Bridging GeoMQTT and REST

Stefan Herle, Ralf Becker & Jörg Blankenbach
Chair for Computing in Civil Engineering & Geo Information Systems
RWTH Aachen University
Aachen, Germany
herle@gia.rwth-aachen.de

***Abstract*— Wireless geo sensor networks (WGSN) use different ways to publish their measured data in the Sensor Web to make time series of this data accessible by means of the World Wide Web (WWW). In earlier papers, we proposed the use of GeoMQTT, a spatiotemporal extension to the widely known and used "Internet-of-Things" (IoT) protocol Message Queue and Telemetry Transport (MQTT). In this paper, we propose a GeoMQTT – Representational State Transfer (REST) bridge to enable users to easily interact with GeoMQTT by using standard HTTP methods. We demonstrate that using such a bridge is especially useful for debugging events issued by sensors nodes using GeoMQTT as a communication protocol.**

Keywords*— *Sensor Web; IoT; GeoMQTT; REST; spatiotemporal pub/sub

I. INTRODUCTION

Monitoring environmental phenomena and events in geoscience has changed dramatically over the last decade since new technological trends improve the capabilities of sensors and sensor platforms. According to [1], there have been three major drivers for this development. First, ubiquitous communication networks have evolved to facilitate access and measuring in remote and inaccessible areas without wires. Furthermore, the miniaturization of computing platforms and, hence, the optimization of power consumption enables sensor platforms to run over a significant extended period of time. Lastly, the sensors and sensor materials themselves have improved to size-reduced and micro-scale sensors.

Wireless geo sensor networks (WGSN) are one manifestation of these technological trends. Being still in an early adoption phase, they shift traditional centralized sensor platforms into lightweight, portable and intelligent systems on a microscale. Distributed sensors and sensor nodes in such networks are able to monitor an extensive geographical area with minimal efforts and costs but deliver point-based data in near real-time.

On the other hand, WGSNs should not be an isolated system. The captured sensor data should be accessible by means of existing methods of the World Wide Web and preferably in a standardized way. This paradigm is called the "Sensor Web". Its goal is to hide "the underlying layers from the applications built on top of it" [2]. Standards for accessing sensor data over Hyper Text Transport Protocol (HTTP) have already been developed by the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) initiative [3]. However, there have been some unsolved issues with transferring the data from the WGSNs to the Sensor Web servers since WGSNs are often not able to handle HTTP requests. The authors of [4] suggest an intermediary layer, the so-called Sensor Bus, to bypass this gap. We took this idea and implemented it using an extension of the lightweight Message Queue and Telemetry Transport (MQTT) protocol, which we call GeoMQTT [5]. A short introduction to MQTT and GeoMQTT is given in the next two sections.

GeoMQTT, like MQTT, is a machine-to-machine (M2M) protocol and, therefore, is tailored to requirements of machines. Users, on the other hand, need simpler methods to interact with machines. The standards of the Sensor Web are one way to achieve this, but it only holds for sensor data. Having a more general access point to the GeoMQTT bus would be a huge benefit, especially for developers. Therefore, we propose an architecture built upon the Representational State Transfer (REST) principle, which bridges GeoMQTT and REST. It is presented in section 4 and 5 of this paper. Last, we give a conclusion and an outlook for future improvements.

II. MESSAGE QUEUE AND TELEMETRY TRANSPORT (MQTT)

The MQTT protocol is a very lightweight protocol often used in the Internet of Things and Service (IoTS). It is especially suitable for constrained devices as well as low-bandwidth, high-latency and unreliable networks. The protocol implements the so-called publish/subscribe interaction scheme, which is an event-based communication model between *publishers* that produce certain information and *subscribers* that register to these information. The term *event* is used for the act of publishing information whilst *notification* denotes the act of delivery to the consumer [6]. A *broker* distributes the events according to the interests of subscribers.

MQTT uses a topic-based publish/subscribe scheme [7] for addressing. Events or publish messages are tagged with a *topic name*, which is an arbitrary string. It can be hierarchically structured with the topic level separator, a forward slash. For instance, a temperature sensor node tags an MQTT message with the topic name *room/237/temperature* and publishes the room temperature in the payload of the message. Subscribers are able to express their interests in events with a *topic filter*,

which is also an arbitrary string and has a similar shape like the topic name. In addition, a single-level wildcard "+" or a multi-level wildcard "#" can be used to register to a set of topics. For example, if a subscriber is interested in the temperature measurements of all rooms, he could use the single-level wildcard in the topic filter *room/+/temperature*. For all rooms and all possible observation properties, he could use the multi-level wildcard with the topic filter *room/#*.

The MQTT Version 3.1.1 offers some core features, such as a Quality of Service (QoS) mechanism, to guarantee the delivery of a message or the Last Will and Testament (LWT) mechanism to notify clients about an "ungracefully" disconnected client. These features are especially useful in unreliable networks.

MQTT is based on TCP/IP, but with the extension MQTT for Sensor Networks (MQTT-SN), it also supports connectionless communication protocols like UDP or ZigBee [8]. As the name suggests, the extension is especially useful in wireless sensor networks (WSN) since it is optimized for tiny battery-operated Sensor/Actuator devices and considers constraints of WSNs such as high link failure or short message payload. MQTT-SN adds two new components to an MQTT network: the MQTT-SN client and a gateway, which acts like a translator between the two protocols.

III. GeoMQTT

We extended MQTT with new message types to support spatiotemporal tagging and filtering of events [5]. Therefore, the extension is called GeoMQTT. It is still a publish/subscribe interaction scheme although not solely topic-based, but also timestamp and geometry-based. The topic mechanism, however, is inherited from ordinary MQTT as described in Section 2.

The introduced *GeoPublish* message, which can be used by producers to generate a spatiotemporal event, is tagged with a timestamp and geometry in addition to the topic name. The format of the timestamp can either be expressed in ISO8061 syntax or in UNIX time, which is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT). The geometry can be specified in different common description languages for geometries such as Well-known Text (WKT) or GeoJSON. Like the MQTT publish message, a GeoPublish message also consists of a payload, which can be arbitrary.

The geo subscription mechanism uses a temporal filter and/or a spatial filter in addition to the topic filter inherited from the ordinary MQTT subscription. The broker forwards the published events only if the subscription meets all specified filters.

The syntax of the *temporal filter* adheres to the ISO8601 intervals and repeating intervals standard defined in [9]. For instance, a time interval can be specified with *2016-03-28T11:15:00Z/PT2H30M*, which subscribes to events issued between 2016-03-28T11:15:00Z and 2016-03-28T13:45:00Z. In addition to the ISO standard, we add support for cron expressions to specify the start timestamp of a period. The syntax adheres to the cron expression defined in the Quartz Job Scheduler [10]. The *spatial filter* is used to filter the events with respect to the tagged geometry in the GeoPublish message. Similar to the GeoPublish message, the geometry is specified using common description languages such as WKT or GeoJSON. The evaluation of the spatial filter in the broker currently uses a simple "covers" relation. To enhance the spatial filter, there are plans that will enable subscribers to specify the Spatial Reference System (SRS) using an EPSG code as well as their preferred spatial relation.

Conflicts may occur between GeoMQTT and MQTT messages. For instance, a client is subscribed to an MQTT topic filter and a GeoPublish message is issued whose topic name matches the filter. Since the subscription only contains a topic filter, but has neither a temporal nor a spatial filter, it raises the question whether to forward the message or not. We implemented a conflict handling strategy, which is also compatible to MQTT clients that do not support the extension [4]. For example, in the conflict mentioned, the temporal and spatial information in the GeoPublish message are ignored. If the topic filter of the MQTT subscription matches the topic name, the message is converted into an MQTT publish message discarding the additional information.

Since we deal with WGSNs that use ZigBee, we also implemented a GeoMQTT-SN version to bridge the sensor nodes with the GeoMQTT broker. We added different message types, which are translated to GeoMQTT messages in the gateway of the WSN. As shown in Figure 1, we use the GeoMQTT-SN protocol in the project EarlyDike to monitor dikes with sensors. Like MQTT, it is also useful in WGSN environments due to its lightweight nature, but adds support for the definition of sampling time and sampling location/geometry in the header of the GeoPublish message.

IV. Bridging GeoMQTT and REST

By adding a REST interface to GeoMQTT, two different semantic models of communication are bridged, the publish/subscribe interaction scheme of MQTT and the request/response pattern of HTTP. According to [11, 12], it is useful to couple the REST-oriented web architecture and the real-time properties of MQTT to close the gap between machines and developers in the IoTS. They implement a so-called QEST broker to expose MQTT topics as REST resources and vice versa. For instance, the REST resource */topics/room/237/temperature* corresponds to the topic *room/237/temperature*. By requesting the resource with an HTTP GET, the response consists of the latest published value issued with the topic. Similarly, an HTTP PUT request at */topics/room/237/temperature* publishes the value of the request body with the corresponding topic.

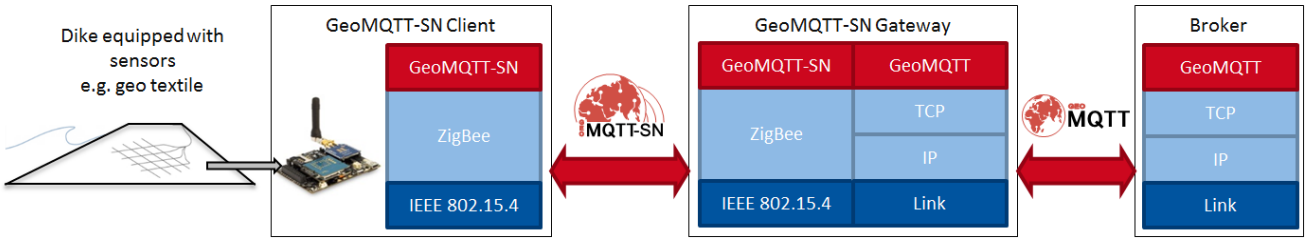

Fig. 1. GeoMQTT-SN architecture and layer stack

We follow a similar approach in the implementation of our REST bridge. Unlike [11], we do not integrate the REST interface directly in our GeoMQTT broker, but use an observer client, which subscribes to all messages/events. Additionally, we use the bridge as a message logger. It does not solely store the latest value on a specific topic, but all messages that are received. The REST-GeoMQTT bridge is shown in Figure 2.

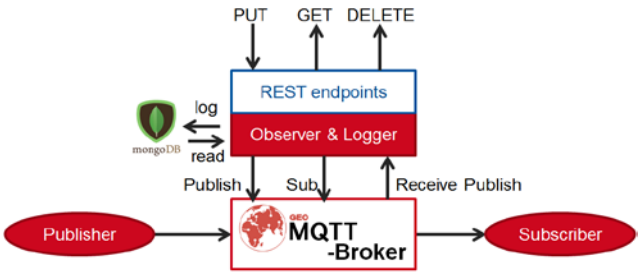

Fig. 2. GeoMQTT - REST bridge

As mentioned above, the observer & logger client subscribes to all MQTT and GeoMQTT messages at the broker. It logs the publish and GeoPublish messages received in separate collections in a MongoDB database. The bridge has two different REST endpoints: one for MQTT and one for GeoMQTT.

For MQTT, the REST resources are mapped to topics according to the approach in [7] (for instance, the resource */publish/room/237/temperature* is mapped to the topic *room/237/temperature*). In an HTTP GET request, it is also possible to use the single-level wildcard "+" or the URL-encoded multi-level wildcard "#" of topic filters. The bridge queries the MongoDB database for logged publish messages and responses with a list of messages in JSON format. Setting the optional URL parameter *size* to 1 allows the users to retrieve only the latest published message that matches the topic filter. Accordingly, with the HTTP PUT request of a resource, the request body is published to the corresponding topic name to the GeoMQTT broker. Wildcards are not allowed in the resource since they are prohibited in topic names in MQTT publish messages.

In GeoMQTT, the topic is handled similarly to the MQTT case (resource */geopublish/temperature* is mapped to topic *temperature*). The HTTP GET request has four optional parameters: *from*, *to*, *geometry* and *size* like before. *from* and *to* are used to specify a time interval whilst *geometry* expects a geometry in WKT format. All OGC Simple Feature Access geometries are supported and extended by a BBOX and BUFFER format. If not specified, the temporal filter and geometry filter are set to wildcards. The bridge queries the MongoDB database with the temporal, geometry and topic filters and returns a GeoJSON FeatureCollection of the logged published messages. Hereby, the geometry filter is evaluated with a "within" relation in respect to the geometries of the GeoPublish messages stored in the database. The HTTP PUT request for the GeoMQTT endpoint expects two required parameters in addition to the corresponding wildcard-free topic name as the resource: the *time* parameter as an ISO8601 timestamp and the *geometry* parameter in WKT syntax. Similar to the MQTT case, the request body and the parameters form a GeoPublish message, which is sent to the GeoMQTT broker.

In addition, we implemented HTTP DELETE for the two endpoints to manage the database. It deletes the matching entities in the database and returns them as a JSON/GeoJSON document. The request parameters are the same as for the HTTP GET requests except for the size parameter.

As mentioned previously, since HTTP uses a request/response mechanism, it cannot fully support a publish/subscribe mechanism. WebSockets could be one possible solution to solve this issue [13]. In fact, we already implemented a GeoMQTT client with WebSockets. But [11] argues that WebSockets do not implement the concept of URI after opening the communication and, therefore, do not support the pure REST approach (resource-topic mapping). They enhanced the implementation by a Long-Polling approach for retrieving real-time updates in the browser without using WebSockets. So far, we have not implemented this solution, but it might be of interest in the future.

V. APPLICATION

The REST bridge is used in the EarlyDike[1] project to log and easily obtain events published by sensor nodes, which are deployed at dike lines to monitor the structure of sea dikes. We set up a Web map application to request the events and plot them in a map. Since the response of the service is a FeatureCollection of events in GeoJSON format, it is quite easy to plot the events, for example, in a Leaflet[2] map container (see Figure 3). The corresponding REST request for the requested data in the application in Figure 3 is the following.

```
http://localhost:8080/rest/geopublish/node/+/temperature?
geometry=LINESTRING(8.589248657226562
54.517893120052946,8.590707778930664, …)
```

The REST endpoint here is */rest/geopublish*. The requested resource corresponds to all stored messages that match the topic filter *node/+/temperature*, where the "+" wildcard is used to replace the sensor node id and, therefore, retrieve all measured temperatures of every available sensor node. Additionally, the stored messages are filtered by the specified geometry, a LINESTRING which represents the southwestern first order dike line of the German North Sea island Pellworm (compare the purplish polyline in the map in Figure 3). The temporal filter is not specified and thus set to a wildcard.

The bridge and application do not represent a substitution for SWE data storage services such as the Sensor Observation Service (SOS). We use it mainly for fast debugging of our sensor networks or to send configuration messages to the sensor nodes.

Fig. 3: Web map application to request GeoMQTT events using the REST bridge

VI. CONCLUSION AND FUTURE WORK

Bridging the publish/subscribe protocol GeoMQTT and a HTTP based REST interface is beneficial to provide an easy access point to the GeoMQTT bus without any further software. By using simple HTTP methods such as GET and PUT, it is convenient especially for developers to retrieve events or publish messages to the GeoMQTT broker. Since we also added logging capabilities to the bridge, users are able to retrieve, not only the latest event issued, but also a history of events.

REST, however, is based on HTTP requests/responses. Therefore, it lacks in retrieving real-time data through push notifications like in the publish/subscribe interaction scheme of GeoMQTT. One solution to solve this issue is the use of WebSockets. This would require additional libraries and does not support the concept of URI. As mentioned previously, a Long-Polling RESTful approach could be used to tackle this issue in the future.

In combination with the Sensor Bus concept for closing the gap between WGSNs and high-level SWE services, the GeoMQTT-REST bridge is a powerful tool. It enables developers of such architectures to log and retrieve events which are issued from the sensor nodes, but also send messages to the bus. However, the bridge does not replace the services for storing sensor data in the Sensor Web, such as the Sensor Observation Service (SOS). Due to the semi-structured nature of the underlying document database MongoDB, it can be used for logging the raw events of a GeoMQTT system or for debugging purposes.

[1] https://www.earlydike.de/
[2] http://leafletjs.com/

REFERENCES

[1] S. Nittel, "A Survey of Geosensor Networks: Advances in Dynamic Environmental Monitoring," *Sensors*, vol. 9, no. 7, pp. 5664-5678, July 2009.

[2] A. Broering *et al.*, "New Generation Sensor Web Enablement," *Sensors*, vol. 11, no. 3, pp. 2652-2699, March 2011.

[3] M. Grothe and J. Kooijman, *Sensor Web Enablement*. Delft, NL:NCG, 2008.

[4] A. Broering, T. Foerster, S. Jirka and C. Priess, "Sensor Bus: An Intermediary Layer for Linking Geosensors and the Sensor Web," *Proc. COM.Geo 2010, 1st Int. Conf. on Computing for Geospatial Research and Applications*, Washington DC, 2010, p.1-8.

[5] S. Herle and J. Blankenbach, "GeoPipes using GeoMQTT" in *Geospatial Data in a Changing World. Selected papers of the 19th AGILE Conference on Geographic Information Science*, T. Sarjakoski, M. Santos and T. Sarjakoski, Ed. Springer International Publishing Switzerland, 2016, pp. 383-398.

[6] P. Eugster, A. Felber, R. Guerraoui and A.-M. Kermarrec "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, June 2003.

[7] *MQTT Version 3.1.1*, OASIS Standard, 2014.

[8] A. Stanford-Clark and H. L. Truong "MQTT for Sensor Networks (MQTT-SN) Protocol Specification Version 1.2," IBM Zurich Res. Lab., Zurich, Nov. 2012.

[9] *Representations of dates and times*, ISO 8601:2004, 2004.

[10] Terracotta Inc. (2016). *Quartz CronTrigger Tutorial* [Online]. Available: http://www.quartz-scheduler.org/documentation/quartz-2.x/tutorials/crontrigger.html

[11] M. Collina, G. E. Corazza and A. Vanelli-Coralli "Introducing the QEST broker: scaling the IoT by bridging MQTT and REST," in IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), Sydney, Australia, 2012, pp. 36-41.

[12] M. Koster (2013, Oct. 4) M2M Protocol Interoperability Using the Smart Object API. *Data Models for the Internet of Things*. Available: http://iot-datamodels.blogspot.de/2013/10/m2m-protocol-interoperability-using.html

[13] *The WebSocket Protocol*, IETF Standard RFC 6455, 2011.