Evaluation aktueller Message Oriented Middleware Technolgien zum Einsatz im IoT-Umfeld

Jannis Priesnitz
Fachbereich Informatik
Hochschule Darmstadt
Schöfferstraße 3
64295 Darmstadt
Email: jannis.priesnitz@stud.h-da.de

***Zusammenfassung*—Message Oriented Middleware (MOM) Technologien sind im Bereich der Maschine zu Maschine Kommunikation zum Standard geworden. Mit Aufkommen des Internet of Things bzw. des in Deutschland geprägten Begriffs Industrie 4.0 wurden diese Technologien zunehmend für den Bereich der Embedded Systems interessant. Hier soll die Vision „vom Sensor in die Cloud“ in die Praxis umgesetzt werden.**

Diese Arbeit soll als Entscheidungsgrundlage für die Auswahl einer MOM-Technologie für ein „Master Projekt Systementwicklung“ im Bereich IoT an der Hochschule Darmstadt in Verbindung mit T-Systems dienen.

***Index Terms*—Message Oriented Middleware, IoT**

I. Einführung

Im Gegensatz zur weit verbreiteten Client-Server Architektur, die beispielsweise im Internet angewendet wird, steht bei Message Oriented Middleware die Kommunikation zwischen Clients im Vordergrund. Hierbei steht ein *Message Broker* im Mittelpunkt, welcher den Nachrichtenfluss der Clients koordiniert. Clients veröffentlichen (*publish*) Nachrichten bezüglich eines Themas (*topic*) und abonnieren (*subscribe*) sich wiederum Themen, zu denen sie Nachrichten erhalten wollen[1]. Der Message Broker funktioniert somit als Grand Central Dispatcher zwischen allen angeschlossenen Clients. Ähnlich wie beim Client-Server-Modell sind die Kommunikationsteilnehmer nicht auf eine Hardwarearchitektur, ein Betriebssystem oder eine Programmiersprache festgelegt. Zudem sieht die Arichtektur eine M-zu-N-Kommunikation vor, bei der beliebig viele Clients auf einem Topic Nachrichten veröffentlichen und empfangen können (Vgl.1).

Grundlage dieser Architektur ist das Advanced Messaging Queue Protocol (AMQP)[2], welches die wichtigen Punkte Sicherheit, Zuverlässigkeit (*Qualities of Service*)und Kommunikationsverhalten spezifiziert. Aktuelle Implementierungen enthalten noch zusätzliche Funktionen zur Performancesteigerung und Ausfallsicherheit.

II. Anforderungen und Evaluationskriterien

A. Hardwarevoraussetzungen

Bei der Evaluation ist im IoT-Kontext zu beachten, dass nicht nur eine hohe Anzahl an Devices kommunizieren, sondern auch, dass diese sehr unterschiedliche Hardwarevoraussetzungen mitbringen.

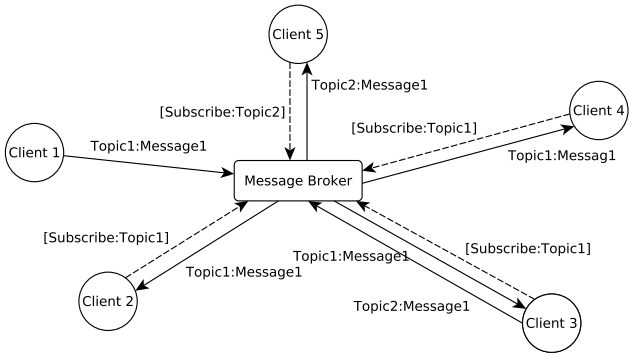

Abbildung 1. Schematische Darstellung der MOM-Architektur

B. Clients

Im IoT-Umfeld ist es üblich, dass Embedded Devices die Daten von verschiedenen Sensoren einsammeln und diese publizieren. Diese Systeme verfügen in der Regel nur über wenige Kilobyte Arbeitsspeicher und haben kein Betriebssystem. Um diese Systeme zu unterstützen, sollten die MOMs über ein C-Binding verfügen, welches in Umgebungen ohne Betriebssystem integriert werden kann und i.A. recourcenschonender ist als interpretierte Sprachen.

C. Anbindung an BigData Server und Clouds

Die von Sensoren generierten Daten werden im Allgemeinen auf Big Data Servern oder in Cloudlösungen abgelegt, die aus diesen Schlüsse ziehen. Hier sollte die Performance des Brokers keinen Flaschenhals des Systems darstellen, was neben Hardware- auch Softwareanforderungen stellt. So wäre z.B. eine Verteilung des Brokers auf mehrere Hardwareplattformen eine interessante Alternative.

III. Ausgewählte MOM-Implementierungen vorgestellt

In diesem Abschnitt werden ausgewählte Implementierungen vorgestellt, welche alle unter nicht kommerziellen Lizenzen vertrieben werden und frei nutzbar sind.

A. Hinweis zur Bewertung

Aufgrund des kurzen Zeitrahmens zur Ausarbeitung dieses Textes konnten nicht alle Entscheidungskriterien wissenschaftlich bewiesen (z.B. durch prototypische Implementierung oder Performancemessungen etc.) und hinreichend begründet werden und beruhen zum Teil auf einem ersten Eindruck und Gefühl. Diese Arbeit versteht sich eher als Basis für detailliertere Evaluationen. Das Fazit soll ebenfalls als Vorschlag auf Basis der bisherigen Erkenntnisse als Diskussionsanregung dienen.

B. Übersicht über berücksichtigte Implementierungen

Die folgenden Technologien wurden im Rahmen der Recherche berücksichtigt, jedoch nicht weiter verfolgt:

- Enduro/X: Interessante Ansätze (echtzeitfähig, POSIX Kernel Queues, ...) sehr junges Projekt und eine kleine Community
- HornetQ: Sehr stark im High Performance Bereich angesiedelt
- ZeroMQ: Läuft ohne Broker auf Basis von Berkley Sockets[3].
- StormMQ: Kein C-Binding, serviceorientiert, starker Fokus auf Sicherheit[4].
- NATS: Zu spät für ein detailiertes Review gefunden. Eher im high Performance Bereich angesiedelt[**?**].
- MSMQ: Starke Abhängigkeiten zu Microsoft[5].
- Gearman: Keine aktive Entwicklung mehr[6].
- Celery: Interessantes Konzept. Einstieg erscheint jedoch kompliziert[7][8].

C. RabbitMQ

Die erste der näher evaluierten Alternativen stellt RabbitMQ dar. RabbitMQ erweist sich den publizierten Informationen nach als sehr universell einsetzbar. Neben zahlreichen Aspekten zur Ausfallsicherheit und Hochverfügbarkeit verfügt es ebenfalls über ein Management Plugin im Broker, welches Konfiguration, Monitoring und Debugging vereinfacht[9]. Des weiteren verfügt RabbitMQ über ein C-Binding, über dessen Kompatibilität zu Embedded Systems kann aktuell jedoch keine Aussage getroffen werden[10][11]. Ein positiver Aspekt ist ebenfalls, dass RabitMQ seit Jahren aktiv weiterentwickelt wird und über eine groß Community verfügt.

D. MQTT

Traut man Internetnachrichtendiensten oder schaut man auf die Agenden von Developertalks, so bekommt man den Eindruck, dass sich MQTT als Standardprotokoll im IoT Umfeld etabliert hat[12][13]. Der Hauptgrund dafür sind vor allem die sehr leichtgewichtigen Clientimplementierungen, welche auf den kleinsten SoCs lauffähig sind.[14] Auch der Broker ist eher auf Einfachheit optimiert und verzichtet beispielsweise auf eine Autorisierung, um die Nachrichtenlänge möglichst kurz zu halten[15]. Außerdem wurde auf Replikationsoptionen, Clustering und Fehlerbehandlungen verzichtet. Vermutlich ist die maximale Performance des MQTT Brokers den anderen Technologien ebenfalls unterlegen. Diese Einfachheit ermöglicht jedoch auch eine zielstrebige Implementierung von Prototypen[16][17].

E. Apache ActiveMQ

Die Hauptvorteile von ActiveMQ bestehen in der Möglichkeit, ein Netz aus Brokern zu betreiben, in dem Clients sich zu einem beliebigen Broker verbinden. Fällt der Broker aus, wird automatisch ein Reconnect auf einem anderen Brocker durchgeführt. Das Routing zwischen den Endpunkt findet dabei vollkommen transparent zum Client statt. Replikation von Brokern wird ebenfalls unterstützt. Dadurch wird ActiveMQ hohen Anforderungen an Performance und Ausfallsicherheit gerecht. Für die Clientintegration stehen zahlreiche Bindings zur Verfügung, welche laut Apache jedoch primär darauf abzielen, legacy Code von Enterpriseapplications zu in neue MOM-Umgebungen integrieren. Auch ActiveMQ wird aktuell noch weiterentwickelt und verfügt über eine ausführliche Dokumentation.[18]

F. Apache Kafka

Apache Kafka stellt mit einem anderen Konzept eine Alternative zu den zuvor vorgestellten Technologien dar. Kafka stellt primär ein Software Paket zur Verarbeitung von Datenströmen dar, welches hoch skalierbar ist und verteilt operieren kann und somit hervorragend im BigData Bereich einsetzbar ist[19][1]. Kafka ist in der Lage, große Nachrichtenvolumen durch Load Balancing und Clustering deutlich schneller zu verarbeiten, als eine MOM Lösung, kann jedoch nicht ohne eines der o.g. MOMs als Datenquelle eingesetzt werden und wäre daher eher Teil einer hybriden Lösung, wie in III-G beschrieben. Kafka soll leicht installierbar sein und verfügt über eine unfangreiche Dokumentation[20].

G. Hybride Lösungen

Wie bereits in der Einführung beschrieben, kommen im Bereich IoT-Systeme mit sehr unterschiedlichen Hardwarevoraussetzungen zum Einsatz. Dies macht eine hybride Lösung aus mehreren Middlewaretechnologien interessant.

Zum einen wäre eine Zwei-Stufen-Lösung denkbar, in der ein oder mehrere Message Broker ein Kafka BigData System mit Daten versorgen, welches diese aufbereitet und in einer Cloud ablegt. Hierbei sollte im aktuellen Projekt jedoch Implementierungsaufwand und Nutzen abgewogen werden.

Zum anderen existieren Schnittstellen zwischen den vorgestellten Systemen, die es möglich machen, die Vorteile der jeweiligen Systeme zu nutzen. Mehrwert und Funktionalität sollten aber auch hier genau evaluiert werden.

IV. Fazit in Bezug auf das konkrete Projekt

Zusammenfassend kann man sagen, dass jede der vorgestellten Message Oriented Middleware Implementierungen, RabbitMQ, MQTT und ApacheMQ, prinzipiell die gestellten Anforderungen erfüllt[21]. Geht man weiter ins Detail, offenbaren

[1] Kafka ist also streng genommen keine Message Oriented Middleware. Da es sich aber u.U. um eine für das Projekt interessante Technologie handelt, soll sie hier trotzdem Beachtung finden.

sich Unterschiede, die für das Projekt relevant sein können. Daher sollte die Entscheidung welche Technologie eingesetzt wird, auf Basis der Embedded Devices, die verwendet werden und des erwarteten Nachrichtenvolumens im gesamten Netzwerk getroffen werden. Außerdem sind implementierungsspezifische Besonderheiten der jeweiligen MOM noch nicht bekannt. Auch eine Implementierung, die hier nicht näher betrachtet wurde, wie EnduroX oder NATS, wäre prinzipiell für den Einsatz im Projekt denkbar. Ein weiterer wichtiger Entscheidungspunkt ist inwieweit Data Streaming Architecturs und Clouddienste zum Einsatz kommen sollen und diese selbst bereit gestellt werden sollen. Für diese Anforderung bietet Apache Kafka ein mächtiges Werkzeug.

V. Die nächsten Schritte

Als nächsten Schritte sollten nun zum einen die Anforderungen an die Message Oriented Middleware auf Basis der genauen Projektbeschreibung ausgearbeitet werden. Zum anderen sollte ein beispielhaftes Setup und eine Referenzimplementierung für die in Frage kommenden Systeme erfolgen, um Limitierungen und Schwierigkeiten möglichst frühzeitig zu erkennen. [25] [22]

Quellen

[1] Remark, "Publish-subscribe with activemq and nms," 2016. [Online; Stand 17. August 2016].

[2] Wikipedia, "Advanced message queuing protocol — wikipedia, die freie enzyklopädie," 2016. [Online; Stand 17. August 2016].

[3] Wikipedia, "Zeromq — wikipedia, the free encyclopedia," 2016. [Online; accessed 17-August-2016].

[4] Wikipedia, "Stormmq — wikipedia, the free encyclopedia," 2015. [Online; accessed 17-August-2016].

[5] Wikipedia, "Microsoft message queuing — wikipedia, the free encyclopedia," 2016. [Online; accessed 17-August-2016].

[6] Wikipedia, "Gearman — wikipedia, the free encyclopedia," 2016. [Online; accessed 17-August-2016].

[7] Wikipedia, "Celery (software) — wikipedia, the free encyclopedia," 2016. [Online; accessed 17-August-2016].

[8] D. R. Greenfeld, "Celery wiki," 2016. [Online; accessed 17-August-2016].

[9] M. Bumb, "What is rabbit mq, why to use and how to install," 2016. [Online; Stand 17. August 2016].

[10] A. Antonuk, "[rabbitmq-discuss] simplest possible embedded message producer," 2016. [Online; accessed 17-August-2016].

[11] T. K. et al., "Rabbitmq for iot [mailing list]," 2016. [Online; accessed 17-August-2016].

[12] S. S. . E. Design, "Understanding the protocols behind the internet of things," 2016. [Online; accessed 17-August-2016].

[13] B. Cabé, "Five things i learnt at iot world 2016," 2016. [Online; accessed 17-August-2016].

[14] Wikipedia, "Mq telemetry transport — wikipedia, die freie enzyklopädie," 2016. [Online; Stand 17. August 2016].

[15] A. Blewitt, "Paho, mosquitto and security of mqtt," 2016. [Online; accessed 17-August-2016].

[16] M. Stal, "Kommunikation über mqtt," 2016. [Online; accessed 17-August-2016].

[17] M. Bernet, "Kommunikation zwischen applikationen im internet of things (iot) [präsentation]," 2016. [Online; accessed 17-August-2016].

[18] Wikipedia, "Apache activemq — wikipedia, die freie enzyklopädie," 2016. [Online; Stand 17. August 2016].

[19] Wikipedia, "Apache kafka — wikipedia, die freie enzyklopädie," 2016. [Online; Stand 17. August 2016].

[20] A. D. Cabrera, "Apache kafka wiki," 2016. [Online; Stand 17. August 2016].

[21] P. Zaitsev, "Exploring message brokers: Rabbitmq, kafka, activemq, and kestrel," 2016. [Online; accessed 17-August-2016].

[22] N. Estrada and H. Astudillo, "Comparing scalability of message queue system: Zeromq vs rabbitmq," in *Computing Conference (CLEI), 2015 Latin American*, pp. 1–6, IEEE, 2015.

[23] T. T. N. D. Team], "Dissecting message queues," 2016. [Online; accessed 17-August-2016].

[24] Pivotal, "Amqp 0-9-1 model explained," 2016. [Online; Stand 17. August 2016].

Weiterführende Literatur

[25] G. Banavar, T. Chandra, R. Strom, and D. Sturman, "A case for message oriented middleware," in *International Symposium on Distributed Computing*, pp. 1–17, Springer, 1999.

[26] O. Standard, "Oasis advanced message queuing protocol (amqp) version 1.0., 2012," *URL http://docs. oasis-open. org/amqp/core/v1. 0/os/amqp-core-complete-v1. 0-os. pdf*.

[27] E. Sha, S.-K. Han, C.-Z. Xu, M. H. Kim, L. T. Yang, and B. Xiao, *Embedded and Ubiquitous Computing: International Conference, EUC 2006, Seoul, Korea, August 1-4, 2006, Proceedings*, vol. 4096. Springer, 2006.

[28] E. Curry, "Message-oriented middleware," *Middleware for communications*, pp. 1–28, 2004.

[29] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz, "A message-oriented middleware for sensor networks," in *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pp. 127–134, ACM, 2004.