IoT 機器間通信の互換性を柔軟に保証するための ソフトウェアアーキテクチャの設計

2015SE098 横山史明　　2015SE089 臼井大輔
指導教員：沢田篤史

1　はじめに

IoT 機器の増加に伴い，スマートスピーカのような IoT 機器同士を連携させる機能を持つ製品も登場した．現在の IoT 機器間通信では HTTP や Constrained Application Protocol (CoAP)[1] などのプロトコルが混在しており，機器同士の連携を妨げている．これにより，ユーザの行動に合わせた連携ができない．

ユーザの行動に合わせた連携のためには, IoT 機器連携を動的に切り替えなければならない．また，IoT 機器連携のためはプロトコルの相違を吸収するアダプタの開発が必要となる．

本研究の目的はユーザの行動に合わせた連携と新機能の追加である．この目的の達成のために，IoT 機器連携のためのアプリケーション開発においてプロトコルの相違を考慮することなく IoT 機器の連携をさせる仕組みの作成と IoT 機器の送信先を動的に切り替えることを研究課題とした．これらの研究課題を IoT 機器間通信の互換性を柔軟にするソフトウェアアーキテクチャを設計することにより解決する．

IoT 機器間におけるプロトコル変換を行うために，アダプタを動的再構成するアーキテクチャを設計した．江坂らが提案した自己適応のための PBR(Policy Based Reconfiguration) パターン [7] を適用し，オブジェクト指向で実装するためにアーキテクチャの詳細をデザインパターン [2] の Mediator パターン，Adapter パターンと Abstract Factory パターンを用いて設計した．本研究で提案するアーキテクチャの妥当性を検証するために，コンテキストに応じて，アダプタの動的再構成と送信先 IoT 機器の変更をするソフトウェアのプロトタイプを実装した．検証により，コンテキストに応じて IoT 機器のメッセージの送信先を変更し，アダプタが再構成できることを確認した．

2　研究背景

2.1　既存技術

既存の IoT 機器同士の連携方法として，家電機器を家庭内のネットワークに接続することで機器同士の連携や遠隔操作を行うホームネットワークシステム (HNS) がある．これは，ECHONET Lite[5] を標準通信プロトコルとしており，ECHONET Lite に対応した機器同士の連携を行っている．HNS では高性能なホームサーバにより，全ての IoT 機器を制御する．ホームサーバから各機器へ制御メッセージを送信することで IoT 機器の連携を行っている．

2.2　関連研究

井垣らの研究 [4] ではネットワーク家電のサービスをネットワークに公開し，サービスを連携させることで家電の連携を行っている．サービス指向アーキテクチャを用いることで機器間が疎結合になり，相互接続性や機器拡張性を向上させている．各家電機器をデバイス層とサービス層で構成している．デバイス層は機器のハードウェア部分を指す．サービス層では，他の機器の機能を制御するために機器の制御インタフェースをラップし，サービスとしてネットワークに公開している．サービス層におけるアプリケーション開発では機器のベンダが公開メソッドの厳密な型定義を行う．一方，連携サービスのシナリオ作成はユーザが行う．

2.3　問題点

現在，IoT 機器が使用しているプロトコルは様々であり，異なるプロトコルを使用している機器との連携が不可能である．それにより，ユーザの行動に合わせた IoT 機器の連携ができない．プロトコルの異なる機器を連携させるようなアプリケーションを開発するためには，プロトコルの相違を考慮しなければならない．

3　IoT 機器制御のための機器間通信を動的再構成するソフトウェアアーキテクチャの設計

3.1　アーキテクチャ設計指針

本研究で扱う IoT 機器は以下の 2 つの条件を満たすものとする．

- IP アドレスをソフトウェアにより制御可能である．
- ソフトウェアインタフェースを変更可能である．

前者は IoT 機器にネットワークインタフェースが備わっていることから，今後の IoT 機器には自身のネットワークインタフェースを変更可能な性能が備わってくると考えられる．後者はスマートスピーカにソフトウェアインタフェースが備わっていることから，今後 IoT 機器はソフトウェアインタフェースも変更可能な性能が備わってくると考えられる．

本研究ではユーザの生活に合わせた IoT 機器の連携とプロトコルの変換を目的にアーキテクチャを設計する．ユーザの生活に合わせた連携のためにコンテキストを時間とユーザの場所としてアーキテクチャを設計する．プロトコルの変換のためのアダプタを動的再構成するために，自己適応のための PBR パターンを適用する．

このアーキテクチャをオブジェクト指向で実現するために，デザインパターンを用いて設計する．連携する IoT 機器を変更するために Mediator パターンを用いる．アダプタのインスタンスを生成するために Abstract Factory パターンを用いる．プロトコル変換のために Adapter パターンを用いる．

3.2 アーキテクチャ設計

プロトコル変換ポリシーにより, プロトコル変換アダプタファクトリがプロトコル変換のためのプロトコル変換アダプタを動的再構成するアーキテクチャを設計した．本研究では江坂が提案した IoT システムのための共通アーキテクチャの一部を変更した．

メディエータでは連携を制御するために連携している IoT 機器の IP アドレスの登録と変更を行う．全ての IoT 機器の IP アドレスをメディエータの IP アドレスとして追加することで IoT 機器からのメッセージをメディエータに送らせる．これによりメディエータが指定した送信先へメッセージを送信できる．

本研究で提案するアーキテクチャの静的構造と動的振舞いを**図 1**，**図 2** に示す．コンテキストを時間と位置とした．プロトコル変換ポリシーは IoT 機器の送信先とアダプタを決定する．プロトコル変換アダプタファクトリはポリシーに応じてアダプタのインスタンスを生成する．プロトコル変換アダプタはプロトコルの変換を行う．送信予定 IoT 機器は IoT 機器インタフェース，プロトコル変換アダプタと has-a 関係になっている．送信前 IoT 機器は IoT 機器インタフェースと has-a 関係になっている．位置，位置センサ，時間，時計，IoT 機器，IoT 機器インタフェース，プロトコル変換アダプタを Configuration とした．

コンテキストの変化によりプロトコル変換ポリシーが起動する．コンテキストに応じてプロトコル変換ポリシーはプロトコル変換アダプタファクトリへメッセージを送信する．プロトコル変換アダプタファクトリはそのメッセージに応じて，プロトコル変換アダプタを活性化させる．それにより，実際に変換するプロトコル変換アダプタが変更され，Configuration が再構成される．受信したメッセージをメディエータがプロトコル変換アダプタでプロトコルを変換する．送信先の IoT 機器へ変換したメッセージを送信する．

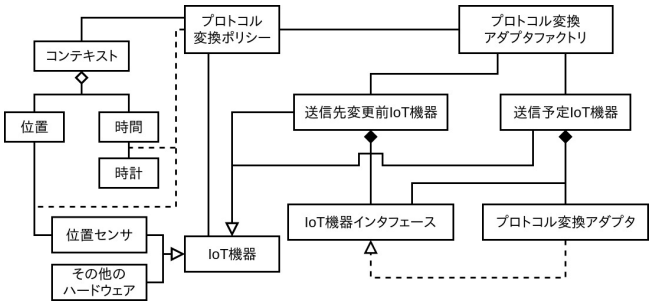

図 1　アーキテクチャ静的構造

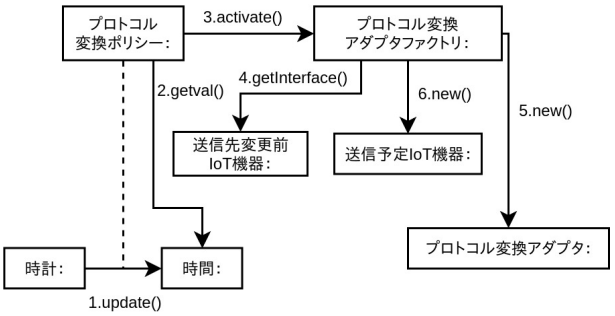

図 2　アーキテクチャ動的振舞い

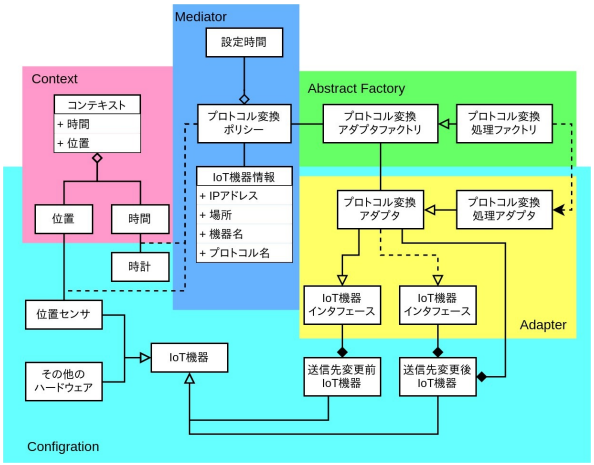

図 3　詳細アーキテクチャ

3.3 詳細アーキテクチャ

本研究で提案するアーキテクチャの詳細を**図 3** に示す．

3.3.1 Mediator 詳細

プロトコル変換ポリシー，設定時間，IoT 機器情報を Mediator パターンに適用した．設定時間は通信先の IoT 機器を変更する時間を保持する．IoT 機器情報クラスは IoT 機器の識別，正確な場所とプロトコル名が必要なので，IP アドレス，家の場所，機器名，プロトコル名を保持する．IoT 機器は同一型番の機器が同一の場所に存在する可能性があるので，識別が必要である．また，プロトコル変換アダプタファクトリへ送信元，送信先のプロトコルを通知する必要があるので，プロトコル名も必要である．

3.3.2 Abstract Factory 詳細

プロトコル変換アダプタファクトリとプロトコル変換処理アダプタを Abstract Factory に適用した．プロトコル変換アダプタファクトリとプロトコル変換処理ファクトリを is-a 関係にした．プロトコル変換アダプタファクトリはプロトコル変換ポリシーから，活性化するアダプタが記述されたメッセージを受け取り，それに応じてプロトコル変換処理アダプタのインスタンスを活性化する．これにより，Configuration が Updated Configuration へ再構成される．

3.3.3 Adapter 詳細

プロトコル変換アダプタ，プロトコル変換処理アダプタと IoT 機器インタフェースを Adapter パターンに適用し，送信されたメッセージをプロトコル変換アダプタでプロトコル変換する．IoT 機器インタフェースは，IoT 機器自身のインタフェースである．プロトコル変換アダプタとプロトコル変換処理ファクトリのインスタンスであるプロトコル変換処理アダプタを is-a 関係にした．プロトコル変換処理アダプタは，送信元 IoT 機器のプロトコ

ルから送信先 IoT 機器のプロトコルへ変換する．プロトコル変換処理アダプタはプロトコル変換メソッドとメッセージを IoT 機器へ送信するメソッドをプロトコル変換アダプタのメソッドをオーバーライドすることで，多相性を実現し，複数のアダプタを同じ制御で扱うことができる．

4 事例検証

4.1 検証方法

提案したアーキテクチャに基づき，コンテキストに応じてアダプタの動的再構成と送信先 IoT 機器の変更をするソフトウェアのプロトタイプ実装を Raspberry Pi 3[3] を用いて行った．PC(a)，PC(b)，Raspberry Pi 3(a)，Raspberry Pi 3(b) の 4 台を使用する．PC(a) を送信元 IoT 機器とし，ソケット通信を行うクライアントにした．PC(b) をメディエータとした．Raspberry Pi 3(a) を送信先 IoT 機器 (a) とし，ソケット通信を行うサーバにした．Raspberry Pi 3(a) を送信先 IoT 機器 (b) とし，HTTP 通信を行うサーバにした．

下記のシナリオで検証を行った．

1 IoT 機器の送信先をメディエータへ変更

- 1.1 クライアントをメディエータ，サーバーを送信先 IoT 機器 (a) にして IP アドレスでソケット通信を行う．
- 1.2 メディエータから送信先 IoT 機器 (a) へ新規 IP アドレスを送信する．
- 1.3 送信先 IoT 機器 (a) の IP アドレスを変更する．
- 1.4 メディエータに送信先 IoT 機器 (a) の変更前の IP アドレスを追加し，変更後の IP アドレスを保持する．

2 アダプタの動的再構成

- 2.1 19 時から 20 時に変化する．
- 2.2 プロトコル変換ポリシーが起動し，送信先を送信先 IoT 機器 (b)，アダプタを HTTP アダプタに決定する．
- 2.3 プロトコル変換ファクトリはポリシーからのメッセージに応じて HTTP アダプタのインスタンスを生成する．

3 IoT 機器同士の連携

- 3.1 送信元 IoT 機器はソケット通信で “get” というメッセージを送信する．
- 3.2 メッセージをメディエータが受け取り，HTTP アダプタでプロトコルの変換を行い，メッセージを送信する．
- 3.3 送信先 IoT 機器 (b) がメッセージを受信する．

代替シナリオ

2a アダプタの動的再構成

- 2a.1 時間が 9 時から 10 時に変化する．
- 2a.2 プロトコル変換ポリシーが起動し，送信先を送信先 IoT 機器 (a)，アダプタを socket アダプタに決定する．
- 2a.3 プロトコル変換ファクトリはポリシーからのメッセージに応じて socket アダプタのインスタンスを生成する．

3a IoT 機器同士の連携

- 3a.1 送信元 IoT 機器はソケット通信で “get” というメッセージを送信する．
- 3a.2 メッセージをメディエータが受け取り，socket アダプタでメッセージを送信する．
- 3a.3 送信先 IoT 機器 (a) がメッセージを受信する．

4.2 検証結果

実装したソフトウェアは上記のシナリオ通りに動作した．IoT 機器の送信先をメディエータに変更した時の実行結果を**図 4**，**図 5** に示す．**図 4** はクライアントの実行結果である．サーバの IP アドレスを変換し，クライアントにサーバの変更前の IP アドレスを追加している．**図 5** はサーバの実行結果である．クライアントから指定された IP アドレスを表示し，その IP アドレスに変更している．

IoT 機器同士の連携の結果を**図 6**，**図 7**，**図 8**，**図 9** に示す．送信元の IoT 機器の実行結果を**図 6** に示す．メディエータにメッセージを送信できたことを確認できた．メディエータの実行結果を**図 7** に示す．現在時間に応じて送信先の決定とアダプタの生成を行っている．送信先を決定しソケット通信を行うことを確認できた．送信元 IoT 機器から “get” というメッセージが受信できたことを確認できた．送信先の IoT 機器の実行結果を **図 8** に示す．メディエータから “get” というメッセージを受信できたことを確認できた．コンテキストによって送信先が変更した場合のメディエータの実行結果を**図 9** に示す．現在時間に応じて送信先の決定とアダプタの生成を行っている．送信先を決定し http 通信を行うことを確認できた．送信元のメッセージを HTTP の GET メソッドに変換し，送信先 IoT 機器へ実行した．

```
15se098@2015-pc:~/IPChangeClient$ java IPChangeClient
アクセス : 成功                          ← IoT機器へIPアドレスを送信し，
[sudo] password for 15se098:
15se098@2015-pc:~/IPChangeClient$ ■      IoT機器のIPアドレスを自身に追加
```

図 4 IP アドレス変更時のメディエータの実行画面

```
pi@raspberrypi:~/IPChangeServer $ java IPChangeServer
クライアントからの接続を待ちます。
クライアント接続。
受信： 10.48.129.100   ← メディエータからIPアドレスを受け取り，
                          自身のIPアドレスを変更する．
```

図 5 IP アドレス変更時の送信先 IoT 機器の実行画面

```
15se089@2015-pc:~/socketclient$ java SocketClient
アクセス : 成功                    ← 送信が成功
15se089@2015-pc:~/socketclient$ ■
```

図 6 送信元 IoT 機器の実行画面

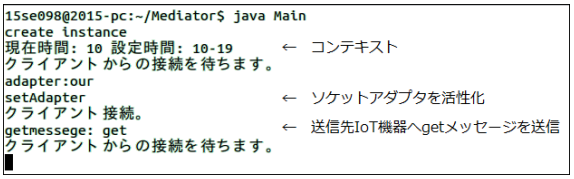

図 7 ソケットアダプタを活性化したメディエータの実行画面

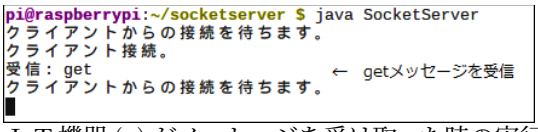

図 8 IoT 機器 (a) がメッセージを受け取った時の実行画面

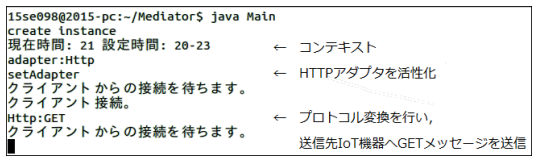

図 9 HTTP アダプタを活性化したメディエータの実行画面

5 考察

5.1 提案したアーキテクチャの妥当性の考察

コンテキストに応じた送信先の変更と，再構成されたアダプタでプロトコル変換を行うことができた．プロトコル変換を可能にしたことにより，プロトコルの相違を考慮することなくアプリケーション開発が可能になる．また，PBR パターンを用いないでアダプタを動的再構成する場合，再構成する方針を一か所で管理しなければ，制御が複雑になり保守性が低くなる．送信先の変更によりユーザの実際の生活に合わせて連携も可能である．以上のことから，本研究の目的が果たされ，我々のアーキテクチャは妥当であると考えられる．

5.2 関連研究との比較

井垣らの研究で提案されているアーキテクチャと本研究で提案したアーキテクチャの比較を行い，利点を考察する．我々の研究では，通信プロトコルの変換を行い連携させるので，より多様な機器との連携が可能である．我々の提案するアーキテクチャではプロトコルレベルの変換だけでなくメッセージの意味レベルの変換も可能である．それに加えてコンテキストに応じた送信先の変更とアダプタの再構成が可能なので，よりユーザの生活に合わせた機器同士の連携が可能である．

5.3 提案したアーキテクチャの実用にむけての考察

現実的な事例を考えた場合，メッセージを分割したり，統合したりするような複雑プロトコルが存在する．我々の提案するアーキテクチャでは，それらに対応するアダプタの作成によりメッセージ自体を分割したり，統合したりできることからこのような複雑なプロトコルにも対応可能である．

アダプタの再構成により，事前に記述すればどのようなプロトコルにも対応可能な仕組みを設計した．これにより，IoT 機器連携のためのアプリケーション開発において，プロトコルの相違を考慮することなく IoT 機器同士の連携を可能になる．

6 おわりに

本研究では，ユーザの行動に合わせた IoT 機器同士の連携と新機能の追加のために，IoT 機器間通信の互換性を柔軟にするソフトウェアアーキテクチャを設計した．動的再構成のために，自己適応のための PBR パターンに適用した．コンテキストに応じて送信先を変更するために Policy で送信先を変更した．オブジェクト指向で実装するために，Mediator パターン，Adapter パターンと Abstract Factory パターンを用いて設計した．設計したアーキテクチャの妥当性を検証するために事例による検証と関連研究との比較をした．

本研究で提案したアーキテクチャでは，プロトコルの増加につれてプロトコル変換アダプタファクトリに記述するプロトコル変換クラスが増加する．我々のアーキテクチャは静的な構造を前提としているので，プロトコル変換クラスの構造が複雑になる．また，プロトコル変換アダプタファクトリが生成するアダプタの数も増加する．我々のアーキテクチャは動的な振る舞いも前提としているので，アダプタの制御も複雑になる．そこで，同一のプロトコルに変換するプロトコル変換クラスを管理するクラスや生成されたアダプタを管理するクラスを作成することで，複雑性が緩和すると考えられる．

本研究で作成したプログラムは単純であり，実用可能な段階ではない．現実的な事例に対応可能なプログラムを記述し，実用性について検証することが必要である．

参考文献

[1] Bormann, C.: CoAP — Constrained Application Protocol — Overview, `http://coap.technology/`, 2019.

[2] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M.: *Design Patterns: Elements of Reusable Object-Oriented Software* , Addison-Wesley, 1994.

[3] Raspberry Pi Foundation: Raspberry Pi — Teach, Learn, and Make with Raspberry Pi, `https://www.raspberrypi.org/`, 2019.

[4] 井垣宏，中村匡秀，玉田春昭，松本健一: サービス指向アーキテクチャを用いたネットワーク家電連携サービスの開発，情報処理学会論文誌，Vol. 46，No. 2(2005)，pp. 314-326.

[5] エコーネットコンソーシアム: ECHONET Lite 規格の特徴と概要—ECHONET, `https://echonet.jp/about/features/`, 2018.

[6] 江坂篤侍: 自己適応を目的としたソフトウェアアーキテクチャの構築と運用に関する研究，南山大学 数理情報研究科 数理情報専攻 博士後期課程 博士論文，(2018).

[7] 江坂篤侍，野呂昌満，沢田篤史: インタラクティブシステムのための共通アーキテクチャの設計，コンピュータソフトウェア，Vol. 35，No. 4(2018)，pp. 3-15.