Message-Oriented-Middleware (MoM)

Matthias Wallnöfer

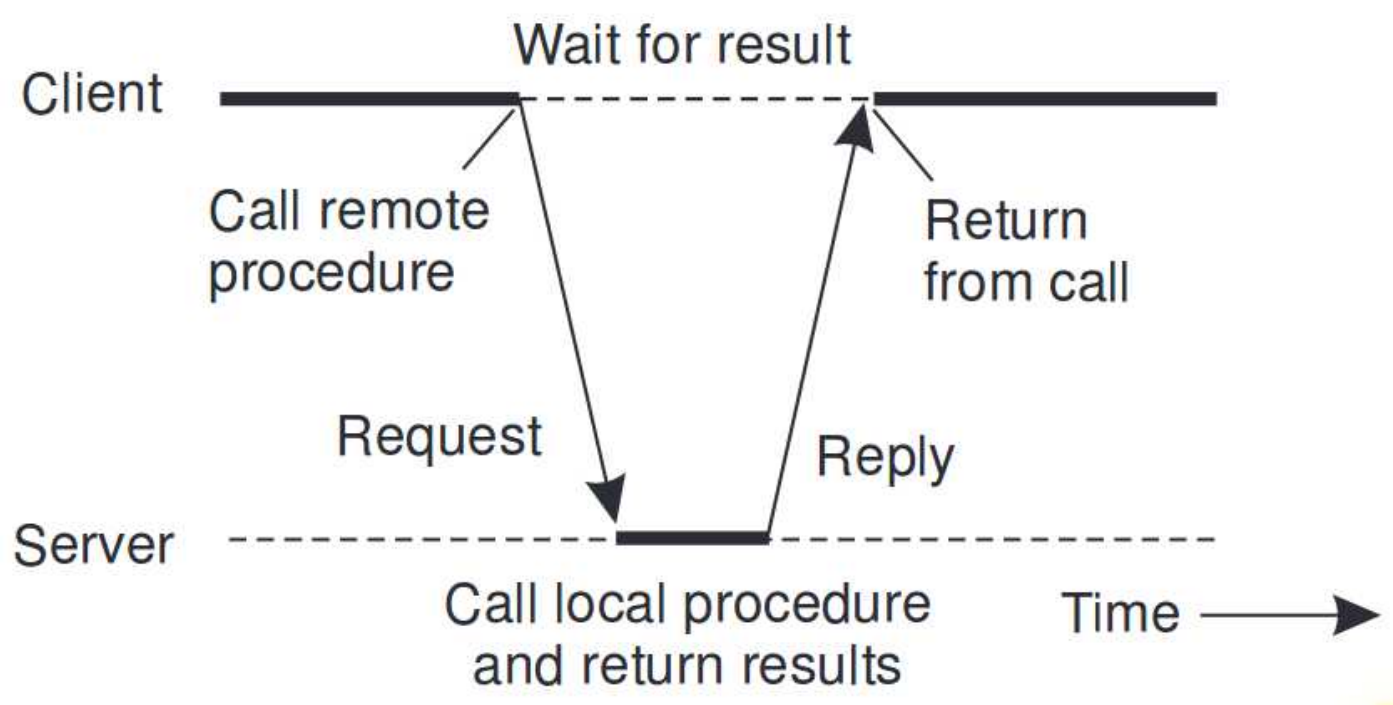
Verwendete Literatur

1)http://www.inf.fu-berlin.de/lehre/SS05/VS/folien/vs10.pdf
2)https://www.techfak.uni-bielefeld.de/~swrede/xml-isy/talks/mom-xmlblaster.pdf
3)A. Tannenbaum, M. van Steen, “Verteilte Systeme - Grundlagen und Paradigmen”
4)https://www.slideshare.net/dobermai/pubsub-for-the-masses-ein-einfhrungsworkshop-in-mqtt-german

RPC

- Haben ein großes Paradigma der Verteilten Systeme kennengelernt: **RPC**
- *Remote Procedure Call* (= Request/Response)
  - … erlauben uns, auf einem anderen Knoten eine Prozedur aufzurufen (Anfrage=*Request*), ihr Parameter zu übergeben und einen Rückgabewert (Antwort=*Response*,*Reply*) in Empfang zu nehmen
  - … sind Middleware, da oberhalb der Internet Socket-API angesiedelt

RPC



Grundkonzept von RPC (Quelle 4)

Vielzahl von Implementierungen/ Erweiterungen (chronologisch): Sun RPC, MS/DCE RPC, RMI, Web Services (XML-RPC, SOAP), HTTP REST

RPC - Nachteile

- Wenig geeignet für asynchrone Kommunikation
- Einschränkung auf klassisches Client-Server-Szenario, was tun bei mehreren Knoten (*peer2peer*)?
- Oftmals feste Verbindung zwischen den Akteuren

MoM

- zu deutsch: *Nachrichtenorientierte Middleware*
- Knoten statt Client-Server
- Nachrichten statt Request-Response
- Synchrone und asynchrone Kommunikation möglich (Knoten entkoppelt)

MoM

- zu deutsch: *Nachrichtenorientierte Middleware*
- Knoten statt Client-Server
- Nachrichten statt Request-Response
- Synchrone und asynchrone Kommunikation möglich (Knoten entkoppelt)
- Aber: Zentrale Koordinationseinheit, *Broker* (Provider) genannt, erforderlich

Beweggründe

❶ Anderer Architekturstil sinnvoll für manche Anwendungen, z.B. Sensorsysteme, Unterstützung von Geschäftsprozessen, ...

- Interaktion von Prozessen über Ereignisse, Nachrichten

→ Pipes? TCP-Verbindungen?

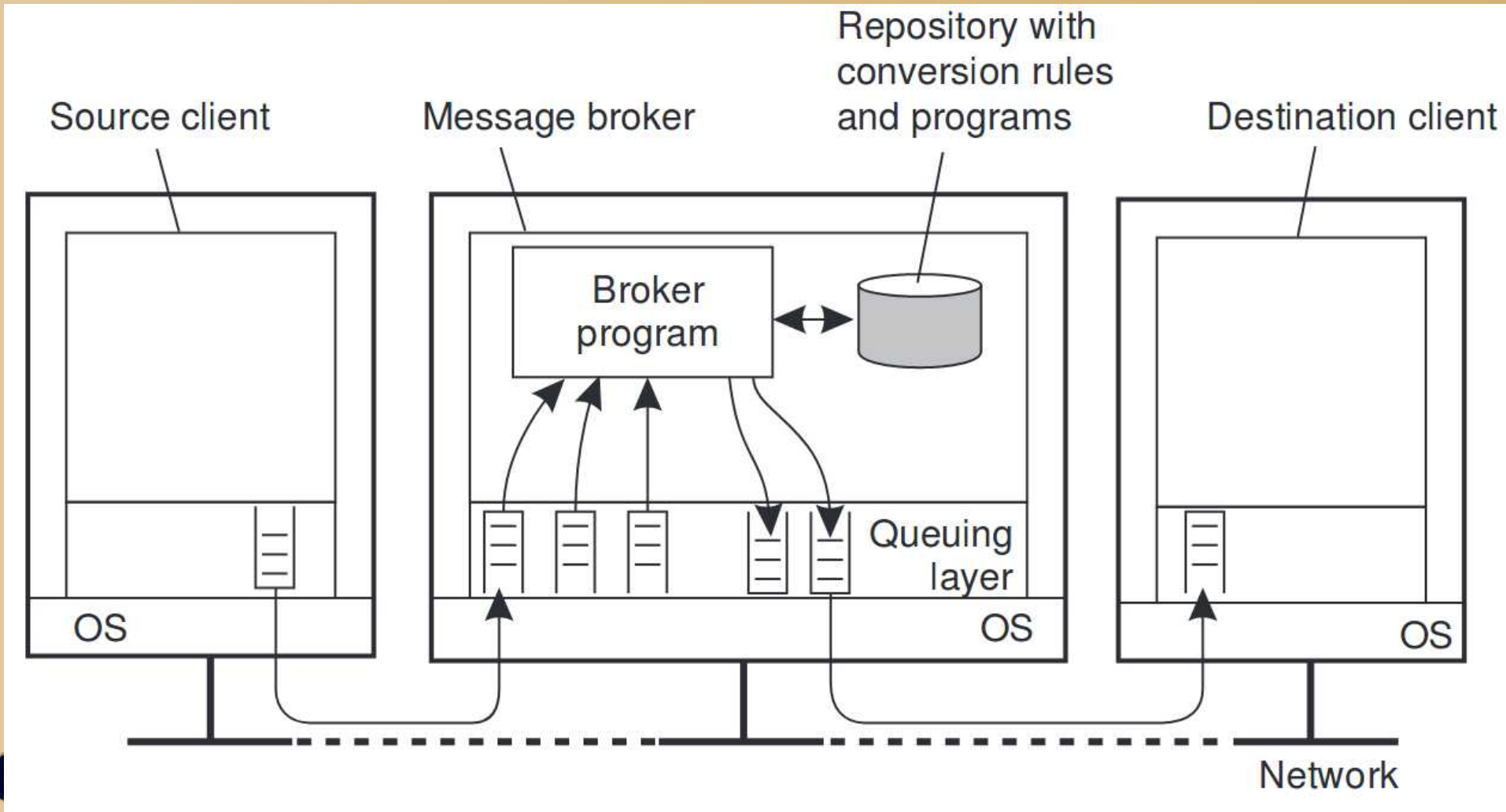
❷ Keine festen Verbindungen zwischen Quellen und Senken von Informationen (mobile Geräte!)

- Dienst für Vermittlung und persistente Zwischenspeicherung

→ E-mail?

(Quelle 2)

Schematischer Aufbau



(Quelle 4)

4 Grundoperationen

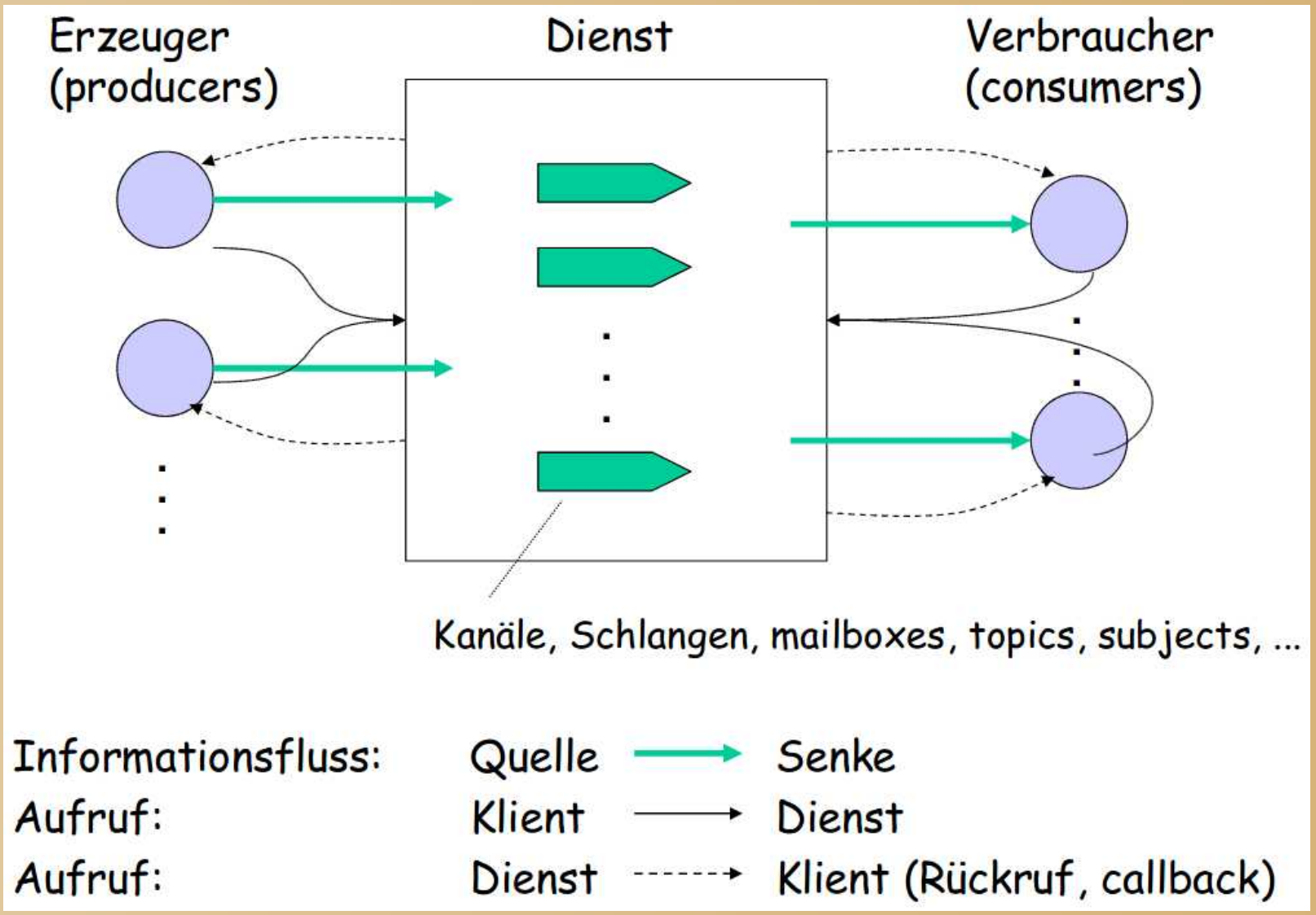
Elementare Funktion	Bedeutung
Put	Eine Nachricht an angegebene Warteschlange anhängen
Get	Blockieren, bis die angegebene Warteschlange nicht leer ist, und die erste Nachricht daraus entfernt
Poll	Eine angegebene Warteschlange auf Nachrichten überprüfen und die erste Nachricht daraus entfernen; kein Blockieren
Notify	Eine Verarbeitungsroutine installieren, die aufgerufen wird,wenn eine Nachricht in die angegebene Warteschlange gestellt wird

(Quelle 3)

Warteschlangen

- Ein Broker verwaltet N WS (*Queues*)
  - Bei Email = Postfächer
- Die WS sind durch Themen (*Topics*) getrennt
- Durch die *Notify*-Operation kann Knoten auf mehreren WS lauschen (Filter-Syntax)
- Erweiterungen:
  - Zutrittskontrollen bei WS
  - Letzter Wille (*Last Will*) wenn Knoten sich abmeldet
  - QoS mit Prioritäten

Einsatzszenario



(Quelle 2)

Umsetzungen

- … gibt es wie Sand am Meer
- Im **Betriebssystem** (1 System, begrenzte Portabilität)
  - POSIX MQs und D-Bus (Linux + UNIX-artig)
  - Windows MQ
- **Low-Level-Bibliotheken** (mehrere Knoten, auf Performance ausgerichtet)
  - Message Passing Interface (MPI)
  - ZeroMQ (0MQ)

Umsetzungen

- **IoT-Bereich** (Elektronik, Ansteuerungen)
  - MQTT
  - NATS
- **High-Level-Frameworks** (hoher Komfort, dafür umfangreich)
  - AMQP-basiert (ActiveMQ, RabbitMQ, StormMQ)
  - Apache Kafka
  - Enterprise Service Bus (ESB)
  - Proprietäre Lösungen von Oracle, IBM, MS
  - Etc. etc.

MQTT

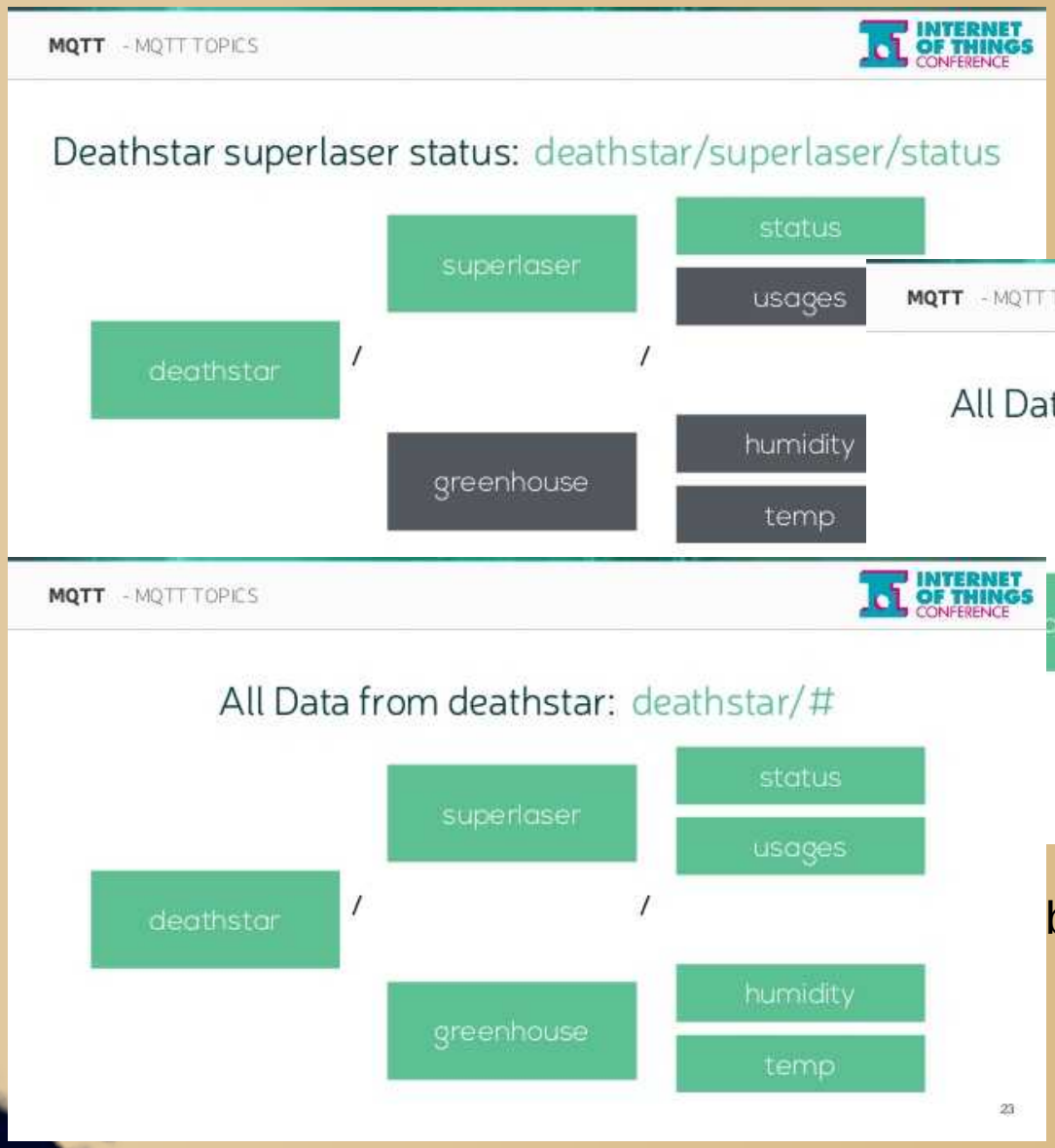
- = *Message Queue Telemetry Transport*
- Website: http://mqtt.org
- Entwickelt von Dr. Andy Stanford-Clark (IBM) und Arlen Nipper (Eurotech) für Überwachung von Ölpipelines
- Einfach gehalten, viele Bibliotheken
- Für Knoten mit begrenzter Leistung konzipiert, auch MCUs
- Kommunikation in unsicheren Netzen mit geringer Bandbreite

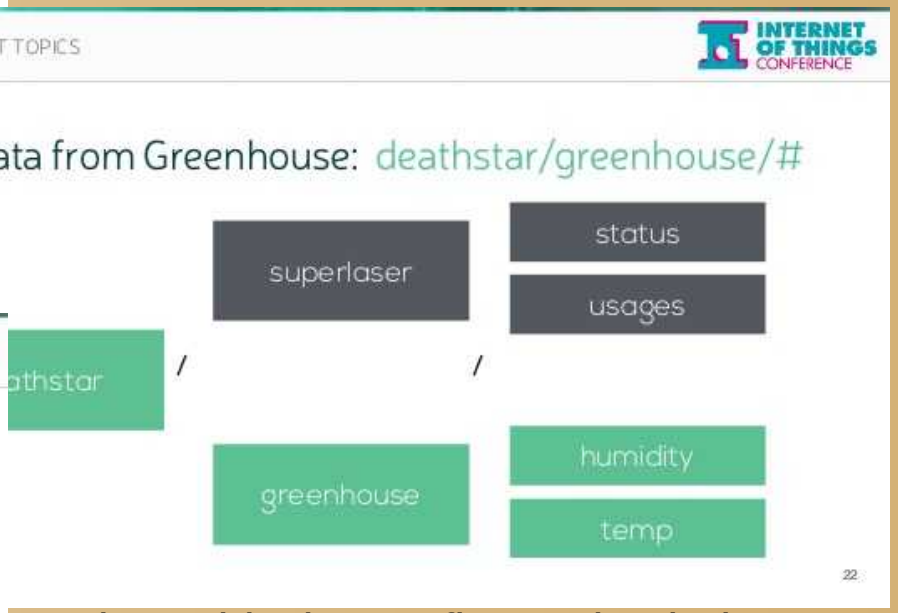
MQTT - Operationen

Allgemein	MQTT	Kommentar
Put	PUBLISH	
Get		In dieser Form nicht vorgesehen, man muss sich zuerst bei der Warteschlage abonnieren ([MQTT-3.3.2-3])
Poll		idem
Notify	SUBSCRIBE/ UNSUBSCRIBE	

- Doku: https://mqtt.org/getting-started/
- Nachrichtenformat (*Payload*) beliebig, kann bspw. Klartext, JSON, XML, aber auch binär sein

MQTT – Warteschlangen

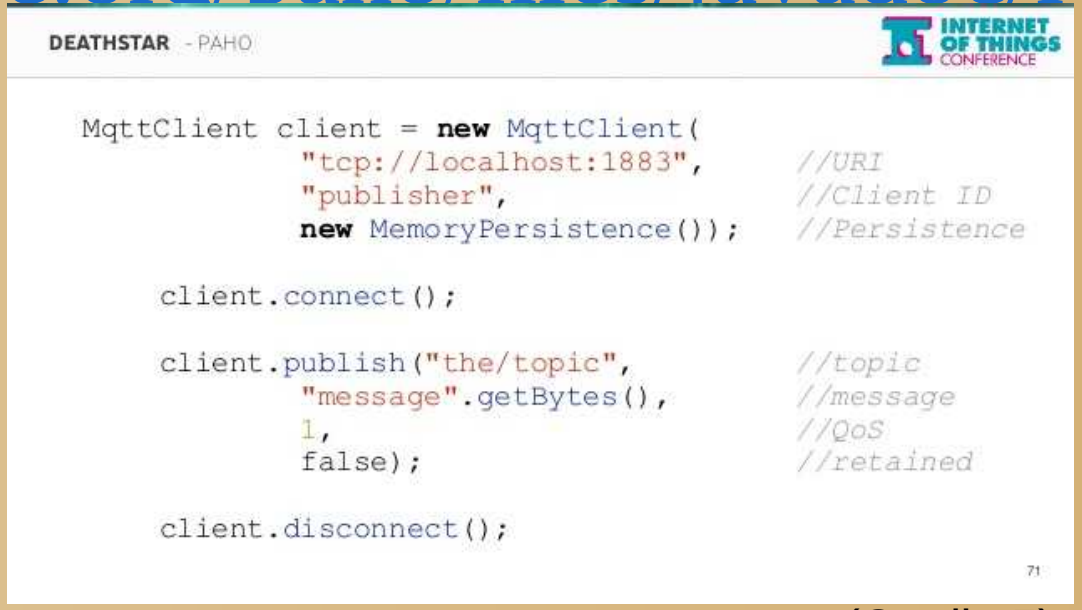




Hierarchischer Aufbau, Platzhalter nur bei Empfang d.h. Subscription (Quelle 5)

MQTT in Java

- Eclipse Paho die erste Wahl:
  https://www.eclipse.org/paho/clients/java/
- Java-Doc-Referenz:
  http://www.eclipse.org/paho/files/javadoc/index.html



(Quelle 5)

MQTT Beispiel

```
public class MqttPublishSample {
        public static void main(String[] args)
{
            String topic        = "MQTT
Examples";
            String content      = "Message from
MqttPublishSample";
            int qos             = 2;
            String broker       =
"tcp://mqtt.eclipseprojects.io:1883";
            String clientId     = "UniqueID-
123456789";
            MemoryPersistence persistence = new
MemoryPersistence();

            try {
                MqttClient sampleClient = new
MqttClient(broker, clientId, persistence);
                MqttConnectOptions connOpts =
new MqttConnectOptions();
                connOpts.setCleanSession(true);
                System.out.println("Connecting
to broker: "+broker);
                sampleClient.connect(connOpts);
System.out.println("Connected");
                System.out.println("Publishing
message: "+content);
                MqttMessage message = new
MqttMessage(content.getBytes());
                message.setQos(qos);
                sampleClient.publish(topic,
message);
                System.out.println("Message
published");
                sampleClient.disconnect();

                System.out.println(
"Disconnected");
                System.exit(0);
            } catch(MqttException me) {
                ...
            }
        }
    }
```

MQTT Broker

- Eclipse stellt einen öffentlichen Broker (Mosquitto) zur freien Verfügung. Hier findet ihr die Einstellungen: https://iot.eclipse.org/projects/sandboxes/
- Dazu gibt es Web- und Client-basierte Inspektoren:
  - HiveMQ Browser Client (http://www.hivemq.com/demos/websocket-client/) NB: Bei Eclipse den Websocket-Port Port *443* wählen.
  - MQTT Client Chrome App (https://www.hivemq.com/blog/mqtt-toolbox-mqtt-client-chrome-app/)
  - Eclipse Paho Mqtt Spy (https://github.com/eclipse/paho.mqtt-spy/wiki