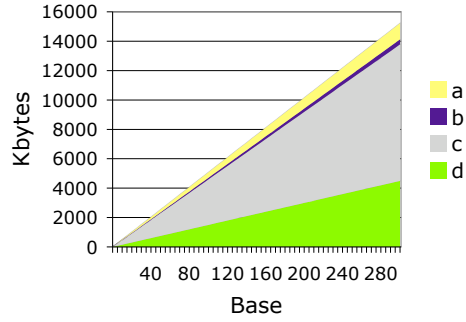


Figure 5.13: Message Traffic in Kbytes

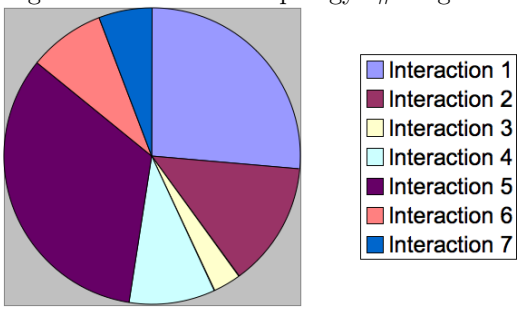


Figure 5.14: Proportions of the Interactions based on Msg. Throughput

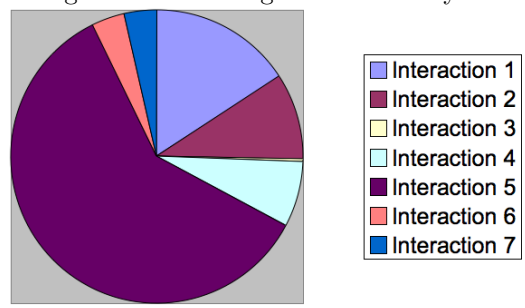


Figure 5.15: Proportions of the Interactions based on Msg. Traffic in KBytes

4. $|\Psi_{HQ}| = 2$
5. $|\Pi| = 100$
6. $\rho = 50\%$
7. $\lambda_i = c_i \cdot \texttt{BASE}$, where c_i is a fixed factor (see Table 5.7) and $1 \leq i \leq 7$

Again, the relative weights of the interactions are set based on the business model of the supply chain scenario (see Figures 5.14 and 5.15). Unlike the horizontal topology, however, the vertical topology places the emphasis on P2P messaging which accounts for 80% of the total message traffic. The aim is to exercise the ability of the system to handle increasing traffic through a destination by processing messages in parallel. This aspect of MOM server performance is more relevant for P2P messaging (queues) than for pub/sub messaging where the message throughput is inherently limited by the speed at which subscribers can process incoming messages.

Table 5.5(b) shows the achieved message mix in the vertical topology. Figure 5.11 presents the same data in graphical form. Figures 5.12 and 5.13 show how the number of messages of each type and the bandwidth they use are scaled as a function of the `BASE` parameter. Again, when scaling the workload the message mix remains constant which is the expected behavior. The sizes of the messages used in the various interactions are computed in the same way as for the horizontal topology (see Table 5.5).

5.1.5 Benchmark Implementation

Event Handlers and Agents

SPECjms2007 is implemented as a Java application comprising multiple JVMs and threads distributed across a set of *client nodes*. For every destination (queue or topic), 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 the queue/topic and receive call backs from the messaging infrastructure as new messages arrive. For maximal performance and scalability, multiple instances of each event handler executed in separate threads can exist and they can be distributed over multiple physical nodes. Event handlers can be grouped according to the physical location (e.g. HQ, SM, DC or SP) they pertain to in the business scenario. In addition to the event handlers, for every physical location, a set of threads is launched to drive the benchmark interactions that are logically started at that location. These are called *driver threads*. The set of all event handlers and driver threads pertaining to a given physical location is referred to as *agent*. For example, each DC agent is comprised of a set of event handlers for the various destinations inside the DC and a set of driver threads used to drive Interaction 2, which is the only interaction with logical starting point at DCs.

Driver Framework

The SPECjms2007 scenario includes many locations represented by many event handlers. In order to drive the JMS server to its capacity, event handlers may well be distributed across many physical machines. The reusable control framework designed for SPECjms2007 aims to coordinate these distributed activities without any inherent scalability limitations. Key design decisions were that

- It should be written as far as possible in plain Java. Since Java is the natural prerequisite of a JMS application this reduces installation and configuration requirements on end users.
- Further to the above, RMI is used as the basis for communication as this is part of the standard Java Standard Edition (Java SE) platform.
- The Controller needs not be on the same machine as any of the performance-critical workloads.
- Users should have maximum choice in how they wish to lay out their workload to achieve optimum performance (within the bounds of the SPECjms2007 run rules [214]).

Figure 5.16 provides a simplified view of a typical test being run on four nodes. In addition to the event handlers, it is made up of several simple components:

Controller The Controller component reads in all of the configuration and topological layout preferences given by the user. This will include items such as the number of different types of event handler and lists of the nodes across which they may be run. With this knowledge, the controller instantiates the topology. It begins this by connecting to a *satellite* process on each node machine identified as part of this test to give it specific instructions.

Satellite The Satellite is a simple part of the framework which knows to build the correct environment to start child Java processes for SPECjms2007. It takes the controller's configuration and starts the *agent* processes relevant to that node. Although each agent is logically discrete from its peers, the satellite will, based upon the initial configuration, combine many agents into a single JVM for reasons of scalability.

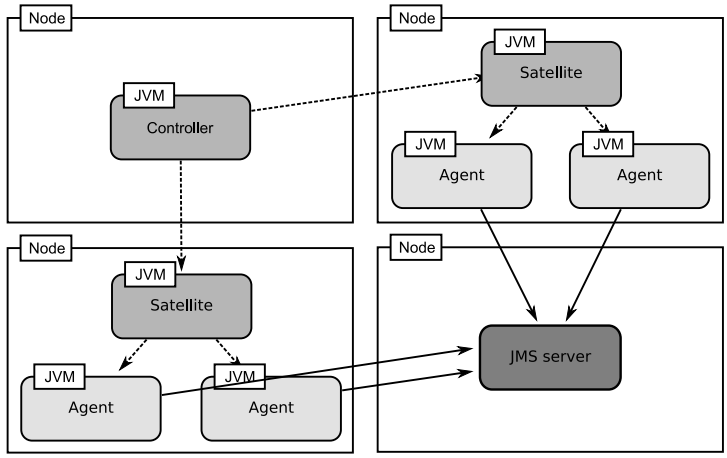


Figure 5.16: Driver Framework

Agents Each logical agent represents one of the locations in the application scenario. This means that, for example, a distribution center agent will contain a set of DC event handlers pertaining to that location. Agents connect back to the Controller who coordinates the stages of the test. Once all agents are connected, the event handlers (implemented as a Java thread each) start connecting to the JMS server and the warm-up phase of messaging begins. The controller manages the life cycle of the test by monitoring progress, coordinating phase changes and collecting statistics from the other components. When complete, it validates and combines the statistics into summary output files and presents the final metric for the test.

Steps of a SPECjms2007 Run

A SPECjms2007 run is a sequence of eight periodes[214], which can be grouped according three phases:

Phase I: Benchmark Preparation

1. *Starting Driver Framework*
2. *Starting Agents*
3. *Starting Event Handlers*

Phase II: Benchmark Run

4. *Warmup Period*
5. *Measurement Period*
6. *Drain Period*

Phase III: Post Run

7. *Stopping Event Handlers*
8. *Post-processing Results*

Period 1: *Starting Driver Framework* The controller component reads in all of the configuration and topological layout preferences given by the user. This will include items such as the number of different types of location and lists of the nodes across which they are distributed.

For vertical and horizontal scaling, many of these values are either fixed or are automatically calculated based upon the scaling factor.

With this knowledge, the controller instantiates the software components of the SUT. It begins this by starting an RMI server and connecting to a satellite process on each node machine identified as part of this test to give it specific instructions. In all places (throughout the benchmark) where lists are given, work is distributed equally using a round-robin algorithm.

Period 2: *Starting Agents* The satellite is a simple part of the framework that knows how to build the correct environment to start the required JVM processes. It takes the controller's configuration and starts the agents relevant to that node. There is an architectural lower bound of one Agent-JVM per class of location (SP, SM, DC, HQ), meaning a minumum of four AgentJVMs in the SUT. Each agent connects back to the controller to signal their readiness.

Period 3. *Starting Event Handlers* The controller signals all agents to initialise their event handler threads and connect to the JMS resources they will be using (this includes both incoming and outgoing Destinations). Each event handler is implemented as a Java thread.

Period 4: *Warmup Period* Load-generating threads (drivers) ramp up their throughput from zero to their configured rate over the Warmup Period. This helps ensure the SUT is not swamped by an initial rush when many of its constituent elements may not yet be fully prepared.

The agents are the only parts of the SUT which perform JMS operations (i.e. talk directly to the JMS middleware).

Period 5: *Measurement Period* The measurement period is also known as the steady-state period. All agents are running at their configured workload and no changes are made. The controller will periodically (thirty seconds by default) check that there are no errors and may also collect periodic realtime performance statistics.

Period 6: *Drain Period* In order to make sure all produced messages have an opportunity to be consumed, the controller signals agents to pause their load-generating threads (drivers). This period is not expected to be long in duration as a noticeable backlog in messages would invalidate audit requirements on throughput anyway.

Period 7: *Stopping Event Handlers* Agents will terminate all event handlers but remain present themselves so that the controller can collect final statistics.

Period 8: *Post-processing Results* Having collected statistics from all parties, the controller begins post-process including auditing and preparation of the final results.

Reporter Framework

The benchmark includes a *reporter framework* that prepares detailed reports about the run. For this purpose, the reporter framework enriches collected measurement data with other information, e.g., throughput predictions and configuration settings.

The reporter framework has two major components:

1. **Final Reporter**

   The controller takes formal measurements at three points during the run (see Figure 5.17). The first two, the beginning and end of the measurement period, are used to audit the messaging throughput. The final measurement, at the end of the drain period, is used to audit the final message counts.

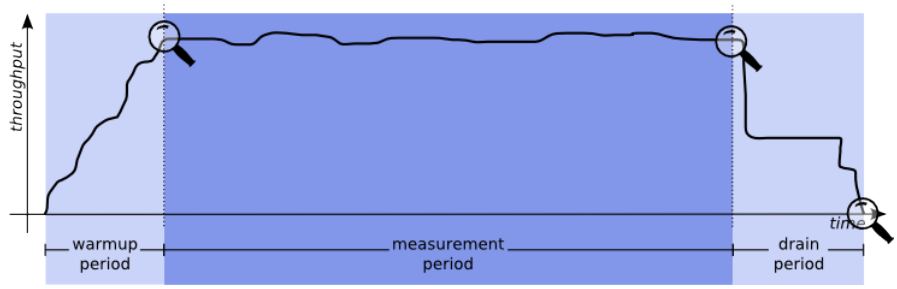


Figure 5.17: Formal Measurement Points during SPECjms2007 Run [214]

```
Collecting Runtime stats 1 for the last 300 sec. statistics complete in 0 seconds
=======Interaction 1 =======
- Predicted Sent/Received:  61672.71 / 61672.71
- Measured Sent/Received: 62509(+1.36 %) / 62510(+1.36 %)
=======Interaction 2 =======
- Predicted Sent/Received:  27341.57 / 31658.66
- Measured Sent/Received: 28143(+2.93 %) / 32604(+2.99 %)
=======Interaction 3 =======
- Predicted Sent/Received:  674.55 / 6745.45
- Measured Sent/Received: 703(+4.22 %) / 7030(+4.22 %)
=======Interaction 4 =======
- Predicted Sent/Received:  21877.14 / 21877.14
- Measured Sent/Received: 21899(+0.1 %) / 21896(+0.09 %)
=======Interaction 5 =======
- Predicted Sent/Received:  77869.59 / 77869.59
- Measured Sent/Received: 79321(+1.86 %) / 79323(+1.87 %)
=======Interaction 6 =======
- Predicted Sent/Received:  1919.96 / 19199.58
- Measured Sent/Received: 1860(-3.12 %) / 18599(-3.13 %)
=======Interaction 7 =======
- Predicted Sent/Received:  1383.68 / 13836.83
- Measured Sent/Received: 1414(+2.19 %) / 14139(+2.18 %)

Time to collect and process periodic runtime stats (msec): 428
```

Figure 5.18: Output of Run Time Reporter

The *final reporter* generates a set of detailed result files. These provide the data in different granularity and allow in-depth analysis of the run.

2. **Run Time Reporter**
   The *run time reporter* collects periodically measurements and statistics on the satellites and forwards them to the controller node, where they are processed to generate run time reports. These reports provides information on the running system and how current message counts differ from model's prediction (see Figure 5.18). By this, an early identification of message backlogs at run time is possible.

Audit Test	Scope	Description
Input rate is within +-5% of configured value	Interaction	For each Interaction, the observed input rate is calculated as count/time for the measurement period. This must be within 5% of the value prescribed by the topology.
Total message count is within +-5% of configured value	Interaction	Using a model of the scenario, the benchmark knows how many messages should be processed as part of each Interaction. The observed number of messages sent and received by all parties must be within 5% of this value.
Input rate distribution deviations do not exceed 20%	Interaction	The percentage of pacing misses (in driver threads) the benchmark will allow. A miss is when the timing code found itself behind where it thought it should be.
90th percentile of Delivery Times under 5000ms	Message Destination	Messages are timestamped when sent and received. The consequent delivery time is recorded as a histogram (for the measurement period only) in each event handler and the 90th percentile of this histogram must be on or under 5000ms.
All messages sent were received	Message Destination	Fails if the final results (taken after the Drain Period) show that not all sent messages were received. For publish-subscribe topics this means received by all subscribers.

Table 5.8: Audit Tests

Auditor

To validate the correctness of a benchmark run, SPECjms2007 comes with two auditor components:

- **Pre-Auditor *(executed in Phase I)*:** The *pre-auditor* checks the availability of JMS message destinations (queues / topcis) and whether messages exist before starting a benchmark run which would invalidate a run (but might otherwise only be detected when the test is completed). If the pre-audit fails for any reason the benchmark is not started.

- **Auditor *(executed in Phase III)*:** The *auditor* component is responsible for making sure that the run has been executed properly and that the results are valid. The Auditor is automatically launched at the end of the run to validate the results. It performs an explicit audit of the results against the specified requirements marking each audit test as $PASS$ or $FAIL$ in the summary reports. A list of all audit tests in provided in Table 5.8.

Minimizing the Impact of Non-MOM-Related Components

As discussed in Section 5.1.1, the SPECjms2007 workload should be focused on measuring the performance and scalability of a MOM server's software and hardware components and minimize the impact of other components that are not directly related to the MOM services. While developing SPECjms2007 two concerns had to be addressed in order to achieve this goal. The first one is how to avoid using a database and the second one is how to minimize the message processing overhead on the client. Given the fact that MOM servers, in their role as mediators in interactions, are typically less loaded than database servers or application servers,

it was a challenge to place the focus of the workload on the MOM-related components, without compromising the workload representativeness.

As to the first concern, the problem is that without a database it is hard to manage any application state and this makes it difficult to emulate the interdependencies between the interactions. We addressed this by building a detailed model of the business scenario, capturing the relative rates of the different operations and the interdependencies between the interactions. This made it possible to emulate database access operations and configure the interactions in such a way that they behave as if a real database were used. Note that, while we are not using a database for managing application state, it is perfectly legitimate to use a database as part of the MOM infrastructure for persistent messaging.

As to the second concern, the major issue was how to minimize the overhead of parsing XML messages on the client. On the one hand, we wanted to use XML for inter-company communication in order to keep things as realistic as possible, on the other hand, using a full-blown XML parser to parse messages would have introduced too much overhead on the client for operations which are not directly related to the MOM services. The solution was to implement an optimized routine that exploits the knowledge of the exact structure of received XML messages and extracts the needed data without having to parse the whole XML documents.

Workload Configurability

An important goal of SPECjms2007 that we discussed in Section 5.1.1 was to provide a flexible framework for performance analysis of MOM servers that allows users to configure and customize the workload according to their requirements. To achieve this goal, the interactions have been implemented in such a way that one could run them in different combinations depending on the desired transaction mix. SPECjms2007 offers three different ways of structuring the workload: *horizontal*, *vertical* and *freeform*. The latter are referred to as *workload topologies* and they correspond to three different modes of running the benchmark offering different levels of configurability. The horizontal topology is meant to exercise the ability of the system to handle an 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 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. Finally, the freeform topology allows the user to use the seven SPECjms2007 interactions as building blocks to design his own workload scenario which can be scaled in an arbitrary manner by increasing the number of physical locations and/or the rates at which interactions are run. In the most general case, the following workload parameters can be configured:

- Number of physical locations (HQs, SMs, DCs, SPs) emulated
- Rates at which interactions are run
- Message size distribution for each message type
- Number of agents for each physical location
- Distribution of agents across client nodes
- Number of JVMs run on each client node
- Distribution of agents among JVMs
- Number of event handlers for each message type
- Number of driver threads for each interaction

- Number of JMS connections shared amongst event handlers
- Acknowledgment mode for non-transactional sessions
- Optional connection sharing by multiple sessions
- Frequency of runtime statistics

While in horizontal and vertical topologies restrictions apply to the above parameters, freeform topology leaves all parameter configurations up to the user. Most importantly, the user can selectively turn off interactions or change the rate at which they are run to shape the workload according to his requirements. At the same time, when running the horizontal or vertical topology, the benchmark behaves as if the interactions were interrelated according to their dependencies in the real-life application scenario. For further details on the benchmark implementation, the reader is referred to [214].

5.2 Case Study I: SPECjms2007

In this section, we present a case study with a deployment of SPECjms2007 using the WebLogic Server 10 JMS platform including a detailed performance analysis considering both the P2P and pub/sub messaging domains. Our evaluation is the first one that uses a standard workload to stress the JMS server. We demonstrate how SPECjms2007 can be exploited for in-depth analysis of selected aspects of the MOM server performance.

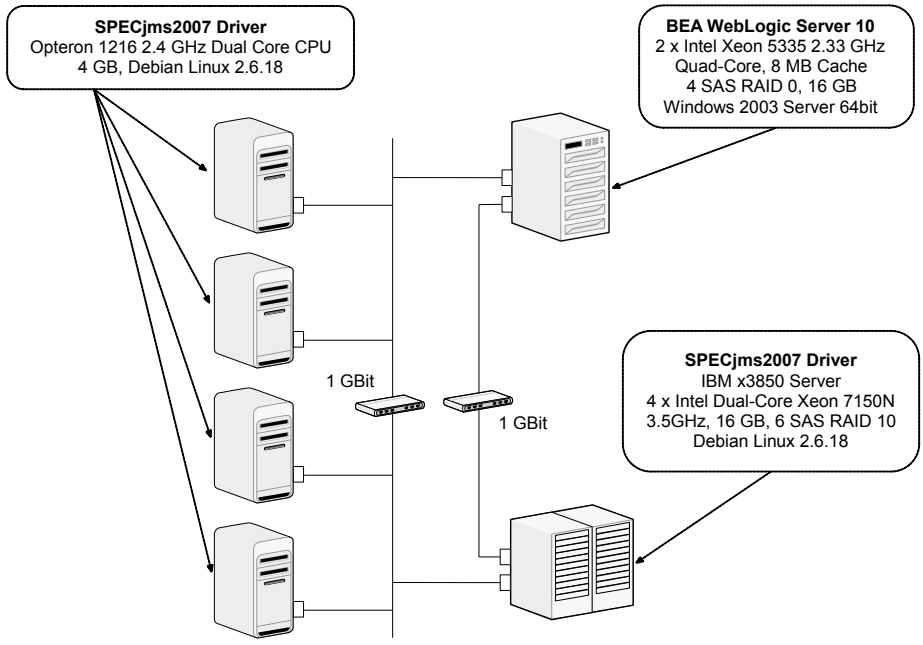


Figure 5.19: Experimental Environment

5.2.1 Experimental Setting

The experimental environment in which we conducted our case study is depicted in Figure 5.19. WebLogic Server was deployed 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 JRockit 1.5 JVM using 8 GByte of heap space. A RAID 0 disk array comprised of four disk drives was used for maximum performance. The WebLogic JMS Server was configured to keep persistent messages in a file-based store on the disk array and to use a 3.8 GByte message buffer to store message bodies in memory. The SPECjms2007 controller and satellite drivers were distributed across five machines, four one-way dual-core Opteron at 2.4 GHz and one four-way dual-core Intel Xeon at 3.5 GHz. All machines were connected to a 1 Gbit network. To further increase the network capacity, a separate Gbit link was installed between the server and the four-way driver machine. The latter was configured to always use this link when accessing the server. The satellite drivers were distributed across the machines in such a way that the network traffic was load-balanced between the two networks.

5.2.2 Horizontal and Vertical Scaling

We first ran some experiments in the horizontal and vertical topologies in order to show the behavior of the server when scaling the workload in the two alternative ways[4]. Figure 5.20 shows the server CPU utilization and the CPU processing time per message (counting both sent and received messages) for the horizontal topology. Figure 5.21 shows the same data for the vertical topology. In both cases, there is a clear linear correlation between the scaling factor (i.e., the BASE) and the server utilization. However, the server utilization grows much faster in the horizontal mode. For a given value of the scaling factor, the CPU consumption of the horizontal topology is between 2.2 and 2.3 times higher than the CPU consumption of the vertical topology. This is expected given that the number of messages injected per second in the horizontal topology is about two times higher than in the vertical topology (see the message traffic analysis in Sections 5.1.4). It is interesting to compare the average CPU time per message (counting both sent and received messages). The latter is about 10% lower for the horizontal topology. The reasons for this will become clear in the next section.

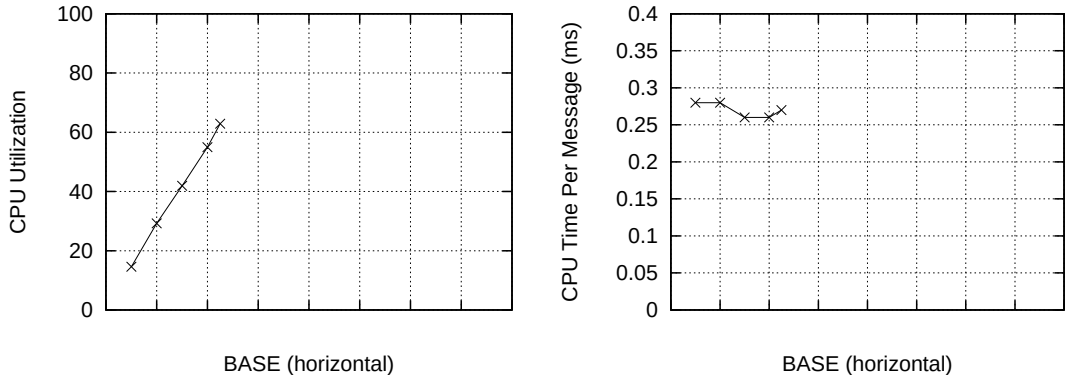


Figure 5.20: Measurement Results for Horizontal Experiments

5.2.3 Customized Vertical Workloads

We now consider two customized workloads based on the vertical topology. The goal is to break down the workload into its P2P and pub/sub components and analyze the server performance when running them in isolation. To this end, the first workload runs only P2P interactions (i.e.,

[4]SPECjms2007 is a trademark of the Standard Performance Evaluation Corporation (SPEC). The results or findings in this chapter 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.

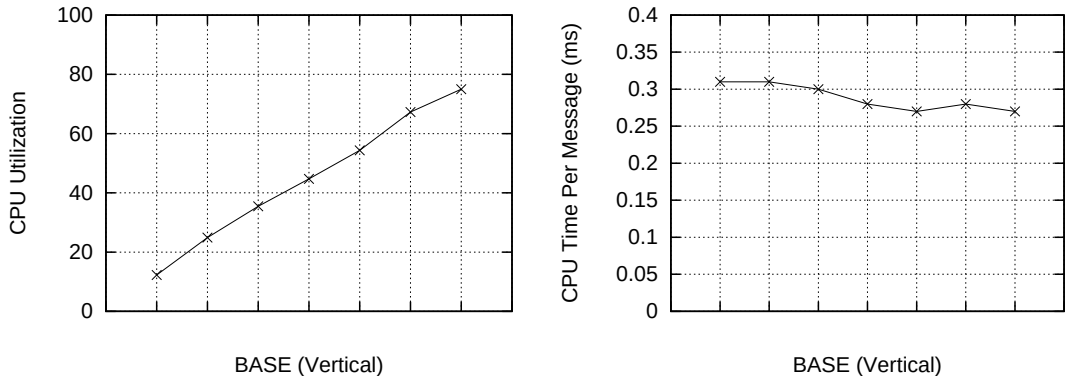


Figure 5.21: Measurement Results for Vertical Experiments

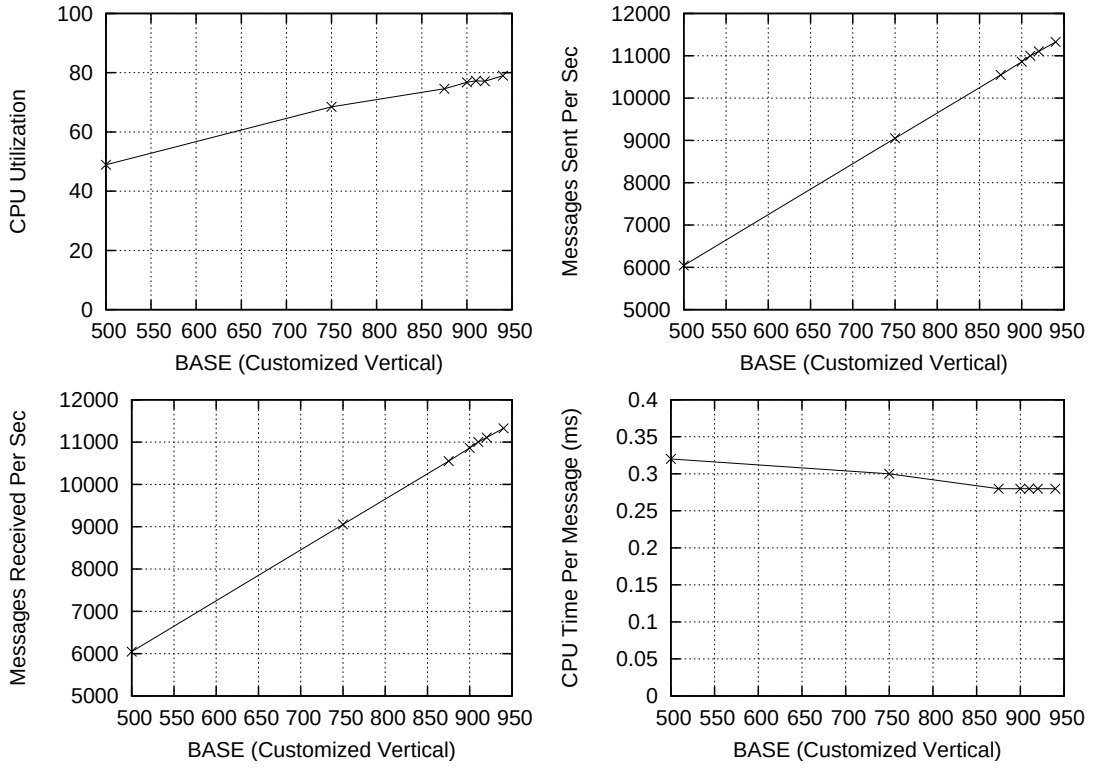


Figure 5.22: Measurement Results for Customized Vertical Experiments with P2P Messaging

1, 4 and 5), whereas the second one runs only pub/sub interactions (i.e., 3, 6 and 7)[5]. In both cases, the relative interaction mix for the considered interactions is the same as for the standard vertical topology. Figures 5.22 and 5.23 show the measurement results. We can see that, as expected, the pub/sub portion of the workload is by far much more light-weight than the P2P portion. This is due to two reasons. On the one hand, for a given value of the BASE, the P2P message traffic injected is much larger than the pub/sub traffic according to the definition of the vertical topology presented in Section 5.1.4. On the other hand, the server overhead per

[5]Note that Interaction 2 is not part of these workloads since it contains a mix of both P2P and pub/sub messaging.

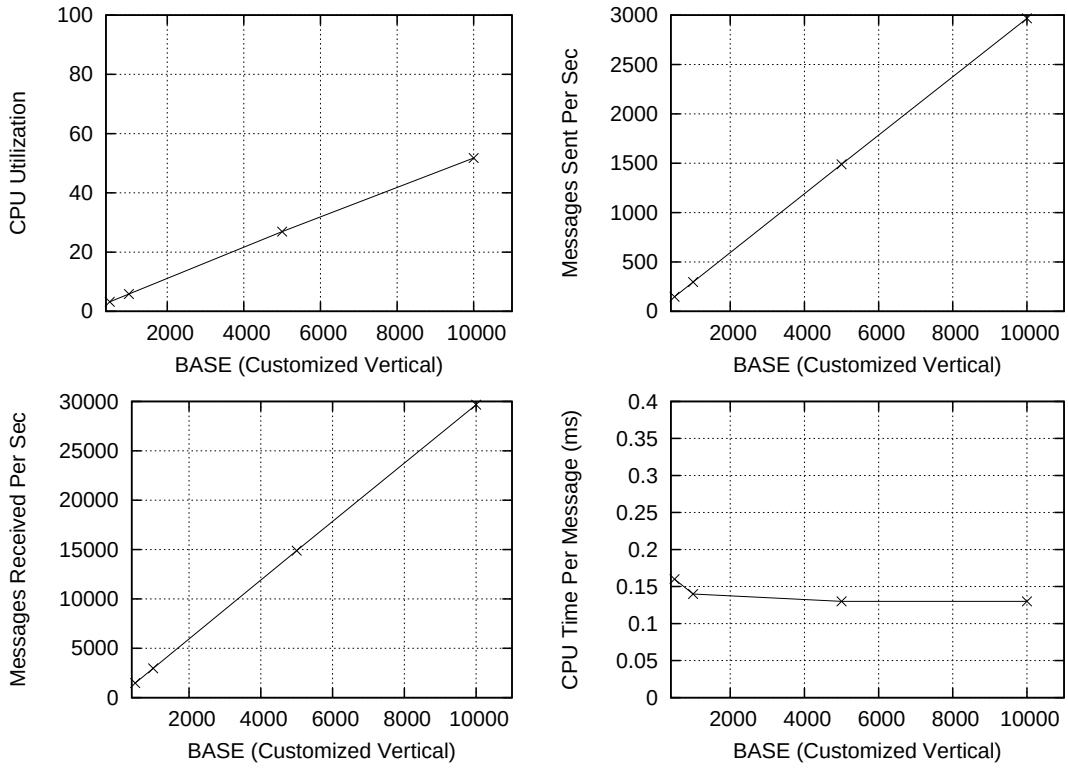


Figure 5.23: Measurement Results for Customized Vertical Experiments with Pub/Sub Messaging

delivered message is significantly lower in the pub/sub case. Looking at the CPU time per message (counting both sent and received messages) in the two workloads, we can see that for both workloads the latter does not change much as we increase the BASE. For P2P messaging it stabilizes at around 0.28ms, whereas for pub/sub messaging it stabilizes at 0.13ms. From this we can conclude that the overhead per P2P message sent/received in the vertical topology is over two times higher than the overhead per pub/sub message sent/received.

This explains why the CPU time per message sent/received in the horizontal topology compared to the vertical topology was measured to be lower in the previous section. This is expected given that the horizontal topology has much less P2P messaging as a proportion of the overall workload than the vertical topology (see Table 5.6).

5.2.4 Publish/Subscribe Messaging

We now study the performance of the server when running only pub/sub messaging. We use the freeform topology and specifically Interactions 3 and 7 to exercise persistent transactional durable (PTD) messaging and non-persistent non-transactional non-durable (NPNTND) messaging, respectively. Table 5.9 shows the configuration for five of the scenarios we analyzed. For each scenario, the emulated number of producers and consumers are shown. Multiple producers and consumers are configured by setting the number of interaction driver threads and the number of emulated SMs, respectively. The producers were run on the 8-core IBM x3850 server, whereas the consumers were distributed among the four one-way Opteron-based servers. In both of the considered interactions, each SM acts as a message consumer and therefore the number of consumers is equal to the number of SMs. In all scenarios there is a single HQ instance and

Scen.	Int.	# Prod.	# Cons.	Injection Rate	Msg. Size	Msg. Group	Fig.
1	7	30	*variable*	1000 msg/sec	*variable*	a	5.24
2	7	30	10	1000 msg/sec	*variable*	a	5.25
3	7	*variable*	*variable*	*unlimited*	0.24 KByte	a	5.26
4	3	1	*variable*	*unlimited*	0.24 KByte	b	5.27
5	3 & 7	1	*variable*	*unlimited*	0.24 KByte	b	5.28

Table 5.9: Configuration for Pub/Sub Scenarios

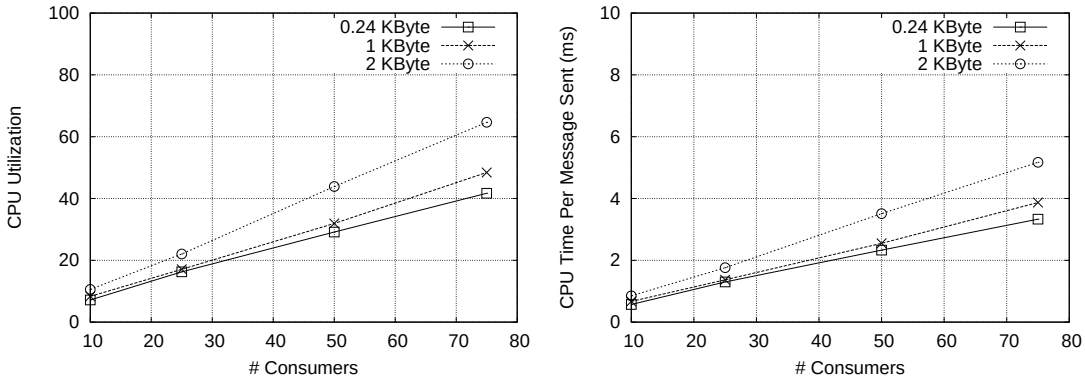


Figure 5.24: Scenario 1: NPNTND Pub/Sub Messaging with Increasing Number of Consumers

a different number of SMs depending on the specified number of consumers. For each scenario, Table 5.9 also shows the message injection rate, the message size and the message type according to the classification in Table 5.3. Given that in both Interaction 3 and 7, each interaction execution results in sending a single message, the specified message injection rate is configured by setting the respective interaction rate. In the cases where 'unlimited' message injection rate is specified, each producer is configured to inject messages at full speed (i.e., with zero delay between successive messages). The results from the experiments are presented in Figures 5.24 to 5.28.

We now take a closer look at the measurement results. We start with NPNTND pub/sub messaging. In the first scenario, we consider the effect of increasing the number of consumers on the server CPU consumption. As expected, the overall CPU utilization and the CPU processing time per message increase linearly with the number of consumers and the rate of increase depends on the message size (Figure 5.24). The larger the message size, the greater the effect the number of consumers has on the overall CPU consumption.

The goal of the second scenario is to evaluate the effect of increasing the message size on the CPU consumption per message and KByte of payload sent. The CPU processing time per message is directly proportional to the message size, however, this does not hold for the CPU time per KByte of payload (Figure 5.25). The latter drops exponentially for message sizes up to 10 KByte and stabilizes around 0.2ms for larger messages. This is due to the fact that for every message there is a constant overhead around 0.4ms (independent of the message size) for parsing the JMS message header. For small messages this overhead dominates the overall processing time, however, as the size of the message grows, the overhead becomes negligible compared to the time needed to deliver the message payload. Thus, for messages larger than 20 KByte, we can estimate the message processing time as $MsgSize * 0.2ms$.

In the third scenario, we analyze the effect of varying the number of producers and consumers (Figure 5.26). Each producer is configured to publish messages at full speed. Given that the number of emulated producers (up to 5) does not exceed the number of available CPU cores on

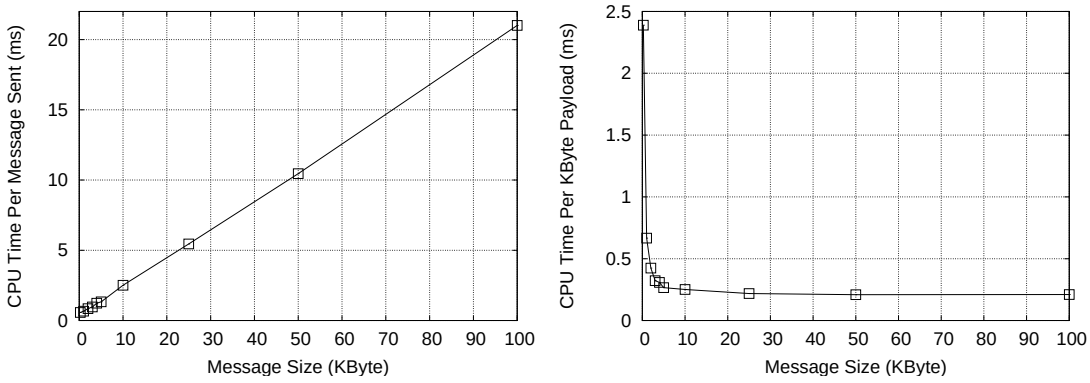


Figure 5.25: Scenario 2: NPNTND Pub/Sub Messaging with Increasing Message Size

the machine hosting the producers, the latter can inject messages in parallel without competing for CPU time on the client side. We consider the server CPU utilization, the throughput in terms of messages sent per second and the CPU processing time per message sent. It is important to note that, in all considered scenarios, the machines hosting the producers and consumers were far from saturated, so that the effect of the client side of the benchmark on the observed system performance was insignificant. From the results we see that increasing the number of message producers and consumers both lead to higher server CPU utilization, however, the number of producers has by far much higher effect on the CPU consumption than the number of consumers. The reason is that whereas the message throughput increases with increasing number of producers, it decreases with increasing number of consumers even if the server is only lightly loaded. The results show that the effect of the decreasing throughput on the CPU consumption cancels out the effect of the increasing number of consumers resulting in stagnation in the server utilization. This is due to synchronization effects. The server has to ensure that successive messages sent by individual producers are delivered in the order in which they are sent. Thus, the more consumers, the higher the synchronization overhead for each producer. Messages sent by different producers, on the other hand, are not affected by this because the server is not required to deliver them in the order in which they were sent. The results also show that the number of producers does not have a significant effect on the average CPU processing time per message. On the other hand, as already shown in the first scenario, the CPU processing time per message is directly proportional to the number of consumers.

In the fourth and fifth scenarios, we evaluate the performance of PTD pub/sub messaging. We first look at the effect of increasing the number of consumers on the server CPU consumption, the mean message delivery latency and the number of messages sent/received per second. The results are shown in Figure 5.27. The server CPU utilization goes up to almost 80% for 150 consumers and stabilizes at this level together with the total number of received messages per second for higher number of consumers. Message processing in this case includes disk I/O operations for persisting the messages. The message delivery latency remains below 15ms for up to 150 consumers. There is a good linear correlation between the received messages/sec and the server CPU utilization. Finally, the rate of sending messages drops by almost a factor of 20 as the number of consumers is increased up to the saturation point.

The fifth scenario compares NPNTND messaging with PTD messaging in terms of the server CPU utilization, the message throughput (number of messages sent per second) and the CPU processing time per message sent. As we increase the number of consumers, the server CPU utilization increases steadily at a decreasing rate. The CPU processing time per message sent increases linearly with the number of consumers and the rate of increase is much higher for PTD messaging than for NPNTND messaging. For 150 consumers, the overhead is over 6 times

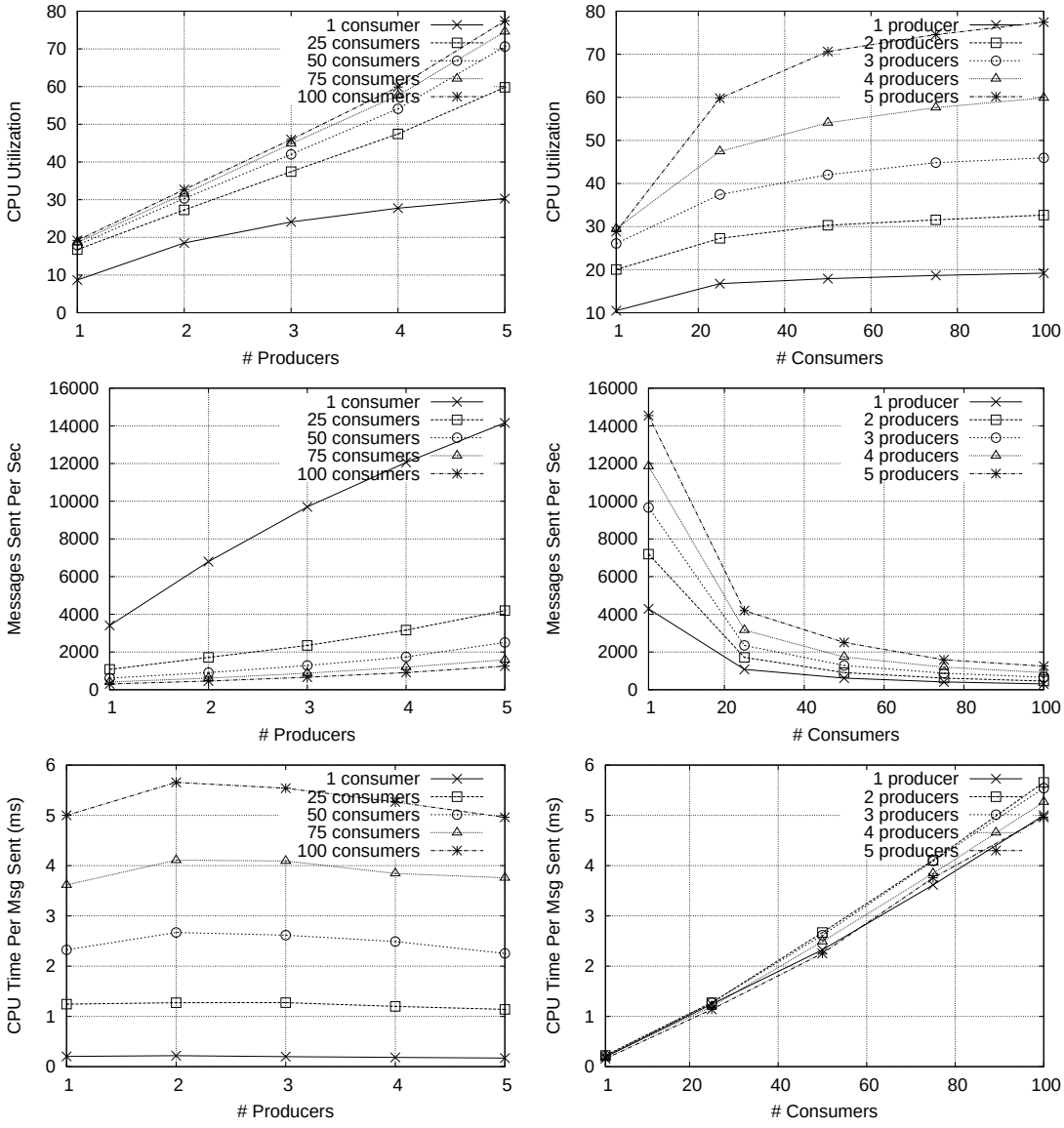


Figure 5.26: Scenario 3: NPNTND Pub/Sub Messaging with Varying Number of Producers and Consumers

higher for PTD messaging than for NPNTND messaging. This is explained by the fact that PTD messaging includes additional overhead not just for persisting messages but also for managing transactions which is directly dependent on the number of consumers.

5.2.5 P2P Messaging

We now study the performance of the server when running only P2P messaging. We use the freeform topology and specifically Interactions 5 and 4 to exercise non-persistent non-transactional (NPNT) and persistent transactional (PT) messaging, respectively. Table 5.10 shows the configuration for three scenarios we analyzed. For each scenario, the number of SMs and HQs are shown as well as the message injection rate, the message size and the message

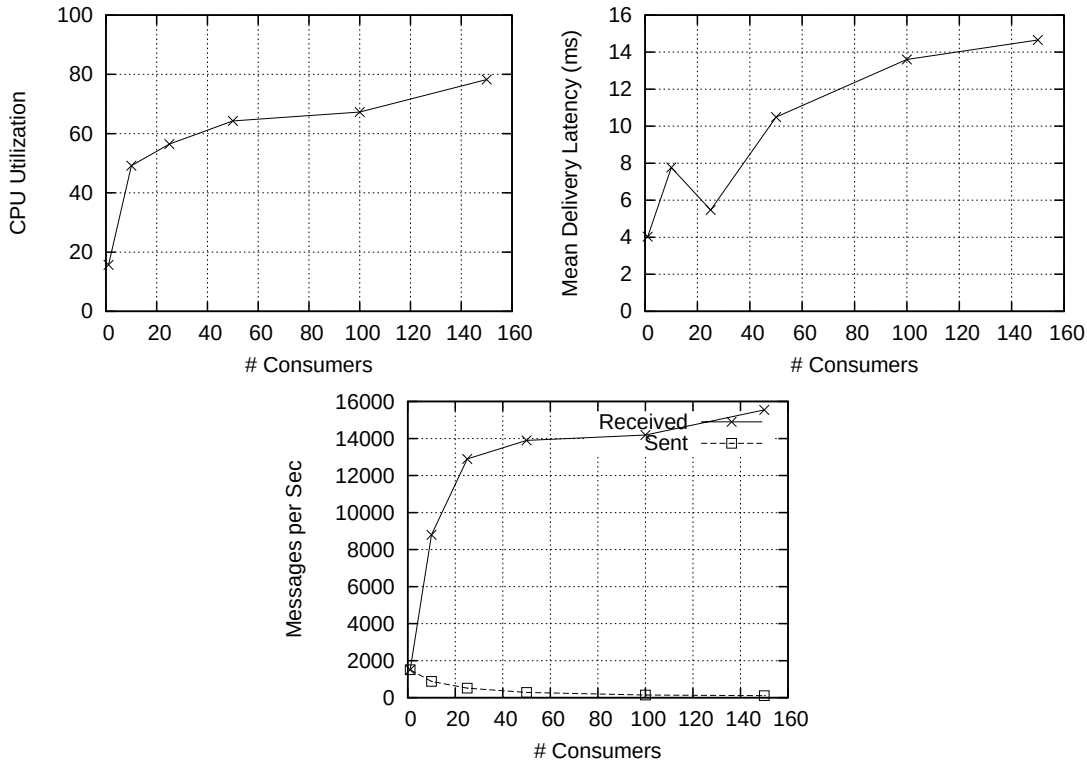


Figure 5.27: Scenario 4: PTD Pub/Sub Messaging with Increasing Number of Consumers

Scen.	Int.	# SMs	# HQs	Injection Rate	Msg. Size	Msg. Group	Figure
1	5	*variable*	*variable*	*unlimited*	2 KByte	*c*	5.29
2	4	*variable*	*na*	*unlimited*	2 KByte	*d*	5.29
3	4	5	*na*	*unlimited*	*variable*	*d*	5.30

Table 5.10: Configuration for P2P Scenarios

type according to the classification in Table 5.3. Given that in both Interaction 4 and 5, each interaction execution results in sending a single message, the specified message injection rate is configured by setting the respective interaction rate. The interaction rate is specified on a per location basis. The analysis results for the three scenarios are presented in Figures 5.29 and 5.30.

We now take a closer look at the results. The first two scenarios compare the performance of NPNT and PT P2P messaging (Figure 5.29). In both scenarios, the number of queues used is varied and the goal is to measure the maximum message traffic per second that can be processed. The first scenario uses Interaction 5 with multiple HQ instances each having its own queue for incoming statInfoSM messages sent by the SMs. In each test, both the number of HQ instances and the number of SMs are set to the desired number of queues. Thus, every SM has a HQ instance and a respective queue that receives its messages. SM agents have 5 producer (driver) threads each. HQ agents have 5 consumer threads each. In order to ensure that the number of producer and consumer threads remains constant, the number of agents per SM/HQ is set in such a way that the number of agents of each type does not change (Section 5.1.5). For example, in the test with 1 queue (1 SM and 1 HQ), there are 20 agents per SM/HQ, in the test with 2 queues, there are 10 agents per SM/HQ and so forth, in all cases leading to 20

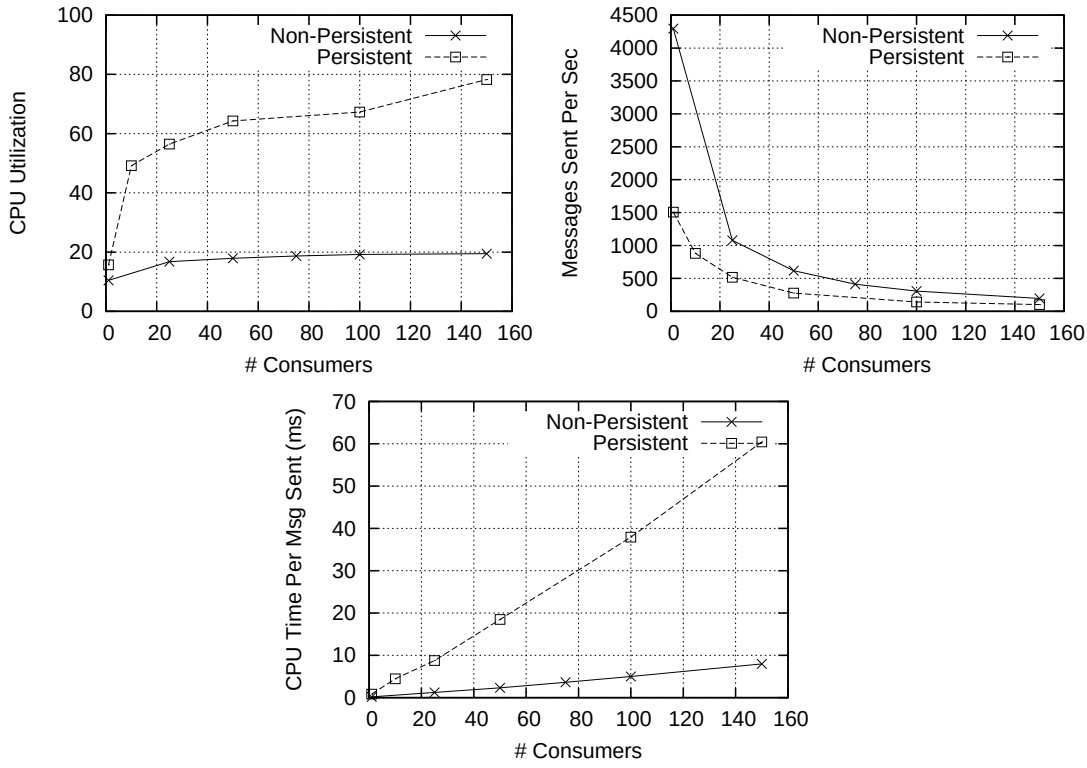


Figure 5.28: Scenario 5: NPNTND vs. PTD Pub/Sub Messaging

agents in total. The second scenario is setup in exactly the same way with exception that it uses Interaction 4 and therefore only SM agents are involved. Each SM agent has 5 producer and 5 consumer threads. The two interactions are configured to use the same message size so that we can compare the results.

As we can see in Figure 5.29, when moving from 1 queue to 2 queues, the message throughtput increases by about 5% for NPNT messaging and about 10% for PT messaging. Increasing the number of queues beyond 2, does not affect the message throughput, the server utilization or the CPU time per message/Kbyte. The server CPU utilization is slightly lower (6-10%) for PT messaging. The latter is expected given that persistent messaging involves disk I/O. The message throughput is about 2.5 times higher for NPNT messaging given that the CPU time used per message/KByte processed is over 2 times lower compared to PT messaging. Overall, the results show that using more than two queues does not lead to any noticable change in the system performance of our configuration.

In the third scenario, we study the performance of PT P2P messaging with variable message size. We use Interaction 4 with a fixed number of SMs and 5 producer and 5 consumer threads per SM. The results are shown in Figure 5.30. As we can see, the CPU processing time per message increases linerarly with the message size whereas the CPU time per KByte quickly drops and stabilizes around 0.1ms per KByte. As we discussed earlier when evaluating pub/sub messaging, the reason for the drop in the overhead per KByte is that there is a constant overhead for parsing the message header which for small messages dominates the overall processing time. The mean delivery latency seems to increase quadratically with the message size.

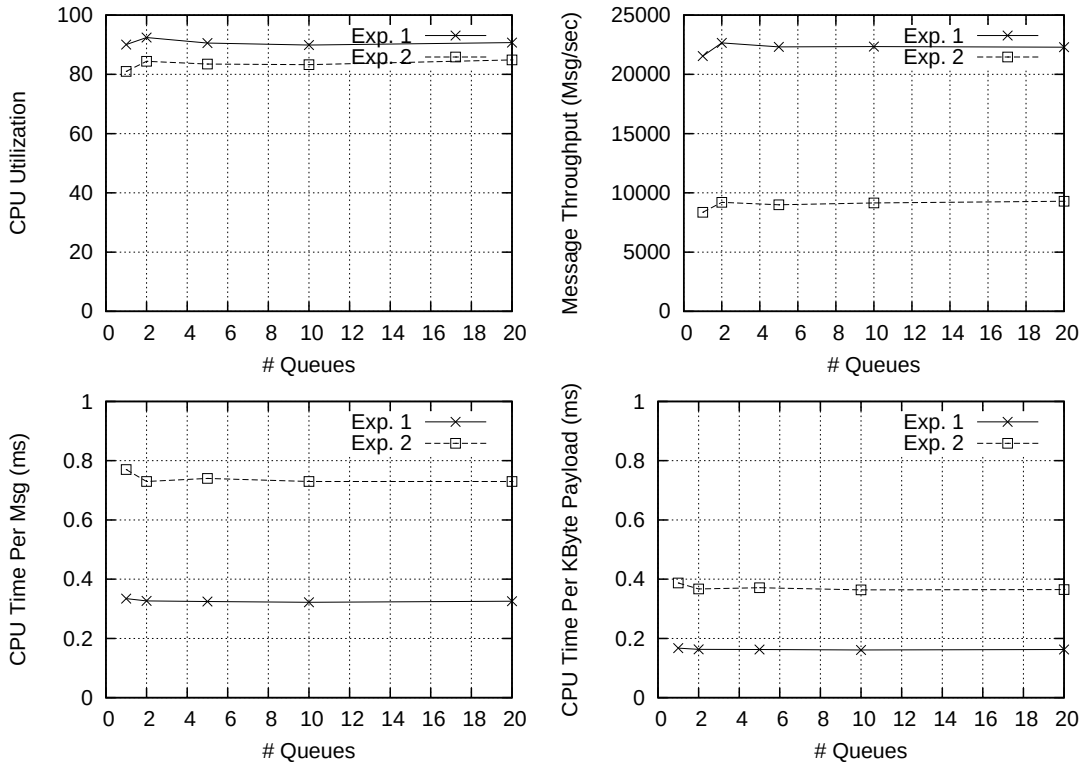


Figure 5.29: Scenarios 1 and 2: NPNT vs. PT P2P Messaging with Increasing Number of Queues

5.2.6 Conclusions of the SPECjms2007 Case Study

In this section, we presented a case study of a leading JMS platform, the WebLogic server, conducting an in-depth performance analysis of the platform under a number of different workload and configuration scenarios. We evaluated the server performance for both the point-to-point and publish/subscribe messaging domains studying the effect of individual workload characteristics on the server CPU utilization, the message throughput, the CPU processing time per message/KByte payload, the message delivery latency, etc. Two groups of scenarios were tested. The first group uses complex workloads based on the standard horizontal and vertical topologies provided by the benchmark. The second group includes scenarios that focus on specific aspects and features of MOM, e.g the overhead of persisting messages, the influence of the message size, the effect of increasing the number of message producers/consumers and the maximum throughput that can be processed through a given number of queues. In most cases, the system scaled in a linear fashion and did not exhibit any unexpected behavior. Interesting observations are the relation between the number of the producer threads and message throughput, the number of topic consumers and CPU load, and the influence of pub/sub and P2P messaging in the vertical scenario.

5.3 jms2009-PS - A Publish /Subscribe Benchmark

While SPECjms2007 includes some limited publish/subscribe communication as part of the workload, the focus of the benchmark is on point-to-point communication via queues which dominate the overall system workload. Moreover, the SPECjms2007 workload implementation

Intr.	Message	Location	T	P	D	Q	TD	ST	Description
1	order	DC	✓	✓	✓	✓	✓	-	Order sent from SM to DC.
	orderConf	SM	✓	✓	✓	✓	✓	-	Order confirmation sent from DC to SM.
	shipDep	DC	✓	✓	✓	✓	✓	-	Shipment registered by RFID readers upon leaving DC.
	statInfo-OrderDC	HQ	✓	✓	✓	✓	✓	-	Sales statistics sent from DC to HQ.
	shipInfo	SM	✓	✓	✓	✓	✓	-	Shipment from DC registered by RFID readers upon arrival at SM.
	shipConf	DC	✓	✓	✓	✓	✓	-	Shipment confirmation sent from SM to DC.
2	callForOffers	HQ	✓	✓	✓	-	✓	✓	Call for offers sent from DC to SPs (XML).
	offer	DC	✓	✓	✓	✓	✓	-	Offer sent from SP to DC (XML).
	pOrder	SP	✓	✓	✓	✓	✓	-	Order sent from DC to SP (XML).
	pOrderConf	DC	✓	✓	✓	✓	✓	-	Order confirmation sent from SP to DC (XML).
	invoice	HQ	✓	✓	✓	✓	✓	-	Order invoice sent from SP to HQ (XML).
	pShipInfo	DC	✓	✓	✓	✓	✓	-	Shipment from SP registered by RFID readers upon arrival at DC.
	pShipConf	SP	✓	✓	✓	✓	✓	-	Shipment confirmation sent from DC to SP (XML).
	statInfo-ShipDC	HQ	✓	✓	✓	✓	✓	-	Purchase statistics sent from DC to HQ.
3	priceUpdate	HQ	✓	✓	✓	-	✓	-	Price update sent from HQ to SMs.
4	inventoryInfo	SM	✓	✓	✓	✓	✓	-	Item movement registered by RFID readers in the warehouse of SM.
5	statInfoSM	HQ	✓	✓	✓	✓	✓	-	Sales statistics sent from SM to HQ.
6	product-Announcement	HQ	✓	✓	✓	-	✓	-	New product announcements sent from HQ to SMs.
7	creditCardHL	HQ	✓	✓	✓	-	✓	-	Credit card hotlist sent from HQ to SMs.

Table 5.11: Configuration parameters supported for each message type.

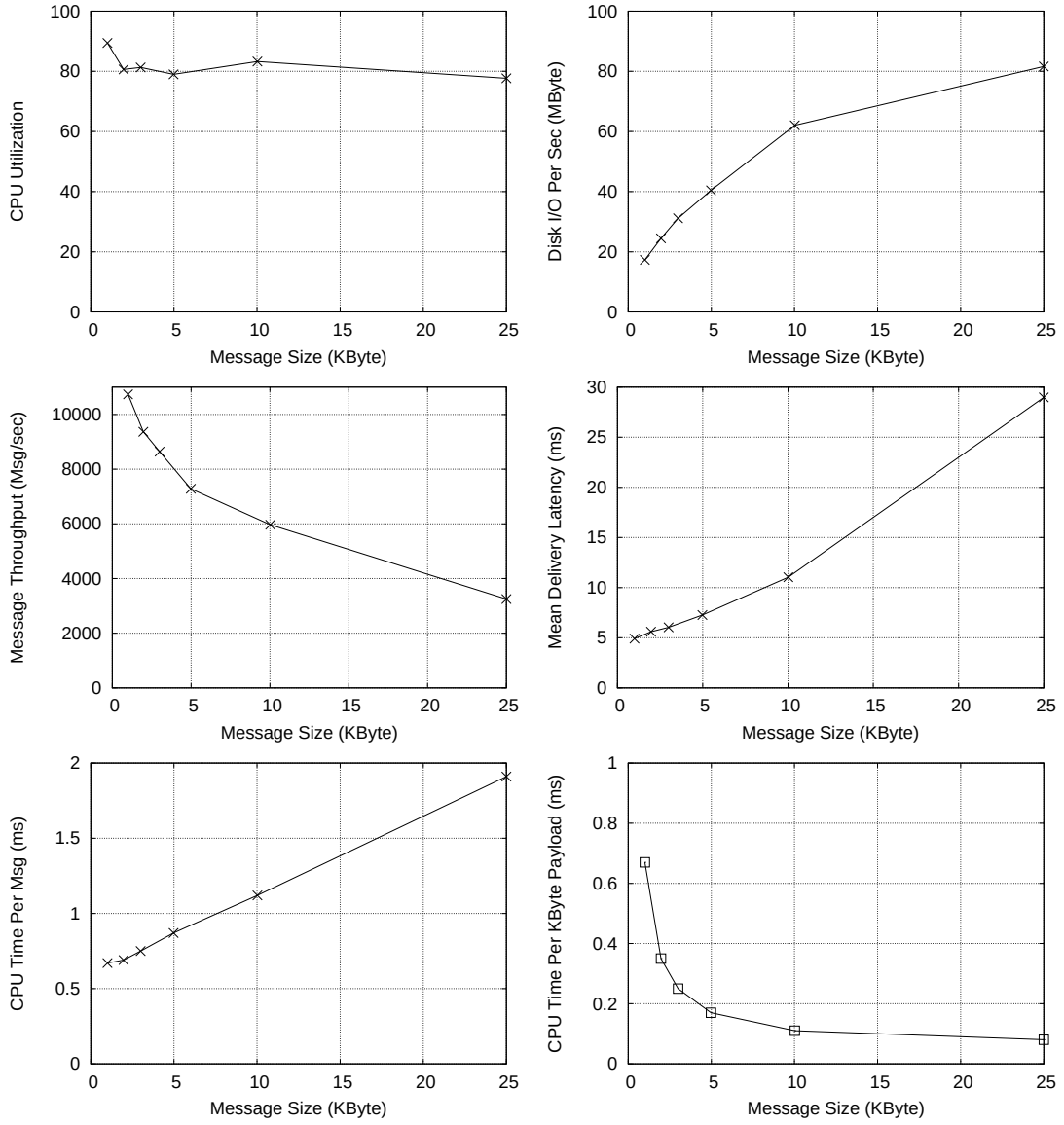


Figure 5.30: Scenario 3: PT P2P Messaging with Increasing Message Size

does not exercise message filtering through JMS selectors, which is an important feature of publish/subscribe messaging that typically causes most performance and scalability issues. To show how to apply this workload on another plattform and to address the need for a workload focused on publish/subscribe messaging, we developed the new *jms2009-PS* benchmark which is based on the SPECjms2007 workload [197, 199]. In this section, we introduce the benchmark and discuss its configuration parameters showing how the workload can be customized to evaluate different aspects of publish/subscribe communication. Overall, we added more than 80 new configuration parameters allowing the user to adjust the workload to his needs. All configurations are identical in terms of the number of subscriptions and the message throughput generated for a given scaling factor. However, they differ in six important points:

1. Number of topics and queues used

2. Number of transactional vs. non-transactional messages
3. Number of persistent vs. non-persistent messages
4. Total traffic per topic and queue
5. Complexity of selectors used (filter statements)
6. Number of subscribers per topic

While the benchmark is targeted at pub/sub workloads, it allows us to use queue-based P2P messaging in cases where messages are sent to a single consumer. This allows the comparison of the costs of queue-based vs. topic-based communication for different message delivery modes. In the case of topic-based communication, several implementations for each interaction are supported. In the first implementation, all types of messages are exchanged using one common topic per interaction. Each message consumer (e.g., orders department in DC 1) subscribes to this topic using a selector specifying two filters, which specify the messages he is interested in: message type (e.g., orders) and location ID (e.g., DC 1). The message type and location ID are assigned as properties of each message published as part of the respective interaction.

In the second implementation, a separate topic is used for each type of message (e.g., one topic for orders, one for invoices). Consequently, message consumers do not have to specify the message type at subscription time, but only their location ID. It is easy to see that the number of subscribers per topic is lower and the filtering is simpler (only one property to check) in the second implementation compared to the first one. In the first implementation, more traffic is generated per topic, while in the second implementation the traffic per topic is less but the system has to handle more topics in parallel. Therefore, the two implementations stress the system in different ways and allow us to evaluate different performance aspects. In addition to these two implementations, the benchmark supports several further implementations which allow to stress additional aspects of topic-based communication. The user can select an implementation by means of the Target Destination (TD) parameter discussed in the next section.

5.3.1 Configuration Parameters

In this section, we describe in detail the new configuration parameters introduced in jms2009-PS. The parameters can be configured on a per message type basis. Table 5.11 shows the parameters supported for each message type. In the following, we briefly describe each parameter.

Transactional [*true*|*false*] (T)

Specifies whether messages should be sent as part of a transaction.

Persistent [*true*|*false*] (P)

Specifies whether messages should be sent in persistent mode.

Durable [*true*|*false*] (D)

Specifies whether a durable subscription should be used by message consumers.

Queue [*true*|*false*] (Q)

Specifies whether a queue or a topic should be used in cases where there is a single message consumer.

Setting	Description	Selector
LocationID-MessageType	A separate topic for each combination of location instance and message type is used, e.g., a topic per DC for order messages: `DC1_OrderT` for DC 1, `DC2_OrderT` for DC 2, etc.	• No selectors are needed.
MessageType	A single topic per message type is used, e.g., a topic `DC_OrderT` for order messages of all DCs.	• TargetLocationID= '*locationID*'
Interaction	A single topic per interaction is used, e.g., a topic `Interaction1_T` for all messages involved in Interaction 1.	• TargetLocationID= '*locationID*' • MessageType= '*messageType*'
LocationType	A single topic per location type is used, e.g., a topic `SM_T` for all messages sent to SMs.	• *TargetLocationID*= '*locationID*' • MessageType= '*messageType*'
LocationID	A separate topic for each location instance is used, e.g., a topic `SM1_T` for all messages sent to SM 1.	• MessageType= '*messageType*'
Central	One central topic for all messages is used, e.g., one topic `T` for all messages that are part of the seven interactions.	• LocationType= '*locationType*' • TargetLocationID= '*locationID*' • MessageType= '*messageType*'

Table 5.12: Target destination options.

Target Destination (TD)

Specifies for each message type the set of topics and respective selectors that should be used to distribute messages to the target consumers. The benchmark supports six different target destination options. Depending on the selected configuration, it automatically takes care of configuring message properties (set by producers) and selectors (set by consumers at subscription time) to guarantee that messages are delivered to the correct consumers. The target destination options supported by jms2009-PS are shown in Table 5.12. For each option, the set of topics and the required selectors are described.

Subscription Type [*IN*|*OR*|*SET*] (ST)

In Interaction 2, a distribution center (DC) sends a `CallForOffers` to suppliers (SP). Each SP offers a subset of all product families and is only interested in the `CallForOffers` messages targeted at the respective product families. There are multiple ways to implement this communication pattern and jms2009-PS supports the following options:

- **Use a separate topic for each product family**: The SP has to subscribe to all topics corresponding to the product families he is interested in and no selector is needed.
- **Use one topic for all product families**: The SP has to subscribe to this topic using a

selector to specify the product families he is interested in. jms2009-PS offers three ways to define the respective subscription:

- **Using multiple OR operators**: The SP places a single subscription using the following selector: *ProductFamily*="PF1" `OR` *ProductFamily*="PF2" `OR` ... `OR` *ProductFamily*="PF*n*"
- **Using a single IN operator**: The SP places a single subscription using the following selector: *ProductFamily* `IN` ("PF1","PF2",...,"PF*n*")
- **Using a set of subscriptions**: The SP subscribes for each product family he is interested in separately:
  ProductFamily="PF1" [···] *ProductFamily*="PF*n*"

5.4 Case Study II: jms2009-PS

5.4.1 Introduction

We now present a case study illustrating how jms2009-PS can be used for performance analysis of messaging servers. The environment in which we conducted our case study is depicted in Figure 5.31. ActiveMQ server was used as a JMS server installed on a machine with two quad-core CPUs and 16 GB of main memory. The server was run in a 64-bit JRockit 1.6 JVM with 8 GB of heap space. A RAID 0 disk array comprised of four disk drives was used for maximum performance. ActiveMQ was configured to use a file-based store for persistent messages with a 3.8 GB message buffer. The jms2009-PS drivers were distributed across three machines. To further increase the network capacity, a separate GBit link was installed between the JMS server and the third driver machine. The latter was configured to always use this link when accessing the server. The drivers were distributed across the machines in such a way that the network traffic was load-balanced between the two networks.

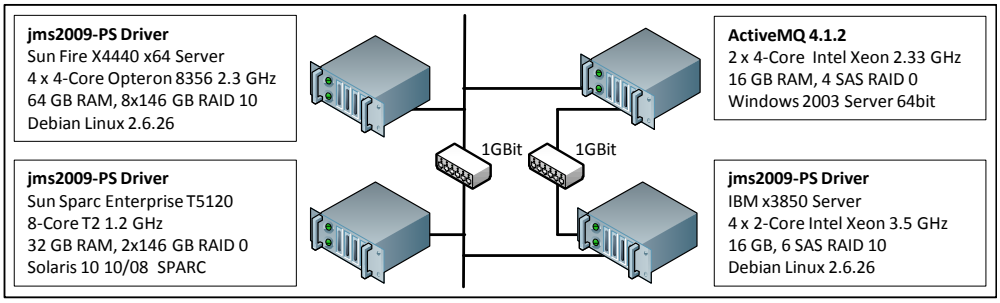


Figure 5.31: Experimental Environment

5.4.2 Test Scenarios

We studied three different scenarios which were identical in terms of the total number of messages sent and received for a given scaling factor (`BASE`). Transactions and persistent message delivery were configured as defined in the SPECjms2007 workload description[201]. The scenarios differ in the number of message destinations and destination types used for communication. Figure 5.32 illustrates the configurations used in the three scenarios for two of the message types: `order` messages sent from SMs to DCs and `orderConf` messages sent from DCs to SMs (cf. Table 5.11).

- **Scenario I (SPECjms2007-like Workload):** The workload is configured similar to the SPECjms2007 workload, i.e., it uses mainly queues for communication. Each location instance has its own queue for each message type and therefore there is no need for selectors.

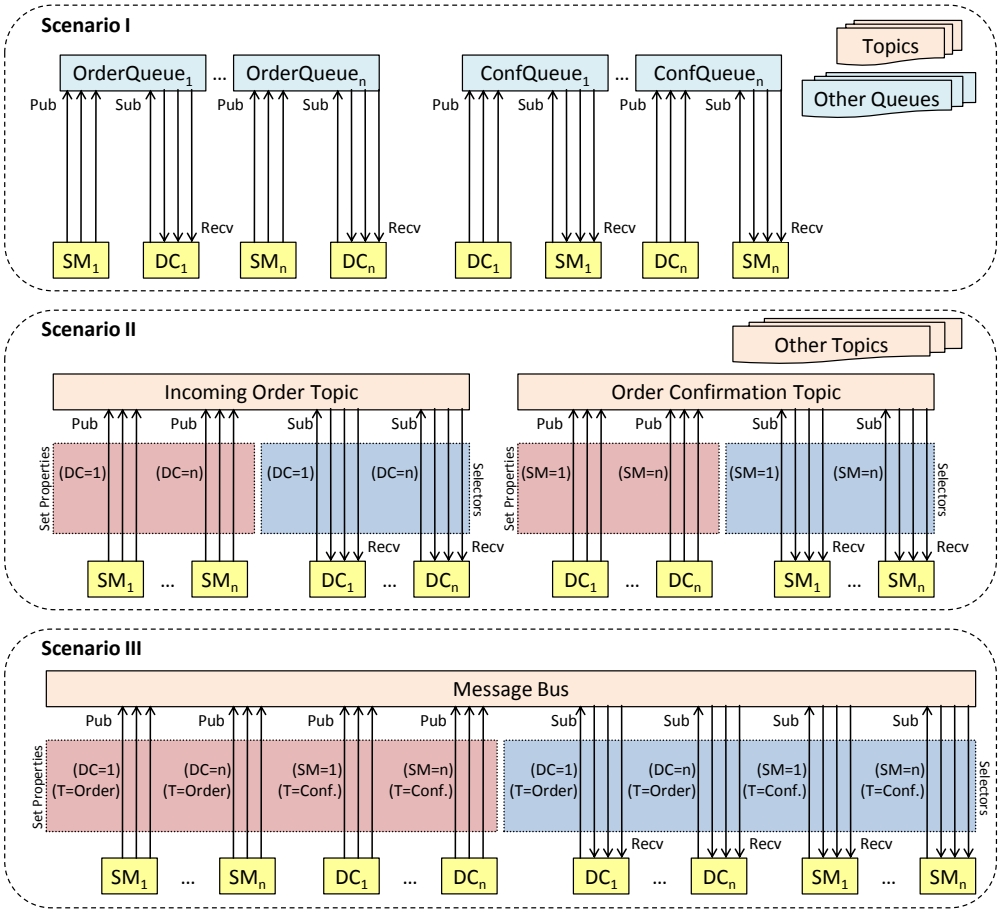
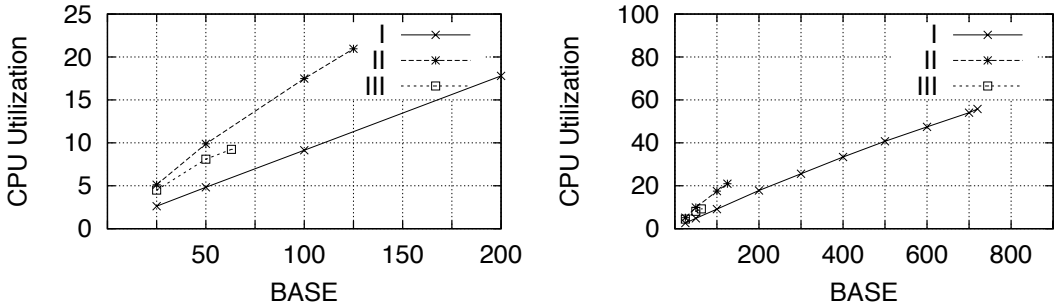


Figure 5.32: Considered Scenarios

- **Scenario II (Pub/Sub with Multiple Topics):** For each message type, a separate topic is used, i.e., the TD configuration parameter is set to `MessageType` (cf. Table 5.12).

- **Scenario III (Pub/Sub with Message Bus):** One topic is used for all messages, i.e., the TD configuration parameter is set to `Central` (cf. Table 5.12).

The three scenarios differ mainly in terms of the flexibility they provide. While Scenario I is easy to implement given that no properties or selectors are necessary, it requires a reconfiguration of the MOM server for each new location or message type since new queues have to be set up. In contrast, Scenarios II and III, which only use topics, provide more flexibility. In Scenario II, a reconfiguration of the MOM server is necessary only when introducing new message types. Scenario III doesn't require reconfiguration at all since a single topic (message bus) is used for communication. In addition, Scenarios II and III support one-to-many communication while the queue-based interactions in Scenario I are limited to one-to-one communication. One-to-many communication based on pub/sub allows to easily add additional message consumers, e.g., to maintain statistics about orders. On the other hand, the use of a limited number of topics in Scenarios II and III degrades the system scalability. As shown in the next section, the jms2009-PS benchmark allows to evaluate the trade-offs that different configurations provide in terms of flexibility, performance and scalability.



Scenario	Max Load	CPU/BASE	Avg. Dlv. Latency (ms)
I	720	0.077	123
II	125	0.168	1587
III	63	0.147	3235

Figure 5.33: Experimental Results

5.4.3 Experimental Results

Figure 5.33 presents the experimental results for the three scenarios described above. It shows the CPU utilization for increasing workload intensities (`BASE`), the maximum load that can be sustained by each scenario, the CPU time per unit of the BASE parameter and the average message delivery latency. The results show the scalability and performance of the three configurations as well as their costs in terms of CPU consumption. Scenario I scales up to BASE 720 and exhibits the lowest message delivery latency ($123ms$). The flexibility provided by Scenario II and III comes at the cost of much worse scalability and performance. The maximum load that can be sustained in Scenario II and Scenario III is respectively 6 and 12 times lower than that in Scenario I. Similarly, the average message delivery latency is about 13 times higher for Scenario II compared to Scenario I and about 26 times higher for Scenario III. Thus, the flexibility provided by Scenario II and III comes at a high price. This is due to two reasons:

1. The use of selectors leads to roughly two times higher CPU processing time per message (see Figure 5.33).

2. The use of topics for communication leads to synchronization delays.

Comparing Scenarios II and III reveals that the selector complexity in this case does not have a significant impact on the CPU processing time per message. What is much more significant is the number of topics used for communication. The single topic in Scenario III clearly leads to a scalability bottleneck and explosion of the message delivery latency. In the third scenario, the throughput was limited by the performance of a single CPU core.

Overall, the results show that topic-based communication using selectors is much more expensive than queue-based communication and, depending on the number of topics used, it limits the scalability of the system. We demonstrated how, by using jms2009-PS, the performance and scalability of different messaging workloads and configuration scenarios can be quantified. The high configurability of the benchmark allows us to tailor the workload to the user's requirements by customizing it to resemble a given application scenario. The user can then evaluate alternative ways to implement message communication in terms of their overhead, performance and scalability.

5.5 Concluding Remarks

In this chapter, we introduced SPECjms2007, the first industry standard benchmark for MOM, and discussed its goals, the business scenario it models and its internal component architecture. We presented a detailed workload characterization with the goal to help users understand the internal components of the workload and the way they are scaled. We proposed a novel methodology for performance evaluation of MOM platforms using standard benchmarks and discussed how the workload can be customized to exercise and evaluate selected aspects of MOM performance. Our workload analysis not only helps to better understand and interpret official benchmark results, but also provides an example of how to define a scalable workload configuration for evaluating selected performance and scalability aspects of MOM.

In addition, we presented a case study of a leading JMS platform, the WebLogic server, conducting an in-depth performance analysis of the platform under a number of different workload and configuration scenarios. Our analysis covered various workload aspects including P2P vs. pub/sub communication, persistent vs. non-persistent messaging, varying message sizes, number of message consumers, number of message producer threads, etc.

Finally, we illustrated how to adjust the SPECjms2007 standard workload to target different aspects of pub/sub communication and introduced jms2009-PS, a new benchmark for pub/sub-based messaging systems built on top of the SPECjms2007 standard workload. Overall, jms2009-PS allows the user to adjust the workload in terms of the number of topics, subscriptions, the number and type of selectors, and message delivery modes. In a case study of an open source middleware we analyzed alternative ways of implementing publish/subscribe communication in terms of their overhead, performance and scalability. The case study showed that the flexibility provided by topic-based publish-subscribe communication comes at a high price. The most critical factor affecting the system performance however was the number of topics used for communication. Having a low number of topics provides maximum flexibility, however, it introduces a scalability bottleneck due to the synchronization delays.

Chapter 6

Performance Modeling of EBS - Case Studies

In this chapter we present two case studies in which we apply our approach for workload characterization and performance modeling of event-based systems introduced in Chapter 4. In the first case study we use the SIENA [40] publish/subscribe system with a basic workload comprising a single message type. In the second more complex case study we model a realistic state-of-the-art event-driven system on a leading commercial middleware platform including all system layers. 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. In both case studies, we show how QPN models can be exploited for performance analysis and capacity planning in the software engineering process. The results demonstrate the effectiveness, practicality and accuracy of the proposed modeling and prediction approach, which provides a powerful tool for ensuring that systems are designed and sized to meet their QoS requirements. Furthermore, we apply our performance modeling patterns (Section 4.2) and make use of QPN extensions (Section 4.3) in the second case study.

6.1 DEBS Case Study

6.1.1 Scenario

We consider a scenario in which DEBS is used to manage the interactions among participants in a supermarket supply chain [202]. The participants involved are the supermarket company, its stores, its distribution centers and its suppliers. Since most of the interactions in the supply chain are asynchronous in nature, they can be implemented using DEBS. Some examples of services that can be handled by the system are supermarket order and shipment handling, inventory management in supermarkets and distribution centers, automated tracking of goods as they move through the supply chain and dissemination of new product announcements, price updates, special offers and discount information from the company headquarters to the supermarkets. Here we consider a simplified version of the SPECjms2007 scenario (see Section 5.1), in which the dissemination of *requests for quotes* are sent when goods in a distribution center are depleted and an order has to be placed to refill stock. A request for quote is published as an event in the system and it is automatically delivered to all suppliers that offer the respective types of goods. It is assumed that suppliers have subscribed to all events belonging to the product groups/categories they sell.

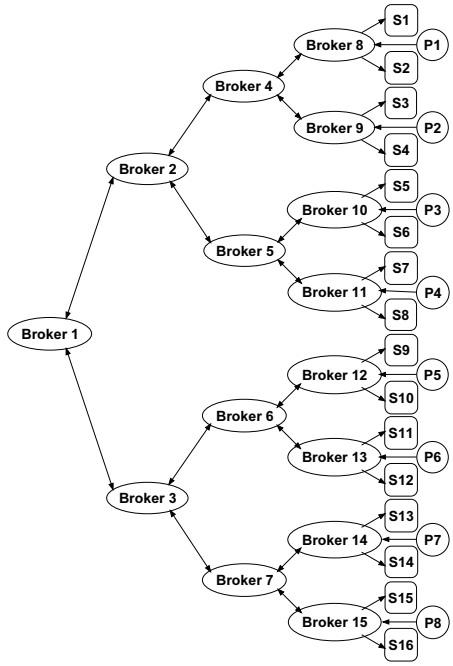


Figure 6.1: Broker Topology

6.1.2 Setup

We have implemented the dissemination of requests for quotes using the SIENA publish/subscribe system [40] enhanced with self-monitoring functionality. We instrumented the system to monitor and collect the event publication rates and routing probabilities needed for characterizing the workload. We implemented an interaction based on the SPECjms2007 scenario and used one of its message schemas (see Appendix **??**).

We used a hierarchical topology with 15 brokers, 8 publishers and 16 subscribers. Brokers were communicating via a Gigabit LAN. The deployment topology is depicted in Figure 6.1.

6.1.3 Experimental Results

Following our methodology presented in the Section 4.1, we first characterized the workload and then built a QPN model of the system and used it to predict the system performance under load. Given that the network utilization was very low, it was omitted from the model, assuming that the network service times did not have any significant impact on the overall system performance. We employed the QPME tool (Queueing Petrinet Modeling Environment) to build and analyze the model. We considered a number of workload and configuration scenarios varying the publication rates and the system topology. Tables 6.1 and 6.2 show the results for two of the scenarios we analyzed. The broker throughput and the event delivery latency for a randomly selected subscriber are shown. As we can see, the predictions are pretty accurate and reflect the real system behavior. The results for the rest of the subscribers as well as for the other scenarios we considered were of similar accuracy to the ones presented here. The model analysis was conducted using SimQPN [127] and took less than two minutes on standard PC hardware.

Broker	*Scenario 1*		*Scenario 2*	
	Model	*Measured*	*Model*	*Measured*
1	94.66	93.46	61.88	62.11
2	94.65	96.15	61.88	62.11
3	89.93	89.29	59.28	59.17
4	90.40	89.29	58.27	57.80
5	83.42	84.03	56.42	56.18
6	85.24	84.75	56.35	56.18
7	71.90	71.94	48.63	48.54
8	78.91	79.37	51.12	51.28
9	67.15	68.03	43.49	43.48
10	67.14	67.11	47.01	46.95
11	59.54	59.88	41.72	41.67
12	58.26	58.82	40.01	40.16
13	73.09	72.46	48.23	48.08
14	56.35	57.47	38.49	38.46
15	63.11	63.29	42.97	42.92

Table 6.1: Broker Throughput (msg. / sec)

Publisher	*Scenario 1*		*Scenario 2*	
	Model	*Measured*	*Model*	*Measured*
1	9.48	8.98	24.60	26.71
2	19.01	18.56	24.79	25.93
3	28.82	27.27	7.90	9.05
4	29.03	27.79	16.39	17.59
5	38.34	37.01	32.61	35.20
6	38.00	37.77	32.63	35.52
7	39.06	38.12	33.27	36.25
8	38.71	37.87	33.28	35.47

Table 6.2: Delivery Latency (ms)

6.1.4 Conclusions

Our case study demonstrated the effectiveness and practicality of our methodology presented in Section 4.1 in the context of a simple scenario. We developed a workload model for this scenario and used operational analysis techniques to characterize the system traffic and derive an approximation for the mean event delivery latency. Our model reflected the behavior of a SIENA publish/subscribe system with 15 brokers, 8 publishers and 16 subscribers and predicted the runtime behavior very well. We showed that our approach can be exploited for performance evaluation of DEBS.

6.2 Modeling SPECjms2007

6.2.1 Introduction

In this section, we present a case study of a state-of-the-art event-driven application deployed on a leading commercial MOM platform - the Oracle WebLogic Server Enterprise Edition. The

application we study is the SPECjms2007 standard benchmark (see Section 5.1). In this case study, we use the benchmark 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.

First, a detailed model of the benchmark based on queueing Petri nets is built in a step-by-step fashion. QPNs allow 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. To validate the approach, model forecasts are compared to measurements on the real system. The results are used to evaluate the effectiveness, practicality and accuracy of the proposed modeling and prediction approach.

By means of the proposed models we were able to predict the performance of the application accurately for scenarios under load conditions with up to 30,000 messages exchanged per second (up 4,500 transactions per sec.). To the best of our knowledge, no models of realistic systems of the size and complexity of the one considered here exist in the literature. The modeling technique can be exploited as a powerful tool for performance prediction and capacity planning during the software engineering lifecycle of event-driven applications.

Both analytical and simulation techniques for solving QPN models exist including product-form solution techniques and approximation techniques (see Section2.3). 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.

6.2.2 Modeling SPECjms2007

We now develop step-by-step a detailed performance model of SPECjms2007 and show how the model can be used to predict the benchmark performance for a given workload and configuration scenario. The model we develop is based on queueing Petri nets (see Section 2.3).

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 6.2. The number of tokens configured in the initial marking of place `BASE` is used to initialize the $BASE$ parameter of the vertical topology (see Section 5.1.4). 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 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

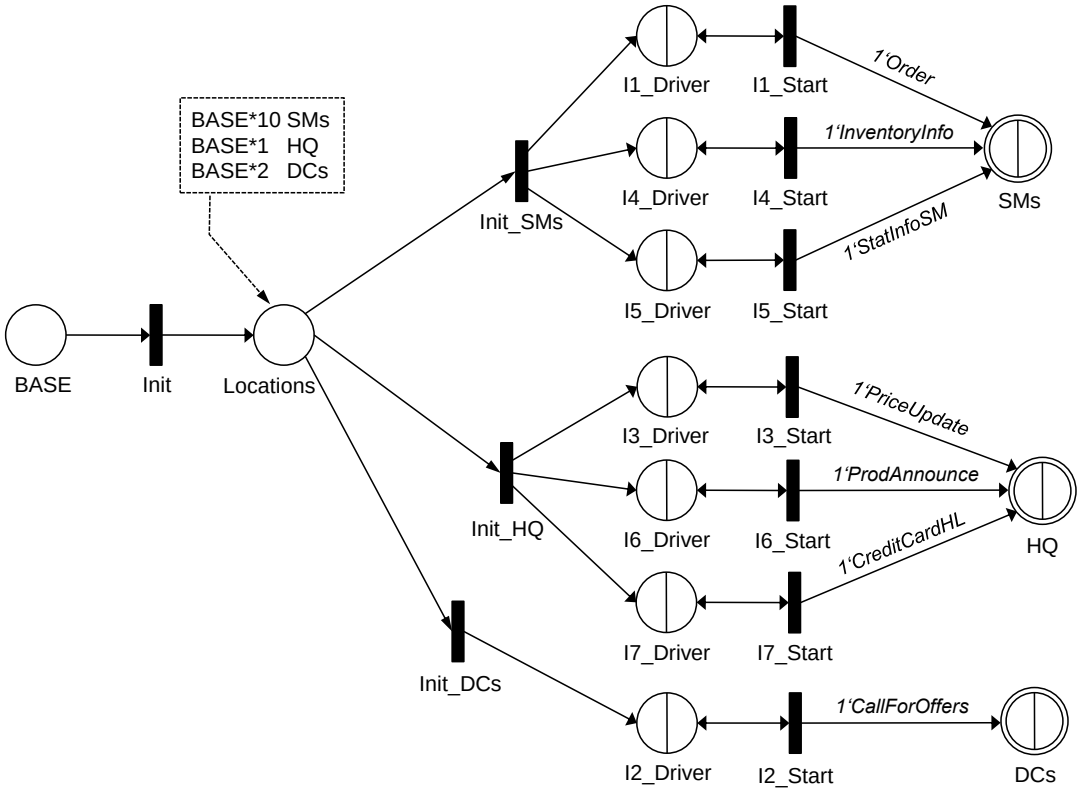


Figure 6.2: Model of Interaction Drivers

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, 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 6.2 has three firing modes corresponding to the three message sizes. The transition firing weights reflect the target message size distribution.

Modeling Interaction Workflows

We now model the interaction workflows. We start with Interactions 3 to 7 since they are simpler to model. Figure 6.3 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 transition `I3_2` which takes the `PriceUpdate` message as input and creates n `PriceUpdateN` messages representing the notification messages delivered to the subscribed SMs (where $n = 10$ for the vertical topology).

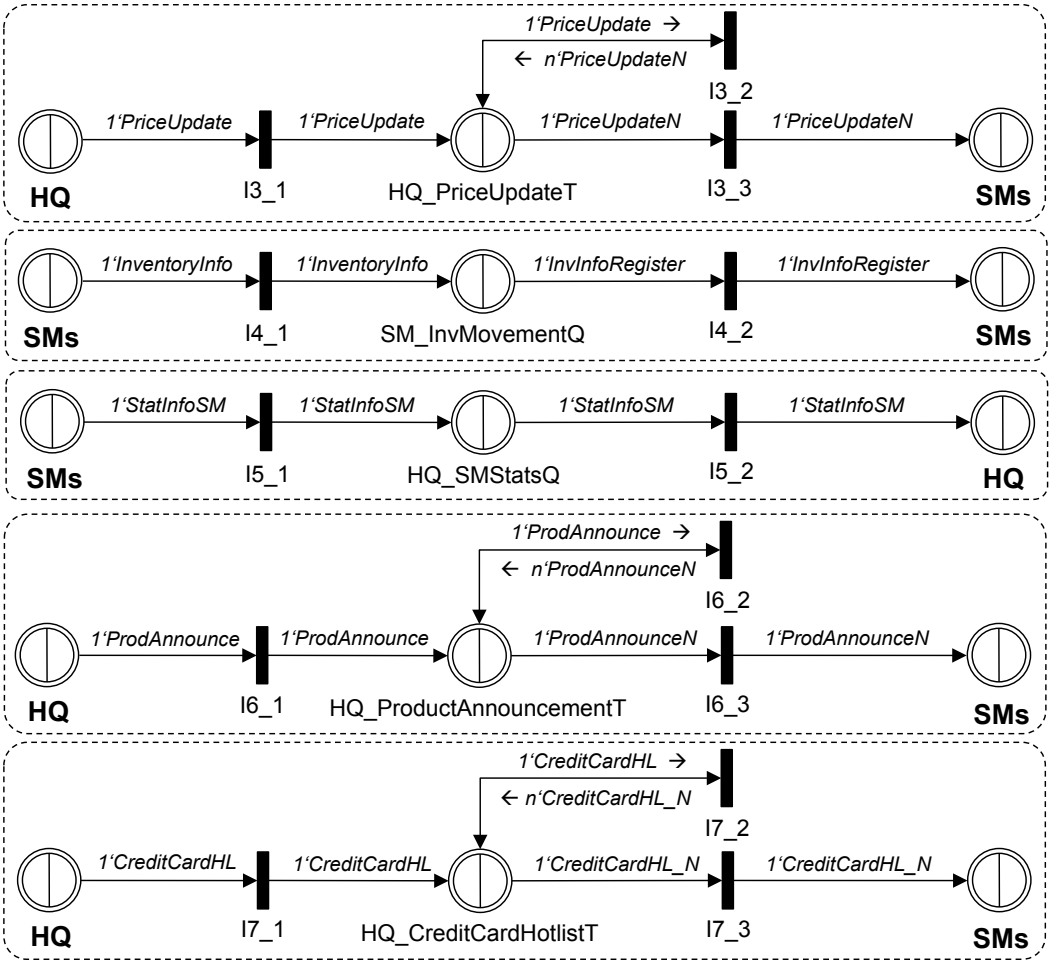


Figure 6.3: Models of Interactions 3, 4, 5, 6 and 7

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 6.4 and 6.5, 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.

Mapping of Logical to Physical Resources

The case study presented exploits the ability to share queues in multiple queueing places by decoupling the logical (software) and physical (hardware) layers of the modeled system. Therefore, the same logical model of the benchmark interactions can be easily customized to different deployment environments. The results presented in this case study can be seen as a validation

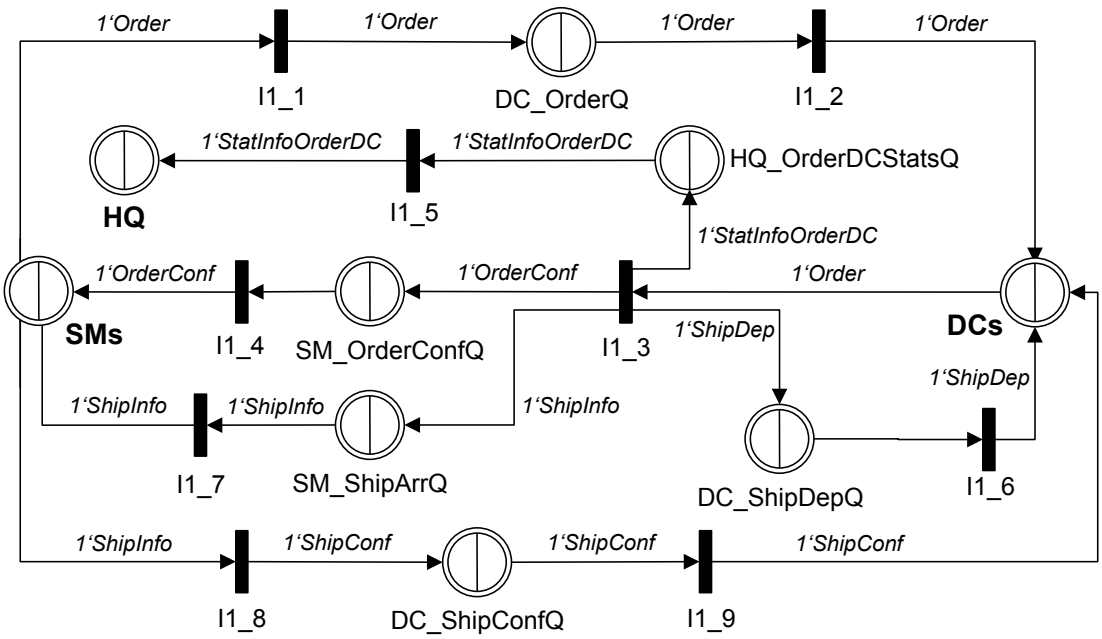


Figure 6.4: Model of Interaction 1

of the QPN extensions introduced in Section 4.3 to our modeling tools and analysis techniques.

By using subnet places to represent the MOM server(s) hosting the individual destinations and the clients (HQ, SMs, DCs and SPs), which exchange messages through the MOM infrastructure, we provide additional flexibility in choosing the level of detail at which the system components are modeled.

6.2.3 Experimental Evaluation

Experimental Environment

To evaluate the accuracy of the proposed modeling approach, we conducted an experimental analysis of the application in the environment depicted in Figure 5.19. Oracle WebLogic server 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 JRockit 1.5 JVM with 8GB of heap space. A RAID 0 disk array comprised of four disk drives was used for maximum performance. The WebLogic 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.

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. Given that all destinations are deployed on a single physical server, the three queueing places for each destination were mapped to three central

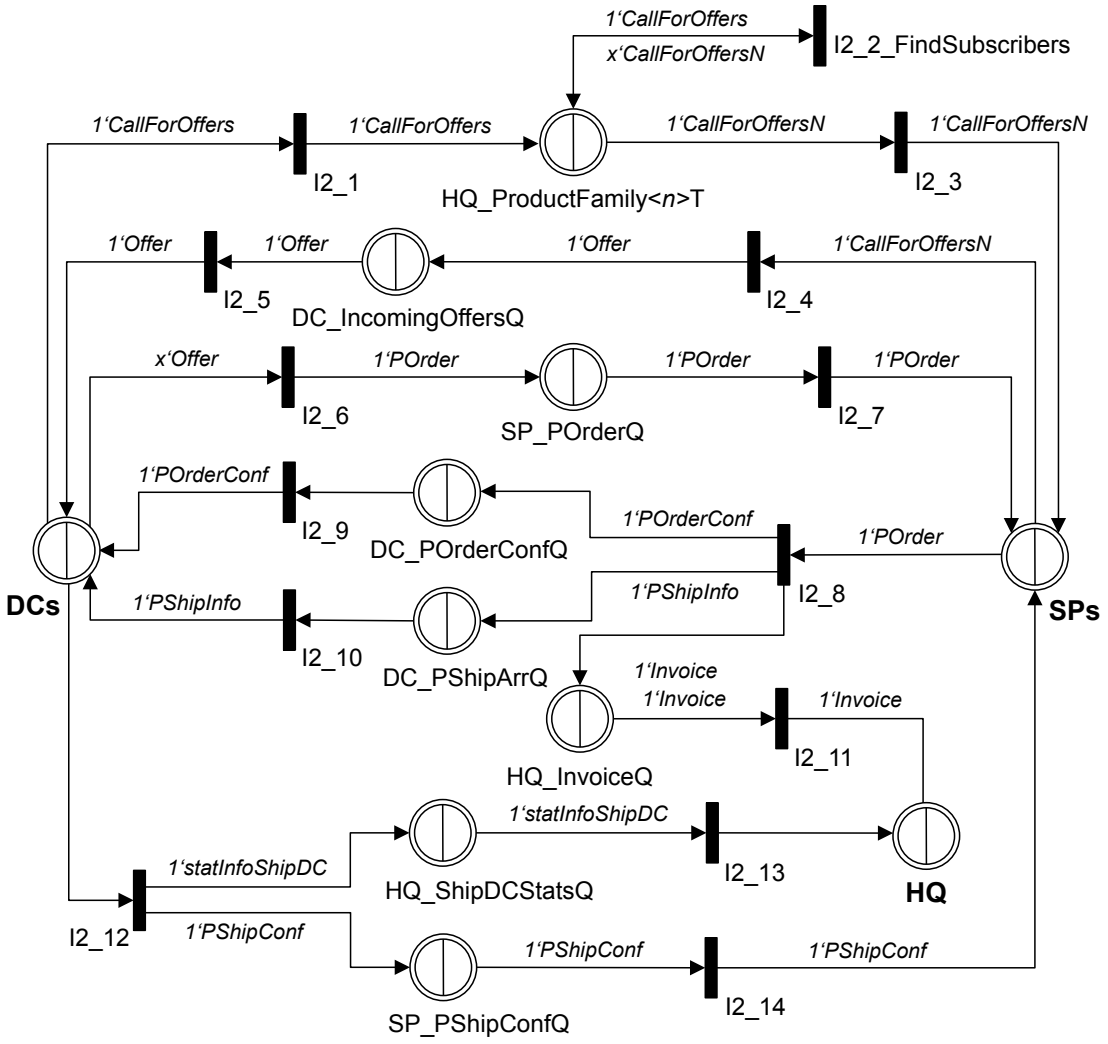


Figure 6.5: Model of Interaction 2

queues representing the respective physical resources of the Oracle WebLogic 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. 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. Since all instances of a given location type were deployed on the same client machine, 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 [128].

Considered Workload Scenarios

We consider several different scenarios that represent different types of messaging workloads. These scenarios stress different aspects of the MOM infrastructure including both workloads

	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 6.3: Scenario Transaction Mix

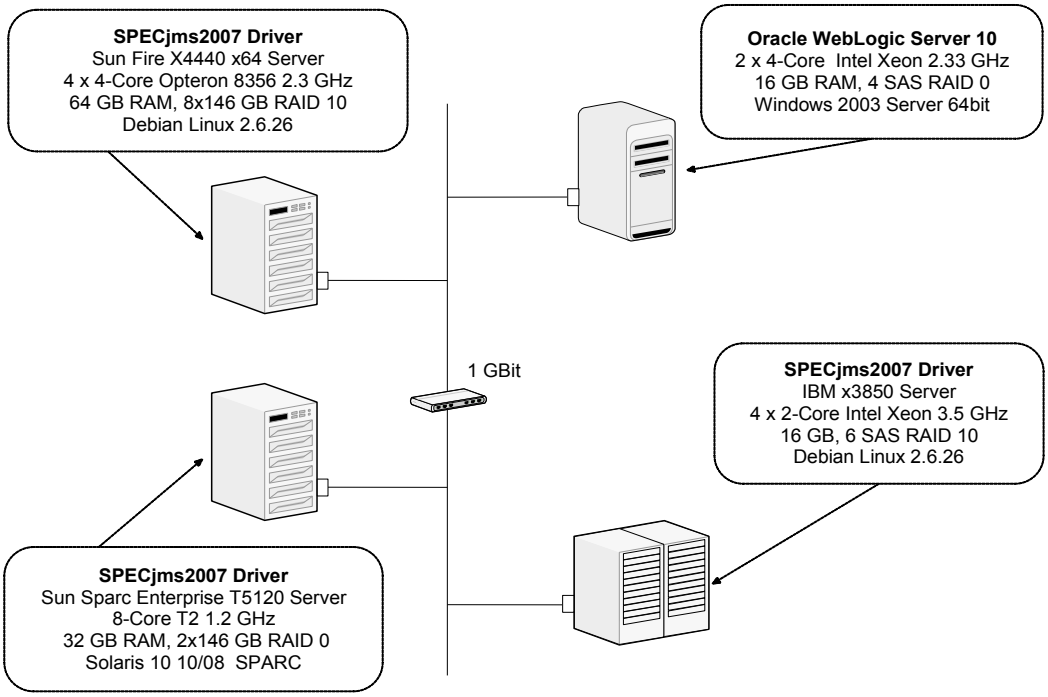


Figure 6.6: Experimental Environment

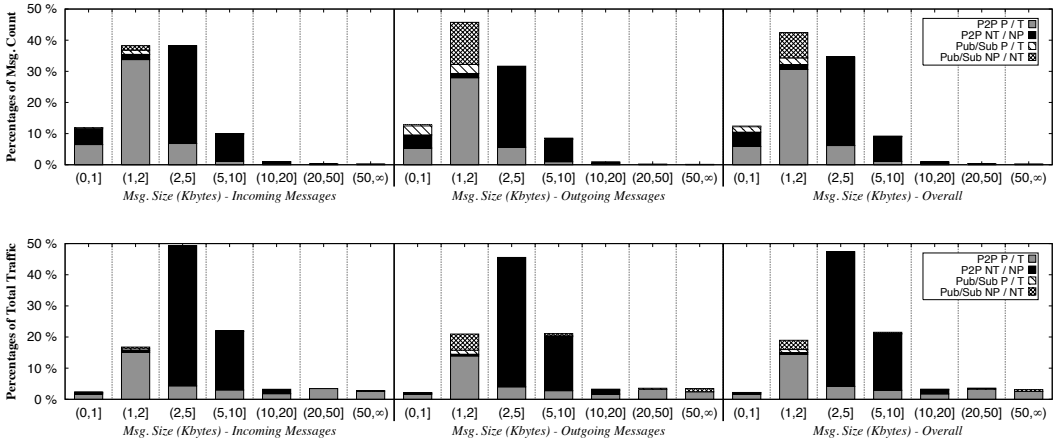
focused on point-to-point messaging as well as workloads focused on publish/subscribe. In each case, the model was analyzed using SimQPN [127] which took less than 5 minutes. We have intentionally slightly deviated from the standard vertical topology to avoid presenting performance results that may be 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.

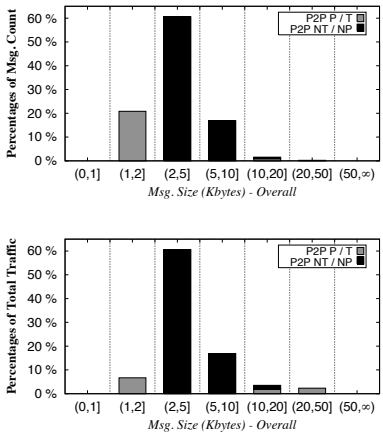
In Table 6.3 and Figure 6.7, we provide a detailed workload characterization of the three scenarios to illustrate the differences in terms of transaction mix and message size distribution.

Experimental Results

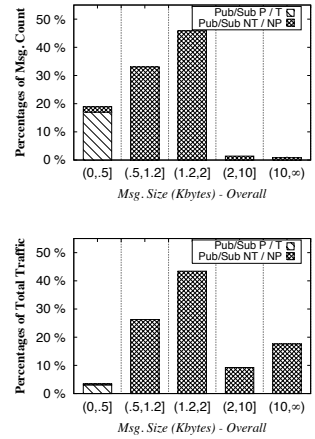
Figure 6.8 shows the predicted and measured CPU utilization of the MOM server for the considered customized vertical topology when varying the $BASE$ between 100 and 700. The total number of messages sent and received per second is shown. As we can see, the model predicts the server CPU utilization very 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 6.5. 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



(a) Scenario 1



(b) Scenario 2



(c) Scenario 3

Figure 6.7: Distribution of the Message Size

(a) Scenario 1

Input BASE	*Inter-action*	*Rate p. sec*	*Avg. Completion T (ms)* Model	Meas. (95% c.i.)	Deviation
300 *med. load*	1	228.57	10.24	10.17 +/- 0.68	0.7%
	2	64	13.28	15.10 +/- 0.71	12.0%
	3	15	3.16	3.49 +/- 0.41	9.4%
	4	486.49	2.64	2.76 +/- 0.31	4.3%
	5	1731.60	1.79	1.97 +/- 0.27	9.1%
	6	42.69	0.97	1.96 +/- 0.29	50.0%
	7	30.77	1.02	2.10 +/- 0.24	51.0%
550 *high load*	1	419.05	20.41	25.19 +/- 2.56	19.0%
	2	117.33	30.73	28.27 +/- 2.05	8.7%
	3	27.50	7.12	7.20 +/- 0.67	1.1%
	4	891.89	7.33	7.35 +/- 0.89	0.3%
	5	3174.60	4.95	6.52 +/- 1.13	24.0%
	6	78.27	4.01	3.26 +/- 0.26	23.0%
	7	56.41	4.05	3.67 +/- 0.34	10.3%

(b) Scenario 2

Input BASE	*Inter-action*	*Rate p. sec*	*Avg. Completion T (ms)* Model	Meas. (95% c.i.)	Deviation
600 *med. load*	4	972.97	2.65	2.66 +/- 0.04	1.0%
	5	3463.20	1.81	1.54 +/- 0.10	17.5%
800 *high load*	4	1297.30	3.49	3.75 +/- 0.17	6.9%
	5	4617.60	2.77	2.62 +/- 0.20	5.7%

(c) Scenario 3

Input BASE	*Inter-action*	*Rate p. sec*	*Avg. Completion T (ms)* Model	Meas. (95% c.i.)	Deviation
6000 *med. load*	3	300	3.74	3.22 +/- 0.09	16.1%
	6	853.89	0.81	0.95 +/- 0.23	14.7%
	7	615.38	1.02	1.31 +/- 0.35	22.0%
10000 *high load*	3	500	4.65	6.75 +/- 0.30	31.1%
	6	1423.15	1.42	1.44 +/- 0.07	1.4%
	7	1025.64	1.70	2.22 +/- 0.10	23.4%

Table 6.4: Detailed Results for Scenarios 1, 2 and 3

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%

Table 6.5: Relative Server CPU Load of Interactions

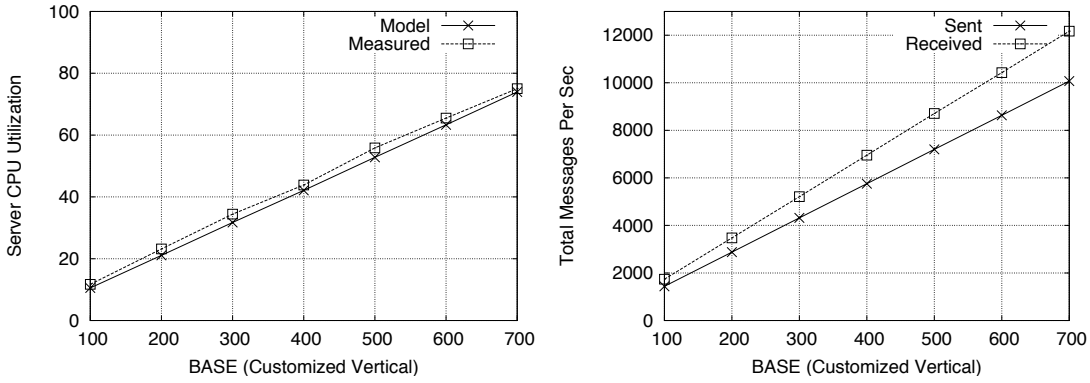
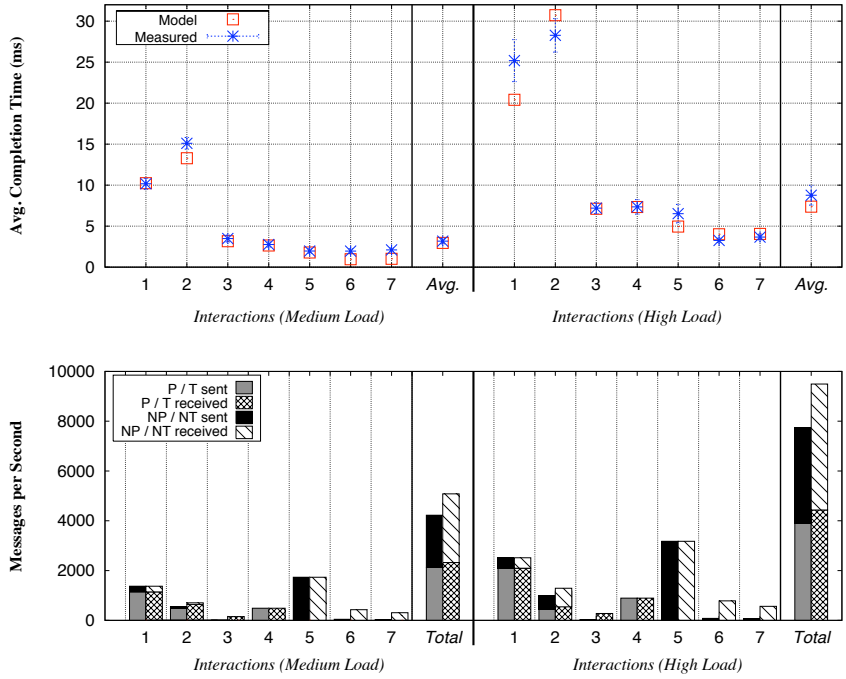


Figure 6.8: Server CPU Utilization and Message Traffic for Customized Vertical Topology

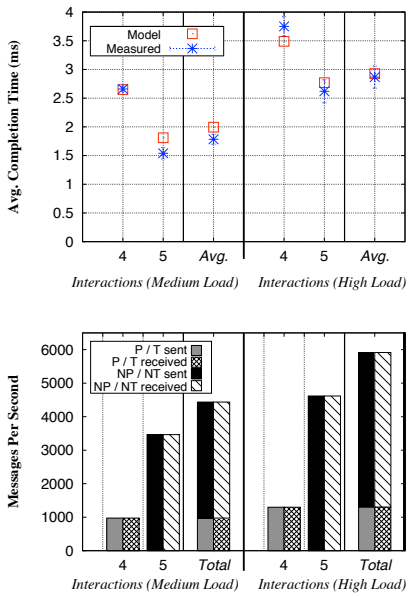
less traffic. This can be expected since the standard vertical topology that we used as a basis places the emphasis on point-to-point messaging (see Section 5.1.4). In the following, we study the three scenarios under different load intensities considering further performance metrics such as the interaction throughput and completion time.

The detailed results for the scenarios are presented in Table 6.4. Additionally, the interaction rates and the average interaction completion times are shown. For each scenario, we consider two workload intensities corresponding to medium and high load conditions configured using the *BASE* parameter. The *interaction completion time* is defined as the interval 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% except for 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 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 low. However, when considered as an absolute value, the error is still quite small.

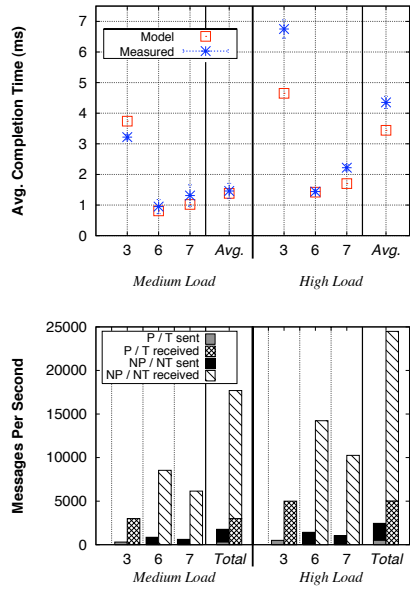
Figure 6.9 depicts the predicted and measured interaction completion times for the three scenarios as well as detailed information on how the total message traffic of each interaction is broken down into sent vs. received messages, on the one hand, and transactional (T) persistent (P) vs. non-transactional (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 due to the fact that each message sent in Interactions 3, 6 and 7 is delivered to 10 subscribers - one for each SM. The results in Figure 6.9 reveal the accuracy of the model when considering different types of messaging. 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 can be attributed 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 in the most cases



(a) Scenario 1



(b) Scenario 2



(c) Scenario 3

Figure 6.9: Model Predictions Compared to Measurements for Scenarios 1, 2 and 3

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.

6.2.4 Conclusion

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 model can be used as a powerful 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.

6.3 Concluding Remarks

In this chapter, we presented two novel case studies of representative state-of-the-art event-based systems showing how our proposed methodology can be exploited for workload characterization, performance modeling and prediction.

In the first case study, we applied our performance methodology to the SIENA publish/subscribe system and validated our approach by providing a workload characterization and performance model for a basic workload. In the second case study, we studied the SPECjms2007 standard benchmark deployed on a leading commercial middleware platform. A detailed model was developed in a step-by-step fashion and ways to customize the model for a particular deployment scenario were demonstrated. 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 in a representative environment comparing the model predictions against measurements on the real system. A number of different scenarios with varying workload intensity and interaction mixes 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 technique can be exploited as a powerful tool for performance prediction and capacity planning during the software engineering lifecycle of message-oriented event-driven systems.

Chapter 7

Conclusions and Outlook

With the growing popularity of EBS and their gradual adoption in mission critical areas, the need for novel techniques for benchmarking and performance modeling of EBS is increasing. Since their reliability is crucial for the whole IT infrastructure, a certain QoS level has to be ensured. The motivation for this thesis was to support the development and maintenance of EBS that meet certain QoS requirements. Given that EBS differ from traditional software in fundamental aspects such as their underlying communications paradigm, specific solutions and concepts are needed. System architects and deployers need tools and methodologies, which allow us to evaluate and forecast system performance and behavior in certain situations to identify potential performance problems and bottlenecks. Benchmarks and performance modeling techniques are usually the means of choice to answer these questions. However, no general performance modeling methodologies focusing on EBS have been published yet. Furthermore, there was a lack of test harnesses and benchmarks using representative workloads for EBS. Consequently, we focused on the development of a performance modeling methodology of EBS as well as on approaches to benchmark them. We summarize now our main contributions and proposed approaches.

To comprehend our contributions, an understanding of the fundamental ideas of EBS is essential. Therefore, we discussed our understanding of events in detail and introduced definitions for different kinds of events and related concepts. Generally spoken, EBS are software systems in which an observed event triggers a reaction. We evaluated the variety of underlying technologies with a focus on DEBS and MOMs and provided a survey of existing DEBS and MOM products and standards. In our review of existing work, we identified a lack of benchmarks and performance modeling approaches for EBS. To support a structural evaluation of benchmarks, we introduced five categories of requirements: (i) *Representativeness:* the benchmark has to be based on a representative workload. (ii) *Comprehensiveness:* exercise all platform features typically used in applications. (iii)*Focus:* place the emphasis on the technology server and minimize the impact of other services, e.g., databases. (iv) *Configurability:* provide a configurable tool for performance analysis. (v) *Scalability:* provide ways to scale the workload in a flexible manner.

None of the existing benchmarks met all our requirements. Therefore, we saw a strong need for independent and standardized benchmarks for EBS fulfilling the requirements. We launched a project inside the SPEC with the goal to develop the first industry standard benchmark for EBS. As underlying technology plattform we chose JMS. This was motivated by the fact that MOMs are widely used in industry and the quasi-standard for MOMs is JMS. Under the lead of TU Darmstadt a team with members from organizations such as IBM, Sun and BEA was formed. Our efforts resulted in the SPECjms2007 standard benchmark. Its main contributions were twofold:

Standard Workload: Based on the feedback of our industrial partners, we specified a comprehensive workload with different scaling options that fullfils all our requirements. It contains

several configuration parameters for evaluating selected performance and scalability aspects of MOM.

Benchmarking Framework: The benchmark was implemented using a newly developed complex framework. The framework offers many additional features, is highly configurable and easy to extend. Examples of its features are a comprehensive reporter and the availability of statistics at run time.

We introduced a methodology for the performance evaluation of MOMs using the SPECjms2007 standard benchmark. We showed how the workload can be customized to exercise and evaluate selected aspects of MOM performance to reflect a given target customer workload. Additionally, we demonstrated our methodology in a case study of a leading JMS platform, the WebLogic server, and conducted in-depth performance analyses of the platform for a number of different workload and configuration scenarios.

The SPECjms2007 business scenario was specified independently from the underlying technology. Therefore, its usage is not limited to a specific type of EBS. We illustrated how the standardized workload can be applied to other EBS using the example of *jms2009-PS*, a benchmark for publish/subscribe-based communication. This benchmark provides a flexible framework for performance analysis with a strong focus on research. Both benchmarks, SPECjms2007 and jms2009-PS, are used in several projects by industry and academia and, since they exercise MOM in a realistic way, they are also used as reference applications.

To the best of our knowledge, no work introducing a general methodology for modeling EBS has been published yet. As a consequence, we investigated whether and how traditional performance modeling approaches are suitable to model the specifica of EBS. We introduced a formal definition of EBS and their performance aspects, which allows us, among other things, to describe workload properties and routing behavior in a structured way. Resulting from our analysis of existing modeling techniques, we proposed a novel approach to characterize the workload and to model the performance aspects of EBS. We used operational analysis techniques to describe the system traffic and derived an approximation for the mean event delivery latency. We then showed how more detailed performance models based on QPNs could be built and used to provide more accurate performance prediction. We chose queueing Petri nets as performance modeling technique because of their modeling power and expressiveness. Our approach allows evaluating and predicting the performance of an EBS and provides detailed system models. It can be used for an in-depth performance analysis and to identify potential bottlenecks. A further contribution was a novel terminology for performance modeling patterns that we used to introduce eleven patterns targeting common aspects of event-based applications. These performance modeling patterns were the first ones published, which a) target EBS applications and b) use QPNs.

To additionally improve the modeling power of QPNs, we suggested several extensions of the standard QPNs, which allow building models in a more flexible and general way and address several limitations of QPNs. By introducing a level of abstraction, it is possible to distinguish between logical and physical layers in our models. This enables to flexibly map logical to physical resources and thus makes it easy to customize the model to a specific deployment. The different layers allow us to reuse one logical model in several physical models or to map several logical models to one physical model. Further, we addressed two limiting aspects of standard QPNs: constant cardinalities and the lack of transition priorities. By introducing non-constant cardinalities of transitions we increased the modeling flexibility and minimized the number of transition modes. The missing support of transition priorities in standard QPNs limits the control of the transition firing order. We addressed this restriction by incorporating priorities for transitions into QPNs and discussed several ways to implement them. Our extensions were integrated in the QPME / SimQPN software tools or are planned for the upcoming release.

Finally, we validated the approach in two case studies. We applied our methodology to model EBS and predicted their performance and system behavior under load successfully. As part of the

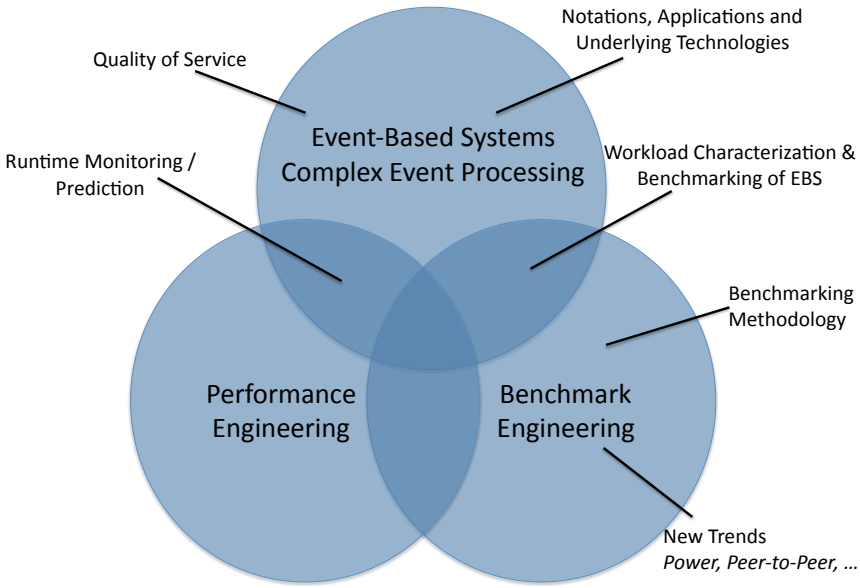


Figure 7.1: Open Research Issues

first case study we extended SIENA, a well-known DEBS, with a runtime measurement framework. We evaluated a system with 15 brokers, 8 publishers and 16 subscribers and predicted the runtime behavior including delivery latency for a basic workload with a single event type. In the second case study, we developed a comprehensive model of the complete SPECjms2007 workload including the persistent layer, point-to-point and publish/subscribe communication. We applied several of our performance modeling patterns. Furthermore, the workload was modeled using the proposed QPN extensions.

We evaluated its accuracy in a commercial middleware environment. To validate our modeling technique we investigated deployments of the benchmark in representative environments comparing the model predictions against measurements on the real systems. A number of different scenarios with varying workload intensity (up to 30,000 messages / 4,500 transaction per second) and interaction mixes were taken into account. By means of the proposed models we were able to predict the performance accurately. To the best of our knowledge, no models of realistic systems of the size and complexity of the one considered in this thesis exist in the literature.

The results of both case studies demonstrated the effectiveness and practicality of the proposed modeling and prediction methodology in the context of a real-world scenario. The advantage of the proposed approach is that it is both practical and general, and it can be readily applied for performance evaluation of DEBS and MOM. The technique can be exploited as a powerful tool for performance prediction and capacity planning during the software engineering lifecycle of message-oriented event-driven systems.

7.1 Ongoing and Future Work

Based on our experience and the findings of this work, we identified three interesting areas for future research:

- *Benchmark engineering*
- *Workload characterization and benchmarking of event-based systems*

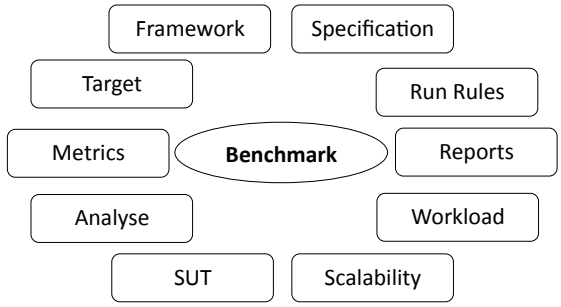


Figure 7.2: Benchmark Aspects

- *Self-adaptive QoS management in event-based systems*

Benchmark Engineering While developing SPECjms2007, we faced a lack of methodology that describes how to develop good and meaningful benchmarks. Since benchmark development has turned into a complex team effort, there is a need for a development methodology. Compared to traditional software, the development process has different goals and challenges. While historical benchmarks were only some hundreds lines long, modern benchmarks are composed of millions of lines of code. Therefore, new concepts and processes are needed which address the whole development and life-cycle management of benchmarks. We refer to them including benchmark methodology and measurement techniques with the term *benchmark engineering*. Furthermore, large scale, highly distributed systems are increasingly used in mainstream applications. However, for these systems traditional benchmarking approaches fail: how can we benchmark a system with 500,000 nodes? What does a typical workload look like and how does it scale? Future research should address these questions. As sample scenarios we propose peer-to-peer systems and highly distributed EBS.

Since it is not feasible to run benchmarks in a realistic environment with thousands of nodes, new methods are needed which allow us to benchmark large scale systems in a realistic way on limited resources. As a consequence, we see a need for research in the area of *simulated benchmarks*.

Workload Characterization and Benchmarking of Event-based Systems For the evaluation and analysis of event-based systems, standardized workloads and benchmarks are required. For MOMs we have contributed SPECjms2007 and jms2009-PS, but for EBS in general only a few benchmarks have been published to date. Standard workloads are not yet available for most domains. Therefore, we see a strong need to continue our efforts in the area of EBS workload characterization and benchmarking. We did a first step by demonstrating a prototype of a benchmark supporting AMQP in [12].

The next logical step is to develop a novel research benchmark for DEBS including a comprehensive workload specification. This poses several new unsolved challenges, on the one hand related to benchmarking methodology of highly distributed systems and workload characterization, and on the other related to the lack of standards. Given that generally accepted standards are missing, comprehensive and exact workload specification (including a description of QoS levels) are needed.

Self-Adaptive QoS Management in Event-based Systems We introduced a performance modeling methodology, which allows the characterization of the workload and to create models for a certain system state in time. However, many EBS are highly dynamic and their topology and workload are constantly changing. These changes have to be reflected in the models to keep

them accurate. Therefore, we propose to work on self-adaptive EBS based on the presented modeling methodology. Such systems should dynamically adjust their configuration to ensure that QoS requirements are continuously met. They should generate performance models at run-time based on monitoring data and, using these models, predict the system performance under the expected workload and adapt itself to guarantee specified QoS levels. Since performance analysis should be carried out on-the-fly, it is essential that the process of generating and analyzing the models is completely automated and efficient. Such a self-adaptive EBS requires novel techniques for system behavior prediction, monitoring and runtime measurements to supervise its own QoS status, to identify possible problems such as bottlenecks and to react to them autonomously, e.g., by automatically allocating resources. Another research challenge is to integrate all QoS attributes in the models and to optimize these models, e.g., regarding resource usage of the system given a certain QoS level.

Bibliography

[1] J. Abbott, K. B. Manrodt, and P. Moore. From Visibility to Action: Year 2004. Report on Trends and Issues in Logistics and Transportation. Technical report, March 2005. http://www.ca.capgemini.com/DownloadLibrary/req-uestfile.asp?ID=432.

[2] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser. MundoCore: A light-weight infrastructure for pervasive computing. *Pervasive and Mobile Computing*, 3(4):332–361, 2007.

[3] AMQP Working Group. AMQP Products. http://www.amqp.org/confluence/display/AMQP/AMQP+Products.

[4] AMQP Working Group. Implementing JMS over AMQP 1-0. https://www.amqp.org/confluence/display/AMQP/1-0+JMS+Mapping.

[5] AMQP Working Group. AMQP 1.0 DRAFT. Specification, 2009. https://www.amqp.org/confluence/display/AMQP/AMQP+Specification.

[6] Apache ActiveMQ. AMQP. http://activemq.apache.org/amqp.html.

[7] Apache ActiveMQ. OpenWire Version 2 Specification. http://activemq.apache.org/openwire-version-2-specification.html.

[8] Apache ActiveMQ. JMeter Performance Test, 2006. http://activemq.apache.org/jmeter-performance-tests.html.

[9] Apache Software Foundation. ActiveMQ . http://activemq.apache.org.

[10] Apache Software Foundation. Apache Qpid. http://qpid.apache.org/.

[11] S. Appel, K. Sachs, and A. Buchmann. Quality of service in event-based systems. In *Proceedings of the 22. GI-Workshop on Foundations of Databases (GvD)*, 2010.

[12] S. Appel, K. Sachs, and A. Buchmann. Towards Benchmarking of AMQP. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS'10)*. ACM, 2010.

[13] F. Araújo and L. Rodrigues. On QoS-Aware Publish-Subscribe. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems, Workshops (ICDCSW): Intl. Workshop on Distributed Event-Based Systems (DEBS'02)*. IEEE Computer Society, 2002.

[14] J. Bacon, J. Bates, R. Hayton, and K. Moody. Using Events to Build Distributed Applications. In *Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments (SDNE'95)*, page 148. IEEE Computer Society, 1995.

[15] J. Bacon, A. Beresford, D. Evans, D. Ingram, N. Trigoni, A. Guitton, and A. Skordylis. TIME: An open platform for capturing, processing and delivering transport-related data. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conf. (CCNC'08)*. IEEE, 2008.

[16] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic Support for Distributed Applications. *Computer*, 33(3):68–76, 2000.

[17] R. Baldoni, R. Beraldi, S. T. Piergiovanni, and A. Virgillito. On the modelling of publish/subscribe communication systems. *Concurrency and Computation: Practice and Experience*, 17(12):1471–1495, 2005.

[18] R. Baldoni, M. Contenti, S. T. Piergiovanni, and A. Virgillito. Modeling Publish/Subscribe Communication Systems: Towards a Formal Approach. In *Proceedings of the Eighth Intl. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS-2003)*. IEEE Computer Society, 2003.

[19] R. Baldoni and A. Virgillito. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey. Technical Report 15-05, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienzia", 2005.

[20] F. Bause. QN + PN = QPN - Combining Queueing Networks and Petri Nets. Technical report no.461, Department of CS, University of Dortmund, Germany, 1993.

[21] F. Bause. Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of the 5th Intl. Workshop on Petri Nets and Performance Models (PNPM'93)*. IEEE Computer Society, 1993.

[22] F. Bause and P. Buchholz. Queueing Petri Nets with Product Form Solution. *Performance Evaluation*, 32(4):265–299, 1998.

[23] F. Bause, P. Buchholz, and P. Kemper. QPN-Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets. In H. Beilner and F. Bause, editors, *Quantitative Evaluation of Computing and Communication Systems*, volume 977 of *Lecture Notes in Computer Science*. Springer, 1995.

[24] F. Bause, P. Buchholz, and P. Kemper. Integrating Software and Hardware Performance Models Using Hierarchical Queueing Petri Nets. In *Proceedings of the 9. ITG / GI - Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen (MMB'97)*, 1997.

[25] 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*, volume 794 of *Lecture Notes in Computer Science*. Springer, 1994.

[26] F. Bause and F. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg Verlag, 2002.

[27] S. Behnel, L. Fiege, and G. Mühl. On Quality-of-Service and Publish/Subscribe. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops: Fifth International Workshop on Distributed Event-based Systems (DEBS'06)*. IEEE Computer Society, 2006.

[28] U. Bellur and A. Shirabate. Performance Prediction and Physical Design of J2EE based Web applications. In *WSEAS CSCC 2004*, 2004.

[29] BiCEP Research Project. FINCoS Framework - User Guide Version 2.2. Technical report, Centre for Informatics and Systems of the University Coimbra, 2009.

[30] G. Biegel and V. Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Proceedings of 2nd IEEE Intl. Conference on Pervasive Computing and Communications (PerfCom'04)*. IEEE, 2004.

[31] P. Bizarro. BiCEP - Benchmarking Complex Event Processing Systems. In Chandy et al. [49].

[32] G. Bolch, S. Greiner, H. d. Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. Wiley-Interscience, 2005.

[33] C. Bornhövd, T. Lin, S. Haller, and J. Schaper. Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In *Proceedings of 30th International Conference on Very Large Data Bases (VLDB'04)*. Morgan Kaufmann, 2004.

[34] G. Bricconi, E. Tracanella, and E. D. Nitto. Issues in Analyzing the Behavior of Event Dispatching Systems. In *Proceedings of the 10th International Workshop on Software Specification and Design (IWSSD'00)*. IEEE Computer Society, 2000.

[35] I. Burcea and H.-A. Jacobsen. L-ToPSS - Push-oriented location-based services. In *Proceedings of the 4th VLDB Workshop on Technologies for E-Services (TES'03)*, 2003.

[36] K. Burton. Visualizing AMQP Broker Behavior with Clojure and Incanter. `http://asymmetrical-view.com/2009/06/02/incanter-amqp-benchmark.html`.

[37] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The 007 Benchmark. *SIGMOD Records*, 22(2), 1993.

[38] M. Carter. MP7G: JMS Performance with WebSphere MQ for Windows V6. `http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24010028`, 2005.

[39] N. Carvalho, F. Araujo, and L. Rodrigues. Scalable QoS-Based Event Routing in Publish-Subscribe Systems. In *Proceedings of the Fourth IEEE Intl. Symposium on Network Computing and Applications (NCA'05)*. IEEE Computer Society, 2005.

[40] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[41] A. Carzaniga and A. L. Wolf. A Benchmark Suite for Distributed Publish/Subscribe Systems. Technical report, Department of Computer Science, University of Colorado, 2002.

[42] S. Castelli, P. Costa, and G. P. Picco. Modeling the communication costs of content-based routing: the case of subscription forwarding. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems (DEBS'07)*. ACM, 2007.

[43] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 20, 2002.

[44] U. Cetintemel, J. Zimmermann, O. Ulusoy, and A. Buchmann. OBJECTIVE: A Benchmark for Object-Oriented Active Database Systems. *Journal of Systems and Software*, 45(1):31–43, 1999.

[45] S. Chakravarthy and Q. Jiang. *Stream Data Processing: A Quality of Service Perspective: Modeling, Scheduling, Load Shedding, and Complex Event Processing (Advances in Database Systems)*. Springer, 2009.

[46] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proceedings of the 20th Int. Conference on Very Large Data Bases (VLDB'94)*. Morgan Kaufmann, 1994.

[47] S. Chandrasekaran and M. Franklin. Streaming queries over streaming data. In *Proceedings of the 28th Int. Conference on Very Large Data Bases (VLDB'02)*. Morgan Kaufmann, 2002.

[48] K. M. Chandy. Event-driven applications: Costs, benefits and design approaches, 2006. Gartner Application Integration and Web Services Summit 2006.

[49] K. M. Chandy, O. Etzion, and R. von Ammon, editors. *Event Processing, 6.5. - 11.5.2007*, volume 07191 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[50] K. M. Chandy and W. Schulte. *Event Processing: Designing IT Systems for Agile Companies*. Mcgraw-Hill Professional, 2009.

[51] D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[52] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. ACM, 2000.

[53] S. Chen and P. Greenfield. QoS Evaluation of JMS: An Empirical Approach. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[54] M. Cilia. *An Active Functionality Service for Open Distributed Heterogeneous Environments*. PhD thesis, Technische Universität Darmstadt, 2002.

[55] M. Cilia, M. Antollini, C. Bornhövd, and A. Buchmann. Dealing with heterogeneous data in pub/sub systems: The Concept-Based approach. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS'04)*, Edinburgh, 2004.

[56] P. S. Corporation. Open Source FUSE Message Broker. `http://fusesource.com/products/enterprise-activemq/`.

[57] A. Corsaro, L. Querzoni, S. Scipioni, T. S. Piergiovanni, and A. Virgillito. Quality of Service in Publish/Subscribe Middleware. 8, July 2006.

[58] Crimson Consulting Group. High-Performance JMS Messaging - A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ, 2003. White Paper.

[59] G. Cugola and J. E. M. de Cote. On introducing location awareness in publish-subscribe middleware. In *Proceedings of 25th IEEE Intl. Conf. on Distributed Computing Systems Workshops(ICDCSW'05)*, pages 377–382. IEEE, 2005.

[60] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27:827–850, 2001.

[61] G. Cugola and G. P. Picco. REDS: a reconfigurable dispatching system. In *Proceedings of the 6th International Workshop on Software Engineering and Middleware (SEM'06)*. ACM, 2006.

[62] P. J. Denning and J. P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Computing Surveys*, 10(3):225–261, 1978.

[63] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale xml dissemination service. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB'04)*. Morgan Kaufmann, 2004.

[64] T. Dunn and R. Branagan. Websphere MQ Integrator for Windows NT and Windows 2000 V2.1. Performance Report. IBM U.K. Hursley Park Laboratories, 2002.

[65] G. Eisenhauer, K. Schwan, and F. Bustamante. Publish-Subscribe for High-Performance Computing. *IEEE Internet Computing*, 10(1):40–47, 2006.

[66] P. T. Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(1), 2007.

[67] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[68] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*. ACM, 2001.

[69] Fedora Project. AMQP Infrastructure. `http://fedoraproject.org/wiki/Features/AMQP_Infrastructure`.

[70] Fiorano. FioranoMQ. `http://www.fiorano.com/products/fmq/products_fioranofmq.php`.

[71] Fiorano Software Inc. JMS Performance Benchmarks - Illustrating the FioranoMQ 2007 Performance Advantage against SonicMQ 7.0, Tibco EMS 4.4 and ActiveMQ 4.1.0. white paper, `http://fiorano.best.vwh.net/whitepapers/jms_performance_comparison.htm`, 2007.

[72] Fiorano Software Inc. JMS Performance Comparison - Performance Comparison for Publish Subscribe Messaging . white paper, `http://www.fiorano.com/whitepapers/fmq/jms_performance_comparison.php?src=pr_mailblast_edm`, 2010.

[73] T. Fox. HornetQ Technical FAQ. `http://community.jboss.org/wiki/HornetQTechnicalFAQ`.

[74] T. Fromm. Ahkera. `http://t-lo.github.com/ahkera/`.

[75] Gallium. InterCOM DDS - product home page. `http://www.gallium.com/products/intercom.htm`.

[76] 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)*, volume 707 of *Lecture Notes in Computer Science*. Springer, 1993.

[77] H. J. Genrich and K. Lautenbach. System Modelling with High-Level Petri Nets. *Theoretical Computer Science*, 13:109–136, 1981.

[78] A. Geppert, M. Berndtsson, D. Lieuwen, and J. Zimmermann. Performance Evaluation of Active Database Management Systems Using the BEAST Benchmark. Technical report, University of Zurich, 1996.

[79] A. Geppert, S. Gatziu, and K. R. Dittrich. A Designer's Benchmark for Active Database Management Systems: oo7 Meets the BEAST. In *Proceedings of the Second International Workshop on Rules in Database Systems (RIDS'95)*, volume 985 of *Lecture Notes in Computer Science*. Springer, 1995.

[80] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann, 1993.

[81] R. Gruber, B. Krishnamurthy, and E. Panagos. The Architecture of the READY Event Notification Service. In *Proceedings of the 19th International Conference on Distributed Computing Systems Middleware Workshop*. IEEE Computer Society, 1999.

[82] R. E. Gruber, B. Krishnamurthy, and E. Panagos. READY: A High Performance Event Notification Service. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*. IEEE Computer Society, 2000.

[83] P. Guerrero, K. Sachs, M. Cilia, C. Bornhövd, and A. Buchmann. Pushing Business Data Processing Towards the Periphery. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*. IEEE Computer Society, 2007.

[84] P. E. Guerrero, K. Sachs, S. Butterweck, and A. Buchmann. Performance Evaluation of Embedded ECA Rule Engines: a Case Study. In *Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008)*, volume 5261 of *Lecture Notes in Computer Science*. Springer, 2008.

[85] X. Guo, X. Ding, H. Zhong, and J. Li. A New Performance Optimization Strategy for Java Message Service System. In *Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration (ICEIS'06)*, 2006.

[86] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, 1995.

[87] 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'08)*. ACM, 2008.

[88] J. Happe, D. Westermann, K. Sachs, and L. Kapov. Statistical inference of software performance models for parametric performance completions. In *Proceedings of 6th International Conference on the Quality of Software Architectures (QoSA 2010)*, volume 6093 of *Lecture Notes in Computer Science*. Springer, 2010.

[89] T. Harrison, D. Levine, and D. Schmidt. The design and performance of a real-time CORBA event service. In *Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '97)*. ACM, 1997.

[90] F. He, L. Baresi, C. Ghezzi, and P. Spoletini. Formal Analysis of Publish-Subscribe Systems by Probabilistic Timed Automata. In *Proceedings of the 27th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems (FORTE 2007)*, volume 4574 of *Lecture Notes in Computer Science*. Springer, 2007.

[91] P. Heidelberger and K. Trivedi. Queueing Network Models for Parallel Processing with Asynchronous Tasks. *IEEE Transactions on Computers*, 31(11):1099–1109, 1982.

[92] R. Henjes, M. Menth, and S. Gehrsitz. Throughput Performance of Java Messaging Services Using FioranoMQ. In *13th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB'06)*, Nürnberg, Germany, 2006.

[93] R. Henjes, M. Menth, and V. Himmler. Impact of Complex Filters on the Message Throughput of the ActiveMQ JMS Server. *Managing Traffic Performance in Converged Networks*, pages 192–203, 2007.

[94] R. Henjes, M. Menth, and V. Himmler. Throughput Performance of the BEA WebLogic JMS Server. *International Transactions on Systems Science and Applications*, 3(3), 2007.

[95] R. Henjes, M. Menth, and C. Zepfel. Throughput Performance of Java Messaging Services Using Sun Java System Message Queue. In *Proceedings of the 20th European Conference on Modelling and Simulation (ECMS'06)*, 2006.

[96] R. Henjes, M. Menth, and C. Zepfel. Throughput Performance of Java Messaging Services Using WebsphereMQ. In *Proceedings of the 26th International Conference on Distributed Computing Systems Workshops (ICDCSW'06)*. IEEE Computer Society, 2006.

[97] Hewlett-Packard. Benchmark report for webMethods Broker 6.5 on an HP Integrity server running HP-UX. Document: 4AA0-9960ENW, Revision 2, 2007. `http://h20195.www2.hp.com/V2/GetDocument.aspx?docname=4AA0-9960ENW`.

[98] P. Hintjens. Introduction to RestMS. `http://www.restms.org/article:introduction-to-restms`.

[99] P. Hintjens. Zyre. `http://www.zyre.com/`.

[100] A. Hinze and G. Buchanan. The challenge of creating cooperating mobile services: experiences and lessons learned. In *Proceedings of the 29th Australasian Computer Science Conference (ACSC2006)*, volume 48 of *CRPIT*. Australian Computer Society, 2006.

[101] 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'09)*. ACM, 2009. Keynote of A. Buchmann.

[102] A. Hinze and A. Voisard. Location- and Time-Based Information Delivery in Tourism. In *8th International Symposium in Spatial and Temporal Databases (SSTD'03)*, 2003.

[103] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[104] K. Huppler. The Art of Building a Good Benchmark. In *First TPC Technology Conference (TPCTC 2009)*, volume 5895 of *Lecture Notes in Computer Science*, 2009.

[105] IBM. WebSphere MQ. `http://www.ibm.com/software/integration/wmq/`.

[106] IBM Hursley. Performance Harness for Java Message Service, 2005. `http://www.alphaworks.ibm.com/tech/perfharness`.

[107] IBM T.J. Watson. The Gryphon Project. `www.research.ibm.com/distributedmessaging/gryphon.html`.

[108] IIT Software. SwiftMQ JMS Enterprise Messaging Platform. `http://www.swiftmq.com/`.

[109] iMatix Corporation. FastMQ Acquisition - press release. `http://www.imatix.com/press:fastmq-acquisition`.

[110] iMatix Corporation. OpenAMQ. `http://www.openamq.org/`.

[111] iMatix Corporation. ZeroMQ. `http://www.zeromq.org/`.

[112] iMatix Corporation. ZeroMQ FAQ. `http://www.zeromq.org/faq`.

[113] H. Jacobsen, A. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES Publish/Subscribe System, 2010.

[114] H. Jacobsen, V. Muthusamy, and G. Li. The PADRES Event Processing Network: Uniform Querying of Past and Future Events (Das PADRES Ereignisverarbeitungsnetzwerk: Einheitliche Anfragen auf Ereignisse der Vergangenheit und Zukunft). *it - Information Technology*, 51(5):250–261, 2009.

[115] M. A. Jaeger and G. Mühl. Stochastic Analysis and Comparison of Self-Stabilizing Routing Algorithms for Publish/Subscribe Systems. In *Proceedings of the13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*. IEEE Computer Society, 2005.

[116] Java News Desk. FioranoMQ 7.5 Offers Largest JMS Performance Gains Ever, Notes CEO Atul Saini. `http://java.sys-con.com/read/46130.htm`, 2004.

[117] JBoss. HornetQ. `http://www.hornetq.org`.

[118] JBoss. JBoss JMS New Performance Benchmark, 2006.

[119] K. Jensen. Coloured petri nets and the invariant-method. *Theoretical Computer Science*, 14:317–336, 1981.

[120] K. Jensen. How to find invariants for coloured petri nets. In *Proceedings on Mathematical Foundations of Computer Science*, volume 118 of *Lecture Notes in Computer Science*. Springer, 1981.

[121] K. Jensen and G. Rozenberg, editors. *High-level Petri nets: theory and application*. Springer, 1991.

[122] Y. Jin, S. D. Urban, and S. W. Dietrich. Extending the OBJECTIVE Benchmark for Evaluation of Active Rules in a Distributed Component Integration Environment. *Journal of Database Management*, 17(4):47–69, 2006.

[123] S. Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction*. PhD thesis, Technische Universität Darmstadt, 2005.

[124] S. Kounev. J2EE Performance and Scalability - From Measuring to Predicting. In *Proceedings of the SPEC Benchmark Workshop 2006 (SPEC'06)*. SPEC, 2006.

[125] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, July 2006.

[126] S. Kounev and A. Buchmann. Improving Data Access of J2EE Applications by Exploiting Asynchronous Processing and Caching Services. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*. Morgan Kaufmann, 2002.

[127] S. Kounev and A. Buchmann. SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4-5):364–394, May 2006. doi:10.1016/j.peva.2005.03.004.

[128] S. Kounev and C. Dutz. QPME - A Performance Modeling Tool Based on Queueing Petri Nets. *ACM SIGMETRICS Performance Evaluation Review (PER), Special Issue on Tools for Computer Performance Modeling and Reliability Analysis*, Jan. 2009.

[129] 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.

[130] S. Kounev and K. Sachs. SPECjms2007 - A novel benchmark and performance analysis framework for message-oriented middleware, March 2008. DEV2DEV Article, O'Reilly Publishing Group.

[131] S. Kounev and K. Sachs. Benchmarking and Performance Modeling of Event-Based Systems. *it - Information Technology*, 51(5):262–269, oct 2009.

[132] 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.

[133] J. Kramer. Advanced message queuing protocol (AMQP). *Linux Journal*, 2009(187), 2009.

[134] Krissoft Solutions. JMS Performance Comparison. `http://www.fiorano.com/comp-analysis/jms_perf_report.htm`, 2006.

[135] D. Kuo and D. Palmer. Automated Analysis of Java Message Service Providers. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 1–14, London, UK, 2001. Springer-Verlag.

[136] K.-D. Lange. Identifying Shades of Green: The SPECpower Benchmarks. *Computer*, 42(3):95–97, 2009.

[137] 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.

[138] C. Liebig, M. Cilia, and A. Buchmann. Event composition in time-dependent distributed systems. In *Proceedings of the 4th Intl. Conference on Cooperative Information Systems (CoopIŚ99)*. IEEE Computer Society, 1999.

[139] H. Liu and H. Jacobsen. A-TOPSS: a publish/subscribe system supporting approximate matching. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. Morgan Kaufmann, 2002.

[140] L. Liu, C. Pu, and W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[141] Y. Liu and I. Gorton. Performance Prediction of J2EE Applications Using Messaging Protocols. In G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. A. Szyperski, and K. C. Wallnau, editors, *Proceedings of the 8th International Symposium on Component-Based Software Engineering (CBSE 2005)*, volume 3489 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[142] Y. Liu and B. Plale. Survey of publish subscribe event systems, 2003. Technical Report TR574, Indiana University.

[143] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[144] D. Luckham and R. Schulte. Event Processing Glossary - Version 1.1, 2009.

[145] P. Maheshwari and M. Pang. Benchmarking message-oriented middleware: TIB/RV versus SonicMQ. *Concurrency and Computation: Practice and Experience*, 17(12):1507–1526, 2005.

[146] B. McCormick and L. Madden. Open architecture publish subscribe benchmarking. In *Proceedings of the OMG Real-Time and Embedded Systems Workshop*, 2005.

[147] D. A. Menascé and H. Gomaa. A Method for Desigh and Performance Modeling of Client/Server Systems. *IEEE Transactions on Software Engineering*, 26(11), Nov. 2000.

[148] M. R. N. Mendes, P. Bizarro, and P. Marques. A framework for performance evaluation of complex event processing systems. In *Proceedings of the Second International Conference on Distributed Event-based Systems (DEBS '08): Demonstration Session*. ACM, 2008.

[149] M. R. N. Mendes, P. Bizarro, and P. Marques. A Performance Study of Event Processing Systems. In R. O. Nambiar and M. Poess, editors, *First TPC Technology Conference (TPCTC 2009)*, volume 5895 of *Lecture Notes in Computer Science*. Springer, 2009.

[150] M. Menth and R. Henjes. Analysis of the Message Waiting Time for the FioranoMQ JMS Server. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS'06)*, 2006.

[151] M. Menth, R. Henjes, C. Zepfel, and S. Gehrsitz. Throughput performance of popular JMS servers. *SIGMETRICS Performance Evaluation Review*, 34(1):367–368, June 2006.

[152] Microsoft. Microsoft Message Queuing. `http://www.microsoft.com/windowsserver2003/technologies/msmq/default.mspx`.

[153] MilSOFT. MilSOFT DDS Middleware. `http://dds.milsoft.com.tr/en/dds-home.php`.

[154] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, TU Darmstadt, 2002. http://elib.tu-darmstadt.de/diss/000274/.

[155] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[156] G. Mühl, A. Schröter, H. Parzyjegla, S. Kounev, and J. Richling. Stochastic Analysis of Hierarchical Publish/Subscribe Systems. In *Proceedings of the 15th International Euro-Par Conference (Euro-Par 2009)*, volume 5704 of *Lecture Notes in Computer Science*. Springer, 2009.

[157] MuleSoft. Mule MQ. `http://www.mulesoft.com/mule-mq-high-performance-low-latency-JMS-messaging-for-the-enterprise`.

[158] N. Mulyar and W. von der Aalst. Patterns in Colored Petri Nets. Technical report, BETA Working Paper Series, WP 139, Eindhoven University of Technology, Eindhoven , 2005.

[159] 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.

[160] my-Channels. Nirvana Enterprise Messaging. `http://www.my-channels.com/products/nirvana/enterprise/`.

[161] 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.

[162] Object Computing Inc. . OpenDDS. http://www.opendds.org/index.html.

[163] Object Management Group (OMG). Event Service Specification Version 1.2 formal/04-10-02. http://www.omg.org/cgi-bin/doc?formal/2004-10-02, 2004.

[164] Object Management Group (OMG). Notification Service Specification Version 1.1 formal/04-10-11. http://www.omg.org/cgi-bin/doc?formal/2004-10-11, 2004.

[165] Object Management Group (OMG). Notification/JMS Interworking Service, version 1.0. http://www.omg.org/technology/documents/formal/notification_jms.htm, 2004.

[166] Object Management Group (OMG). Data Distribution Service for Real-time Systems Version 1.2 formal/07-01-01. http://www.omg.org/technology/documents/formal/data_distribution.htm, 2007.

[167] J. O'Hara. Toward a Commodity Enterprise Middleware. *ACM Queue*, 5(4):48–55, 2007.

[168] Oracle. Open Message Queue. https://mq.dev.java.net/.

[169] Oracle. Oracle Data Replication and Integration. http://www.oracle.com/technology/products/dataint/index.html.

[170] Oracle. Sun GlassFish Message Queue. http://www.sun.com/software/products/message_queue/index.xml.

[171] Oracle. WebLogic. http://www.oracle.com/us/products/middleware/application-server/index.htm.

[172] OW2 Consortium. JORAM: Java (TM) Open Reliable Asynchronous Messaging. http://joram.ow2.org/.

[173] S. Pallickara and G. Fox. NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware'03)*, volume 2672 of *Lecture Notes in Computer Science*. Springer, 2003.

[174] N. Paton, editor. *Active Rules in Database Systems*. Springer, New York, 1999.

[175] J. Pereira, F. Fabret, H.-A. Jacobsen, F. Llirbat, and D. Shasha. WebFilter: A High-throughput XML-based Publish and Subscribe System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 723–724. Morgan Kaufmann, 2001.

[176] 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.

[177] R. Pettit IV and H. Gomaa. 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.

[178] R. Pettit IV and H. Gomaa. 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.

[179] P. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambridge, Feb. 2004. Technical report. UCAM-CL-TR-590.

[180] P. Pietzuch, D. Eyers, S. Kounev, and B. Shand. Towards a Common API for Publish/Subscribe. In *Proceedings of the Inaugural International Conference on Distributed Event-Based Systems (DEBS'07)*. ACM, June 2007.

[181] Prism Tech. OpenSplice DDS. `http://www.opensplice.com`.

[182] Prism Tech. DDS Touchstone Benchmark Suite - User Guide, 2008. `http://sourceforge.net/projects/dds-touchstone`.

[183] Progress Software Corporation. SonicMQ. `http://web.progress.com/en/sonic/sonicmq.html`.

[184] Rabbit Technologies Ltd. RabbitMQ. `http://www.rabbitmq.com/`.

[185] Real-Time Innovations. RTI Data Distribution Service (DDS). `http://www.rti.com/products/dds/index.html`.

[186] Real-Time Innovations. RTI Message Service. `http://www.rti.com/products/jms/index.html`.

[187] Real-Time Innovations. RTI Data Distribution Service - Performance and Scalability Benchmarks: C++ on Linux, 2008. `http://www.rti.com/products/dds/benchmarks-cpp-linux.html`.

[188] Red Hat. Red Hat Enterprise MRG. `http://www.redhat.com/mrg/`.

[189] A. Rhoades, G. Schrader, and P. Poulin. Benchmarking Publish/Subscribe Middleware for Radar Applications. In *Proceedings of the Eleventh Annual High Performance Embedded Computing (HPEC'07)*, 2007.

[190] A. Rindos, M. Loeb, and S. Woolet. A performance comparison of IBM MQseries 5.2 and Microsoft Message Queue 2.0 on Windows 2000. IBM SWG Competitive Technical Assessment, Research Triangle Park, NC, 2001.

[191] Roberto Chinnici and Bill Shannon. Java Platform, Enterprise Edition (Java EE) Specification, v6. Technical report, 2009.

[192] J. Robie. Method and an apparatus to deliver messages between applications - patent application. Application number: 11/897,607, Publication number: US 2009/0063418 A1, Filing date: Aug 31, 2007.

[193] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: Content-based Filtering and Aggregation of Blogs and RSS Feeds. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'07)*, 2007.

[194] D. Rosenblum and A. Wolf. A design framework for internet-scale event observation and notification. *SIGSOFT Softw. Eng. Notes*, 22(6):344–360, 1997.

[195] A. I. T. Rowstron, A. M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Networked Group Communication, Third International COST264 Workshop (NGC'01)*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2001.

[196] K. Sachs. Evaluation of Performance Aspects of the SAP Auto-ID Infrastructure. Master's thesis, Technische Universität Darmstadt, 2004.

[197] 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: BenchmarX'10*, Lecture Notes in Computer Science. Springer, 2010.

[198] K. Sachs and S. Kounev. Kaffeekunde - SPECjms2007 misst Message Oriented Middleware. *iX - Magazin für professionelle Informationstechnik*, 2008(2), February 2008.

[199] K. Sachs, S. Kounev, S. Appel, and A. Buchmann. A Performance Test Harness For Publish/Subscribe Middleware. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance 2009): Demo Competition*. ACM, 2009.

[200] K. Sachs, S. Kounev, S. Appel, and A. Buchmann. Benchmarking of Message-Oriented Middleware. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS 2009)*. ACM, 2009.

[201] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation*, 66(8):410–434, Aug 2009.

[202] K. Sachs, S. Kounev, J. Bacon, and A. P. Buchmann. Workload Characterization of the SPECjms2007 Benchmark. In *Proceedings of the Fourth European Performance Engineering Workshop (EPEW'07)*, volume 4748 of *Lecture Notes in Computer Science*. Springer, 2007.

[203] K. Sachs, S. Kounev, M. Carter, and A. Buchmann. Designing a Workload Scenario for Benchmarking Message-Oriented Middleware. In *Proceedings of the 2007 SPEC Benchmark Workshop (SPEC'07)*. SPEC, January 2007.

[204] SAP. SAP NetWeaver. `http://www.sap.com/platform/netweaver/`.

[205] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *AUUG 97 Conference*, 1997.

[206] T. Sivaharan, G. S. Blair, and G. Coulson. GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing. In *Proceedings of the OnTheMove (OTM) Conferences*, volume 3760 of *Lecture Notes in Computer Science*. Springer, 2005.

[207] T. Sivaharan, G. S. Blair, and G. Coulson. GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing. In *Proceedings of the OnTheMove (OTM) Conferences*, volume 3760 of *Lecture Notes in Computer Science*, pages 732–749. Springer, 2005.

[208] A. Slominski, Y. Simmhan, A. Rossi, M. Farrellee, and D. Gannon. XEVENTS/XMESSAGES: Application Events and Messaging Framework for Grid. Technical report, Indiana University Computer Science Department, 2002.

[209] Software AG. webMethods Broker. `http://www.softwareag.com/corporate/products/wm/enterprise_integration/broker/overview/default.asp`.

[210] Sonic Software Corporation. JMS Performance Comparison: SonicMQ(R) vs TIBCO Enterprise(TM) for JMS, 2003. White Paper, `http://www.onwhitepapers.com/redirect.php?wid=4A0D2BDBBE89205241534CCB0AA8ED56`.

[211] Sonic Software Corporation. SonicMQ(R) 5.0.2 Outperforms TIBCO Enterprise(TM) for JMS 3.1.0 in Nine Key Publish/Subscribe Benchmark Tests. press release, http://goliath.ecnext.com/coms2/gi_0199-3273161/SonicMQ-is-Faster-Than-TIBCO.html, 2003.

[212] Sonic Software Corporation. Benchmarking E-Business Messaging Providers, 2004. White Paper, http://www.onwhitepapers.com/redirect.php?wid=4A0ABD6CBE8920524153D95D6F02C48C.

[213] Sonic Software Corporation. Sonic Test Harness - home pahe, 2005. http://communities.progress.com/pcom/docs/DOC-29828.

[214] Standard Performance Evaluation Cooperation (SPEC). SPECjms2007 Benchmark Documentation. http://www.spec.org/jms2007/.

[215] Standard Performance Evaluation Cooperation (SPEC). SPECjms2007 Results Published by SPEC. http://www.spec.org/jms2007/results/.

[216] Stomp Project. Stomp Protocol Specification, Version 1.0. http://stomp.codehaus.org/Protocol.

[217] Stomp Project. StompConnect. http://stomp.codehaus.org/StompConnect.

[218] R. E. Strom, G. Banavar, T. D. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. C. Sturman, and M. Ward. Gryphon: An Information Flow Based Approach to Message Brokering. In *Proceedings of the 9th International Symposium on Software Reliability Engineering (ISSRE'98)*, 1998.

[219] H. Subramoni, G. Marsh, S. Narravula, P. Lai, and D. Panda. Design and Evaluation of Benchmarks for Financial Applications using Advanced Message Queuing Protocol (AMQP) over InfiniBand. In *Workshop on High Performance Computational Finance at Super Computing (Workshops on SC08)*, Nov. 2008.

[220] Sun Microsystems. Sun and Sonic Software Demonstrate Outstanding Performance Running SonicMQ on Sun's Solaris 10 OS for x86 Platform. Press release, 2004.

[221] Sun Microsystems, Inc. Java Message Service (JMS) Specification - Version 1.1. Technical report, 2002.

[222] Sun Microsystems, Inc. Java Platform, Enterprise Edition (Java EE) Specification, v5, May 2006.

[223] H. Taylor, A. Yochem, L. Phillips, and F. Martinez. *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*. Addison-Wesley Professional, 2009.

[224] F. Tian, B. Reinwald, H. Pirahesh, T. Mayr, and J. Myllymaki. Implementing a scalable XML publish/subscribe system using relational database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*. ACM, 2004.

[225] TIBCO. TIBCO Enterprise Message Service. http://www.tibco.com/software/messaging/enterprise-message-service.

[226] TIBCO. TIBCO Rendezvous. http://www.tibco.com/software/messaging/rendezvous/.

[227] P. Tran, J. Gosper, and I. Gorton. Evaluating the sustained performance of COTS-based messaging systems. *Software Testing, Verification and Reliability*, 13(4):229–240, 2003.

[228] P. Tran, P. Greenfield, and I. Gorton. Behavior and Performance of Message-Oriented Middleware Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCSW'02)*, Washington, DC, USA, 2002. IEEE Computer Society.

[229] Twin Oaks Computing. CoreDX DDS Data Distribution Service Middleware. `http://www.twinoakscomputing.com/coredx`.

[230] S. Vinoski. Advanced Message Queuing Protocol. *IEEE Internet Computing*, 10(6):87–89, 2006.

[231] A. Virgillito. *Publish/Subscribe Communication Systems: From Models to Applications.* PhD thesis, Universita La Sapienza, 2003.

[232] D. Walker-Morgan. The Red Hat Patent Problem and AMQP. `http://www.h-online.com/open/features/The-Red-Hat-Patent-Problem-and-AMQP-746549.html`.

[233] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[234] World Wide Web Consortium (W3C). W3C XML Query (XQuery). `http://www.w3.org/XML/Query/`.

[235] World Wide Web Consortium (W3C). XML Path Language (XPath) 2.0 (Second Edition). `http://www.w3.org/TR/2009/PER-xpath20-20090421/`.

[236] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. C. Schmidt. Evaluating technologies for tactical information management in net-centric systems. In *Proceedings of the Defense Transformation and Net-Centric Systems Conference 2007*, volume 6578 of *SPIE*. Society of Photo Optical, Apr. 2007.

[237] J. Zimmermann and A. Buchmann. Benchmarking Active Database Systems: A Requirements Analysis. In *Proceedings of the OOPSLA'95: Workshop on Object Database Behavior, Benchmarks and Performance*, 1995.

Erklärung

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades "Dr.-Ing." mit dem Titel "Performance Modeling and Benchmarking of Event-Based Systems" selbstständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, den 14.07.2010 Kai Sachs