Vol.2018-SE-198 No.12
2018/3/9

IoT システムのためのエッジアーキテクチャ設計方法論の提案と評価

濱野 真伍[†1]　青山 幹雄[†2]

概要：IoT システムの発展に伴い，デバイスからクラウドにメッセージを収集する中間層としてエッジが注目されている．しかし，エッジアーキテクチャには多様な非機能要求があり，それらを同時に満たすエッジアーキテクチャ設計は容易ではない．本稿では，非機能要求を満たす特性とアーキテクチャ設計概念からなるアーキテクチャデザインパターンを用いて，エッジアーキテクチャを段階的に設計する方法を提案する．提案方法を車載システムへ適用し，具体的なアーキテクチャの設計を行った．設計したアーキテクチャのプロトタイプを実装し，非機能要求を評価することで，提案方法の有効性を示す．

キーワード：IoT，エッジコンピューティング，設計方法論，ソフトウェアアーキテクチャ，Publish/Subscribe アーキテクチャ，アーキテクチャパターン

A Design Methodology of Edge-Computing Architecture for IoT Systems

SHINGO HAMANO[†1]　MIKIO AOYAMA[†2]

1. はじめに

IoT(Internet of Things)[1]システムの発展に伴い，大量かつ多種多様なデバイスがインターネットに接続されている．デバイスからクラウドへメッセージを収集する中間層としてエッジコンピューティング(以下エッジ)が注目されている．エッジを実現するアーキテクチャモデルとして Publish/Subscribe (以下 Pub/Sub)があり, Pub/Sub の具体的な仕様として MQTT が標準化されている[10]．エッジアーキテクチャはクラウドのアーキテクチャと異なり，スケーラビリティなどの多様な非機能要求がある．しかし，多様な非機能要求を満たすエッジアーキテクチャの設計方法は未確立である．

本稿では，複数の非機能要求を満たすためのエッジアーキテクチャの段階的設計方法論を提案する．提案方法を車載システムへ適用し，具体的なエッジアーキテクチャの設計と，そのプロトタイプを実装する．異なる非機能要求に対応する複数のシナリオをプロトタイプで実行し非機能要求を評価することで，提案方法の有効性を示す．

2. 研究課題

本稿ではエッジアーキテクチャを上位層からクラウド，エッジ，デバイスの 3 層とする．この 3 層アーキテクチャを対象として以下の 2 点を研究課題(RQ)とする．

(1) **RQ1**: エッジアーキテクチャの多様な非機能要求を実現可能な設計方法.
(2) **RQ2**: (1)の方法を車載システムに適用し，具体的なアーキテクチャの設計による有効性の評価.

†1 南山大学大学院 理工学研究科 ソフトウェア工学専攻
Graduate Program of Software Engineering, Nanzan University
†2 南山大学 理工学部 ソフトウェア工学科
Dep. of Software Engineering, Nanzan University

3. 関連研究

3.1 oneM2M 参照アーキテクチャ

oneM2M は IoT プラットフォームの標準化を目的として設立した団体である．参照アーキテクチャは様々なエンティティが相互接続可能であることを示し，上位層からクラウドなどのインフラストラクチャノード，ゲートウェイなどのミドルノード，デバイスなどのデバイスノードの 3 層からなる[12]．ノード間通信に MQTT を適用するプロトコルバインディング[13]が定義されている．

3.2 エッジコンピューティングアーキテクチャ

エッジ[16]はクラウドの概念をネットワークのエッジへ拡張し,クラウドとデバイスの間にエッジサーバを導入し，低レイテンシーや高スケーラビリティを実現するアーキテクチャである．フォグコンピューティング[3]と呼ばれる場合もあるが本稿ではエッジと呼ぶことにする．

エッジに処理を分散しデバイスでのサービス実行の遅延を低減するためのエッジアーキテクチャが提案されている[14]．また，複数のエッジノードに処理を均等に分散するフレームワーク[18]が提案されている．

3.3 Publish/Subscribe アーキテクチャ

ブローカを介して非同期でメッセージを配信するアーキテクチャである[17]．メッセージ受信者はブローカにメッセージ配信要求(Subscription)を送信し，ブローカによってフィルタリングされたメッセージを受信する．IoT 向けの仕様としてトピックベースでメッセージを配信し QoS をサポートする MQTT[10]がある．MQTT は OASIS で標準化され ISO/IEC で標準化されている．MQTT の実装としてブローカは Eclipse Mosquitto[8]，クライアントは Eclipse Paho-MQTT[9]がある．MQTT を適用したシステムに QEST ブローカ[7]がある．QEST ブローカは MQTT ブローカに

REST サーバを統合し，非同期配信と同期配信が併用可能なメッセージングブローカである．QEST ブローカはスケーラビリティを確保するために，並行可能が可能である．

3.4 品質特性駆動アーキテクチャ設計

品質特性駆動アーキテクチャ設計(ADD: Attribute Driven Design)は非機能要求を満たすために再帰的にモジュール分割を行う手法である[2,5]．モジュール分割の各段階で実現手法とアーキテクチャパターンを選択してアーキテクチャを設計する．

4. アプローチ

IoT におけるエッジへの非機能要求は多様である．複数の非機能要求を同時に満たすエッジアーキテクチャの設計は容易ではない．本稿では，図 1 に示すように非機能要求を満たすための特性とエッジアーキテクチャの設計概念からなるアーキテクチャデザインパターン(AD パターン)によって複数の非機能要求を満たすエッジアーキテクチャが設計可能になることに着目する．前提条件として，アーキテクチャ構成要素から設計概念を抽出し，設計概念と特性を結びつけたアーキテクチャデザインパターンと，その集合であるデザインパターンリポジトリ(AD パターンリポジトリ)は定義済みであるとする．エッジアーキテクチャ設計のために ADD を拡張するアプローチをとる．非機能要求を，それを満たすための特性に分割する．特性ごとに AD パターンからエッジアーキテクチャ設計概念を選択し，エッジを段階的に繰り返して拡張することで複数の非機能要求を満たすエッジアーキテクチャを設計する方法を提案する．

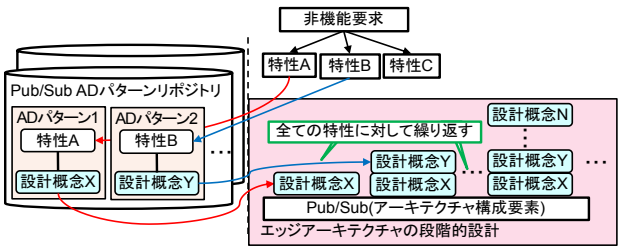

図 1 アプローチ
Figure 1 Approach

5. アーキテクチャ設計方法論

本稿では車載システムを例として，提案する設計方法論とその有効性を示す．

5.1 適用するシステム

本稿では，車両の位置情報と速度情報から走行ルートを導出するシステムを例として具体的なエッジアーキテクチャを設計する．以下の 6 点を前提条件とする．

(1) 車両をデバイスとし，位置情報と速度情報エッジに送信する．
(2) メッセージ配信には Pub/Sub を適用する．
(3) デバイスは大量に存在し，短い間隔でメッセージを送信する．
(4) エッジは地理的に分散して設置する．
(5) デバイスがメッセージを送信するエッジは変化する．
(6) エッジはデバイスからクラウドへのメッセージ配信のみを行う．

また，本システムのエッジに対する非機能要求は以下の 2 点である．

(1) スケーラビリティの確保: 1 つのエッジ内で大量のメッセージを処理可能である必要がある．
(2) メッセージ配信の完全性の確保: 正確な走行ルートを導出するためにエッジが受信したメッセージは全てクラウドに配信可能である必要がある．

5.2 アーキテクチャ設計プロセス

図 2 に本例題のアーキテクチャ設計プロセスを示す．提案する設計方法ではアプローチに基づき 4 つのプロセスでアーキテクチャを設計する．以下に各プロセスの詳細を示す．ただし，本稿において非機能要求の獲得，アーキテクチャ構成要素から設計概念の抽出，AD パターンの設計，AD パターンリポジトリの定義は行われているものとする．

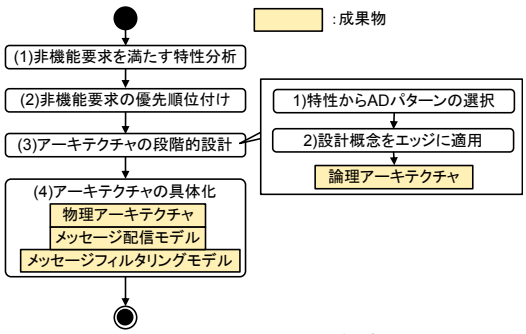

図 2 アーキテクチャ設計プロセス
Figure 2 Architecture Design Process

(1) 非機能要求を満たす特性分析: 非機能要求を満たすために必要な性質を特性とし，非機能要求ごとに特性を分析し獲得する．
(2) 非機能要求の優先順位付け: 獲得済みの非機能要求ごとに設計の優先順位をつける．
(3) アーキテクチャの段階的設計: (2)で定義した優先順位ごとに以下の 2 つのサブプロセスを繰り返す．
1) 特性から AD パターンの選択: (1)で獲得した特性を満たす AD パターンを選択する．
2) 設計概念をエッジに適用: 1)で選択した AD パターンの設計概念をエッジに適用する．

2 つのサブプロセスを全ての特性に対して適用したものを論理アーキテクチャとする．

(4) アーキテクチャの具体化: 論理アーキテクチャの構成要素と設計概念のコンポーネントを具体化する．また，論理アーキテクチャから物理アーキテクチャやメッセージ配信モデルなどを設計する．

5.3 アーキテクチャデザインパターン(AD パターン)

AD パターンはエッジアーキテクチャの構成要素に基づく拡張方法である設計概念と，非機能要求を分割した特性を結びつけるものである．各パターンは設計概念と同じ名前を持つ．AD パターンの集合を AD パターンリポジトリとする．AD パターンリポジトリはエッジの構成要素ごとに定義する．

本例題では構成要素に Pub/Sub を用いる．図 3 に Pub/Sub AD パターンリポジトリと 4 つの AD パターンを示す．また，図 4 に Pub/Sub の構成要素から抽出した AD パターンの設計概念の詳細を示す．

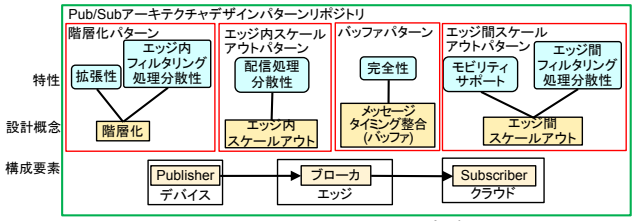

図 3 Pub/Sub AD パターンリポジトリ
Figure 3 Pub/Sub AD Pattern Repository

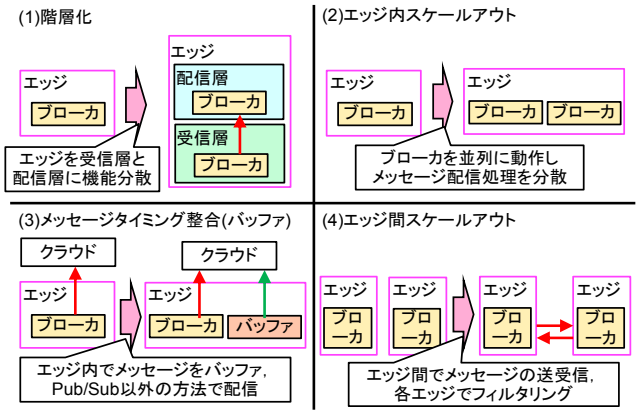

図 4 アーキテクチャ設計概念の詳細
Figure 4 Detail of Architecture Design Concept

(1) 階層化: ブローカの機能をデバイスからのメッセージを受信する受信層とクラウドへのメッセージを配信する配信層に機能分散する．

(2) エッジ内スケールアウト: エッジのブローカの数を増加し，Pub/Sub のメッセージ配信処理を分散する．

(3) メッセージタイミング整合(バッファ): エッジでメッセージをバッファし，Pub/Sub とは異なる配信方法でメッセージを配信することでタイミングを整合する．

(4) エッジ間スケールアウト: 複数のエッジ間で Pub/Sub によるメッセージの送受信を行う．この際にメッセージのフィルタリングも行う．

5.4 非機能要求を満たす特性分析

獲得済みの非機能要求を詳細化し特性とする．1 つの非機能要求は 1 つ以上の特性に詳細化される．特性はエッジアーキテクチャの構成要素に基づいて詳細化される．特性は再帰的に詳細化することが可能である．特性は AD パターンリポジトリの特性と一致するまで詳細化する．これを非機能要求を満たす特性分析とする．非機能要求の実現には 1 段階詳細化した特性を全て満たすことであるとする．一方で特性を再帰的に詳細化した場合の関係には以下の 2 つのパターンがある[6].

(1) And パターン: 対象の特性を実現するためには，1 段階詳細化した特性を全て満たすことが必要なパターン．And パターンは詳細化した特性が同じアーキテクチャ構成要素に適用される場合に用いる．

(2) Or パターン: 対象の特性を実現するためには，1 段階詳細化した特性のうちいずれかを満たすことが必要なパターン．Or パターンは詳細化した特性が異なるアーキテクチャ構成要素に適用される場合に用いる．

本例題における非機能要求の詳細化および特性分析，特性間のパターンを図 5 の(1)に示す．また本例題における特性の詳細を以下に示す．

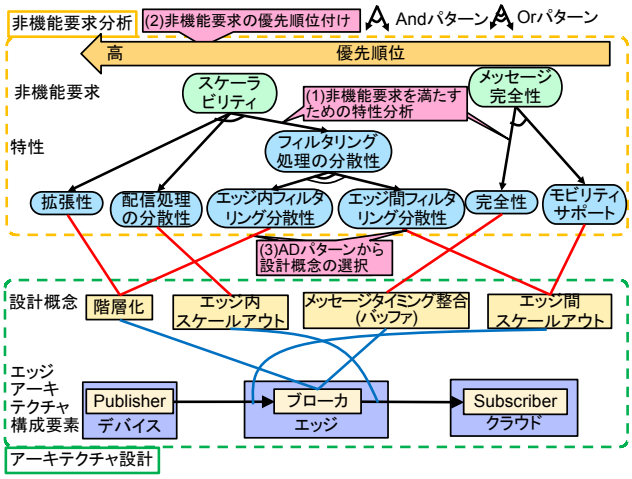

図 5 アーキテクチャ設計方法
Figure 5 Architecture Design Method

(1) スケーラビリティ: 本例題におけるスケーラビリティはエッジが処理するメッセージ数に関係なく，メッセージ配信のパフォーマンスの保持が可能である状態とする．スケーラビリティ実現のために以下の 3 つの特性を獲得した．

1) 拡張性: 1 つのエッジ内において，接続されるデバイス数や処理するメッセージ数に応じて処理を分散するために拡張可能である必要がある．
2) 配信処理の分散性: Pub/Sub ではメッセージ配信時に配信先を参照するため，受信時と比較して配信時の負荷が大きくなる．そこで，メッセージの配信処理を複数のブローカに分散可能である必要がある．
3) フィルタリング処理の分散性: デバイスの数や種類が多い本例題では，メッセージのフィルタリング条件が複雑なためフィルタリング処理を分散する必要がある．そこで，フィルタリング処理の適用場所によって以下の 2 つの特性に詳細化し Or パターンで結合する．

I) エッジ内フィルタリング処理分散性: 1 つのエッジ内

部において階層的なフィルタリング処理により，それぞれのブローカが参照する配信先の数を少なくする必要がある．

II) エッジ間フィルタリング処理分散性: 他のエッジへのメッセージ送信時に，メッセージをフィルタリングすることで，送信先のエッジが不要なメッセージを受信しないようにする必要がある．

(2) メッセージの完全性: 本例題におけるメッセージの完全性は，デバイスから受信したメッセージを全てクラウドに配信可能であり，メッセージを配信するエッジはクラウドからの Subscription を受信したエッジで配信可能である状態とする．メッセージの完全性実現のために以下の 2 つの特性を獲得した．

1) 完全性: 一般に Pub/Sub のブローカにはメッセージを保存する機能はない．エッジは Pub/Sub のメッセージを配信のみでは，メッセージの完全性を保証できない．そのため，エッジは受信したメッセージを Pub/Sub 以外の方法でクラウドに配信可能である必要がある．
2) モビリティサポート: デバイスは異なるエッジ間を移動可能であり，メッセージを送信するエッジは固定されていない．クラウドから Subscription を受信したエッジでメッセーの完全性を保証することができない．そのため，エッジ間でメッセージの送受信を行い，クラウドからの Subscription を受信するエッジからメッセージが配信可能である必要がある．

5.5 非機能要求の優先順位付け

獲得済みの非機能要求をアーキテクチャに適用するための優先順位をつける．この優先順位付けにより以降のアーキテクチャの段階的設計の設計順位とする．非機能要求の優先順位付けの後に非機能要求ごとに特性の優先順位をつける．この特性の優先順位ごとに論理アーキテクチャの拡張を行う．本例題の優先順位を図 5 の(2)に示す．図 5 では図中の左に非機能要求や特性が優先順位の高いものを配置する．本例題の優先順位の詳細を以下に示す．

(1) スケーラビリティ: 本例題では大量のデバイスとメッセージを Pub/Sub を用いて配信することが主要な目的であり，メッセージの完全性を実現するためにはスケーラビリティが必要であるため，スケーラビリティを優先順位 1 位とする．また各特性の優先順位は以下のように定義した．

1) 拡張性
2) 配信処理分散性
3) エッジ内フィルタリング処理分散性
4) エッジ間フィルタリング処理分散性

(2) メッセージの完全性: メッセージの完全性を実現するためには Pub/Sub 以外のメッセージ配信方法が必要であり，それを実現するためにはスケーラビリティが必要であるため，優先順位 2 位とする．また，各特性の優先順位は以下のように定義した．

1) 完全性
2) デバイスモビリティ

本例題ではどの非機能要求も，1 つのエッジ内にのみ関連する特性の優先順位高くし，複数のエッジ間に関連する特性の優先順位を低くした．

5.6 アーキテクチャの段階的設計

アーキテクチャの設計は，5.4 節で獲得した特性を満たす AD パターンの選択し，5.5 節で定義した優先順位ごとに設計概念を適用する．選択した AD パターンの全ての設計概念を適用したアーキテクチャを論理エッジアーキテクチャとする．

5.6.1 AD パターンから設計概念の選択

5.4 節で分析した特性を満たす AD パターンを AD パターンリポジトリから選択する．AD パターンリポジトリはアーキテクチャの構成要素に従って選択する．AD パターンリポジトリを選択したら，AD パターンに従って図 5 の(3)に示すように優先順位に従って配置した特性と設計概念を結ぶ．その際，特性から設計概念を選択するため，1 つは 1 つの設計概念によって実現される．しかし，1 つの設計概念は複数の特性を満たすことが可能である．本例題における AD パターンリポジトリは Pub/Sub AD パターンリポジトリを用い，特性と設計概念を結んだ結果は図 5 である．

5.6.2 論理アーキテクチャの段階的設計

図 5 に示すようなアーキテクチャ設計方法に従って段階的に設計概念を適用しエッジアーキテクチャを設計する方法を以下に示す．

(1) 非機能要求のうち実現していないかつ，最も優先順位が高いものを選択する．
(2) (1)で選択した特性のうちアーキテクチャに実現済みでないかつ，最も優先順位が高いものを選択する．
(3) (2)で選択した特性と結ばれている設計概念をエッジに適用する．
(4) (3)で適用した設計概念が満たすことができる特性を全て実現済みとする．
(5) (2)~(4)を(1)で選択した非機能要求の特性を全て実現済みになるまで繰り返す．
(6) 全ての非機能要求が実現できていない場合(1)に戻る．

本例題における論理アーキテクチャの段階的設計とエッジの拡張手順を図 6 に示す．

(1) 階層化: 特性のうち最も優先順位が高い拡張性を満たすための階層化を適用する．階層化は図 6 の(1)に示すようにエッジを受信層と配信層に機能分散し，それぞれに Pub/Sub のブローカを導入する．これによりデバイスとメッセージ配信は疎結合となりし，メッセージ配信の拡張性を満たすことができる．階層化により表 1 に示す 2 つの Pub/Sub でメッセージを配信し，受信層と配信層のそれぞれでメッセージのフィルタリングを行う．これによりエッジ内のフィルタリング処理の分散性も実現する．

表 1 メッセージ配信の 2 つの Pub/Sub
Table 1 Two Pub/Sub of Messaging

メッセージ配信	Publisher	ブローカ	Subscriber
デバイス→エッジ	デバイス	受信層	配信層
エッジ→クラウド	受信層	配信層	クラウド

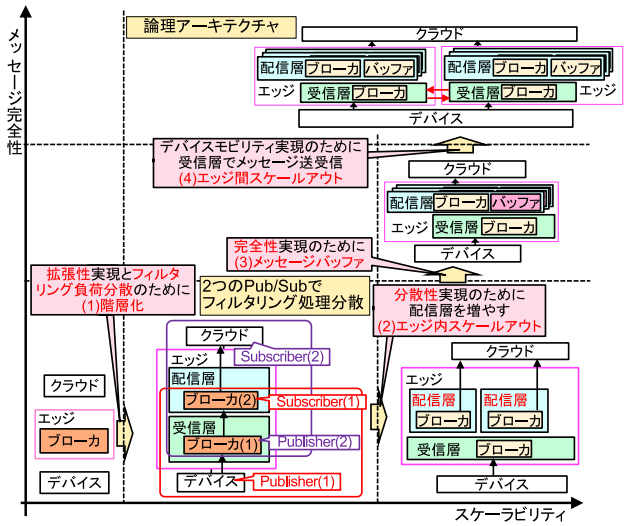

図 6 論理アーキテクチャの段階的設計
Figure 6 Incremental Design of Logical Architecture

(2) エッジ内スケールアウト: 次に配信処理の分散性を実現するために図 6 の(2)に示すエッジ内で配信層(ブローカ)の数を増加するエッジ内スケールアウトを適用する．これにより，各配信層が処理するメッセージ数を削減し，メッセージ配信負荷を分散する．

フィルタリング処理の分散性は Or 結合で詳細化されている．そのため階層化とエッジ内スケールアウトの 2 つの設計概念を適用により，スケーラビリティを 1 段階詳細化した全ての特性が実現済みになり，スケーラビリティを実現したと言える．次にメッセージの完全性を実現するための段階的設計を示す．

(3) メッセージタイミングの整合(バッファ): 完全性の特性を実現するために図 6 の(3)に示すようにエッジでメッセージをバッファする．メッセージのバッファはクラウドへのメッセージ配信を行い，スケールアウト可能な配信層で行う．これにより，エッジは Pub/Sub のタイミングとは異なるタイミングでメッセージを配信可能になり，メッセージの完全性を実現する．

(4) エッジ間スケールアウト: デバイスモビリティを実現するために図 6 の(4)に示すように，デバイスに近く，メッセージ保存による負荷が発生しない受信層でメッセージをフィルタリングし送受信する．これにより，配信層は異なるエッジの受信層からメッセージを受信する必要がなく，デバイスモビリティを保証する．また，メッセージをフィルタリングして他のエッジに送信するため，エッジ間フィルタリング処理の分散性も実現する．

5.7 アーキテクチャの具体化

5.6 節で設計した論理アーキテクチャを具体化する．本例題では物理アーキテクチャ，メッセージフィルタリングモデル，メッセージ配信モデル設計する．ただし，メッセージ配信モデルは物理アーキテクチャで定義したコンポーネント名で設計する．

5.7.1 コンポーネントの具体化

アーキテクチャの具体化のために，論理アーキテクチャで定義した設計概念を満たすためのコンポーネントを具体化する．

(1) Pub/Sub: プロトコルに標準化されている MQTT を適用する．トピックベースのフィルタリングではワイルドカードを用いることができるため，フィルタリング処理の分散に適している．

(2) メッセージバッファ: メッセージバッファは以下の 2 つの役割からなる．

1) メッセージの保存: メッセージを一時保存するストレージとして様々なデータ構造に対応したドキュメント指向データベース(ドキュメント DB)を適用する．
2) 保存メッセージの配信: 保存したメッセージの配信には REST を適用する．これにより，エッジは Pub/Sub と異なるタイミングでメッセージの配信が可能になる．

5.7.2 物理アーキテクチャ

論理アーキテクチャから図 7 に示す物理アーキテクチャを設計する．図 7 中の赤矢印は MQTT のメッセージ配信，緑線は REST による同期配信，黒実線は保存，黒破線は参照関係を示す．各エッジはエッジ内スケールアウトにより，1 つの受信層と複数の配信層で構成される．受信層，配信層の概要と機能を以下に示す．

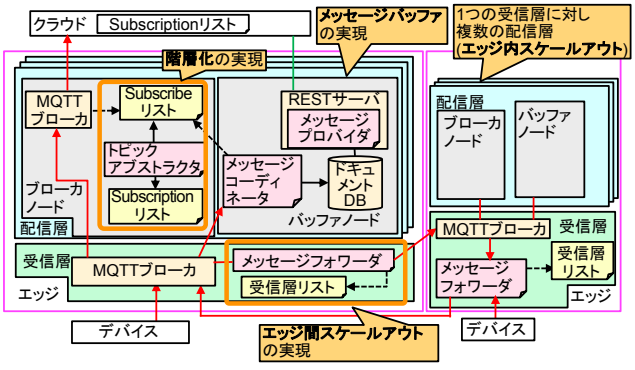

図 7 物理アーキテクチャ
Figure 7 Physical Architecture

(1) 受信層: 次の 3 つの機能を配置する．1)デバイスからのメッセージ受信．2)他のエッジにメッセージをフィルタリングして送信．3)配信層へメッセージをフィルタリングして配信．

(2) 配信層: ブローカノードとバッファノードから成る．各ノードの詳細を以下に示す．1) ブローカノード: クラウドへメッセージをフィルタリングし非同期配信，配信層が

指定するトピックの生成．2)バッファノード: メッセージを一定期間保存しクラウドへ同期で配信する．

AD パターンの設計概念を実現するコンポーネントは，アーキテクチャ上で独立して導入可能である．以下に設計概念ごとに導入するコンポーネントを示す．

(1) 階層化

1) Subscription リスト: クラウドから受信した Subscription のトピックのリストである．
2) トピックアブストラクタ:Subscription リストのトピックの一部を抽象化し，Subscribe リストに保存する．
3) Subscribe リスト: 配信層が受信層から受信するメッセージのトピックのリストである．
4) MQTT ブローカ(配信層):受信層のブローカから MQTT の Bridge 機能でメッセージを受信しクラウドに配信する．トピックは Subscribe リストを参照し指定する．

(2) メッセージバッファ

1) メッセージコーディネータ: 同配信層の Subscribe リストを参照し，受信層のブローカの Subscriber となる．受信したメッセージはドキュメント DB に保存する．
2) ドキュメント DB: バッファノードで受信したメッセージを保存するデータベースである．
3) メッセージプロバイダ, REST サーバ: クラウドからのリクエストに従って，ドキュメント DB のメッセージを返却する．

(3) エッジ間スケールアウト

1) メッセージフォワーダ: 同受信層のブローカから全メッセージを受信する．メッセージ毎に受信層リストを参照し，他の受信層のブローカに送信する．
2) 受信層リスト: トピックの一部と送信先のエッジのアドレスをキーバリュー形式で保存する．全てのエッジの受信層は同じ受信層リストを持つ．

5.7.3 メッセージフィルタリングモデル

図 8 に受信層と配信層のフィルタリングモデルを示す．

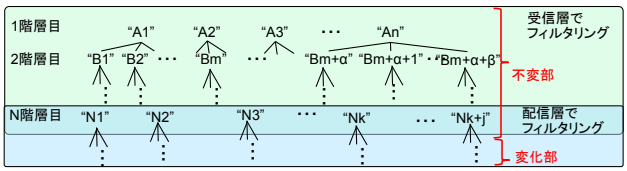

図 8 メッセージフィルタリングモデル
Figure 8 Message Filtering Model

受信層ではトピックの 1 階層目から N 階層目まで，配信層ではトピックの N 階層目から最下位層までのメッセージのフィルタリングを行う．そのため，配信層が指定する Subscription は N 階層目以下にマルチレベルワイルドカードを用いる．これにより配信層は受信層でフィルタリングされたメッセージのみを受信可能である．また，トピックは意味が異なる場合があるため，メッセージはトピックの意味を示す URI を記述する．

5.7.4 メッセージ配信モデル

エッジがデバイスからメッセージを受信してからクラウドに配信するまでのメッセージ配信モデルを図 9 に示す．ただし，図 9 では，デバイスのメッセージはエッジ A からクラウドに配信する．Bridge 機能による接続と，クラウドとメッセージフォワーダの Subscription は送信済みである．

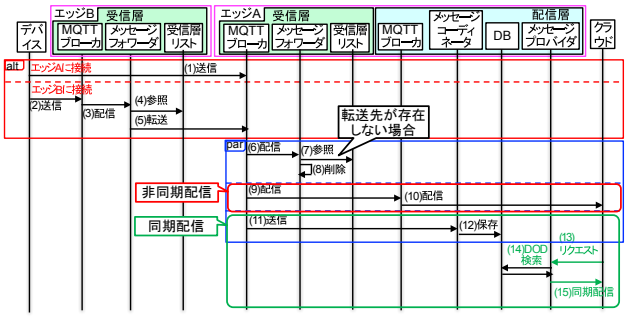

図 9 メッセージ配信モデル
Figure 9 Message Delivery Model

デバイスがエッジ A とは異なるエッジ B メッセージを送信する場合，エッジ B のメッセージフォワーダはエッジ A にメッセージを転送する．一方でエッジ A のメッセージフォワーダはメッセージを削除する．これにより配信層はデバイスがどのエッジにメッセージを送信する場合でも，クメッセージを 1 つの受信層から受信できる．さらに，配信層の MQTT ブローカとメッセージコーディネータは独立してメッセージを受信することで，非同期配信と同期配信を分離しているため，互いに影響を受けない．

6. プロトタイプの実装

提案方法によって設計したアーキテクチャにおける非機能要求の実現を確認するためにプロトタイプを実装した．

6.1 プロトタイプの実装環境

プロトタイプに使用したハードウェアコンポーネントを表 2 に，ソフトウェアコンポーネントを表 3 に示す．ただし，デバイスは Ubuntu OS 上に仮想的に生成した．

表 2 ハードウェアコンポーネント
Table 2 Software Components

クラウド, デバイス	
OS	Ubuntu 14.04
メモリ	2GB
CPU	Intel® CoreTM2 Duo CPU E7300 @ 2.66Ghz×2
エッジ	
デバイス	Raspberry Pi model B+
OS	Raspbian 9.3
メモリ	512MB
CPU	ARM1176JZF-S @ 700MHz
ストレージ	microSD カード 8GB

表 3 ソフトウェアコンポーネント
Table 3 Software Components

コンポーネント名	使用ソフトウェア	バージョン
MQTT ブローカ	Apache Mosquitto	1.4.9
MQTT クライアント	Apache Paho-MQTT	1.3.1
ドキュメント DB	MongoDB	2.4.14
受信層リスト	Radis	2.7.13
REST サーバ	Apache	3.2.6

6.2 プロトタイプの規模

エッジに実装したアプリケーションを表 4 に示す.

表 4 実装アプリケーション
Table 4 Implement Application

アプリケーション名	実装言語	LOC
メッセージコーディネータ	Python 2.7	53
メッセージプロバイダ	PHP	67
トピックアブストラクタ	Python2.7	29
メッセージフォワーダ	Python 2.7	43

7. 例題への適用

設計方法に従って具体化したアーキテクチャがスケーラビリティとメッセージの完全性を実現していることを確認するために 3 つのシナリオを実行した. シナリオの概要を図 10 に示す. 各シナリオは 3 回ずつ実行した. スケーラビリティを確認するために, 配信層(シナリオ 1 はエッジ全体)における CPU 使用率とロードアベレージを計測した. また, 全てのシナリオにおいてエッジの数は 2 つとしエッジ間スケールアウトと, メッセージバッファは実行する.

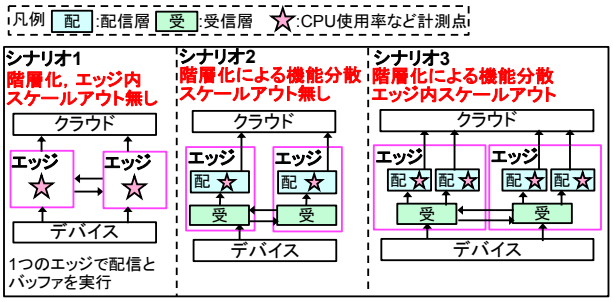

図 10 実行シナリオの概要
Figure 10 Outline of scenarios

デバイスの動作は以下に定義する.

(1) 120 秒間の間に 20 ミリ秒間隔で, 計 6,000 個のメッセージを送信する.
(2) メッセージを送信するエッジは乱数で決定する.
(3) MQTT プロトコルの QoS は 0 とする.
(4) 送信するメッセージのログをデバイス上で保存する.

各シナリオ実行後に配信層から同期配信でメッセージを受信し, デバイスのログ及び Pub/Sub で受信したメッセージと比較する.

8. 評価

提案方法によって設計したアーキテクチャが非機能要求を満たしていることを確認する.

8.1 スケーラビリティの評価

(1) CPU 使用率による評価

図 11 に各シナリオの OS を除く CPU 使用率を示す. 図 11 で横軸は 1 件目のメッセージを受信してからの経過時間を示す. 図 11 中の吹出しは 10 秒後から 110 秒後までのシナリオ毎の CPU 使用率の平均値を示す.

シナリオ 2 ではシナリオ 1 と比べ CPU 使用率の 49%削減を確認した. これは階層化でエッジ間のフィルタリング処理を分離したためである. また, シナリオ 3 はシナリオ 2 と比較して CPU 使用率の 54%削減を確認した. これはエッジ内スケールアウトによって各配信層が処理するメッセージ数が約 3,000 から約 1,500 に減少したためである. そのため, 配信層における CPU 使用率は単位時間あたりに処理するメッセージ数と比例関係にあることが考えられる. さらにシナリオ 3 では CPU 使用率の分散が最も小さくなることからメッセージ配信のパフォーマンスが維持できていると言える.

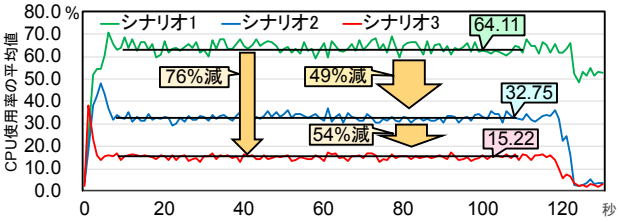

図 11 CPU 使用率
Figure 11 CPU Usage

(2) ロードアベレージによる評価

式 1 に本例題におけるロードアベレージの定義を示す.

$$\left(\sum_{\text{計測時刻}}^{\text{計測時刻の1分前}} \frac{\text{実行中のプロセス数} + \text{待ちプロセス数}}{\text{1コアあたりのCPUの最大処理プロセス数}}\right) / \text{1分間の計測回数} \quad (1)$$

図 12 に各シナリオにおけるロードアベレージを示し, 表 5 に線形近似を示す. 図 12 の横軸は 1 件目のメッセージを受信してからの経過時間を示す.

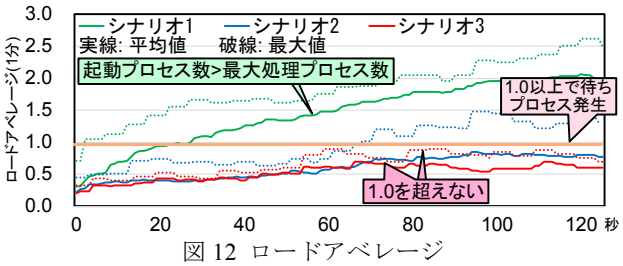

図 12 ロードアベレージ
Figure 12 Road Average

表 5 ロードアベレージの線型近似
Table 5 Liner Prediction of Road Average

シナリオ	平均値	最大値
シナリオ 1	y = 0.013x + 0.669	y = 0.010x + 1.220
シナリオ 2	y = 0.005x + 0.275	y = 0.009x + 0.391
シナリオ 3	y = 0.003x + 0.361	y = 0.005x + 0.390

シナリオ 1 では時間経過とともに, ロードアベレージが増加しているため, 起動プロセス数が最大処理プロセス数を超えておりスケーラビリティが実現できていない. 一方シナリオ 2 とシナリオ 3 では, ロードアベレージの平均値が 1.0 を超えることはない. しかし, シナリオ 2 の最大値が 1.0 を超えることがあるため, 待ちプロセスが発生することがあるが, シナリオ 3 ではでは最大値が 1.0 を超えることはない. これは, 配信層で処理するメッセージの負荷分散により, 各配信層でメッセージ配信と保存で実行するプロセス数が削減できたためである.

CPU 使用率とロードアベレージの計測により提案方法

で設計したアーキテクチャはスケーラビリティを実現していることを確認した．

8.2 メッセージの完全性の評価

各シナリオのデバイスで生成したメッセージ 6,000 件の内”Jcar”をトピックに持つメッセージは 1,200 件である．クラウドが”Jcar”を指定して非同期配信，同期配信で受信したメッセージも共に 1,200 件であり，デバイスで生成したメッセージと差異はなかったことを確認した．そのため，提案方法で設計したアーキテクチャではメッセージの完全性を保証することを確認した．

8.3 本例題における提案方法の評価

本例題で提案方法に従って設計したアーキテクチャはスケーラビリティとメッセージの完全性の 2 つの非機能要求の実現を確認した．よって，提案方法は複数の非機能要求を満たすエッジアーキテクチャ設計に有効である．

9. 考察

(1) QEST ブローカとの比較

本例題で設計したアーキテクチャと MQTT の設計に REST の設計を導入した QEST ブローカ[7]の性能を 5.4 節で獲得した特性の視点で比較した結果を表 6 に示す．

表 6 関連研究との比較
Table 6 Compare with QEST Broker

視点	提案アーキテクチャ	QEST ブローカ
拡張性	＋＋	＋
配信処理分散性	＋＋	＋
フィルタリング分散性	＋＋	－
完全性	＋＋	＋
デバイスモビリティ	＋	(記述なし)

表 6 より提案方法では非機能要求を特性に分割し AD パターンを用いてアーキテクチャ設計を行うことで，複数の特性を同時に満たすことができる．よって複数の非機能要求を同時に満たすアーキテクチャ設計が可能と言える．機能アーキテクチャを設計してから具体化を行うことで，コンポーネントに制限されない設計が可能と言える．

(2) 品質特性駆動アーキテクチャ設計との比較

提案した設計方法と ADD[2,5]の比較により，提案方法には以下の 2 つの有効性があると言える．

1) エッジアーキテクチャの設計に有効: 提案方法では，アーキテクチャの設計概念に基づき AD パターンを選択するため，スケーラビリティなどエッジ固有の非機能要求を満たす設計が可能であると考える．
2) アーキテクチャ設計の効率化が可能: 提案方法では獲得済みの非機能要求に優先順位をつけ AD パターンの設計概念を適用する．設計概念はそれぞれ独立しているため，段階的な設計が容易であり，論理アーキテクチャ設計の効率化が可能であると考える．

10. 今後の課題

今後の課題は以下の 4 点である．

(1) 提案する設計方法を例題以外のシステム適用可能であるかどうかの検証．および，他のエッジアーキテクチャ設計方法と比較による妥当性評価．

(2) AD パターンおよび AD リポジトリの生成方法の検討．

(3) トレードオフとなる非機能要求や特性がある場合の設計方法の検討．

(4) アーキテクチャの構成要素が複数あり，特性を満たす AD パターンが複数存在する場合における AD パターン選択方法の検討．

11. まとめ

多様な非機能要求を満たすエッジアーキテクチャの設計方法は未確立である．本稿では，非機能要求を詳細化した特性と設計概念からなる AD パターンを用いたエッジアーキテクチャの段階的設計方法論を提案した．提案方法を車載システムへ適用し，プロトタイプの実装と例題シナリオの実行により，スケーラビリティとメッセージの完全性のを評価し，提案方法の有効性を示した．

本研究の適用対象はエッジに限らない．多様な非機能要求のあるシステムのソフトウェアアーキテクチャ設計方法を提案した点に本研究の意義がある．

参考文献

[1] A. Al-Fuqaha, et al., Internet of Things, Proc. of CST ’15, IEEE, Jun. 2015, pp. 2347-2376.
[2] L. Bass, et al., Software Architecture in Practice 3rd ed., Addison Wesley, 2013.
[3] F. Bonomi, et al., Fog Computing and Its Role in the Internet of Things, Proc. of MCC ’12, ACM, Aug. 2012, pp. 13-16.
[4] R. Buyya, et al. (eds.), Internet of Things, Morgan Kaufmann, 2016.
[5] H. Cervantes, et al., Designing Software Architectures, Addison Wesley, 2016.
[6] L. Chung, et al., Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design, Proc. of 1st Int’l Workshop on Architectures for Software Systems, Apr. 1995, pp. 31-43.
[7] M. Collina, et al., Introducing the QEST Broker: Scaling the IoT by Bridging MQTT and REST, Proc. of PIMRC ’12, IEEE, Sep. 2012, pp. 36-41.
[8] iot.eclipse.org, Mosquitto, https://mosquitto.org/.
[9] iot.eclipse.org, Paho-mqtt, https://www.eclipse.org/paho/.
[10] ISO/IEC 20922:2016, Information Technology - Message Queuing Telemetry Transport (MQTT) V. 3.1.1, 2016.
[11] M.H. Klein, et al., Attribute-Based Architecture Styles, Proc. of WICSA1, Kluwer Academic Pub., Feb. 1999, pp. 225-243.
[12] oneM2M, Functional Architecture version 2.10.0, Aug. 2016.
[13] oneM2M, MQTT Protocol Binding version 2.4.1, Sep. 2016.
[14] J. Ren, et al., Serving at the Edge: A Scalable IoT Architecture Based on Transparent Computing, Proc. of MNET ’17, IEEE, Aug. 2017, pp. 96-105.
[15] L. Richardson, et al., RESTful Web Services, O’Reilly, 2007.
[16] W. Shi and S. Dustdar, The Promise of Edge Computing, IEEE Computer, Vol. 49, No. 5, May. 2016, pp. 78-81.
[17] S. Tarkoma, Publish/Subscribe Systems, Wiley, 2012.
[18] Y. Xu, et al., Fairness in Fog Networks: Achieving Fair Throughput Performance in MQTT-Based IoTs, Proc. of CCNC ’17, IEEE, Jul. 2017, pp. 191-196.