Designing a Workload Scenario for Benchmarking Message-Oriented Middleware

Kai Sachs*, Samuel Kounev*†, Marc Carter‡, Alejandro Buchmann*
*Databases and Distributed Systems Group
TU Darmstadt, Germany
Email: {sachs,skounev,buchmann}@dvs1.informatik.tu-darmstadt.de
† Computer Laboratory
University of Cambridge, UK
Email: Samuel.Kounev@cl.cam.ac.uk
‡ IBM Hursley Labs
Hursley Park, Winchester, UK
Email: mcarter@uk.ibm.com

***Abstract*— Message-oriented middleware (MOM) is increasingly adopted as an enabling technology for modern information-driven applications like event-driven supply chain management, transport information monitoring, stock trading and online auctions to name just a few. There is a strong interest in the commercial and research domains for a standardized benchmark suite for evaluating the performance and scalability of MOM. With all major vendors adopting JMS (Java Message Service) as a standard interface to MOM servers, there is at last a means for creating a standardized workload for evaluating products in this space. This paper describes a novel application in the supply chain management domain that has been specifically designed as a representative workload scenario for evaluating the performance and scalability of MOM products. This scenario is used as a basis in SPEC's new SPECjms benchmark which will be the world's first industry-standard benchmark for MOM.**

I. Introduction

Message-oriented middleware (MOM) is at the core of a vast number of financial services and enterprise applications, and is gaining increasing traction in other areas, such as manufacturing, transportation, computer gaming, health-care and supply chain management, as well as in technology domains, such as Enterprise Service Bus (ESB), Enterprise Application Integration (EAI) and Service Oriented Architectures (SOA) [5]. Novel messaging applications pose some serious performance and scalability challenges. For example, the next generation of event-driven supply chain management based on RFID technology [7] (for instance SAP's AutoID infrastructure [3]) will be highly reliant on scalable and efficient backend systems to support the processing of acquired real-time data and its integration with enterprise applications and business processes [13]. Large retailers, like Wal-Mart, Metro or Tesco, are expected to have throughput rates of about 60 billion messages per annum [2]. The performance and scalability of the underlying MOM platforms used to process these messages will be of crucial importance for the successful adoption of such applications in the industry.

In order to guarantee that applications meet their Quality of Service (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 [10]:

- 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, i.e. it must provide a level playing field for performance comparisons.
- It must generate reproducible results.
- It must not have any inherent scalability limitations.

While a number of proprietary benchmarks for MOM servers (for example [16], [8], [1], [9]) have been developed and used in the industry for performance testing and product comparisons (see [15], [14], [6], [12], [4]), these benchmarks do not meet the above requirements. The reason is that most of them use artificial workloads that do not reflect 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. Currently, no industry-standard benchmark exists for evaluating the performance and scalability of MOM platforms.

In this paper, we describe a novel application in the supply chain management (SCM) domain that has been designed to serve as a representative workload scenario for evaluating the performance and scalability of MOM products. The application models a set of supply chain interactions between a supermarket company, its stores, its distribution centers and its suppliers. These interactions are used as a basis in SPEC's new SPECjms benchmark which will be the world's first industry-standard benchmark for MOM products that offer a JMS (Java

Message Service [17]) interface. SPECjms is developed at SPEC's OSG-Java Subcommittee with the participation of IBM, TU Darmstadt , Sun, Sybase, BEA, Apache, Oracle and JBoss.

The rest of the paper is organized as follows. We start by looking at the requirements and goals of the SPECjms benchmark. We then present the supermarket supply chain scenario used as basis in SPECjms. We discuss in detail the roles, use cases and interactions defined in the scenario and the way they can be used to stress and evaluate the different aspects of JMS performance. Finally, we look at some implementation issues and challenges that had to be addressed.

II. Workload Requirements and Goals of the SPECjms Benchmark

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

1) Representativeness
2) Comprehensiveness
3) Focus
4) Configurability
5) Scalability

We now briefly discuss each of these groups of requirements in turn.

a) 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 SPECjms benchmark is that it is based on a representative workload scenario. 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.

b) Comprehensiveness: The second important requirement is that the workload is comprehensive in that it should exercise 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.
- 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.

c) 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 [11]. This is especially true in the case of SPECjms, 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.

d) Configurability: As mentioned earlier, in addition to providing standard workload and metrics for JMS performance, SPECjms 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 SPECjms. 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 great 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.

e) Scalability: The SPECjms application scenario must provide a way to scale the workload along the following two dimensions:

1) Horizontal scaling
2) Vertical scaling

In the horizontal scaling, the workload is scaled by increasing the number of destinations (queues and topics) while

keeping the traffic per destination constant. In the 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 his own set of scaling points.

III. Application Scenario for SPECjms: Supermarket Supply Chain

The application scenario chosen for SPECjms 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 was defined based on the requirements discussed in the previous section. It offers an excellent basis for defining interactions that stress different subsets of the functionality offered by JMS servers, e.g. different message types as well as both point-to-point and publish/subscribe communication. The scenario also offers a natural way to scale the workload, e.g. by scaling the number of supermarkets (horizontal) or by scaling the amount of products sold per supermarket (vertical).

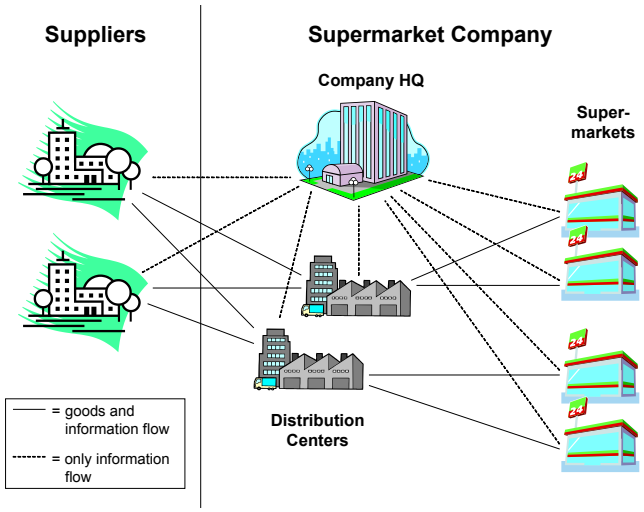


Fig. 1. Overview of the Modeled Scenario and its Roles

The following four roles of participants involved in the scenario are defined:

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 1.

A. 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.

B. Distribution Centers (DCs)

The distribution centers supply the supermarket stores which sell goods to end customers. 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.
- Providing statistical data to HQ (e.g. for data mining).

C. Supermarkets (SMs)

The supermarkets sell goods to end customers. The scenario focuses on the management of the inventory of a supermarket including its warehouse (back room). Not every supermarket offers the same products. Some supermarkets could be smaller than others, so that they do not have enough room for all products, others may be specialized for some product groups like food. We assume that every supermarket is supplied by exactly one of the distribution centers.

D. Suppliers (SPs)

The suppliers deliver goods to distribution centers of the supermarket company. 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. To keep things simple, it is assumed that each SP offers either all products of a given product family or none of them.

IV. Modeled Interactions

The following interactions are modeled in SPECjms:

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.

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

A. Order / Shipment Handling (SM & DC)

This use case describes order and shipment activities between SMs and DCs. It includes two major steps:

1) SM sends an order to DC
2) DC ships goods to SM

In the first step, a SM sends an order to its DC. After receiving the order, the DC sends a confirmation, ships the goods to the SM and notifies the HQ about the shipment. When the shipment arrives at the SM, the local inventory is updated and a confirmation is sent to the DC. The interaction uses only point-to-point communication and is illustrated in Figure 2(a)). A specification of the interaction flow follows:

Interaction 1 (Point-To-Point):

1) A SM sends an order to its DC.
2) The DC sends a confirmation to the SM and dispatches shipment.
3) The DC sends a non-transactional, non-persistent message to the HQ (transaction statistics).
4) The shipment arrives at the SM and confirmation is sent to the DC.

B. (Purchase) Order / Shipment Handling (DC & SP)

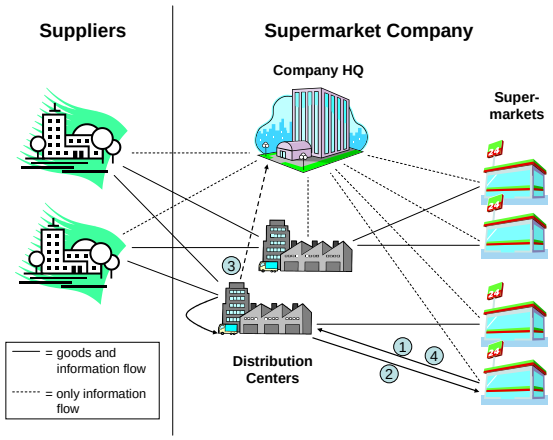
This use case describes purchase order and shipment activities between DCs and SPs. It includes four major steps:

1) DC sends a call for offers.
2) SPs send offers to DC.
3) DC selects one offer and sends a purchase order to SP.
4) SP ships goods to DC.

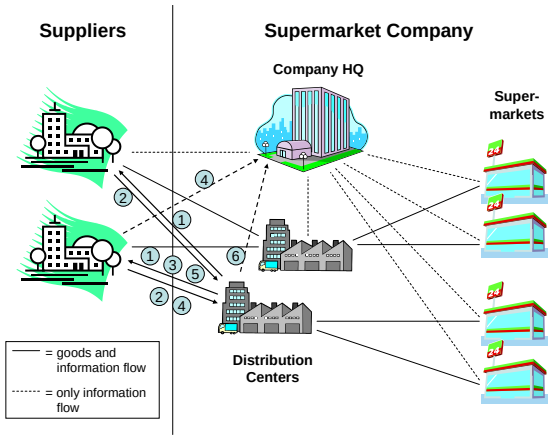
When goods in a DC are depleted, the DC has to place an order to a SP to refill stock. First, a SP has to be selected. The DC sends a *call for offers* to all SPs that are supplying the required products. The SPs send offers to the DC, the DC selects one of them and places a purchase order based on the offer. The selected SP receives the order, delivers the goods to the DC and sends an invoice to the company HQ. After receiving the shipment, the DC acknowledges receipt and reports it to the HQ. The interaction uses both point-to-point and publish/subscribe communication and is illustrated in Figure 2(b). A specification of the interaction flow follows:

Interaction 2 (Publish/Subscribe & Point-To-Point):

1) DC sends a call for offers.
2) All SPs offering the requested products (of the respective product family) send an offer.
3) Based on the offers, the DC selects a SP and sends a purchase order to it.
4) The SP sends an order confirmation to the DC and an invoice to the HQ.
5) The SP dispatches a shipment to the DC.



(a) Interaction 1 - Communication between SM and DC



(b) Interaction 2 - Communication between SP and DC

Fig. 2. Interactions 1 and 2

6) The shipment arrives at the DC and confirmation is sent to the SP.
7) The DC sends a non-transactional, non-persistent message to the HQ (transaction statistics).

C. Price Updates

Selling prices are changed by the company administration from time to time. To communicate this, the company HQ sends messages with pricing information to the supermarkets. The communication here is one-to-many and is based on publish/subscribe messaging.

Interaction 3 (Publish/Subscribe):

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

D. Inventory Management

When goods leave the warehouse of a supermarket (to refill a shelf), the warehouse application must be informed, so that

the inventory can be updated.[1]

Interaction 4 (Point-To-Point):

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.

E. Sales Statistics Collection

SMs send sales statistics to the HQ (type and amount of goods purchased by customers visiting the store). 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.

Interaction 5 (Point-To-Point):

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

F. New Product Announcements

New products are announced by the company administration from time to time. To communicate this, the company HQ sends messages with product information to the supermarkets offering the respective range of products (e.g. food, computers, mp3-players). This communication is based on publish/subscribe messaging.

Interaction 6 (Publish/Subscribe):

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

G. Credit Card Hot Lists

HQ sends credit card hot lists to SMs (complete list once every hour and incremental updates as required). This interaction is used to exercise non-durable, non-persistent publish/subscribe messaging.

Interaction 7 (Publish/Subscribe):

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

V. Implementation Details

In this section, we discuss the way the workload scenario we presented in the previous sections has been implemented as part of the SPECjms benchmark.

A. Message Types and Destinations

There are several possibilities as to how many queues are used in the implementation of interactions that use point-to-point messaging:

[1]Note: Because the incoming goods are part of another interaction (as described in Section IV-A), they are not considered here.

Intr.	Destination Name	Message
1	*DC_OrderQ*	order
1	*DC_ShipDepQ*	shipDep
1	*DC_ShipConfQ*	shipConf
2	*DC_IncomingOffersQ*	offers
2	*DC_POrderConfQ*	pOrderConf
2	*DC_PShipArrQ*	pShipInfo
1	*SM_ShipArrQ*	shipInfo
1	*SM_OrderConfQ*	orderConf
4	*SM_InvMovementQ*	inventoryInfo
2	*SP_POrderQ*	pOrder
2	*SP_PShipConfQ*	pShipConf
1	*HQ_OrderDCStatsQ*	statInfoOrderDC
2	*HQ_ProductFamilynT*	callForOffers
2	*HQ_ShipDCStatsQ*	statInfoShipDC
2	*HQ_InvoiceQ*	invoice
3	*HQ_PriceUpdateT*	priceUpdate
5	*HQ_SMStatsQ*	statInfoSM
6	*HQ_ProductAnnouncementT*	productAnnouncement
7	*HQ_CreditCardHLT*	creditCardHL

TABLE I
QUEUES AND TOPICS USED IN THE INTERACTIONS

1) One queue per message type and physical location (e.g. one queue per SM for registering incoming goods from the DC, and a separate queue for order confirmations from the DC).
2) One queue per physical location (one queue per SM, one queue per DC, one queue per SP and one queue for the HQ).
3) One queue per physical location and communicating party (e.g. one queue per DC for communication with SPs, and one for communication with SMs).

In SPECjms the first option was chosen. Table I shows the destinations (queues and topics) used in the implementation of the interactions. Destination names are prefixed with the location type they belong to (SP, SM, HQ or DC) and suffixed with 'T' or 'Q' depending on the destination type (*T*opic or *Q*ueue)[2].

The different types of messages used are detailed in Table II. All message types supported by the JMS specification with exception of ByteMessages (which are rarely used in practice) are exercised by the workload. It was decided to use AUTO_ACKNOWLEDGMENT as default acknowledgment mode for non-transactional sessions. The reason is that most real-world applications do not use the other acknowledgment modes (CLIENT_ACKNOWLEDGMENT or DUPS_OK_ACKNOWLEDGMENT). However, SPECjms still offers the possibility to change the acknowledgment mode for selected interactions, although this is not allowed for published benchmark results.

B. Event Handlers

An *Event Handler (EH)* is a Java class that contains the application logic executed to process messages sent to a queue or a topic. Event handlers register as listeners for the queue/topic

[2]Since the Headquarters are responsible for managing the different topics, all topic names are prefixed with HQ.

Intr.	Message	Destination	Type	Size	Properties	Description
1	order	Queue	ObjectMessage	small to medium	Persistent, Transacted	Incoming orders from SMs including ordered products and other order information.
1	orderConf	Queue	ObjectMessage	small to medium	Persistent, Transacted	Confirmation sent by DC to SM confirming receipt of an order.
1	shipInfo	Queue	TextMessage	small to large	Persistent, Transacted	EPCs representing items in a shipment (registered by RFID readers as shipment leaves DC / enters SM).
1	shipDep	Queue	TextMessage	small to large	Persistent, Transacted	EPCs representing items in a shipment (registered by RFID readers as shipment leaves DC / enters SM).
1	shipConf	Queue	ObjectMessage	small to medium	Persistent, Transacted	Confirmation sent to DC to confirm shipment arrival at SM.
2	callForOffers	Topic	TextMessage	small to medium	Persistent, Transacted	Call for Offers send to all SP offering a product family (XML document).
2	offer	Queue	TextMessage	small to medium	Persistent, Transacted	Offer created by SPs based on callForOffers (XML document).
2	pOrder	Queue	TextMessage	small to medium	Persistent, Transacted	Incoming purchase order based from DC on an offer (XML document).
2	pOrderConf	Queue	TextMessage	small to medium	Persistent, Transacted	Confirmation sent by SP to DC confirming receipt of a purchase order (XML document).
2	pShipInfo	Queue	TextMessage	small to large	Persistent, Transacted	XML document describing an arriving shipment from SP (registered by RFID readers as they enter DC).
2	pShipConf	Queue	TextMessage	small to medium	Persistent, Transacted	Confirmation sent to SP to confirm shipment arrival at DC (XML document).
2	invoice	Queue	TextMessage	small to medium	Persistent, Transacted	Purchase order invoice sent by SP (XML document).
1	statInfoOrderDC	Queue	StreamMessage	small	Non-Persistent, Non-Transacted	Statistical information sent to HQ containing information on the transactions and flow of goods between SMs and DCs.
2	statInfoShipDC	Queue	StreamMessage	small	Non-Persistent, Non-Transacted	Statistical information sent to HQ containing information on the transactions and flow of goods between SPs and DCs.
3	priceUpdate	Topic	MapMessage	small	Persistent, Transacted, Durable	Price update sent to SM by HQ.
4	inventoryInfo	Queue	TextMessage	small to large	Persistent, Transacted	EPCs representing observed item movements in the inventory of a SM (registered by RFID readers).
5	statInfoSM	Queue	ObjectMessage	small to very large	Non-Persistent, Non-Transacted, AutoAck	Statistical information sent to HQ containing sales statistics.
6	product-Announcement	Topic	StreamMessage	small to large	Non-Persistent, Non-Transacted, Non-Durable	Product announcements sent to SM by HQ.
7	creditCardHL	Topic	StreamMessage	small to large	Non-Persistent, Non-Transacted, Non-Durable	Credit card hotlist sent to SMs by HQ.

TABLE II
MESSAGE TYPES USED IN THE INTERACTIONS

and receive call backs from the messaging infrastructure (JMS provider) as new messages arrive. A separate event handler is defined for each queue and topic. For maximal performance, multiple instances of an event handler could exist to enable parallel processing.

The event handlers at the different physical locations are part of the various applications emulated by the benchmark, e.g. the four event handlers of a DC represent the *DC-App* used in the diagrams in Figures 4(a) and 4(b). An event handler has a similar name to its respective destination but is distinguished by an 'EH' suffix in place of 'Q' or 'T'.

Example: $\mathrm{DC\_OrderQ} \Rightarrow \mathrm{DC\_OrderEH}$

C. Driver Framework

The SPECjms 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 SPECjms 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 J2SE platform.
- The controller need 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 SPECjms run rules).

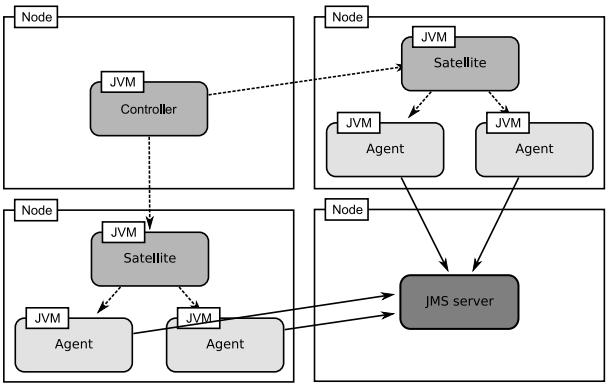


Fig. 3. Driver Framework

Figure 3 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.

1) 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.

2) Satellite: The satellite is a simple part of the framework (another Java application) which knows to build the correct environment to start child Java processes for SPECjms. It takes the controller's configuration and will start 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.

3) Agents: Each logical agent represents one of the locations in the application scenario (Section III). 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 co-ordinates the stages of the test. Once all agents are connected, the event handlers (implemented as a Java thread each) will 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.

D. Minimizing the Impact of Non-MOM-Related Components

As discussed in Section II, the SPECjms workload should be focused on measuring the performance and scalability of a JMS server's software and hardware components and minimize the impact of other components that are not directly related to the MOM services. 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 JMS servers, in their role as mediators in interactions, are typically less loaded than database servers or application servers, it was a real 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. The way we addressed this is 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.

E. Providing a Framework for Performance Analysis of JMS

Another important goal of SPECjms that we discussed in Section II was to provide a flexible framework for performance analysis of JMS servers that allows users to configure and customize the workload according to their requirements. To achieve this goal, the interactions were implemented in such a way that one could run them in different combinations depending on the desired transaction mix. SPECjms provides three different *topologies* which correspond to three different modes in which the benchmark can be run:

1) Horizontal
2) Vertical
3) Freeform

The horizontal topology allows the workload to be scaled horizontally by increasing the number of physical locations (SMs, DCs, SPs) while keeping the traffic per location constant. In the vertical topology, a fixed set of physical locations

Interaction	**2**	**3**	**6**	**7**
No. of topics:	*n* (one for each product family)	One	One	One
No. of producers:	*m* (one per DC)	One (HQ)	One (HQ)	One (HQ)
No. of consumers:	*p* (one per SP)	*n* (one per SM)	*n* (one per SM)	*n* (one per SM)
Transacted	Yes	Yes	No	No
Persistent	Yes	Yes	No	No
Durable	Yes	Yes	No	No

TABLE III

OVERVIEW OF THE ATTRIBUTES OF PUBLISH/SUBSCRIBE COMMUNICATION IN THE INTERACTIONS

is used and the workload is scaled by increasing the rate at which goods are sold at supermarkets. Finally, the freeform topology allows the user to define his own topology and transaction mix and scale the workload in an arbitrary manner by choosing a set of scaling points. Note that while in the horizontal and vertical topology scaling is supported in a manner that preserves the relation to the modeled real-life business scenario, this does not necessarily apply to the freeform topology since the goal there is to give the user maximum flexibility.

SPECjms provides the following configuration parameters that can be set by the user:

- The number of physical locations (HQ, SM, DC, SP) emulated.
- The number of agents representing a single physical location.
- The driver nodes on which agents are run.
- The number of JVMs run on each node and the way agents are distributed among them.
- The number of javax.jms.Connection objects shared amongst event handler classes within a single agent.
- Whether connections are shared by multiple sessions in an event handler.
- Number of event handlers in an agent of each type.
- Number of driver instances for each interaction.
- Rate at which interactions are invoked.
- Total number of invocations of each interaction (as an alternative to specifying a rate).
- Message size distributions for each interaction.
- Connection factory used on each event handler or driver object.
- Whether AUTO_ACKNOWLEDGMENT should be used as acknowledgment mode for non-transactional sessions.
- Frequency of verifying message integrity after transmission (CRC check).

While in the horizontal and vertical topologies there are some restrictions as to which parameters can be changed by the user, these restrictions are relaxed in the freeform topology. This allows the user to precisely configure the workload and transaction mix to be generated and the way it is distributed among client processes. 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.

F. Interaction Details

1) Interactions 1 & 2: Interactions 1 and 2 are depicted in Figures 4(a) and 4(b), respectively.

2) Interaction 3: In Interaction 3, publish/subscribe messaging is used. The HQ communicate with SMs using a topic.

3) Interaction 4: Like Interactions 1 and 2, this interaction is point-to-point based. There is one queue per SM.

4) Interaction 5: This interaction is again point-to-point based. SMs use the HQ_StatsQ queue (see Interactions 1 and 2) to send sales statistics to HQ. Alternatively, we could use a separate queue for these statistics. In any case, a non-persistent queue should be used.

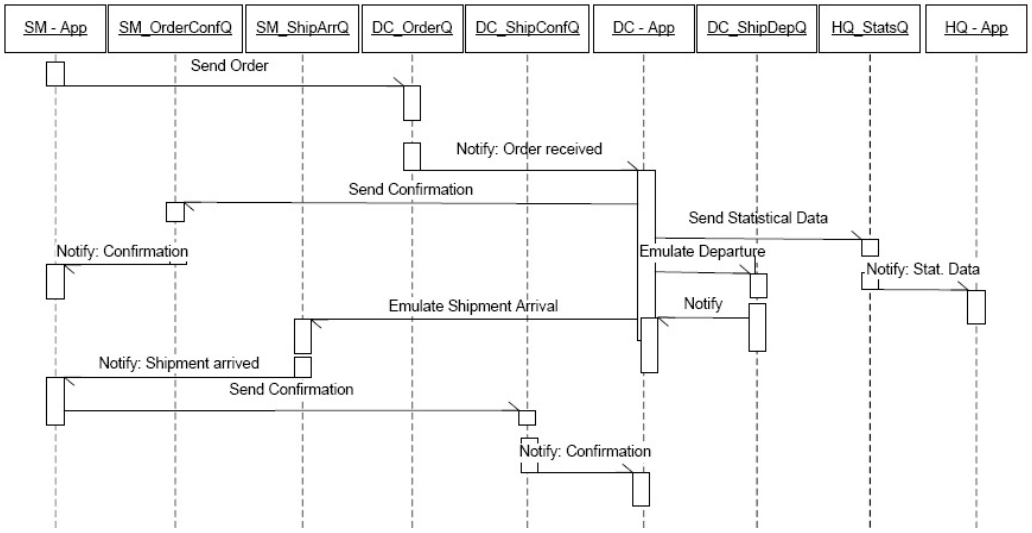
5) Interaction 6: In this interaction the HQ communicate with SMs via publish/subscribe messaging.

6) Interaction 7: Credit card hot lists are sent to SMs through publish/subscribe messaging.

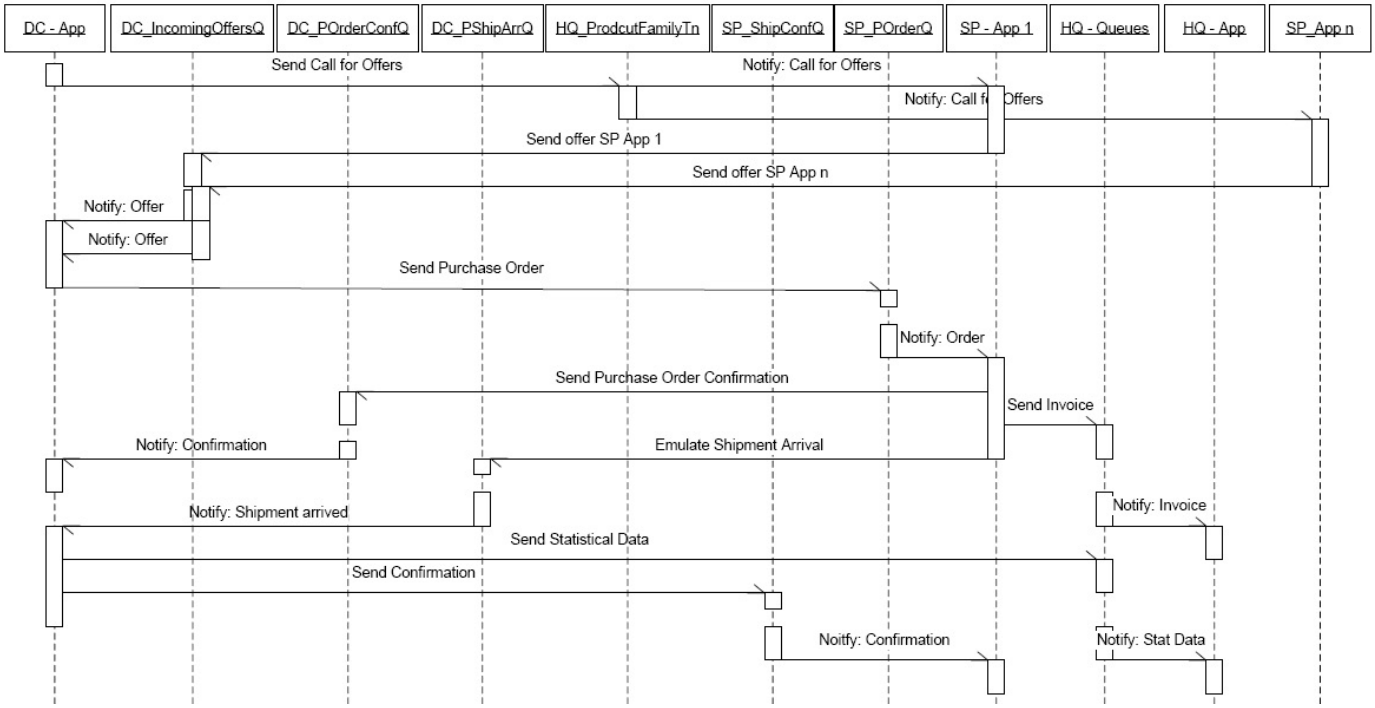
VI. SUMMARY

We presented a novel application in the supply chain management domain that has been designed to serve as a representative workload scenario for evaluating the performance and scalability of MOM products. The application models a set of interactions in the supply chain of a supermarket company. These interactions are used as a basis in SPEC's new SPECjms benchmark which will be the world's first industry-standard benchmark for MOM products. We started by looking at the requirements and goals of the SPECjms benchmark. We then discussed in detail the roles, use cases and interactions defined in the scenario and the way they can be used to stress and evaluate the different aspects of JMS performance. Finally, we looked at some implementation issues and challenges that had to be addressed.

SPECjms provides a level playing field for performance comparisons of competitive MOM products. However, producing and publishing standard results for marketing purposes will be just one usage scenario of the benchmark. Many users will be interested in using the benchmark to fine 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. SPECjms has been designed to



(a) Sequence diagram for Interaction 1



(b) Sequence diagram for Interaction 2

Fig. 4. Use Case 1 & 2

support all of these usage scenarios by providing a flexible performance analysis framework which allows the user to precisely configure the workload and transaction mix to be generated. In this sense, SPECjms will not only provide an industry-standard benchmark for MOM servers, but it will also offer a full-blown performance analysis tool for JMS-based MOM infrastructures.

VII. Acknowledgments

This work was partially funded by the German Research Foundation (Deutsche Forschungsgemeinschaft). The authors would like to acknowledge the contributions of the members of the SPECjms Working Group to the specification and development of the SPECjms benchmark. Special thanks to Tim Dunn from IBM, George Tharakan from Sun Microsystems, Evan Ireland from Sybase, Tom Barnes and Russell Raymundo from BEA, and Adrian Co from Apache. We are also especially thankful to Lawrence Cullen, Robert Berry, Alan Adamson and John Stecher from IBM, Steve Realmuto from BEA and Ricardo Morin from Intel for their continued support of the SPECjms project.

REFERENCES

[1] ActiveMQ. JMeter performance test. http://incubator.apache.org/activemq/jmeter-performance-tests.html, 2006.

[2] K. Alexander, T. Gillian, K. Gramling, M. Kindy, D. Moogimane, M. Schultz, and M. Woods. IBM Business Consulting Services - Focus on the Supply Chain: Applying Auto-ID within the Distribution Center. White paper IBM-AUTOID-BC-002, 2003.

[3] C. Bornhövd, T. Lin, S. Haller, and J. Schaper. Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In *30th International Conference on Very Large Databases (VLDB'04), Aug. 29 - Sep. 3, 2004, Toronto, Canada*, 2004.

[4] M. Carter. JMS Performance with WebSphere MQ for Windows V6.0. http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24010028, 2005.

[5] D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004. ISBN: 059600675.

[6] Crimson Consulting Group. High-Performance JMS Messaging - A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ. http://www.sun.com/software/products/message_queue/wp_JMSperformance.pdf, 2003.

[7] K. Finkenzeller. *RFID Handbook : Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley & Sons, 2nd edition, may 2003.

[8] IBM Hursley. Performance Harness for Java Message Service. http://www.alphaworks.ibm.com/tech/perfharness, 2005.

[9] JBoss. JBoss JMS New Performance Benchmark. http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossJMSNewPerformanceBenchmark, 2006.

[10] S. Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction*. Shaker Verlag, Dec. 2005. ISBN: 3832247130.

[11] S. Kounev and A. Buchmann. Improving Data Access of J2EE Applications by Exploiting Asynchronous Processing and Caching Services. In *Proceedings of the 28th International Conference on Very Large Data Bases - VLDB2002, Hong Kong, China, August 20-23*, 2002.

[12] Krissoft Solutions. JMS Performance Comparison. http://www.fiorano.com/comp-analysis/jms_perf_report.htm, 2006.

[13] K. Sachs. Evaluation of Performance Aspects of the SAP Auto-ID Infrastructure. Master's thesis, Department of Computer Science, Darmstadt University of Technology, 2004.

[14] Sonic Software Corporation. JMS Performance Comparison: SonicMQ(R) vs TIBCO Enterprise(TM) for JMS. White Paper, Nov. 2003. http://www.onwhitepapers.com/redirect.php?wid=4A0D2BDBBE89205241534CCB0AA8ED56.

[15] Sonic Software Corporation. Benchmarking E-Business Messaging Providers. White Paper, Jan. 2004. http://www.onwhitepapers.com/redirect.php? wid=4A0ABD6CBE8920524153D95D6F02C48C.

[16] Sonic Software Corporation. SonicMQ Test Harness. http://www.sonicsoftware.com/products/sonicmq/performance_benchmarking/index.ssp, 2005.

[17] Sun Microsystems Inc. Java Message Service (JMS) Specification Version 1.1. http://java.sun.com/products/jms/docs.html, 2002.