Quality of Service in Event-based Systems

Stefan Appel
TU Darmstadt, Germany
appel@dvs.tu-darmstadt.de

Kai Sachs
TU Darmstadt, Germany
sachs@dvs.tu-darmstadt.de

Alejandro Buchmann
TU Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

ABSTRACT

Future software systems must be responsive to events and must be able to adapt the software to enhance business processes. Examples are production and logistics processes that must be rescheduled based on relevant traffic information. It is therefore essential that relevant events are detected and transported to the software components responsible for dynamic changes. The trust in such reactive systems depends to a large extent on the *Quality of Service (QoS)* provided by the underlying event system. This paper introduces QoS aspects in event-based systems and discusses ways of identifying and evaluating QoS needs. This is done by identifying the different system layers, and the quality requirements at each layer based on the layer's functionality.

1. INTRODUCTION

Quality of Service (QoS) in *event-based systems (EBSs)* is important when developing solutions to dynamically support and monitor business processes. A user requires reliability guarantees in order to trust the output of a system supporting business critical operations. The QoS aspects discussed in this paper are driven by the research project *ADiWa*[1]. The goal of ADiWa is letting events from the Internet of Things (IoT) dynamically influence and enhance business processes. Industrial partners provide different use cases from the areas of logistics, business services, retail and production; for these scenarios, event-based system prototypes are developed that enhance the underlying business processes automatically.

In event-based systems event producers and event consumers are decoupled and communicate via asynchronous communication patterns. Event consumers may subscribe to events of interest. Whenever those events are detected, the subscriber is notified. Since the invocation of business critical processes is now triggered by events, the QoS of the event mechanism becomes a key aspect. Defining QoS and developing the necessary monitoring mechanisms is a major challenge. First, the system boundaries must clearly be defined: is only event transport of interest, or have event production, composition and consumption taken into consideration as well. Once the scope of the event system is defined, overall quality criteria and criteria for each component must be defined, monitored and enforced. Quality of Service in EBSs is strongly influenced both by functional properties, such as ordered delivery of events, and non-functional properties, such as throughput an availability.

In this paper a general overview of QoS in event-based systems is given and an architecture that supports the different types of QoS in an EBS is presented. The main focus is on QoS of the notification service (*Message-Oriented Middleware, MOM*, sometimes referred to as *Event Bus*) as well as the QoS in terms of *Complex Event Processing (CEP)*. The paper is organized as follows: first, related work is presented pointing out current directions of research. Based on those, a generic approach is chosen to identify QoS factors in event-based systems. The paper continues with a system architecture describing the different qualities occurring within the scope of an EBS. Finally performance evaluation and monitoring of event-based systems is addressed as a controlling instrument for ensuring QoS.

2. RELATED WORK

Research in event-based systems can be categorized into three broad areas: event production, event transportation and event consumption. Typical event producers are wireless sensor nodes and RFID readers but also complex event processors (CEPs) producing, e.g., composite events. Event consumers are, for example, complex event processing engines or business applications in general. To connect producers and consumers a middleware is necessary and also responsible for transporting the event notifications. Events are represented by event objects which are packaged into notifications [7, 12]. A schematic view of an EBS is shown in Figure 1.

For the communication between producers and consumers the paradigm of choice is publish/subscribe, allowing a decoupled many-to-many communication. The pub/sub middleware must provide a certain QoS [8]. At present, one of the most widely used technologies providing pub/sub capabilities is the Java Message Service (JMS) [23]. A variety of different implementations exist; popular ones are ActiveMQ

[1]Allianz Digitaler Warenfluss (ADiWa). Funded by the German Federal Ministry of Education and Research under grant 01IA08006

GvD Workshop 2010, 25.-28.05.2010, Bad Helmstedt, Germany.

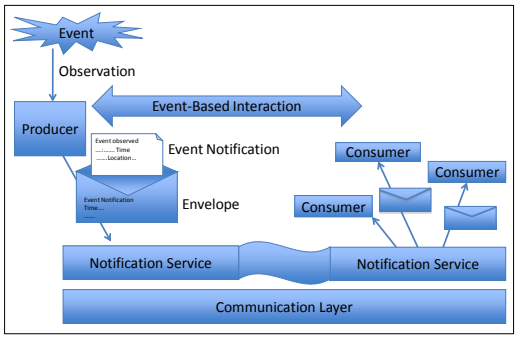

Figure 1: Structure of Event-based Systems

[24], HornetQ [20], IBM Websphere, TIBCO Enterprise Message Service or Oracle WebLogic Server. All these products follow the JMS specification and provide the corresponding QoS functionalities. For JMS these are support for transactions, persistence and durability.

Besides JMS, the *Data Distribution Service (DDS)* [17] and the *Advanced Message Queuing Protocol (AMQP)* [1] are other standards for MOM. AMQP is a not yet finally specified network wire-level protocol which enables interoperability among different MOM implementations and programming languages; it is comparable to JMS in terms of reliability and messaging capabilities. The focus of DDS are real-time distributed systems whereas JMS was originally designed for business applications. In terms of QoS, DDS does support more options than JMS does; [17] gives an overview over the supported QoS policies, e.g., DDS allows the specification of transport priorities for notifications. Depending on the area of application JMS, AMQP, or DDS might be the messaging standard of choice.

Although JMS, AMQP, and DDS support a variety of QoS features, important QoS needs, e.g., ordering events from multiple producers or the occurrence of false positives and false negatives, are not addressed and still topics of research and of interest for future systems. JMS and DDS are used in QoS research: [13] presents concepts for the management of QoS in DDS, [22] uses the SPECjms2007 benchmark to evaluate compliance and performance of JMS middleware.

In addition to JMS, AMQP, and DDS, which are used in real-world applications, many research prototypes of middleware systems exist to evaluate new concepts and features. While JMS follows a centralized, topic-based approach, current research systems are highly distributed and support, for example, content or concept based pub/sub [2]. The use of these distributed event-based systems (DEBSs) is necessary to cope with high volume of events expected to occur in future applications. With the spread of mobile smart devices, events are produced and received anywhere making some sort of intelligent middleware indispensable. Research prototypes of DEBSs are for example PADRES [11], SIENA [6], HERMES [19] or REBECA [2]. Prominent research topics include routing [10] and filtering [9] in DEBSs. Further, the support of transactions [16] is an important property in order to build systems supporting business-critical processes. A system overview of DEBS addressing, amongst others, routing, filtering and transactions is given in [5] and [18].

Generally, QoS has not been addressed extensively in research; a basis is provided in [4] where basic QoS metrics (e.g., latency, bandwidth, delivery guarantees) are discussed. In the ADiWa project we tried to apply these metrics directly and found out that they are very technical on one hand, and incomplete on the other, since they do not address many of the critical functional aspects that have an impact on QoS. Therefore, we used a list of EBS features as a starting point to determine the functionality needs of EBSs. From them we derived the QoS requirements.

3. QOS IN EVENT-BASED SYSTEMS

3.1 Features in EBSs

ADiWa started with the identification of business processes which can be dynamically supported by an event-based system. From the technical perspective this is a high-level view and many common QoS criteria, like performance of the EBS, are not applicable at this early stage. To address this issue we categorized features of EBSs to enable a QoS-aware specification and development process. The used features are based upon an analysis of real-world event-based systems [12]. Examples are: support for mobility, support for transactions, early filtering, real time capabilities, or support for interval events. As first step those more than 40 single features were grouped into five main categories: event types (e.g., support for interval events), general requirements (e.g., handling of out of order events), (legal) limitations and specifications (e.g., privacy protection), technical properties (e.g., support of prioritization), and runtime requirements & performance (e.g., latency). These features determine which type of EBS is needed with respect to the relevant business cases. Thus, rather than building a Swiss army knife EBS supporting all imaginable features and QoS needs, the requirements are matched during the requirements engineering and development process.

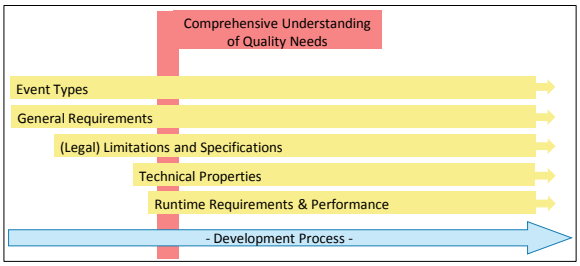

Figure 2: Features of Event-based Systems

Figure 2 shows the above mentioned categories with respect to the time line of a development process. As it can be seen, general requirements as well as event types should be identifiable rather early, during the first steps of requirements engineering. Later on, legal limitations and further system specifications can be determined before technical properties can be addressed. As last step, runtime requirements and performance needs can be identified. A comprehensive understanding of QoS is only possible after all required features have been identified. A detailed specification of QoS requirements can now be produced.

3.2 Characterizing the EBS

Based on the relevance of the various features in a given application scenario a specific EBS can be configured that conforms to the required QoS. Figure 3 shows the conceptual approach of defining QoS for certain features. For each feature one or more metrics are necessary to measure and monitor this feature. For some features the metric to measure QoS might be as simple as *Is supported [yes/no]*, for other features multiple metrics might be necessary. An example is the support of the feature early filtering with filter merging and the tradeoff between reduced number of filters and increased number of false positives.

By merging filters we reduce the number of filters that must be maintained and evaluated but increase the number of false positives and the number of forwarded messages because the filter is now less precise. As shown in Figure 2, early filtering results in multiple QoS attributes such as accuracy and efficiency. Efficiency is ultimately measured through resource utilization of the broker node.

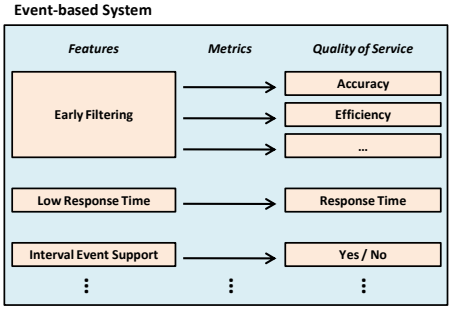

Figure 3: Defining QoS

A major problem that must be considered when addressing QoS in EBSs is the fact that features and derived QoS requirements are not orthogonal, i.e. they may influence each other. However, the more concrete the requirements for an EBS are specified and as more design decisions are made during the design process, the easier it becomes defining and monitoring QoS.

4. A QOS AWARE INFRASTRUCTURE

So far we limited the scope to event transport (MOM) and complex event processing (CEP). As shown in Figure 4, the complete system includes also the event production layer and the dynamic business process layer.

At the lowest level the events enter into system; at this point there is an associated production quality. This quality is determined by the event producers, for example wireless sensor nodes. These nodes can monitor the environment and have, for example, a limited accuracy (Temperature +/- 1 degree) which might influence later event processing. The highest levels of the system are the business processes driven by events where the goal is to dynamically adapt these processes according to occurring events. The adaption of processes is associated with QoS again: *controlling quality*. Associated QoS metrics describe how well processes are adapted and whether changes lead to improvements. This can be measured with *Key Performance Indicators (KPI)* assigned to processes. The two remaining levels of QoS concern the inner parts of the system: the event transporting layer as well as the event processing layer.

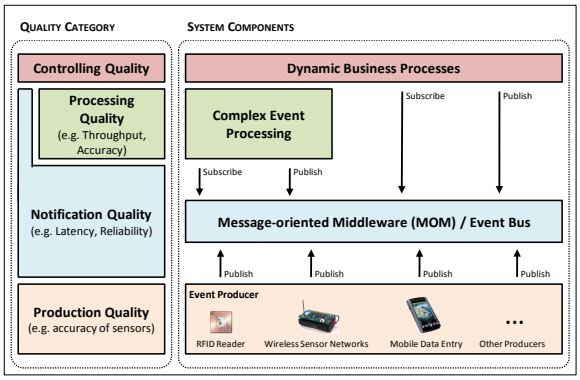

Figure 4: QoS Architecture

Since QoS affects all layers of a system a system-wide approach for managing quality information is needed. Every single event has its related *production quality* and is then transported and processed with a certain quality. During all these steps quality information for each event has to be maintained either directly at the event level or at higher granularity like event streams. The complexity of the QoS information suggests that in addition to attaching quality data to events a central registry for QoS data is helpful (Figure 5).

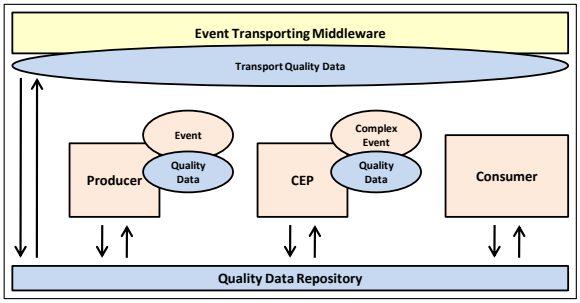

Figure 5: Types of Quality Data

QoS data can be archived. This QoS history is then accessible and the middleware might draw conclusions during the event transport: events originating from a sensor reporting critical situations might require a highly reliable real-time handling within the middleware. To support features along these lines an appropriate event format and a system-wide understanding of the existence of QoS is necessary.

A special challenge in terms of QoS is the support of transactions since transactional processing does involve producers, middleware, and consumers and thus affects all QoS layers. Transactions are, for instance, needed in case event producers require acknowledgments from a consumer before publishing another event (e.g., shipment event before billing event). To support this functionality in a flexible and reliable way, transaction management has to be supported by the middleware [16]. Without middleware mediated transactions it would not be possible to inform the event producer

whether a message was just lost or whether there is no appropriate subscriber. Depending on this information further steps can differ: resending the message vs. reacting to not available subscriber.

5. EVALUATING & MONITORING MOM

To ensure the compliance of a system with the QoS requirements techniques based on modeling, benchmarking and monitoring are used.

5.1 Modeling

Various techniques exist for the modeling of event-based systems. A promising approach is the use of Queuing Petri Nets (QPNs) [3, 15], which combine the advantages of Queuing Networks and Petri Nets. The basic building blocks of a QPN are places, transitions and tokens. Tokens flow through the net via transitions; at each place the "processing" of a token takes a certain amount of time, the so called service time. In the context of event-based systems, an event is represented by a token. After building the model and calibrating the service times, simulation tools (e.g., SimQPN [14]) allow predicting the throughput and latency of the whole system in a reasonable time. In addition, resource utilization (e.g., CPU utilization) of single components, like brokers, can be estimated.

5.2 Benchmarking

Another approach is the use of benchmarks to evaluate the performance and QoS of systems. At first, a benchmark can test whether the promised QoS is provided by the underlying EBS, e.g., in terms of latency. Second, a well designed benchmark has a scalable workload allowing determining the limits of systems and comparing implementations of different vendors in an independent way. One of the challenges when developing a benchmark is choosing an appropriate workload. Since building separate workloads for various application scenarios is usually not possible, a benchmark uses a workload with properties which are characteristic for many applications. Further, it can be designed to support software developers and architects during the development process. When choosing a current middleware which supports pub/sub, e.g., following the JMS standard, the developers still have the choice whether to use a single topic for all messages or various topics. In terms of ADiWa the decision would be whether to use an event bus where all events flow through the same stream (destination), or whether to structure events and use various streams (destinations) forming the overall event bus. To evaluate, amongst others, these different alternatives jms2009-PS was developed [21]; it uses the SPECjms2007 workload as a basis and emphasizes on pub/sub messaging. SPECjms2007 is the current industry-standard benchmark for MOM servers based on the JMS and models a supermarket supply chain where RFID technology is used to track the flow of goods.

jms2009-PS provides more than 80 configuration parameters allowing the user to customize the workload in terms of the number of destinations (topics), the number of subscriptions, the number and type of selectors, and the message delivery modes (QoS). With these parameters alternative ways to implement pub/sub communication can be evaluated in terms of their overhead, performance and scalability.

5.3 Monitoring

Besides modeling and benchmarking the monitoring of a productive event-based system is important to ensure QoS. Collected data can be incorporated with models, validated by simulations, and utilized to develop a characteristic benchmark workload. Further, monitoring allows the early detection of potential bottlenecks and gives developers the possibility to adapt and extend the system in advance. Monitoring is done by collecting runtime information. This can be accomplished in a pub/sub manner: each event producer or consumer does not only publish events but also statistical information. In addition, the event transporting middleware itself can publish data accounting for QoS. Parties interested in statistical data can then simply subscribe to messages and perform further analysis. The middleware can act as a consumer, too. By this a self-adapting middleware can be realized which dynamically reconfigures to constantly assure the required QoS.

To allow the reporting of statistical data middleware implementations need to be adapted. Unfortunately monitoring a system introduces an overhead. The overhead increases with the granularity of monitoring, thus a tradeoff between monitoring granularity and potential monitoring benefits has to be found in productive systems. It is also possible to build a system with adjustable monitoring capabilities. If such a monitoring framework exists, it becomes possible to increase the monitoring granularity either on a regular basis or whenever an in-depth problem analysis is needed.

6. CONCLUSION

Dynamically enhancing business processes using events originating from a variety of producers requires a middleware for transporting the data. Within the ADiWa project requirements for those middleware systems are a topic of research. In this paper an approach for identifying, defining and ensuring QoS needs is presented. At first, relevant features of an EBS are identified; this can be accomplished stepwise as the requirements engineering process proceeds. Based upon the determined functionalities, QoS metrics can be developed and evaluated using, e.g., benchmarking, modeling, and monitoring mechanisms. This QoS-aware software development process assures that quality needs are taken into consideration early in the design process. In addition to the identification of QoS needs the management of QoS relevant data is a challenge. QoS data does occur at various places in systems and thus collecting relevant parts of it in a central repository is necessary. Based upon archived QoS data the system can be evaluated and tuned.

7. REFERENCES

[1] AMQP Working Group. Advanced Message Queuing Protocol, 2010. `http://www.amqp.org`.

[2] J. Antollini, M. Antollini, P. Guerrero, and M. Cilia. Extending REBECA to support concept-based addressing. In *Proceedings of ASIS*, 2004.

[3] F. Bause. Queueing Petri Nets - A formalism for the combined qualitative andquantitative analysis of systems. In *Proceedings of Petri Nets and Performance Models*, 1993.

[4] S. Behnel, L. Fiege, and G. Muehl. On quality-of-service and publish-subscribe. In

Proceedings of ICDCSW, 2006.

[5] A. Buchmann, C. Bornhoevd, M. Cilia, L. Fiege, F. Gaertner, C. Liebig, M. Meixner, and G. Muehl. Dream: Distributed reliable event-based application management. In M. Levene and A. Poulovassilis, editors, *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*. Springer, 2004.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of PODC*, 2000.

[7] K. Chandy and W. Schulte. *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill, Inc., 2010.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2), 2003.

[9] 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 SIGMOD*, 2001.

[10] C. Fengyun and J. P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *Proceedings of INFOCOM*, 2004.

[11] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The PADRES distributed publish/subscribe system. *Feature Interactions in Telecommunications and Software Systems*, VIII, 2005.

[12] A. Hinze, K. Sachs, and A. Buchmann. Event-based applications and enabling technologies. In *Proceedings of DEBS*, 2009.

[13] J. Hoffert, D. Schmidt, and A. Gokhale. A QoS policy configuration modeling language for publish/subscribe middleware platforms. In *Proceedings of DEBS*, 2007.

[14] 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), 2006.

[15] S. Kounev, K. Sachs, J. Bacon, and A. P. Buchmann. A methodology for performance modeling of distributed Event-Based systems. In *Proceedings of ISORC*, 2008.

[16] C. Liebig and S. Tai. Middleware mediated transactions. In G. Blair, D. Schmidt, and M. Takizawa, editors, *Proceedings of DOA*, 2001.

[17] Object Management Group (OMG). OMG Data Distribution Service (DDS) Specifications. 2007. `http://www.omg.org/spec/DDS/`.

[18] P. Pietzuch, D. Eyers, S. Kounev, and B. Shand. Towards a Common API for Publish/Subscribe. In *Proceedings of DEBS*, 2007.

[19] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of ICDCSW*, 2002.

[20] Red Hat, Inc. JBoss HornetQ, `http://www.jboss.org/hornetq`, 2009.

[21] K. Sachs, S. Appel, S. Kounev, and A. Buchmann. Benchmarking publish/subscribe-based messaging systems. In *Proceedings of BenchmarX*, 2010.

[22] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation*, 66(8), 2009.

[23] Sun Microsystems. Inc. Java Message Service Specification Final Release 1.1, 2002.

[24] The Apache Software Foundation. Apache ActiveMQ, `http://activemq.apache.org/`, 2009.