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

M2016SE002　濱野　真伍
指導教員　青山　幹雄

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 がある．エッジアーキテクチャはスケーラビリティなど多様な非機能要求がある．しかし，これらを満たすエッジアーキテクチャ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 点を研究課題とする

(1) 多様な非機能要求を実現するエッジアーキテクチャの設計方法.

(2) (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.

3 関連研究

3.1 エッジコンピューティング

エッジ[3,9]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の導入により高いスケーラビリティなどを実現するアーキテクチャである．エッジに処理を分散しデバイスのサービス実行の遅延を低減するアーキテクチャが提案されている[11].

3.2 Publish/Subscribe アーキテクチャ

Publish/Subscribe アーキテクチャ(Pub/Sub)はブローカを介し非同期でメッセージを配信するアーキテクチャである[10]．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，フィルタリングされたメッセージを受信する．IoT 向けの実装に ISO/IEC[8]で標準化された MQTT 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 サーバを統合したメッセージングブローカである．

3.3 品質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(AD 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の設計方法を提案する．

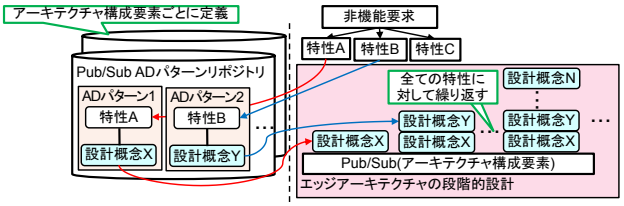

図 1 アプローチ

5 アーキテクチャ設計方法論

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 5 点を前提条件とする．

(1) 車両をデバイスとする.

(2) メッセージ配信には Pub/Sub を適用する.

(3) デバイス数は大量で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) デバイスは異なるエッジ間を移動する.

本例題のエッジへの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) メッセージの完全性の確保: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 にアーキテクチャ設計プロセス示す．提案する設計方法では 4 つのプロセスでアーキテクチャを設計する.

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，非機能要求を詳細化した特性の組み合わせ．各パターンは設計概念と同じ名前を持つ.

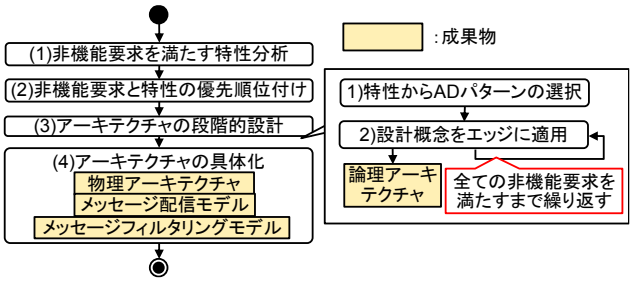

図 2 アーキテクチャ設計プロセス

AD パターンの集合を AD パターンリポジトリとし，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 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．

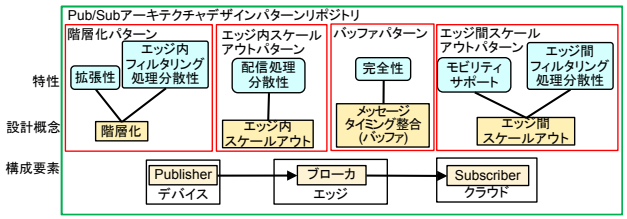

図 3 Pub/Sub AD パターンリポジトリ

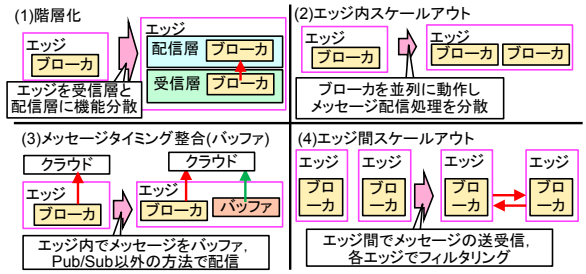

図 4 アーキテクチャ設計概念の詳細

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満たすために詳細化した性質を特性とする．特性はエッジアーキテクチャの構成要素に基づいて再帰的に詳細化する．特性は AD パターン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．非機能要求の実現は 1 段階詳細化した特性を全て実現することとする．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: 対象の特性の実現には，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．

(2) Or パターン: 対象の特性の実現には, 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．

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．

(1) スケーラビリティ: 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．スケーラビリティは 3 つの特性に詳細化した．

1) 拡張性: 1 つのエッジ内で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 の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:フィルタリング処理を分散する必要がある．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．

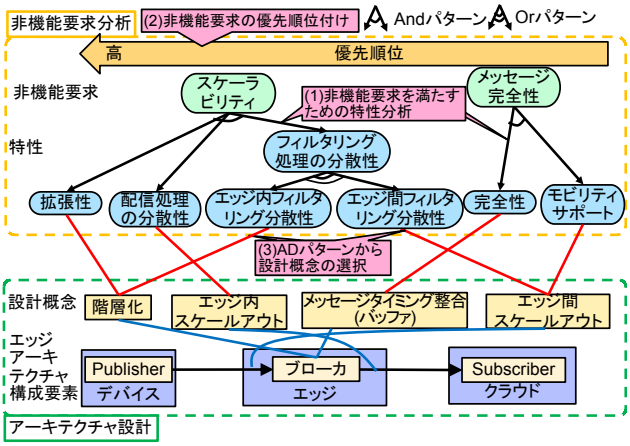

図 5 アーキテクチャ設計方法

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をする必要がある．

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フィルタリングをする必要がある．

(2) メッセージの完全性: デバイスから受信したメッセージを全て特定のエッジからクラウドに配信可能である状態とする．メッセージ完全性は 2 つの特性に詳細化した．

1) 完全性: エッジはメッセージを 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:エッジ間でメッセージ送受信を行い，特定のエッジからのメッセージ配信の必要がある．

5.5 非機能要求と特性の順位づけ

非機能要求をアーキテクチャに適用する優先順位をつける．この優先順位を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．

5.6 アーキテクチャの段階的設計

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．その後，AD パターンに従って 5.5 節で優先順位を定義した特性と設計概念を結ぶ．特性から設計概念を選択するため，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．本例題では 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 のアーキテクチャ設計方法に従った論理アーキテクチャの段階的設計プロセスを図 6 に示す．また，図 7 に本例題の論理アーキテクチャの段階的設計を示す．

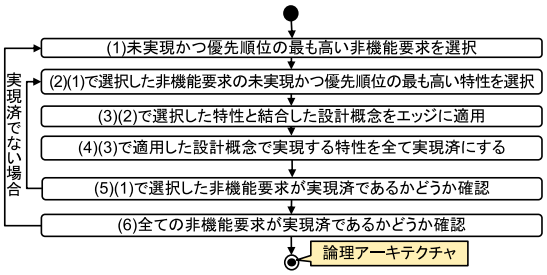

図 6 論理アーキテクチャの段階的設計プロセス

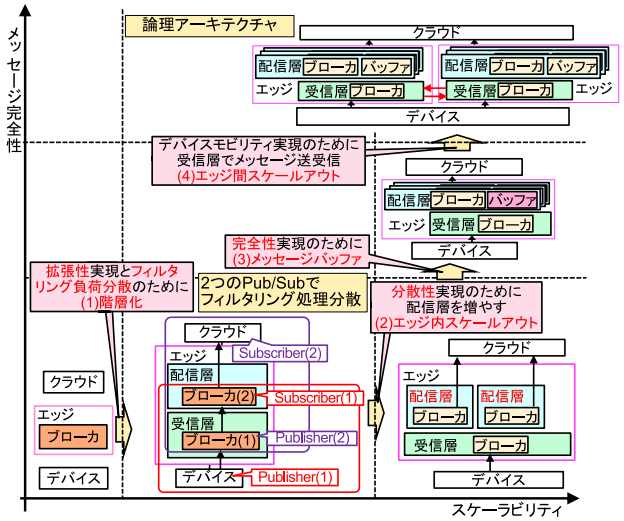

図 7 論理アーキテクチャの段階的設計

(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. 図 7 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. 階層化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.

(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 7 の(2)に示すエッジ内で配信層の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 つの設計概念の適用により，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になる.

(3) メッセージタイミングの整合(バッファ): 図 7 の(3)に示すように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メッセージをバッファする. これにより，エッジは Pub/Sub 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: 図 7 の(4)に示すように，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, 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

5.7.1. 物理アーキテクチャ

論理アーキテクチャから図 8 に示す物理アーキテクチャを設計する. 物理アーキテクチャでは Pub/Sub に MQTT, バッファにはドキュメント指向データベース(ドキュメント DB)と REST を用いる. 図 8 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. 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.

5.7.2. メッセージ配信モデル

エッジが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，クラウドは事前にエッジ A に Subscription を送信済みとする.

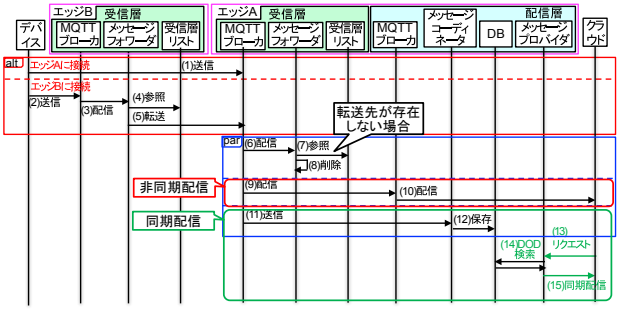

図 9 メッセージ配信モデル

デバイスがエッジ 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メッセージを 1 つの受信層から受信できる. 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，互いに影響を受けない.

6 プロトタイプの実装

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 1 に，ソフトウェアコンポーネントを表 2 に示す.

表 1 ハードウェアコンポーネント

デバイス	Raspberry Pi model B+
OS	Raspbian 9.3
ストレージ	microSD カード 8GB

表 2 ソフトウェアコンポーネント

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	Eclipse Mosquitto	1.4.9
MQTT クライアント	Eclipse 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

エッジに実装したアプリケーションを表 3 に示す.

表 3 実装アプリケーション

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

アーキテクチャ設計方法に従って具体化したアーキテクチャの非機能要求確認のために図 10 に示す 3 つのシナリオをそれぞれ 3 回実行した. デバイスは 2 分間に 20 ミリ秒間隔で計 6,000 個のメッセージをエッジに送信する.

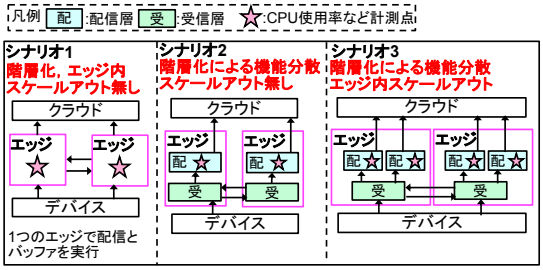

図 10 実行シナリオの概要

8 評価

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 に各シナリオの配信層における OS を除く CPU 使用率を示す．図 11 の吹出しはメッセージの受信を起点とし，10 秒後から 110 秒後の CPU 使用率の平均値を示す．

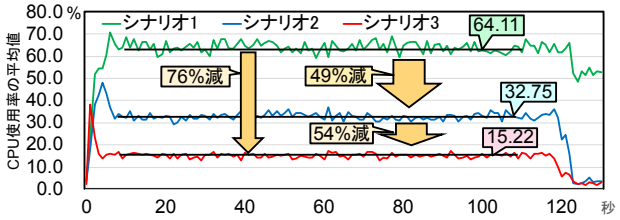

図 11 CPU 使用率

シナリオ 2 ではシナリオ 1 と比べ，階層化のフィルタリング処理を分散により CPU 使用率の 49%削減を確認した．シナリオ 3 はシナリオ 2 と比べ，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の約半減により CPU 使用率の 54%削減を確認した．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．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．

(2) ロードアベレージによる評価

図 12 に各シナリオの配信層における式 1 で定義したロードアベレージを示す．

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

シナリオ 1 は時間経過とともに増加しているため，起動

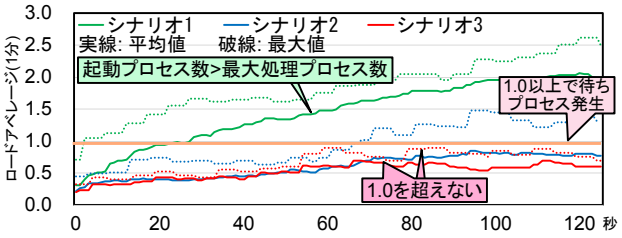

図 10 ロードアベレージ

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．シナリオ 2 とシナリオ 3 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．シナリオ 2 の最大値が 1.0 を超えることがあるが，シナリオ 3 ではでは 1.0 を超えることはないため，待ちプロセスが発生することはない．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．

8.2 メッセージの完全性の評価

各シナリオでクラウドが非同期配信，同期配信で受信したメッセージに重複は無く，デバイスで生成したメッセージと差異はなかったため，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．

9 考察

(1) 関連研究との性能比較

提案アーキテクチャと QEST ブローカ[7]の性能を非機能要求の視点で比較し，その結果を表 4 に示す．

表 4 関連研究との比較

視点	提案アーキテクチャ	QEST ブローカ
スケーラビリティ	＋＋	＋
メッセージ完全性	＋＋	－

表 4 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．また，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) ADD[2,5]との比較による有効性

1) エッジアーキテクチャの設計に有効: 提案方法はアーキテクチャ設計概念に基づき AD パターンを選択するため，エッジ固有の非機能要求を満たす設計が可能と考える．

2) アーキテクチャ設計の効率化: 提案方法は特性ごとの独立した設計概念を適用により，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と考える．

10 今後の課題

今後の課題は以下の 3 点である

(1) 提案するエッジアーキテクチャ設計方法の妥当性評価

(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がある場合の設計方法．

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を提案した．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設計の，複数の非機能要求の評価により，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] ISO/IEC 20922:2016, Information Technology - Message Queuing Telemetry Transport (MQTT) V. 3.1.1, 2016.

[9] W. Shi and S. Dustdar, The Promise of Edge Computing, IEEE Computer, Vol. 49, No. 5, May. 2016, pp. 78-81.

[10] S. Tarkoma, Publish/Subscribe Systems, Wiley, 2012.

[11] 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.