A Logistics Workload for Event Notification Middleware

Stefan Appel and Kai Sachs

TU Darmstadt, Germany
lastname@dvs.tu-darmstadt.de

Abstract. The event-based paradigm plays an important role to reflect logistics processes in modern IT infrastructures. Events occur at many stages, e.g., when goods tagged with RFID chips are scanned, when transportation vehicles move or when sensors report environmental observations. These events have to be delivered to interested consumers by a reliable notification middleware, which is crucial for a successful implementation of event-based applications. Specified service levels have to be fulfilled and to guarantee them, an exhaustive evaluation and analysis of the underlying event notification middleware is required. This can be achieved by applying well-defined test scenarios that allow us to analyze different aspects of the middleware in an independent and representative way.

In this paper we present a realistic workload originating from a real world scenario in the logistics domain. Our workload is suited to test event notification middleware under realistic conditions; it stresses different aspects of the middleware while being scalable.

1 Introduction

Designing systems that follow the event-based paradigm are necessary to develop new types of applications [6, 3]. These application have to handle large amounts of data originating from, e.g., sensor networks or the Internet of Things. The devices collect a variety of data that is potentially interesting for different applications. For example, nowadays mobile phones are equipped with GPS sensors and accelerometers allowing applications for monitoring the environment [9, 15]. All these events, or more precisely, their representations, the event notifications, need to be transported from the event producers to event consumers [6]. To reach a high amount of flexibility throughout this communication process, event producers and consumers need to be decoupled physically and logically. Therefore, event notifications are routed from event producers to event consumers by a notification middleware (also called notification service, see Figure 1). This underlying notification middleware allows us to decouple producers and consumers and is responsible for a reliable message transportation [10].

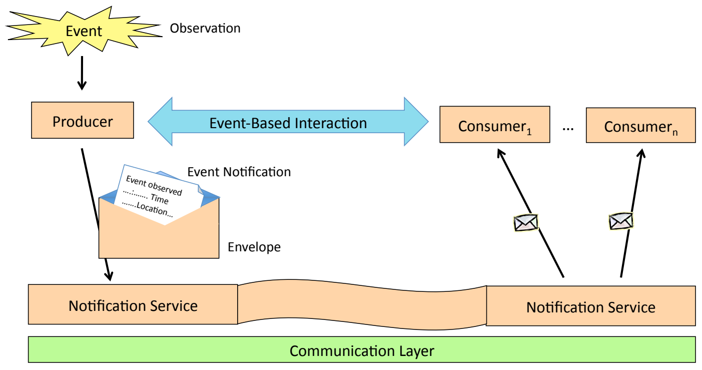


Fig. 1. Event-based System - Schematic Overview

Driven by the research in *ADiWa*[1] we identified the need for testing and analyzing event notification middleware. Within ADiWa business processes adapt and react dynamically to events.

In this paper, we present a novel workload based on a logistics scenario. Our goal is to support exhaustive testing and analysis of a notification middleware by stressing the system in different ways. The remainder of this paper is structured as follows: we first present related work in the area of quality of service (QoS) and workload characterization of event-based systems (EBS). In the next section we discuss requirements a workload has to fullfil with a focus on the logistics domain. We then introduce a logistics workload which models three interactions among different entities and derive variable message rates to describe the workload. The paper concludes with a short summary of our results and an outlook on future research.

2 Related Work

Several test harnesses and benchmarks for different EBS were published, e.g., [8, 5, 7]. However, previous work in the area of benchmarking mostly focuses on the design and development of test frameworks, but not on the definition of workloads. An example for a well-defined workload scenario used for the industry standard benchmark SPECjms2007 can be found in [13]. The application scenario models seven business interactions of a supermarket's supply chain where RFID technology is used to track the flow of goods. SPECjms2007 includes some limited publish/subscribe communication as part of the workload, but mainly focusing on point-to-point communication. The setup of consumer, producer and subscriptions is static and does not change at runtime. A benchmark for publish/subscribe systems built on top of the SPECjms2007 workload is

[1] Alliance Digital Product Flow (ADiWa). Funded by the German Federal Ministry of Education and Research under grant 01IA08006.

jms2009-PS [12]. However, the implemented topology is static and subscriptions do not change at runtime. For a comprehensive overview of existing benchmarks and test harnesses we refer to [11]. An overview of relevant QoS metrics in the context of distributed and decentralized publish-subscribe systems is provided in [2]. Further, QoS of EBS is discussed in [1].

3 Workload Requirements

The major goal of this paper is to provide a standard workload and metrics for measuring and evaluating the performance and scalability of event notification middleware in dynamic environments. To achieve this goal, a workload must be designed to meet a number of important requirements. These requirements can be grouped in the following five categories [11]:

1. *Representativeness:* It has to reflect the way middleware services are exercised in real-life systems.
2. *Comprehensiveness:* All middleware features and services typically used in applications have to be exercised. Features and services stressed should be weighted according to their usage in real-life systems.
3. *Focus:* The emphasis has to be on the event notification middleware and the impact of other components and services has to be minimized.
4. *Configurability:* The workload should be configurable to provide a base for exhaustive system analysis.
5. *Scalability:* The workload must not have any inherent scalability limitations and provide ways to scale the workload in a flexible manner.

Based on these categories, we specified a set of requirements, which differ in major points from previous work. For example:

1. *Independent Participants:* Event producers and consumers should be logically decoupled.
2. *Communication Plattform:* An event service bus [4] should be used for communication.
3. *Dynamic Environment:* Subscriptions, subscribers and message producers should not be static and change frequently.

Keeping these requirements in mind we specified a novel workload for performance analysis of event notification middleware based on a real-world scenario.

4 Logistics Workload

In this section we present a logistics workload to evaluate event notification middleware. The workload models three different interactions in a company. Figure 2 gives a schematic overview and shows the information flow as well as the flow of goods. A logistics workload stresses different aspects of the middleware; especially high fluctuation rates of event producers and consumers are characteristic.

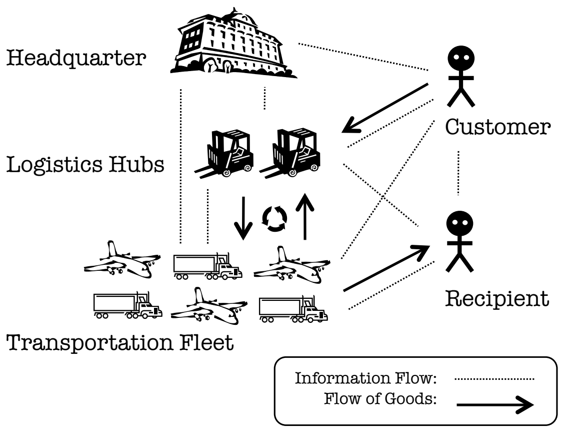


Fig. 2. Flow of Information and Goods

In the following sections we first introduce event notification producers and and interacting entities involved in our scenario. Three different interactions between the entities are simulated to evaluate the event notification middleware. For this we derive interaction rates and publish/subscribe parameters.

4.1 Event Notification Producers

In our scenario the event notifications are generated by different entities. Either humans or three different types of devices, RFID Readers, On-board Units or Environment Sensors, are involved in the event detection and notification generation process:

Humans (`H`) To initiate a shipment process an order has to be triggered by humans. The addresses of sender and recipient are submitted to and verified by the company. Afterwards, the order is acknowledged and the transportation process starts.

RFID Reader (`RFID`) We assume a scenario where RFID tags are attached to all transported goods. These tags are read when goods enter and leave hubs as well as when they are delivered.

On-board Units (`OBU`) Modern trucks are equipped with on-board units including GPS receivers so that the vehicle position can be tracked. The update interval is a tradeoff between accuracy and system utilization in terms of bandwidth, CPU utilization and network traffic. For our scenario we assume an update interval of three minutes.

Environment Sensors (`ENV`) To ensure that goods are transported appropriately, environment sensor can be installed to monitor, e.g., the temperature. Customers are interested in whether the conditions were met throughout the transportation process.

4.2 Interacting Entities

The above listed notification producers act in the context of entities. The following parties participate in interactions modeling the message flow:

1. *Company Headquarters* (`HQ`):
   Messages generated by humans.
2. *Logistics Hubs* (`LH`):
   Messages generated by RFID readers and environment sensors.
3. *Customers* (`C`):
   Messages generated by humans.
4. *Transportation Vehicles* (`V`):
   Messages generated by on-board units and environment sensors.

We consider that customers are performing monitoring and tracking of goods. For better usability, we specified a fixed ratio between different entities based upon data from a real-world company [14]. The base value for scaling the entities is the number of hubs $\texttt{BASE}_L$.

$$\begin{aligned}
|Logistics\ Hubs| &:= \texttt{BASE}_L \\
|Headquarters| &:= 1 \\
|Vehicles| &:= 53 \cdot \texttt{BASE}_L \\
\left|\frac{Customers}{day}\right| &:= 1000 \cdot \texttt{BASE}_L \\
\left|\frac{Shipments}{day}\right| &:= 8384 \cdot \texttt{BASE}_L
\end{aligned}$$

4.3 Interaction Patterns

Throughout the delivery process different interactions between the entities take place. Each interaction involves messages, subscriptions and quality of service requirements. We identify three interaction patterns for which we derive the message rates depending on the number of hubs:

Interaction 1 (I1): Shipment Order and Proof of Delivery
Interaction 2 (I2): Real-time Shipment Tracking
Interaction 3 (I3): Freight Monitoring

While I1 is a common interaction in logistics companies, I2 and I3 are currently popular trends, but are still in the early adopter phase. We expect the wide-spread deployment of real-time tracking and freight monitoring in the future.

Table 1. Interactions

(a) *Interaction 1* - Shipment Order and Proof of Delivery

Message Publisher	**Message Subscriber**	**Type**	**QoS**
Customer	Headquarters	ShipmentOrder	Reliable
Headquarters	Customer	OrderConfirmation	Reliable
Headquarters	Logistics Hub	PickupOrder	Reliable
Logistics Hub	Headquarters	ShipmentScan	Reliable
⋮	⋮	⋮	⋮
Recipient	Headquarters	ProofOfDelivery	Reliable

(b) *Interaction 2* - Real-time Shipment Tracking

Message Publisher	**Message Subscriber**	**Type**	**QoS**
Logistics Hub	Customer	ShipmentScan	Reliable
Vehicle	Customer	PositionData	Unreliable

(c) *Interaction 3* - Freight Monitoring

Message Publisher	**Message Subscriber**	**Type**	**QoS**
Vehicle	Customer	EnvironmentData	Reliable
Logistics Hub	Customer	EnvironmentData	Reliable

I1 - Shipment Order and Proof of Delivery The most important interaction is receiving shipment orders from customers and providing a proof of delivery (PoD) once the shipment is successfully delivered. The first event in this interactions is a message originating from a customer requesting pick up of goods (`ShipmentOrder`). Afterwards, shipment IDs are generated and the goods are tagged with RFID labels. The order confirmation message containing shipment IDs is then sent to the customer (`OrderConfirmation`). A hub close to the customer is selected and a pickup order is sent (`PickupOrder`). As soon as the goods leave the customer, the headquarters is responsible for tracking the goods and thus it receives messages whenever a shipment enters or leaves a hub (`ShipmentScan`). Finally, a proof of delivery message is generated once the shipment receives its final destination (`ProofOfDelivery`). With this message the interaction ends. Table 1(a) shows the message exchanges within I1; during the shipment process, the goods enter and leave several hubs. Thus multiple `ShipmentScan` messages occur within this interaction.

In terms of QoS, I1 has to be reliable. This means that the delivery of all messages has to be ensured in case of failures. This requires persistence mechanisms and recovery strategies integrated in the middleware.

I2 - Real-time Shipment Tracking Real-time shipment tracking is an interaction pattern involving multiple entities and advanced application logic. It requires the knowledge of shipment IDs; based upon those IDs, it is possible to track the flow of goods. For tracking, the first step is subscribing to `ShipmentScan` messages matching the IDs of the shipments to track to receive messages originating from RFID readers. These events indicate whenever a shipment leaves or enters a hub. As soon as the shipment enters a transportation vehicle an additional subscription has to be issued to receive position events from the respective transportation vehicle.

Table 1(b) shows the messages within I2. Besides `ShipmentScan` messages, which need to be delivered reliably, `PositionData` messages are generated. The latter do not require reliable delivery since positions updates are generated regularly and only the last position is of major interest.

I3 - Freight Monitoring Some goods require special treatment throughout the transportation process. For example, fresh products have to be kept below a certain temperature, other goods can only be transported in an upright position. Thus, it is essential to monitor goods and to identify improper treatment. An early detection is desirable to inform customers as fast as possible. This gives the opportunity to react quickly while the goods are still on their way. Real-time monitoring stresses the middleware and thus it is included in this workload. We choose temperature monitoring since it is one characteristic application of freight monitoring.

Table 1(c) shows the message exchange within I3. Opposed to `PositionData` messages in I2, `EnvironmentData` messages require reliable transportation since a violation of environmental conditions has to be reported.

4.4 Workload Generation

To simulate the message flow within a company, we derive rates at which messages enter the system. These rates are determined by the scenario design and scale with the number of hubs, $\texttt{BASE}_L$. Based upon the number of shipments per day (cp. Section 4.2) the rates can be calculated. We further make the following assumptions:

1. Time from pickup of goods to delivery is 3 days.
2. The goods are in transit 1.5 days (real transportation time), the goods are processed at hubs the other 1.5 days.
3. Vehicles and environment sensors submit messages every 3 minutes.
4. The goods pass 2 hubs until the recipient is reached.
5. Shipment scans and transportation are uniformly distributed over time.
6. All shipments are tracked in real-time.
7. The environmental conditions of 30% of the shipments are monitored.

This results in multiple messages for each shipment as shown in Table 2. Position and environment data messages are relevant for multiple shipments, thus the use of publish/subscribe mechanisms is the delivery paradigm of choice.

To generate the workload, five components are necessary, one for each message type. Each message driver component produces messages with certain rates, consumes them and simulates join and leave of producers and consumers at

Table 2. Messages per Shipment

Message Type	Messages per Shipment
`ShipmentOrder`	1
`OrderConfirmation`	1
`PickupOrder`	1
`ShipmentScan`	4
`ProofOfDelivery`	1
`PositionData`	720
`EnvironmentData`	1440

certain rates. Therefore, the message driver components have to ensure that published messages are consumed according to the scenario specification.

For describing the behavior of workload generating components (drivers) we use the following terms:

– *Active Entity:* Producer/consumer which publishes/subscribes to messages.
– *Messages min^{-1}:* Total number of published messages per minute.
– *Parallel Subscriptions:* Number of subscriptions in parallel.
– *Parallel Publishers:* Number of parallel message publishers.
– *Subscription join/leave min^{-1}:* Rate at which subscribers leave, respectively join the system.
– *Publisher join/leave min^{-1}:* Rate at which publishers leave, respectively join the system.
– *Pub/Sub Factor:* The number of recipients for each published message.

Table 3 lists all the message driver components along with the derived service rates.

ShipmentOrder Driver Shipment orders are generated by customers, the destination is always the headquarters. Multiple customers issue shipment orders in parallel and customers enter, respectively leave, the system constantly. Each costumer only sends one shipment order. The pub/sub factor of one indicates that each messages is delivered to one destination, the headquarters.

OrderConfirmation Driver As for the shipment orders, order confirmations are a one-to-one communication between headquarters and customers. Each customer receives one order confirmation.

PickupOrder Driver Pickup orders are messages from the headquarters to hubs; each hub is supposed to receive the same number of messages. Since hubs and the headquarters are static parts of the infrastructure, a change rate of zero is assumed.

Table 3. Event Driver and their Characteristics

Type	**Characteristic**	**Rate**	**Active Entity**
ShipmentOrder	Messages min^{-1}	$5.82 \cdot BASE_L$	`C`
	Parallel Subscriptions	1	`HQ`
	Parallel Publishers	$8.43 \cdot BASE_L$	`C`
	Subscription join/leave min^{-1}	0	`HQ`
	Publisher join/leave min^{-1}	$0.69 \cdot BASE_L$	`C`
	Pub/Sub Factor	1	`HQ`
OrderConfirmation	Messages min^{-1}	$5.82 \cdot BASE_L$	`HQ`
	Parallel Subscriptions	$8.43 \cdot BASE_L$	`C`
	Parallel Publishers	1	`HQ`
	Subscription join/leave min^{-1}	$0.69 \cdot BASE_L$	`C`
	Publisher join/leave min^{-1}	0	`HQ`
	Pub/Sub Factor	1	`C`
PickupOrder	Messages min^{-1}	$5.82 \cdot BASE_L$	`HQ`
	Parallel Subscriptions	$BASE_L$	`LH`
	Parallel Publishers	1	`HQ`
	Subscription join/leave min^{-1}	0	`C`
	Publisher join/leave min^{-1}	0	`HQ`
	Pub/Sub Factor	1	`LH`
ShipmentScan	Messages min^{-1}	$7.76 \cdot BASE_L$	`LH`
	Parallel Subscriptions	$25152 \cdot BASE_L$	`C`
	Parallel Subscriptions	1	`HQ`
	Parallel Publishers	$BASE_L$	`LH`
	Subscription join/leave min^{-1}	$5.82 \cdot BASE_L$	`C`
	Subscription join/leave min^{-1}	0	`HQ`
	Publisher join/leave min^{-1}	0	`LH`
	Pub/Sub Factor	1 / 1	`HQ` / `C`
ProofOfDelivery	Messages min^{-1}	$5.82 \cdot BASE_L$	`V`
	Parallel Subscriptions	1	`HQ`
	Parallel Subscriptions	$25152 \cdot BASE_L$	`C`
	Parallel Publishers	$53 \cdot BASE_L$	`V`
	Subscription join/leave min^{-1}	$5.82 \cdot BASE_L$	`C`
	Subscription join/leave min^{-1}	0	`HQ`
	Publisher join/leave min^{-1}	0	`LH`
	Pub/Sub Factor	1 / 1	`HQ` / `C`
PositionData	Messages min^{-1}	$8.83 \cdot BASE_L$	`V`
	Parallel Subscriptions	$25152 \cdot BASE_L$	`C`
	Parallel Publishers	$53 \cdot BASE_L$	`V`
	Subscription join/leave min^{-1}	$5.82 \cdot BASE_L$	`C`
	Publisher join/leave min^{-1}	0	`V`
	Pub/Sub Factor	474.74	`C`
EnvironmentData	Messages min^{-1}	$2.91 \cdot BASE_L$	`LH`
	Messages min^{-1}	$2.91 \cdot BASE_L$	`V`
	Parallel Subscriptions	$7546 \cdot BASE_L$	`C`
	Parallel Publisher	$BASE_L$	`LH`
	Parallel Publisher	$53 \cdot BASE_L$	`V`
	Subscription join/leave min^{-1}	$1.75 \cdot BASE_L$	`C`
	Publisher join/leave min^{-1}	$1.75 \cdot BASE_L$	`V`
	Pub/Sub Factor	142.36	`C`

ShipmentScan Driver Goods are scanned whenever they enter or leave hubs. The resulting messages are part of I1 and I2 and thus consumed by the headquarters as well as by customers. Many customers are subscribed simultaneously since transportation of goods lasts three days. Each message is consumed by the headquarters and by one customer.

ProofOfDelivery Driver The proof of delivery (PoD) denotes the arrival of the shipment at the designated recipient. The final delivery is performed by vehicles, whereas the driver triggers the generation of the PoD message. PoD messages are consumed by customers as well as by the headquarters.

PositionData Driver Position data is sent by all vehicles. Multiple shipments are transported within one vehicle, this motivates the high pub/sub factor; each position data message has to be received by around 474 customers. We assume that all shipments within a vehicle belong to different customers. The total number of subscriptions is determined by the goods being in move in parallel.

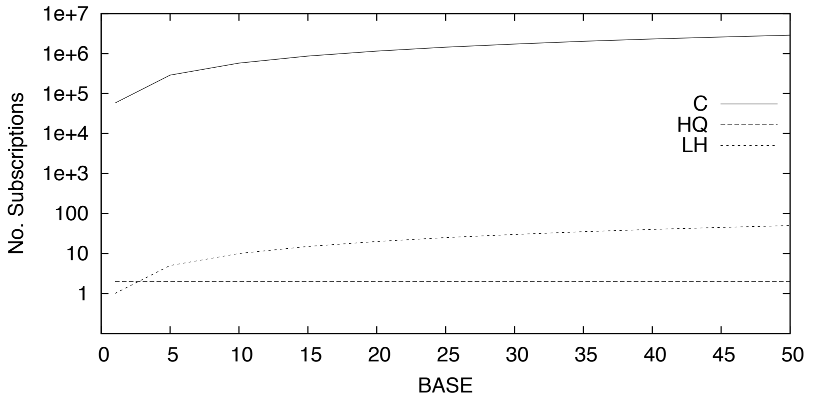
EnvironmentData Driver We assume that 30 percent of all shipments require monitoring, e.g., of temperature. The monitoring is either performed within the trucks or the hubs which then generate the environment data messages; the messages are consumed by multiple customers.

4.5 Message Contents

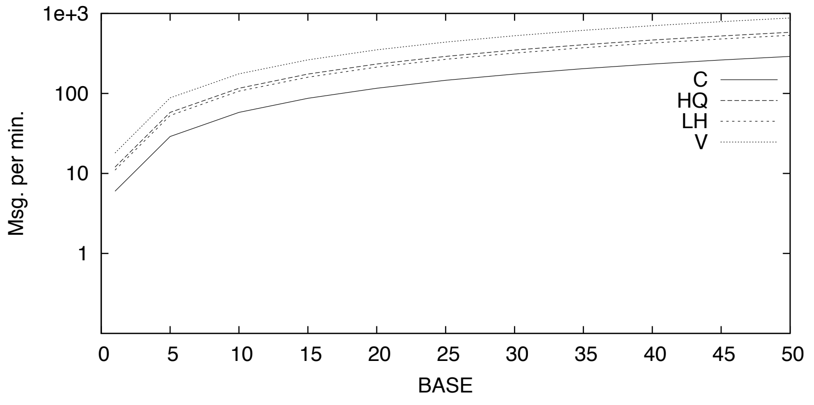
All messages include timestamps and identification information of the message producer. Shipment orders contain further address information necessary to deliver the goods. The order confirmation, as reply to the order message, contains the shipment identification data necessary for tracking- and monitoring-subscriptions. The pickup order contains the address data and, in addition, the ID of the hub being responsible for picking up goods. Shipment scans contain the ID of shipments as well as identification information of the hub the RFID reader is installed at. The proof of delivery contains recipient related data, e.g., the name of the person acknowledging the reception of the shipment. Position data messages contain GPS coordinates in addition to the mandatory vehicle identification data. Environment data messages contain the environment monitoring values, e.g., temperature data, as well as all shipment IDs of goods monitored at a specific hub or vehicle.

4.6 Workload Characeristics

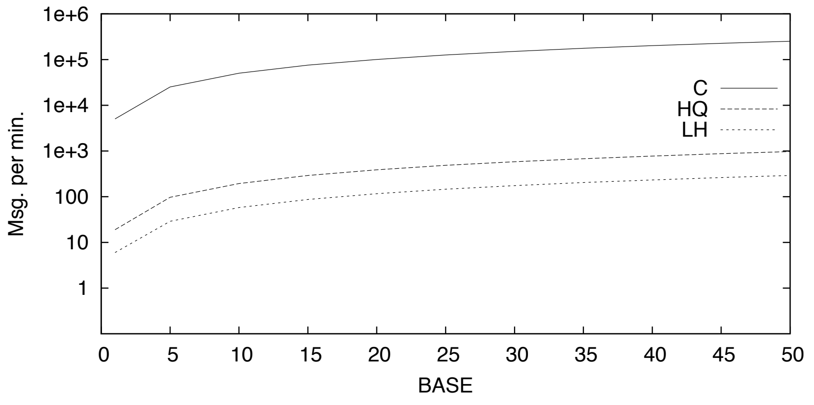
To illustrate the scaling behavior of our workload, Figure 3 shows different characteristics in terms of number of subscriptions and messages per entity. At this, entities are seen as a whole, e.g., `LH` refers to all hubs, `C` refers to all customers together.



(a) Subscriptions per Entity



(b) Published Messages per Entity



(c) Received Messages per Entity

Fig. 3. Workload Characteristics

Figure 3(a) shows the number of subscriptions per entity. While a constant number of subscriptions is issued by `HQ`, the number of subscriptions issued from `C` and `LH` grows linearly with an increasing base. Although the overall number of subscriptions increases, the number of subscriptions per single entity, e.g., per customer, remains constant. Characteristic of our workload is the large number of subscriptions; many customers subscribe in order to receive information about their shipments. With $\texttt{BASE}_L = 10$ already more than 500.000 subscriptions of customers exist.

In Figure 3(b) the number of published messages per minute is shown. Since our workload scales with $\texttt{BASE}_L$, the number of hubs, the amount of goods and customers increases accordingly leading to a linear increase in the number of messages. The same holds for the number of received messages as shown in Figure 3(c). The combination of both figures illustrates the publish/subscribe characteristics of our workload; while the number of published messages for $\texttt{BASE}_L = 10$ ranges from around 60 to 180 per entity, the number of received messages for customers goes up to 50.000. This difference originates from the nature of publish/subscribe communication. In our workload `PositionData` and `EnvironmentData` messages are of interest for many different customers, e.g., position data is relevant for all parcels within one vehicle.

As for the number of subscriptions, the number of messages per single entity remains constant except for the `HQ` entity. Since the number of `HQ` is constant, an increasing number of `LH` and `C` leads to an increased message load of `HQ` and thus stresses the system yet in another dimension.

5 Conclusion and Outlook

In the ADiWa project we identified the middleware as a key component for a successful adoption of event-based architecture in business environments. In particular, a reliable event transportation mechanism is needed that is highly scalable and performs according to the business needs. To ensure that a notification middleware provides the QoS specified in the service level agreements, a detailed analysis and evaluation has to be performed. This can be achieved by applying comprehensive workloads to compare various setups with respect to different system dimensions in a realistic and independent way.

In this paper we introduced a novel workload for event notification services in highly dynamic environments. We specified three realistic business interactions (including event types and communication patterns) covering several processes of the logistics domain. Characteristic of the logistics domain is a large number of subscribers (e.g., customers) whereas each subscriber receives only a small amount of messages (e.g., tracking information of specific shipments). Further, subscribers and publishers join and leave the system constantly, i.e., the event notification middleware has to orchestrate a highly dynamical environment. In contrast to synthetic workloads often used to demonstrate the capabilities of systems, our workload is specified independently from a particular event notification middleware or standards and allows us to evaluate different features of a middleware in a realistic way. Furthermore, previous realistic workload definitions targeting event-based systems assumed that publishers and consumers stay connected to the middleware. While this assumption is true for many scenarios, in this paper we considered scenarios with a dynamic changing environment with event producers and consumers joining and leaving the system. With each join or leave the event notification routing has to be adapted and potentially calculated subscription aggregates have to be revised. Handling this task efficiently is a key point in providing a high level of service quality. Our workload is the first considering these dynamic environments and allows evaluating system quality in highly dynamic environments.

As part of the ADiWa project, we are working on a prototype implementation of our workload and plan to analyze the service quality of the event notification middleware.

References

1. 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.
2. 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.
3. K. M. Chandy and W. Schulte. *Event Processing: Designing IT Systems for Agile Companies*. Mcgraw-Hill Professional, 2009.
4. D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.
5. 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.
6. 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.
7. B. McCormick and L. Madden. Open architecture publish subscribe benchmarking. In *Proceedings of the OMG Real-Time and Embedded Systems Workshop*, 2005.
8. 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.
9. J. Moore, T. Collins, and S. Shrestha. An open architecture for detecting earthquakes using mobile devices. volume 1, pages 437 –441, apr. 2010.
10. G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
11. K. Sachs. *Performance Modeling and Benchmarking of Event-Based Systems*. PhD thesis, TU Darmstadt, 2010.
12. 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.

13. 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.
14. United Parcel Service of America, Inc. UPS facts website, Aug 2010. http://www.ups.com/content/us/en/about/facts/worldwide.html.
15. U. Varshney. Pervasive healthcare and wireless health monitoring. *Mob. Netw. Appl.*, 12(2-3):113–127, 2007.