IoT アプリケーションの柔軟な構成変更のための ポリシ記述方式の提案

2017SE023 石原渉太郎　2017SE104 山本凌司
指導教員：沢田篤史

1　はじめに

近年 IoT 家電が普及しているが，相互運用性と柔軟性が確保されているのは，特定の製品群の中だけに留まっている．相互運用性が保証された製品群でスマートホーム化する方法以外では，スマート家電を有効活用することが難しい．柔軟性においても，特定のアプリケーション内でのみ利用者の状況を考慮することができるが，それより広い範囲で考慮するのは難しい．上述の問題に対処するために，インターネットを繋いだ IoT 機器のために相互運用性，柔軟性を保証するアーキテクチャが提案されている [1]．既存のアーキテクチャでは，アダプタの動的構成により相互運用性，柔軟性を確保する仕組みが提案されている．そこでは，コンテキストに応じた切り替えや再構成を実施でき，必要なプログラムを提供されるライブラリを利用して記述する必要がある．さらに既存のアーキテクチャを利用する際に構成変更のための要素は，ポリシに記述される．しかし，ポリシはプログラミング言語を使用して記述する必要があり，利用者が容易に設定できない．

本研究の目的は，利用者が IoT 機器の構成要素を書くための記述方式を考案し，記述方式の解釈実行系を既存のアーキテクチャに組み込むことである．表形式でポリシ記述を行えるようにすることで，利用者が簡便に構成変更を行うことが可能となる．ポリシ記述の解釈と評価のためのコンポーネントを実現するために，Interpreter パターン [2] を用いた．Interpreter パターンを用いて実現される解釈コンポーネントは表形式のポリシ記述を一行ずつ評価するので，プログラムの動作をすぐ確かめることができる．また，利用者が記入した表を一行ずつ変換と実装を逐次的に繰り返し行い処理を進めていき，文法的なミスや誤字があるとプログラムがストップするので，エラーが発生した個所を特定しやすい．さらに，表形式を適用することで利用者が IoT 機器の関係を理解しやすく，保守や改良，カスタマイズを行うことができる．

本研究で提案したポリシ記述を解釈実行するためのプロトタイプシステムの実装を行った．利用者が記入した表により柔軟な構成変更を行うことを示し，既存のアーキテクチャに組み込むことで妥当性を確認した．

2　IoT アプリケーションにおける相互運用性と柔軟性に関する課題

2.1　スマートホーム IoT アプリケーション

IoT 機器やセンサーを使用した住居であるスマートホームが増えており，IoT 機器の相互運用性を保証する既存研究がある．

井垣ら [3] はソフトウェアの機能をサービスと見立て，そのサービスをネットワーク上で連携させてシステムの全体を構築していく SOA（Service-Oriented Architecture）を用いて，IoT 機器をサービス層で結合することで，機器間の相互接続性が向上する手法を提案した．

山田ら [4] は情報家電エージェントを使用し，スマート家電にエージェントを配置して相互運用性を保証する家電協調アーキテクチャを提案した．

いずれの研究においても柔軟性については，特定のアプリケーション内でのみ利用者の状況を考慮することができるが，それ以外のアプリケーションを利用した際に考慮するのが難しい．保守性が高く，柔軟で相互運用可能な IoT アプリケーションを開発するためには，これらの要素技術の統合を系統的に支援するための共通基盤が必要である．この共通基盤が横山らが提供したソフトウェアアーキテクチャである．

2.2　相互運用性と柔軟性を保証する既存のソフトウェアアーキテクチャ

横山らは，相互運用性と柔軟性の高いスマートホームアプリケーションのための共通基盤の提供を目的としたソフトウェアアーキテクチャを提案している [1]．

相互運用性については，3 つのレベル (ネットワーク接続，メッセージングプロトコル，意味) で変換できるようにアダプタパターンを適用している．データや制御をやり取りする IoT 機器にアダプタを関連付け，必要なプロトコル等の変換のやり取りを担わせることができる．

柔軟性については，PBR パターン [5] を適用し，コンテキストに応じた動的再構成を行うことで解決を試みている．PBR パターンは，Application Component，Policy，Configuration Builder，Configuration の要素から構成される．静的構造では，ポリシと再構成の仕組みをそれぞれコンポーネント化する．動的振舞いでは，コンポーネント間のメッセージ通信により動的再構成を実現する．Application Component は，特定のコンサーンによって規定されるコンポーネントである．Policy は，動的再構成するための方針を記述するためのコンポーネントである．Configuration Builder は，Configuration を再構成するための記述をするコンポーネントである．Configuration は，再構成後の Application Component 群のコンポーネントである．再構成の仕組みとして，Application Component 間のメッセージ通信をきっかけに，Policy がメッセージを横取りし，Configuration Builder に Configuration のイ

ンスタンスを生成させ，そのインスタンスにメッセージを送信する．Configuration Builder は，Policy からのメッセージに応じて，Configuration を再構成する．

動的適応のための基本構造を図 1 に示す．この基本構造

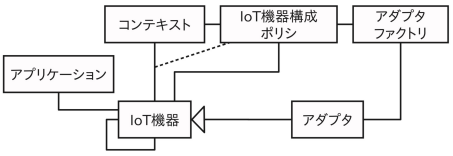

図 1　動的適応のための基本構造 [1]

は，IoT 機器クラス，アプリケーションクラス，コンテキストクラス，IoT 機器構成ポリシクラス，アダプタファクトリクラス，アダプタクラスから構成される．コンテキストクラスは，コンテキスト情報を管理する．IoT 機器構成ポリシクラスは，コンテキストクラスを元に IoT 機器を動的再構成するための方針を記述する．アダプタファクトリクラスでは，アダプタを生成する．アダプタクラスは，ネットワーク接続，メッセージングプロトコル，意味の 3 つのレベルの変換を行うための記述をする．これにより，人感センサーなどの IoT 機器から取得したコンテキストの変化に応じて，動的に IoT 機器を再構成する構造を定義した．

2.3　既存アーキテクチャの問題点

IoT 機器を切り替えるためにコンテキスト（人の位置）を判断するソースコードの if 文は，複数回記述する必要がある．IP アドレスやメッセージングプロトコルなどの変換のためのアダプタを生成する処理やファクトリに再編成のメッセージを送信する処理などが if 文の重複した (入れ子状) ソースコードにより複雑になる．これらから，ポリシの記述が複雑になってしまうことが明らかであり，問題点といえる．また，利用者が IoT 機器を利用する際に，通知先を変更する役割としてのポリシをカスタマイズすることが想定されている．しかし現状では，ポリシをプログラミングする必要があり一般の家電利用者が IoT 機器を利用する際に，通知先を変更する役割としてのポリシを容易に設定することは難しい．

本研究では，利用者が IoT 機器の構成変更を書けるような記述方式について検討する．これにより，プログラミングの知識をもたない一般の家電利用者でも，ポリシを記述し保守や改良ができるようになり，横山らのスマートホーム IoT アプリケーションのアーキテクチャにおけるポリシの記述方法をより簡便にすることを検討する．

3　相互運用性と柔軟性を確保するためのポリシの記述の提案

3.1　表形式によるポリシ記述

本研究では，表形式を適用することで利用者が IoT 機器の関係性を理解しやすく，保守しやすくなり要求を満たすことができると考えた．また表形式を適用し IoT 機器，コンテキスト（人の位置），機能，製品コードの 4 つの要素の組み合わせから，ポリシの記述を行い，必要なアダプタのインスタンスを，個々の IoT 機器と通知の出力方法を識別できる名前で生成することで，ポリシの記述も容易になる．さらに表形式にすることの利点として，新しい IoT 機器の追加，故障や使わない IoT 機器の削除が容易であること，現在連携している IoT 機器が常に見やすく理解しやすいことが挙げられる．通知を発信する IoT 機器は，機器名と製品コードを表に入力する．通知を出力する IoT 機器の入力の具体例を表 1 に示す．IoT 機器は，通知を出力させたい機器名を入力する．コンテキスト（人の位置）は，通知を出力させたい IoT 機器がスマートホームのどこに置かれており，どの人感センサーが感知したときにその IoT 機器に出力させたいのかを明確にするために入力する．機能は，その IoT 機器にどのように通知を出力させたいのか，通知の出力方法を明確にするために入力する．製品コードは，スマートホームに同じ IoT 機器が 2 台以上ある場合に，その IoT 機器を正確に特定し通知を出力するために入力する．

表 1　通知を出力する IoT 機器

IoT機器	コンテキスト（人の位置）	機能	製品コード
スマートスピーカー	居間	音声	ABC
テレビ	居間・視聴時	文字	DEF
テレビ	寝室・視聴時	文字	GHI
給湯器	風呂	音声	JKL
コンピュータ	居室	文字・音声	MNO
照明	寝室	点灯させる	PQR
スマホ	外出時・見当たらない	文字・音声	STU
自動車	運転時	音声	VWX

3.2　アダプタ名（インスタンス名）の決定方法

横山らのスマートホーム IoT アプリケーションのアーキテクチャにおいて通知機能を変換する機能変換器，メッセージ形式を変換するメッセージングプロトコル変換機，IP アドレスとポートを変換するネットワークプロトコル変換機の 3 つの変換機能が搭載されているアダプタが必要である．そのアダプタ名の決定方法を考える．あらかじめ全ての通知の出力方法に番号を振ることにする．番号を振ることで，プログラミング言語の知識がない利用者でも理解しやすく，保守しやすいと考えた．IoT 機器の頭文字を取り，出力方法に応じてその番号で挟む．コンテキスト（人の位置）の頭文字は，出力する IoT 機器の頭文字の前に-で繋ぐことで分かりやすくする．

4　ポリシ記述を解釈実行するためのプロトタイプシステムの設計と実装

4.1　システムの設計

既存のアーキテクチャのポリシ記述に着目してシステムを開発した．ポリシ記述の解釈と評価のためのコンポー

ネントを実現するために，Interpreter パターンを用いた．システムの基本構造を図 2 に示す．本研究の基本構造は，

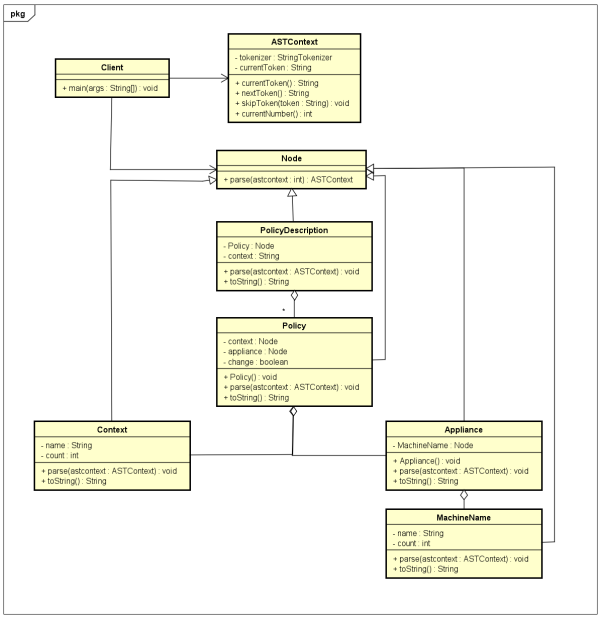

図 2　Interpreter パターンの基本構造

Client クラス，Node クラス，ASTContext クラス，PolicyDescription クラス，Policy クラス，Appliance クラス，MachineName クラス，Context クラス，ParseException から構成される．Client クラスは，動作テストを行う．Node クラスは，構文木の各部分 (ノード) を構成する最上位のクラスである．Node クラスは抽象メソッド parse だけが宣言されており，この parse は構文解析を行うメソッドである．ASTContext クラスは，構文解析のために必要なメソッドを提供する．PolicyDescription クラスは，複数のポリシを格納する．コンテキストに応じてポリシ記述が変化するので，複数のポリシを保持している．Policy クラスは，Context クラスと Appliance クラスを元に IoT 機器を動的再構成するための方針を記述する．Context クラスは，表のコンテキスト部を解釈し，入力したコンテキストが現在のコンテキストと一致するかを判別する．そして，現在のコンテキストと一致すれば値を返す．Appliance クラスは MachineName クラスで返された IoT 機器の名前を解釈し，格納する．MachineName クラスは表の IoT 機器名部を解釈し，コンテキストに応じて送信先となる IoT 機器の名前を返す．ParseException クラスは，構文解析中の例外のためのクラスである．

これにより，コンテキストに応じて利用者が記入した表から送信先となる IoT 機器に構成変更を行う構造を定義した．

4.2　全体の実装

実行結果を図 3 に示す．本研究は，Raspberry Pi を用いてコンテキストの値を定めた．Client クラスで利用者が記入した表を読み込み，一行ずつ構文解析し，その結果を

```
<終了 > Main [Java アプリケーション] C:¥Users¥
text = " コンテキスト IoT機器"
node = [[IoT機器 コンテキスト]]
text = " 居間 スマートスピーカー"
node = [[スマートスピーカー 居間]]
text = " 居間・視聴時 テレビ"
node = [[テレビ 居間・視聴時]]
text = " 寝室・視聴時 テレビ"
node = [[テレビ 寝室・視聴時]]
text = " 風呂 給湯器"
コンテキスト : 風呂
送信先が 給湯器 へ変更されました。
node = [[給湯器 風呂]]
text = " 居室 コンピュータ"
node = [[コンピュータ 居室]]
text = " 寝室 照明"
node = [[照明 寝室]]
text = " 外出時・見当たらない スマホ"
node = [[スマホ 外出時・見当たらない]]
text = " 運転時 自動車"
node = [[自動車 運転時]]
```

図 3　実行結果

表示している．表示の中で，text = で始まっている部分が利用者が記入したコンテキスト，IoT 機器の名前で，node = で始まっている部分が構文解析後の表示である．本研究の Interpreter パターンを用いた基本構造では，利用者がソースコードに触れずに表にコンテキストや送信先となる IoT 機器の名前を追加するだけで構成変更を行うことができるので，このようなエラー処理の方法を考案した．これにより，利用者が柔軟な構成変更を行うことができる．本研究で行ったプロトタイプシステム実装の範囲では，コンテキストと通知先の IoT 機器の名前だけで構成変更を行い，アダプタの生成までは実現できなかったが，IoT 機器の繋ぎ先を切り替えることは可能となった．これより本研究の提案は妥当であったと考える．

5　考察

本研究では，横山らのアーキテクチャにおけるポリシ記述を改良するために，表形式のポリシ記述を Interpreter パターンで解釈実行する方法を提案した．Interpreter パターンは，複雑な構文には適応できないといわれているが，本研究で提案した構文は表が 3 つとシンプルであり，複雑化することはないので，実装可能性は十分にある．Visitor パターンや我々が作成した CSV ファイルを直接配列に読み込んで検索する方法は，Interpreter パターンと比較して効率が良い．

一方で Interpreter パターンでは，CSV ファイルを指定するだけではなく，IoT 機器からメッセージが送信され IoT 機器を繋ぎなおす際に，ポリシ記述を追加していくことができる利点がある．また，Visitor パターンでは，表のコンテキストが現在のコンテキストと一致しない場合や，送信先となる IoT 機器の名前が登録されていない場合，ソースコードにコンテキストや送信先となる IoT 機器の名前を追加する必要がある．しかし，本研究はプログラミング知識のない家電利用者もポリシ記述をカスタマイズすることが可能となることが目的なので，Interpreter パターンを用いることが妥当だと考えた．

ポリシの記述に表形式を用いることで，ポリシをプログラムに埋め込まれる形で実現せざるをえない状況に対して，プログラミング言語の知識がない一般の家電利用者も

簡便にポリシの記述を書くことを可能にした．

提案したポリシ記述を解釈・評価するためのプロトタイプシステムでは，ポリシ記述の一部を利用して IoT 機器を切り替えることができた．実装した範囲では，接続先の IoT 機器を識別することができたことから，提案したポリシ記述は妥当であると評価できる．

また，スマートホーム化を行うための様々な web サービスやデバイスが提案されている．相互運用性，柔軟性を保証するという観点から本研究と比較する．IFTTT[6] とは，スマートフォン上のアプリやインターネット上のサービス等を連携させ，一連の作業を自動化するサービスである．IFTTT では事前にトリガー (起動条件) とアクション (実行内容) を事前に登録すると，サービス同士の連携を自動で行う．スマートフォンでアカウントを登録するだけで利用可能となるので，柔軟性を考慮していると言える．ただ，一つのトリガーで複数のアクションを登録する場合，複雑となる．本研究では表を書き換えるだけで複数の構成変更を行うことができるので，優れていると言える．

SwitchBot[7] とは，すべてのスイッチとボタンを機械的に制御する IoT 機器である．スマートフォンからの遠隔操作で，従来の壁スイッチや家電のスイッチを物理的に押して，様々な設備をネットを通じて操作することができる．SwitchBot 専用のアプリをスマートフォンに追加するだけで誰でも利用することができる．ポリシに相当する再構成論理は，専用のアプリケーションでのみ可能という問題がある．ただ，スマートフォン上で GUI を用いて連携条件の指定が容易にできるので柔軟性を考慮していると言える．本研究では，表にコンテキストと送信先となる IoT 機器名を書くだけで構成変更ができ，表形式を用いることで送信先となる IoT 機器名の追加や削除も容易であることから，柔軟性を考慮していると言える．

Eclipse sensiNact[8] とは，異なるデバイスが IoT 上で情報を交換し，互いに相互運用できる環境を作ることを目的としている．本研究では，相互運用性に関しては，利用者が表に送信先となる IoT 機器名を記入することにより，二つ以上の IoT 機器が構成変更を行うという情報交換ができ，利用可能である．これにより，相互運用性が保証された製品群以外でも構成変更を行うことができるので，相互運用性を保証していると言える．Eclipse sensiNact では，拡張可能なデータモデル (センサーデータ，アクション，状態変数，プロパティの 4 つのリソース) を採用することで，相互運用性を確保し，本研究では，表形式にアダプタの 3 つのレベル (機能・メッセージングプロトコル・ネットワークプロトコル) を記入することにより，相互運用性を確保する．

6 おわりに

本研究では，利用者が IoT 機器の構成要素を書ける記述方式を考案し，記述方式の解釈実行系を既存のアーキテクチャに組み込んだ．表形式でのポリシ記述方法を検討した．表形式でポリシ記述を行えるようにすることで，新しい IoT 機器の追加，故障や使わない IoT 機器の削除が容易になり，保守や改良，カスタマイズが可能となる．これにより，利用者が簡便に構成変更を行うことが可能となる．また，既存のアーキテクチャに組み込むことで柔軟な構成変更が可能となり，既存のアーキテクチャがさらに利用者にとって扱いやすくなる．

Interpreter パターンを用いることで，一行ずつ実行を行うので，プログラムの動作をすぐ確かめることができる．さらに，利用者が記入した表を一行ずつ変換と実装を逐次的に繰り返し行い処理を進めていき，文法的なミスや誤字があるとプログラムがストップするので，エラーが発生した個所を特定しやすい．これらから，Interpreter パターンを使用する妥当性を示した．

今後の課題として，本研究で提案したポリシ記述の提案を組み込んだ既存のアーキテクチャが実装するか確認する必要がある．組み立み妥当性は確認されたが実際に動かす必要がある．また，本研究で提案したアダプタの生成は実現できなかったので，アダプタの生成の今後考えていく必要がある．また，一つのコンテキストに全く同じ機器が複数個ある場合に対して，製品コードなどを用いて構成変更を行う仕組みを考える必要がある．

参考文献

[1] 横山史明，沢田篤史， 野呂昌満， 江坂篤侍，“IoT の柔軟な相互運用性を実現するソフトウェアアーキテクチャの提案”，ソフトウェア工学の基礎 XXVI(日本ソフトウェア科学会 FOSE2019)，pp. 93-102，2019.

[2] Gamma, E., Richard, H., Johnson, R., and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1995.

[3] 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，pp. 314-326，2005.

[4] 山田知秀，飯島正，山口高平，“オントロジーを利用した情報家電エージェント協調アーキテクチャ”，第 19 回人工知能学会全国大会, pp. 1B2-03，2005.

[5] 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, pp. 3-15, 2018.

[6] IFTTT help every thing work better together, https://ifttt.com.（Accessed 2020.12.22）

[7] SwitchBot(スイッチボット) とは，https://switchbot.shop.（Accessed 2020.12.22）

[8] Eclipse sensiNact — projectseclipse.org, https://projects.eclipse.org/projects/technology.sensinact.（Accessed 2020.12.22）