FAETERJ – RIO

ANDRÉ DA SILVA LEMOS

ARTHUR LOURENÇO MACHADO

GUSTAVO VIEIRA BALDUINO

JOÃO VITOR DE FREITAS CASTRO

JUAN LUIS FONSECA DA SILVA

MIDDLEWARE

SISTEMAS DISTRIBUÍDOS

RIO DE JANEIRO

2021

ANDRÉ DA SILVA LEMOS

ARTHUR LOURENÇO MACHADO

GUSTAVO VIEIRA BALDUINO

JOÃO VITOR DE FREITAS CASTRO

JUAN LUIS FONSECA DA SILVA

MIDDLEWARE

SISTEMAS DISTRIBUÍDOS

Trabalho de Fundamentos de Redes e Sistemas Distribuídos para a composição de nota da AV2 em TASI apresentado à FAETERJ-RIO.

Orientadora: Prof. Maria Cláudia.

RIO DE JANEIRO

2021

SUMÁRIO

1. INTRODUÇÃO

Nos dias de hoje, é nítido toda a variedade de linguagens de programação existentes em nosso cotidiano. Por isso, grande parte dos softwares encontrados hoje não se comunicam entre si, pois são feitos de maneiras diferentes, utilizando linguagens diversas. Assim, faz-se necessário o uso de uma ponte, que consiga interligar dois ou mais sistemas para que estes consigam comunicar-se entre si.

Portanto, o presente trabalho tem como objetivo mostrar toda a complexidade existente na comunicação entre dois ou mais sistemas distribuídos e o quão importante é o Middleware para isto.

2. O QUE É MIDDLEWARE

O Middleware permite a comunicação entre duas ou mais aplicações, podendo ser uma biblioteca, um programa ou um serviço. Sua função é fazer uma ponte entre o sistema operacional e os serviços que ele executa.

Basicamente, o middleware pode ser definido como uma “cola de software”, uma vez que ele consegue fazer a “união” entre diversas aplicações, mesmo que elas tenham métodos de comunicação diferentes. Em outras palavras, ele funciona transportando os dados de um sistema para o outro.

Digamos que dois servidores diferentes, A e B, fazem parte de um sistema. O servidor A roda um código em PHP, um site, como a loja de “skins” do jogo LoL (League of Legends) e o servidor B roda um código em C que recebe as informações de compra do usuário e atualiza os bancos de dados dos servidores que rodam o cliente do LoL para que ele exiba nas partidas a “skin” comprada.

Para que esse processo ocorra, o servidor A precisa se comunicar com o servidor B. Os programadores precisariam criar um código em PHP e em C que entendessem um ao outro, que definissem um protocolo de comunicação e abrissem uma conexão usando a rede para enviar os dados entre eles, esse processo demandaria tempo e recursos.

É aí que entra o Middleware, funcionando como a ponte para a comunicação entre os servidores.

Um Middleware também pode ser dividido em três categorias:

- Middleware baseado em RPC (Remote Procedure Call)
- Middleware baseado em ORB (Object Request Broker)
- Middleware baseado em MOM (Message Oriented Middleware)

2.1 PROBLEMAS QUE O MIDDLEWARE RESOLVE

O Middleware resolve um grande problema no desenvolvimento de sistemas, que é construir sistemas distribuídos diretamente sobre a camada de transporte da rede. Por ser muito difícil, se tornou necessário uma infraestrutura capaz de fornecer os meios para que as aplicações se comuniquem. Assim é possível evitar que os programadores tenham que se preocupar em fazer um protocolo para que os

diferentes sistemas se comuniquem (código específico de comunicação para enviar e receber dados entre aplicações), permitindo que eles possam focar nas funções específicas de cada aplicação.

3. TIPOS DE MIDDLEWARE

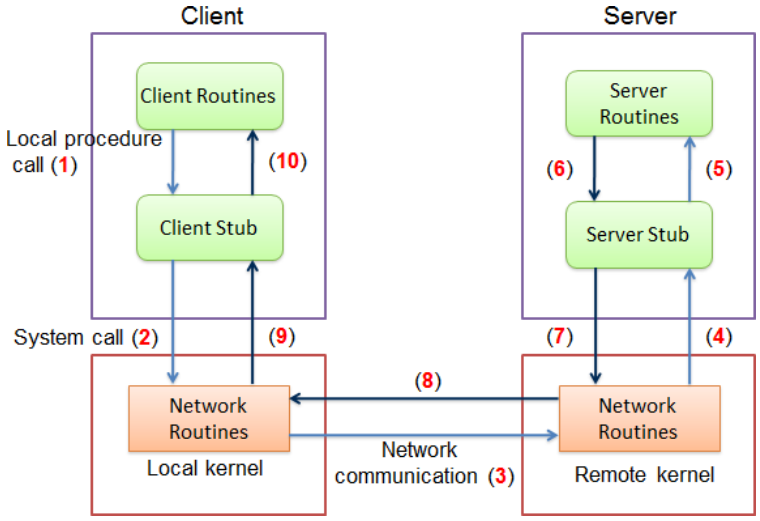
3.1 CHAMADA DE PROCEDIMENTO REMOTO (RPC)

Como o seu nome já diz, todas as requisições feitas por esse paradigma de comunicação são realizadas remotamente por um outro host conectado à rede, também conhecida como chamada de funções ou sub-rotina. Portanto o procedimento utiliza-se do modelo cliente/servidor, ao qual haverá sempre uma requisição por parte do cliente e a resposta do servidor. Assim ao decorrer da requisição enquanto não houver uma resposta por parte do servidor, a aplicação em virtude fica bloqueada, pois a princípio é uma chamada síncrona, com exceção dos casos em que a requisição seja um XMLHttpRequest.

O RPC ao longo dos anos evoluiu bastante e é um modelo bastante utilizado por várias aplicações, como: SunNFS, Java RMI, SOAP e CORBA. Devido a sua evolução também influenciou no desenvolvimento de protocolos como HTTP e REST.

3.1.1 COMO FUNCIONAM AS CHAMADAS?

Os pontos no RPC podem ser um cliente e um servidor, dois nós em uma rede ponto a ponto, dois hosts em um sistema de computação em grade ou até mesmo dois microsserviços. A comunicação RPC não se limita a apenas dois hosts, mas pode ter vários hosts ou endpoints envolvidos.

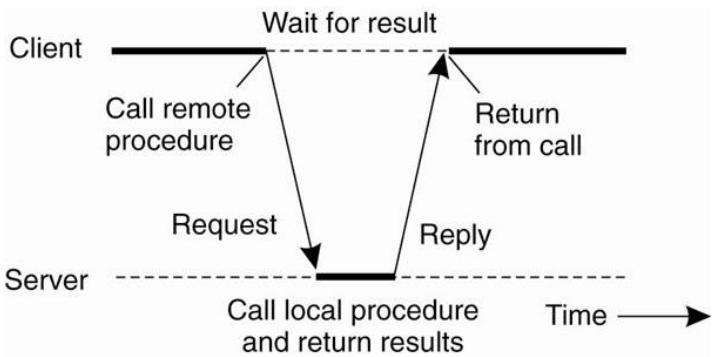


A imagem acima exemplifica um simples sistema de relacionamento RPC. Nota-se que estão presentes as rotinas e os stubs.

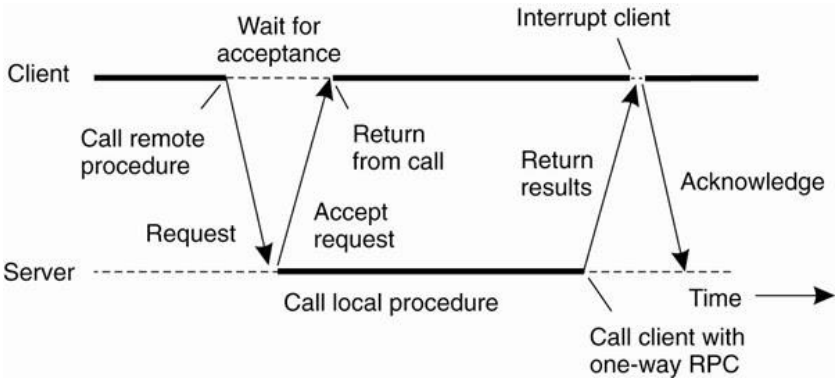
Os stubs ficam encarregados de fornecer uma interface para a rotina da aplicação em que se está fazendo uma requisição, assim quando há uma requisição do cliente para o servidor ou mesmo vice e versa, o stub fica encarregado de serializar os dados que serão transferidos pela rotina de rede através de uma comunicação de rede.

3.1.2 O RPC HOJE EM DIA

Um paradigma proposto pela primeira vez na década de 1980, que inicialmente tinha como propósito funcionar apenas em redes LANs, ao passar dos anos foram adicionados implementações para o seu funcionamento na WEB, devido à algumas funcionalidades que estavam deixando a desejar. Foram implementados a capacidade de fazer chamadas assíncronas, tratamento de falhas de rede e heterogeneidade para diferentes linguagens.



Na chamada síncrona, o cliente envia uma solicitação ao servidor e a aplicação é bloqueada até que uma resposta seja retornada.



A chamada assíncrona deixa a aplicação rodando mesmo quando há uma requisição, e depois de um certo tempo a resposta interrompe o programa para mostrar os resultados.

3.2 OBJECT REQUEST BROKER (ORB)

Utilizado no modelo de arquitetura CORBA, um Object Request Broker atua como um “broker” entre uma chamada do cliente e o serviço requisitado, ou seja, a solicitação é feita sem que o cliente saiba de todos os detalhes do servidor que está sendo requisitado. A troca de informações sobre interfaces e endereçamento são trocados durante uma execução de chamada. É basicamente um Middleware mais sofisticado, que inclui procedimentos de RPCS, Middleware orientado a mensagens, procedimentos de armazenamento de banco de dados e serviços ponto a ponto.

3.2.1 COMO FUNCIONA O OBJECT REQUEST BROKER (ORB)

Um ORB a princípio, utiliza o repositório de interface do CORBA para descobrir como localizar e se comunicar com um componente. Ao se criar um componente, há uma declaração de interfaces públicas ou o compilador da linguagem de programação fica encarregado de traduzir as instruções apropriadas. Depois as instruções são armazenadas como metadados para que uma interface possa funcionar de uma forma adequada e compatível com o serviço.

Alguns dos serviços que um ORB utiliza:

- Serviço de ciclo de vida, que define como criar, copiar, mover e excluir um componente;
- Serviço de notificação de eventos;
- Serviço de consulta, que permite um componente consultar um banco de dados, de acordo com algumas especificações, como SQL3, ODMG etc.

4. O QUE É MIDDLEWARE ORIENTADO A MENSAGEM (MOM)?

Middleware orientado. a mensagem (Message Oriented Middleware) é uma infraestrutura de software ou hardware que dá suporte à comunicação entre os componentes de um sistema distribuído por meio da passagem de mensagem. Uma mensagem pode ser a notificação da ocorrência de um evento ou uma solicitação de execução de uma operação.

O MOM permite que módulos de aplicativos sejam distribuídos em plataformas heterogêneas e reduz a complexidade do desenvolvimento de aplicativos que abrangem vários sistemas operacionais e protocolos de rede. O Middleware cria uma camada de comunicação distribuída que isola o desenvolvedor do aplicativo dos detalhes dos vários sistemas operacionais e interfaces de rede. Essa camada de Middleware permite que componentes de software que foram desenvolvidos independentemente e que são executados em diferentes plataformas de rede interajam uns com os outros. Os aplicativos distribuídos em nós de rede diferentes usam a interface do aplicativo para se comunicar. Além disso, ao fornecer uma interface administrativa, este novo sistema virtual de aplicativos interconectados pode se tornar confiável e seguro.

No Middleware orientado a mensagem o foco é a comunicação assíncrona persistente. Tem a capacidade de armazenamento de médio prazo para mensagens, não precisando que o remetente ou o receptor estejam ativos durante a transmissão da mensagem.

Em geral, o remetente só tem a garantia que, a certa altura, sua mensagem será inserida na fila do receptor. Não tendo garantias de quando ou se a mensagem será lida pelo receptor.

O MOM funciona com as aplicações se comunicando inserindo mensagens em filas específicas (com cada aplicação tendo sua fila particular para a qual outras aplicações podem enviar mensagens). As mensagens são repassadas por uma série de servidores de comunicação estas são entregues ao destinatário, mesmo que ele não esteja em funcionamento quando a mensagem foi enviada.

Essas mensagens podem conter qualquer tipo de dado, assim devem ser adequadamente endereçadas. Sendo esses endereçamentos feitos com o fornecimento de um nome exclusivo da fila destinatária no âmbito do sistema.

4.1 ARQUITETURA DO MIDDLEWARE ORIENTADO A MENSAGEM

Sua arquitetura é composta pela fila de fonte, fila de destino e gerenciadores de fila.

Fila de fonte: é na qual o remetente envia a mensagem, sendo filas locais do remetente, apenas podendo ser colocada as mensagens na mesma máquina ou em uma máquina na mesma LAN.

Fila de destino: é a especificação dentro de uma mensagem para o destino em que ela deve ser transferida.

Gerenciadores de fila: interagem diretamente com as aplicações que estão recebendo ou enviando as mensagens.

Sistemas colaborativos distintos podem trocar mensagens por meio de um fila de mensagens ou pela notificação de eventos.

4.2 VANTAGENS DO MIDDLEWARE ORIENTADO A MENSAGEM

- O paradigma de comunicação por mensagens é simples, natural e fácil de entender;
- A reconfiguração de sistemas é simplificada, pois os participantes não precisam conhecer os endereços uns dos outros – basta saberem onde é mantida a fila de mensagens;
- Participantes da comunicação não precisam se sincronizar para trocar dados, o que reduz o tempo ocioso durante a comunicação;

- Participantes não precisam estar permanentemente conectados à rede – basta conectar para enviar e receber mensagens.

4.3 DESVANTAGENS DO MIDDLEWARE ORIENTADO A MENSAGEM

- Exigência de um elemento central responsável pelo gerenciamento das filas de mensagens;
- A comunicação assíncrona pode retardar a entrega de mensagens;
- Quanto maior a heterogeneidade do sistema, mais custos são associados ao mesmo por ter de comprar vários módulos, maior manutenção também lhe está associada.

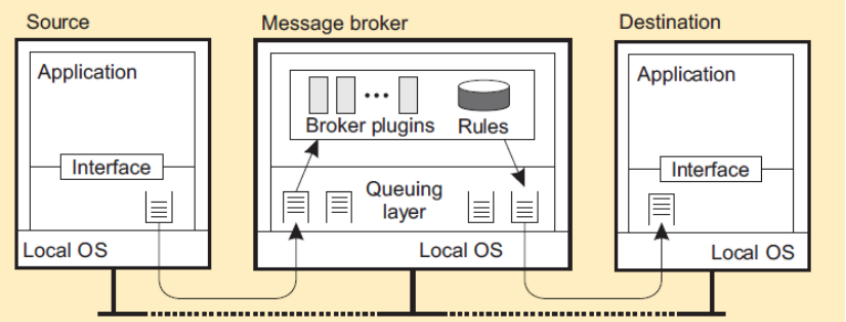
5. MESSAGE BROKER

5.1 O QUE É MESSAGE BROKER?

É o elemento principal envolvido na comunicação do MOM, que é basicamente um servidor de mensagens, que intermedia a interação entre os participantes e gerencia as filas de mensagens.

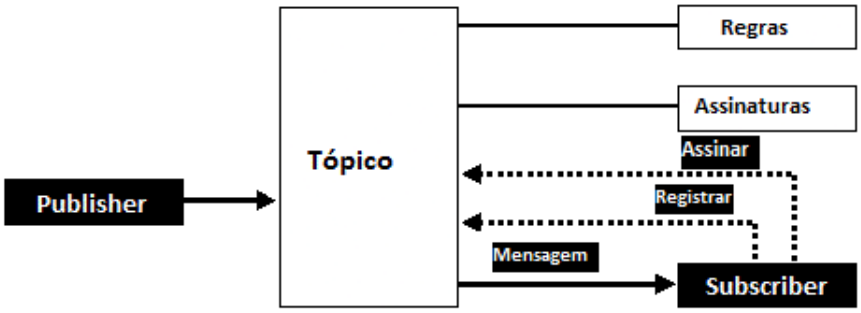
Sua principal função é gerenciar as filas de mensagens, garantindo que a mensagem seja enfileirada, armazenada e que fique lá enquanto necessário. Podemos dizer que funciona como uma caixa de correio, e as mensagens são as cartas que serão depositadas (publicadas) ali e retiradas (consumidas) por alguém que tenha interesse em ler essas cartas.

5.2 ARQUITETURA DO MESSAGE BROKER



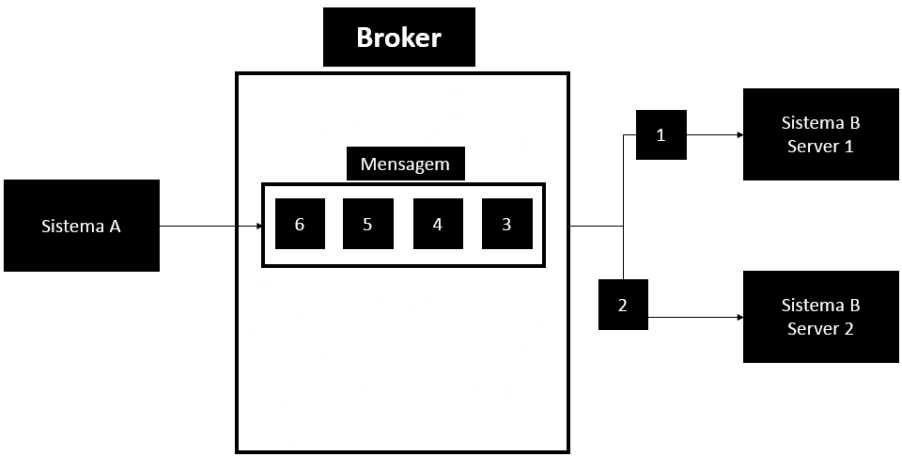
Essa seria a arquitetura de um broker. A source (produtor) envia uma mensagem (evento), que pode ser qualquer tipo de formato em bytes. O broker recebe essa mensagem, onde pode ter algum plugin para converter as mensagens de um protocolo para o outro, que além de converter podem fazer o casamento das aplicações (publish/subscriber). Depois desse processo a mensagem vai para a fila ou tópico. E assim sendo enviada para Destination (receptor).

5.3 MODELO PUBLISH/SUBSCRIBE E TÓPICOS



No modelo publish/subscriber, o sistema produtor envia a mensagem na forma de publish com um tópico X para o broker, e o broker envia essa mensagem para os sistemas que tiverem assinado o tópico X. Essa filtragem de tópicos reduz o tráfego na rede e elimina a necessidade de tratar as mensagens que não interessam ao receptor.

5.4 SISTEMA DE FILA



Não sendo o caso publish/subscriber, as mensagens vão para fila normal e distribuídas normalmente para os receptores, e caso exista mais de um servidor receptor, o broker irá dividir a carga das mensagens igualmente para os servidores receptores.

5.5 VANTAGENS DO MESSAGE BROKER

Uma das principais vantagens de se usar o broker pela confidencialidade e controle de acesso, é que as mensagens podem ser criptografadas. As filas podem ter um controle no seu acesso, impondo restrições a quem produzir e receber as mensagens. Mesmo que o sistema receptor esteja fora do ar, o sistema produtor poderá continuar enviando mensagens pois o broker continuará armazenando e gerenciando as mensagens. E também não sobrecarregará o sistema receptor de nenhuma forma pois somente estará entregando as mensagens que o sistema receptor conseguir processar.

6. ADVANCED MESSAGE QUEUE PROTOCOL (AMQP)

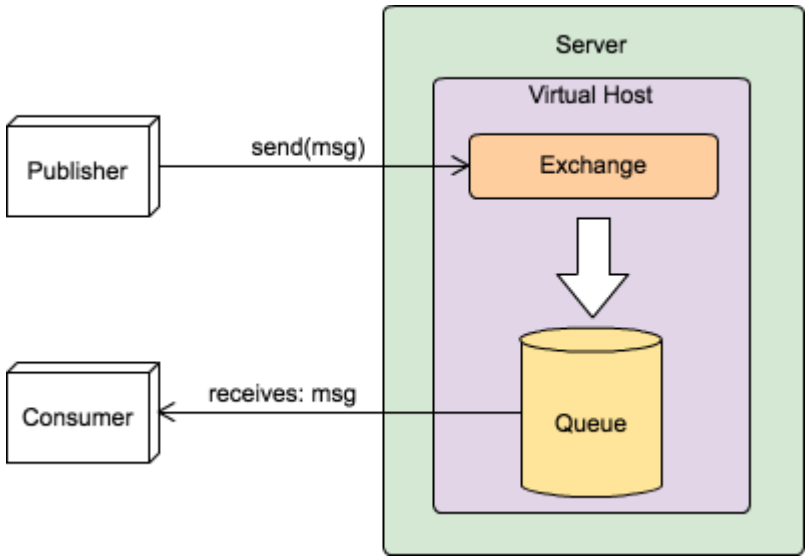
AMQP é um protocolo de mensageria em rede que visa solucionar a interação entre diferentes endpoints de uma forma confiável, de fácil escalabilidade e de forma transparente.

O protocolo teve seu surgimento após uma demanda de uma solução que fosse menos complexa e custosa, pois anteriormente os fornecedores produziam suas próprias soluções e que consequentemente resultava em uma falta de compatibilidade entre as operações. Portanto fora desenvolvido o AMQP que visa principalmente ser um padrão aberto que possui essa interoperabilidade entre diversas aplicações, com arquiteturas diferentes e linguagens.

6.1 COMO FUNCIONA O AMQP

O AMQP funciona de modo que as mensagens são enviadas pelos publishers às exchanges, depois as exchanges são encarregadas de distribuir cópias das mensagens às filas aplicando regras chamadas de bindings. Posteriormente os

brokers ficam encarregados de enviar as mensagens aos consumidores que estão requisitando às filas.



Ao publicar uma mensagem, os publishers têm total liberdade para definir algumas atribuições, como metadados, que posteriormente podem ser utilizados pelos brokers, ou mesmo pelas aplicações que estão requisitando as mensagens.

Como todas as requisições são feitas pela rede, ao qual possa estar suscetível a falhas, o protocolo AMQP possui um tratamento de acknowledgments das mensagens, ao qual quando uma mensagem chega ao consumidor o broker é notificado.

AMQP é um protocolo programável, ou seja todas as entidades e esquemas de roteamento são definidos principalmente pelas próprias aplicações. Isso dá uma liberdade aos desenvolvedores, mas devem ficar atentos a potenciais conflitos.

6.3 TIPOS DE EXCHANGE NO AMQP

Dependendo da aplicação a ser desenvolvida, o AMQP provém de diversos tipos de Exchanges:

- Default Exchange: Geralmente usado para aplicações simples, não há payload de atributos, e as mensagens são entregues de forma “direta”;

- Direct Exchange: Tem como função entregar as mensagens para as filas com base na chave de roteamento, geralmente usado em unicast e distribuir tarefas para diversas instâncias de uma aplicação;
- Fanout Exchange: Roteia mensagens para filas vinculadas e a chave de roteamento é ignorada, muito usada para broadcast de mensagens.Ex: Jogos MMO que necessitam atualizar placares;
- Topic Exchange: Geralmente usado para rotear mensagem em uma ou várias filas, usado para aplicações que necessitam de um multicast.Ex: Distribuição de dados geográficos, como ponto de vendas;
- Headers Exchange: Também ignora a chave de roteamento, porém o gerenciamento é feito pelas atribuições nos cabeçalhos das mensagens,ou seja a mensagem é vinculada de acordo com uma checagem.

6.4 EXEMPLO DE SOFTWARE MIDDLEWARE

6.4.1 RABBITMQ

RabbitMQ é um message-broker (servidor de mensagens) que, originalmente, implementa o protocolo AMQP.

Foi escrito na linguagem Erlang e possui suporte a clusterização (diversas instâncias do programa rodando em diferentes máquinas físicas, agindo como um só). Existem bibliotecas de conexão escritas em quase todas as linguagens conhecidas.

6.4.2 FILAS

O RabbitMQ implementa as filas como descrito anteriormente no protocolo AMQP. Um cliente publicador (publisher) faz o envio de uma mensagem para uma exchange, que encaminha a mensagem para as filas utilizando suas regras (bindings), os clientes consumidores inscritos nessas filas (subscribers) recebem essa

mensagem e retornam um acknowledgement (confirmação de recepção da mensagem).

Uma mensagem possui um título (*label*) e seu conteúdo (*payload*). O conteúdo pode ser qualquer coisa, desde um JSON, até um arquivo de mídia.

A ordenação das mensagens é feita com FIFO, porém a ordem pode ser afetada no caso de mensagens com prioridade diferente e nos casos de re-entrega (retorno da mensagem para a fila pelo consumidor).

6.4.3 PROPRIEDADES

As filas neste software possuem algumas propriedades para definir seu comportamento, são elas:

Nome: todas as filas possuem um nome para que as aplicações subscribers possam se referir ao solicitar a inscrição. O nome pode ser definido tanto pela aplicação, quanto pelo próprio servidor

Durabilidade: as filas podem ser duráveis ou não. Uma fila durável não perde seu conteúdo caso haja um restart no broker, em outras palavras, isso define se o conteúdo das filas será armazenado apenas em memória, ou se também será armazenado em disco para que haja persistência.

Exclusividade: uma fila exclusiva é criada por um cliente e só pode ser usada pela conexão que a criou. Filas desse tipo são excluídas automaticamente quando a conexão que as criou é fechada.

Auto-Remoção: uma fila auto-removível é deletada se já tiver pelo menos um consumidor e todos já se desinscreveram

7. CONCLUSÃO

Dessa maneira, é possível concluir que nos dias de hoje o Middleware é mais do que necessário para o funcionamento não só de todos os sistemas que comunicam-se uns com os outros, como de toda a internet. Se cada programador necessitasse criar não só o próprio sistema, como também a forma que esse sistema se comunicaria com outros, demandaria muito tempo e recursos. Portanto, o objetivo do Middleware é facilitar o desenvolvimento de aplicações, tipicamente as distribuídas, assim como facilitar a integração de sistemas legados ou desenvolvidos de forma não integrada automática.

8. REFERÊNCIAS

https://www.gsd.inesc-id.pt/~ler/docencia/tm0405/slides/MessageOrientedMiddleware-MOM.pdf

https://www.inf.ufsc.br/~frank.siqueira/INE6514/MOM-Folhetos.pdf

https://pt.linkedin.com/pulse/arquitetura-mom-message-oriented-middleware-e-algumas-jogi-takechi

https://pt.wikipedia.org/wiki/Message_Oriented_Middleware

https://homepages.dcc.ufmg.br/~geanderson/aulas/dad/aula-04-dad.pdf

https://sites.google.com/site/proffdesiqsistemasdistribuidos/aulas/caracterizacao-de-sistemas-distribuidos

https://drive.google.com/file/d/1au6vL3-hNOeg_S09_hU5R5j7AWgWteSF/view

https://drive.google.com/file/d/1JRYifhExTvX9TODA2zy7bghtHDzHED8a/view

https://www.rabbitmq.com/

https://adolfoeloy.com/rabbitmq/amqp/2016/09/07/rabbitmq-study.pt.html

https://medium.com/@devbrito91/mensageria-1330c6032049

http://wiki.secondlife.com/wiki/Message_Queue_Evaluation_Notes#Group_Chat_Use_Case

https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

https://github.com/baldugus/gochat

https://www.youtube.com/watch?v=_Uo14nxB_iA

https://www.youtube.com/watch?v=NmSL7llD1OI

https://www.youtube.com/watch?v=5ewQBi1rx58

https://www.youtube.com/watch?v=ItHnMnrIVh0