Performance Modelling of Message-Oriented Middleware with Priority Queues

Snigdha Singh
snigdha.singh@kit.edu
Karlsruhe Institute of Technology

Larissa Schmid
larissa.schmid@kit.edu
Karlsruhe Institute of Technology

Anne Koziolek
koziolek@kit.edu
Karlsruhe Institute of Technology

Abstract

Message-Oriented Middleware (MOM) with priority queues reduces the latency of critical events. In general, MOM uses a FIFO queuing methodology. But, different application scenarios require certain critical events with higher priority to be served earlier over low-priority events, so that the subscriber of the event consumes the high-priority event with less delay. In the context of the Palladio Component Model (PCM), MOM-based systems have been modelled considering message queue length and latency as metrics for performance prediction and simulation. However, the approaches did not consider modelling MOM with priority queues and their impact on performance. We will first, discuss the existing approaches in PCM which support performance prediction for MOM-based systems and then propose how they can be extended to support performance predictions for MOM with priority queuing. We will then conclude which approach is best suited to extend by assessing their capabilities to predict performance metrics relevant for priority queuing, especially the delay of individual events at the subscriber end.

1 Introduction

RabbitMQ is a widely used message-broker which uses a first-in-first-out (FIFO) strategy for message queue consumption. That means messages are consumed at the receiver end in the exact order that they were posted to the queue at the sender. RabbitMQ is based on AMQP protocol that generates queues with several properties to define the behaviour of the queue. Such properties are queue name, queue durability, queue message length limit. These properties of the queue determine the performance of the system.

However, in case of a messaging scenario in which a consumer receives more important messages (high-priority) and less important messages (low-priority) in the same queue, the message-broker is not designed to schedule and process these priority messages. This introduces the processing delay of high-priority messages and increases the latency. Increased latency reduces system performance.

RabbitMQ has started supporting the priority queue implementation from its core version of 3.5.0. Implementing the priority queue as an additional quality attribute of the queue can help maximize application performance by separating messages based on processing priority. For example, higher-priority messages can be prioritized to be handled by consumers first, while less important messages can be handled by the consumer later. Thus improving the overall performance of the message-oriented system.

Palladio Component Model (PCM) is an approach to predict the system performance at design time, without access to the application implementation. PCM has already been extended to model event-driven service-oriented systems to evaluate the performance and other quality attributes of the system.

The ”event-extension” approach added new model elements to PCM for modelling the event-driven systems for performance prediction [2]. With the help of extended model elements, architects can configure the message-based communication using the PCM meta-model language based on different messaging patterns (publish-subscribe / point-to-point) and can predict the performance and other quality attributes at the design phase.

The ”message-queuing-simulation” approach further extended the PCM to model and simulate MOM in a more detailed manner [4]. It uses a separate messaging simulation which enables the explicit queuing strategies and asynchronous calls.

Both modelling approaches are based on FIFO messaging strategies. In both meta-model languages, they did not consider the priority of the messages and how to process them with minimum delay.

In our approach, we explored the possibility of extending the PCM so as to support the modelling and simulation of priority queues. We compared the event-extension approach and message-queuing-simulation approach to figure out how the messages can be served based on their priority number at the subscriber end. Among other things, theoretically, we discussed, how much delay can be occurred due to the priority scheduling at the run time for both the approaches.

In the next section, we discussed in detailed, the already existing queue-based modelling and simulation approaches in PCM to predict the performance of event-driven communication. Followed by modelling a further extension of PCM to support priority queue. In the last section, we conclude the paper by suggesting the best possible PCM-extension to support modelling priority queues comparing queuing strategies.

2 Event-Extension

Rathfelder introduced events as first-class elements in PCM [1]. In event-based systems, the business logic is implemented and executed in the *EventHandler* method. The communication between sender and receiver happens via events that can be sent and received. In the PCM an *EventGroup* was introduced for this purpose. This group behaves similar to an interface. An *EventGroup* can have one or more *EventTypes*. An *EventTypes* can have a payload. The component that sends events from an *EventGroup* needs a *SourceRole*. The component, that receives the events of an *EventGroup*, has a *SinkRole*. The *SinkRole* and *SourceRole* behave similar to a RequiredRole and ProvidedRole. The components can send and receive events. In order to do that, model elements are needed to enable the sending and receiving. The sender can be activated with an *EmitEvent* action that sends an *EventType* within a SEFF (Service Effect Specification) via the *SourceRole*. The receiver requires a *SourceRole* and a SEFF, which serves as event handler.

The PCM offers two possibilities to handle messaging models. If components have to communicate in point-to-point methods, then events are sent directly to the components. If components have to communicate with each other in a publish-subscribe scenario, an *EventChannel* is required. An *EventChannel* refers to exactly one *EventGroup*. The sending assembly context connects its *SourceRole* to the *EventChannel*. The receiving assembly context connects its *SinkRole* to the *EventChannel*.

A model-transformation method is then used to enable performance analysis of the event-driven system [2]. The interactions between the messaging components and performance attributes are captured and modelled with PCM for performance completion.

The limitation of the work is that queues of the middleware are not explicitly considered and certain effects are not evaluated for messaging middleware. For example, in case of increased latency in a message queue, the delay can not be predicted accurately using the existing approach. The impact of message priority on performance is also not considered in the current methods.

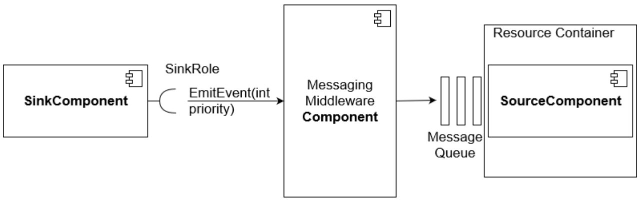

Figure 1: Priority scheduling for the events extension

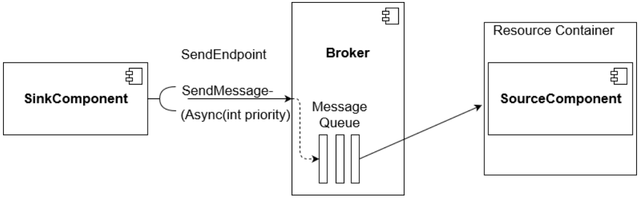

Figure 2: Priority scheduling for the messaging simulation

In order to enable priority messaging in the event-extension approach, a *number* of the message can be added as a new attribute to the *EmitEvent* action. Furthermore, the messaging middleware components (MOM) would need to be extended to forward the priority-number to the receiver side. As mentioned in Fig. 1, the queues are then scheduled to be processed with the PCM extension of the preemptive-priority scheduling approach [3]. In their approach, the priorities have been assigned statically at the processing end, however, we want to extend to assign priorities dynamically on the sender side. At the receiver end, the SEFF of the receiver calls the *IPriority* interface of the resource container. This schedules the messages and processes them at the receiver end based on the priority-number.

The latency and queue length are the metrics of interest here. Latency can be measured as the total time between emitting the event at the sender and processing the message at the receiver. The queue length can be determined by the length of the queue in front of the resource container. However, if a component is consuming from more than one queue at a time, the length of individual queues cannot be determined because they result in a single queue in front of the resource the component is located on. This is one of the limitations of the discussed approach.

3 Message-Queuing-Simulation

Building on the event-extension by Rathfelder, we proposed an extension of the PCM to support truly asynchronous calls and explicit queuing [4]. Components can send messages via the *SendMessageAsync* action which inherits from the *EmitEvent* action. In the assembly view, brokers and routers are explicit modelling elements connected to components by message channels, which can be either point-to-point or publish-subscribe. They route incoming messages based on their *EventGroup* and routing key to outgoing message channels. Each message channel contains a *QueuingProperties* model element where details of the resulting queue in the simulation can be specified. Supported attributes are the maximum queue length and whether the queue should be durable. Furthermore, a scheduling policy that determines in which order messages are taken out of the queue and forwarded to consumers can be chosen. However, until now, only FIFO is supported as scheduling strategy.

The messaging infrastructure is transformed into entities of the messaging simulation model prior to the execution of the simulation. When the *SendMessageAsync* action is executed, a simulation interface is invoked, storing all PCM-relevant data and forwarding message ID and size to the messaging simulation. The message is then routed by the specified broker and router to one or more queues. Receiving components can specify a per-queue prefetch, defining how many messages they accept without acknowledging the previous ones. Components can acknowledge messages directly after receiving them or after they have processed them. Whenever a message arrives at a queue, it is checked whether the consumer of the queue is ready to receive the message. Otherwise, it is queued up and delivered when the consumer is available again.

For supporting priority queuing, the *SendMessageAsync* action needs to support the specification of a priority. This could be achieved by a *VariableCharacterisation* for the priority which is then handed to the messaging simulation. It is the same as adding the *number* to the *EmitEvent* action. Furthermore, priority queuing must be added as available scheduling policy for the queuing properties of a message channel. Simulation-wise, this new scheduling strategy must be considered during the transformation of the PCM model to entities of the messaging simulation model. Since queues are separate simulation entities which forward messages to consumers, they also need to support priority queuing. This can be achieved by implementing a multi-level queue for each logical queue, where one level defines a certain priority. The possible extension can be realised as in Fig. 2. Similar to the event-extension approach, latency and queue length can be measured here as well.

4 Discussion

In the event-extension approach, the queues are outside of the resource container, however, in the case of the message-queuing-simulation extension the queues are inside the message broker. In the case of a component consuming from more than one queue at a time, queues being located on the message broker have the advantage that the length of all individual queues can be measured. This is because they do not end up in a single queue in front of the resource container. The queues are processed based on their priorities in the message-broker as compared to outside of the resource container in case of the event-extension.

The modelling of the message-queuing-simulation priority extension will be much easier and semantically more clear as compared to the event-extension. This is because of the reason that priority queuing can directly be modelled in the assembly view type instead of in the resource environment view type. The major limitation with the event-extension approach to support priority queue is its lack of ability to measure the individual queue length at the receiving end.

In addition, it would be difficult to have a component to consume from two different queues if one is using priority queuing and the other one is not. For example, in case of a real-time scenario where there is a queue with alarm messages of different priorities and a queue with maintenance messages with no priority, that should be processed with FIFO scheduling. It may so happen that the maintenance messages with no priority may never get processed. This is because the messages are queued and processed based on the priority on the resource level at receiving end and not in the message-broker in separate queues.

5 Conclusion

In this paper, we proposed the possible extension of PCM to support priority queuing in message queues for MOM to predict the performance. We discussed the two existing approaches for modelling MOM and how they could be extended. We concluded that the message-queuing-simulation approach will be a better alternative compared to the event-extension approach. As the future work, we plan to implement the extension approach with a real-time case study and measure the latency and queue length for validation.

References

[1] C. Rathfelder. *Modelling Event-Based Interactions in Component-Based Architectures for Quantitative System Evaluation.* Karlsruhe Series on Software Design and Quality / Ed. by Prof. Dr. Ralf Reussner. KIT Scientific Publishing, 2013.

[2] C. Rathfelder et al. “Modeling event-based communication in component-based software architectures for performance predictions”. In: *Software & Systems Modeling* 13.4 (2014), pp. 1291–1317.

[3] J. Fernández-Salgado et al. “Integration of a preemptive priority based scheduler in the Palladio Workbench”. In: *Journal of Systems and Software* 114 (Apr. 1, 2016), pp. 20–37.

[4] L. Schmid. “Modeling and Simulation of Message-Driven Self-Adaptive Systems”. MA thesis. Karlsruher Institut für Technologie (KIT), 2020.