IoT アプリケーションのためのコンテキスト指向ソフトウェアアーキテクチャに関する研究

M2021SE003 本田一輝
指導教員：野呂昌満

1 はじめに

近年，IoT アプリケーションは，コンテキスト指向で協調するマイクロサービス群として構成する方法が一般的となってきている [1]．IoT アプリケーションにおいて，多種多数なデバイスやサービスを協調させるために，コンテキスト指向で定義する必要がある．協調するマイクロサービス群は，特定のコンポーネントにおいてサービスとして定義され，他のマイクロサービス群と連携させる構造を定義できる．アプリケーションの開発においては，システム全体の構造を明らかにした上で具体化を行う必要がある [2]．すなわち，アプリケーションアーキテクチャを設計し，評価を行うことが必要となる．一方で，IoT アプリケーションのアーキテクチャには，コンテキスト指向，サービス指向，サービスのマッシュアップなどの技術を統合した例は未だない．

本研究の目的は，これらの技術を統合したソフトウェアアーキテクチャを定義することである．多種多様な IoT アプリケーションに利用可能な構造を定義し，アーキテクチャを中心に IoT アプリケーション開発をするための枠組みを整理する．アーキテクチャの実現を考え，実行効率および記述性の向上のために並行処理記述を利用する方法を提案する．サービス指向を前提としているので，サービスのマッシュアップ技術について考察する必要がある．サービスのマッシュアップにおいて，Business Process Execution Language（以下，BPEL）などのビジネスプロセス言語を利用するのではなく，並行処理記述の利用を試みる．研究課題は以下の通りである．

RQ1 IoT アプリケーションのためのアーキテクチャ定義

RQ2 並行処理記述によるサービスのマッシュアップに関する考察

RQ3 事例検証による提案アーキテクチャの有用性と並行処理記述の妥当性検討

2 IoT アプリケーションのためのアーキテクチャ

2.1 IoT のリファレンスアーキテクチャ[3]

図 1 に WSO2 が提案する IoT のリファレンスアーキテクチャを示す．IoT のリファレンスアーキテクチャは，システムとデバイス間の統合をサポートするために提案されている．OSI 参照モデルに基づいて，コンポーネント間の通信について階層構造で整理している．それに対する横断的関心事として，アスペクト指向の概念に基づいて横断する層でデバイスマネージャとセキュリティ関連のビューが配置されている．

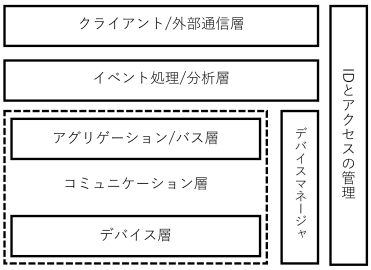

図 1 IoT のリファレンスアーキテクチャ[3]

2.2 サービス指向リファレンスアーキテクチャ[4]

図 2 に Louis らが定義したサービス指向リファレンスアーキテクチャを示す．このリファレンスアーキテクチャは，サービスとマイクロサービスを統合するためのアーキテクチャである．Container Orchestration により，マイクロサービス群としてサービス化することが可能になる．

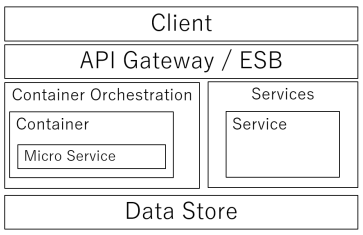

図 2 サービス指向リファレンスアーキテクチャ[4]

2.3 IoT アプリケーションのための概念アーキテクチャ

IoT とサービス指向のリファレンスアーキテクチャを参照し，コンテキスト指向を統合する．多種多様なデバイスやサービスを協調させると，デバイスやサービスの組み合わせの数が大きくなる．BPEL のようなビジネスプロセス言語で定義されている同期記述のための言語要素は，最適粒度の記述に適しておらず，却って記述が煩雑になる．デバイスやサービスの同期を最適粒度で行うために，コンテキスト指向を取り入れることが必須である．IoT とサービス指向，コンテキスト指向を統合した上で，デバイスなどの管理側面について考察するべきと考え，管理側面の詳細については考えないこととして，概念アーキテクチャを定義する．図 3 に本研究で提案する IoT アプリケーションのための概念アーキテクチャを示す．

Client，Data Store，Edge，IoT Devices に対応するコンポーネントは，コンテキストの更新を行う．Container Orchestration，Services に対応するコンポーネントは，サービス指向リファレンスアーキテクチャを取り入れ，

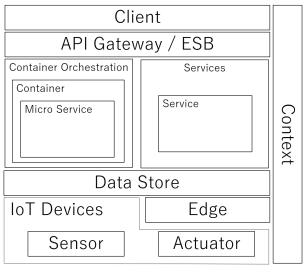

図 3 IoT アプリケーションのための概念アーキテクチャ

Client からの要求とコンテキストの参照によって，Client からの要求に対応するためのサービスを提供する．IoT アプリケーションに関わるあらゆるコンポーネントが，コンテキストと関連する構造を実現することで，クライアントとデバイスの状況に合わせた柔軟な対応と，実行効率の向上が可能になる．

3 事例

本研究では IoT アプリケーションの具体例として，チケット予約システムを扱うこととした．チケット予約システムでの支払方法は，現金とキャッシュレス決済がある．チケットの受け取りは発券機またはスマートフォンなどの端末で行われる．それらはユーザがチケットを予約，購入する際に指定する．

チケット予約システムは，チケット予約アプリケーションと発券機，スマートフォンなどの端末が連携する，サービス指向かつ IoT に関連したシステムである．以上のことから，チケット予約システムは本研究で定義した IoT アプリケーションのための概念アーキテクチャの具体例として妥当である．

3.1 チケット予約システムのための具象アーキテクチャ

図 3 で定義した IoT アプリケーションのための概念アーキテクチャをもとに，具体例とするチケット予約システムのための具象アーキテクチャを定義する．図 4 にチケット予約システムのための具象アーキテクチャを示す．

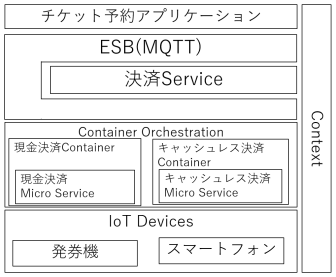

図 4 チケット予約システムの具象アーキテクチャ

チケット予約アプリケーションからの要求が ESB を通じて決済 Service へと伝わり，決済 Service が現金またはキャッシュレス決済のマイクロサービスをマッシュアップするという構造である．チケット予約アプリケーションへ登録された購入情報と，現金またはキャッシュレスでの支払い情報をコンテキストとして，対象となるデバイスで発券する．

横断する層として Context を配置することで，ESB で接続されたコンポーネント同士が，Context を参照することができる．発券機やスマートフォンから，決済方法や決済方法に合わせた支払情報を入手してコンテキストとする．

一般的に，アーキテクチャは，複数のビューから構成されていることが一般的であり，それぞれの関心事を別々に記述することで関心事の関連を整理することができる．本研究で定義したチケット予約システムアーキテクチャにおいて，コンテキスト指向，サービス指向，IoT それぞれの視点におけるコンポーネントについて整理を行う．これにより，横断的関心事がどのように関連しているのかを整理する．

図 5 に Context View を示す．コンテキストに応じて振る舞いが変わるコンポーネントを明らかにするために作成した．チケット予約システムにおける Context Level と Base Level に属するコンポーネントを抜き出して，その関連を示している．決済 Service 内部のコンテキストと Behavior Activator は，Context Level に含まれており，Behaviors は Base Level に含まれている．コンテキストの変化に応じて Behavior Activator がマイクロサービスの Behavior を決定することができる構造になることを明らかにした．

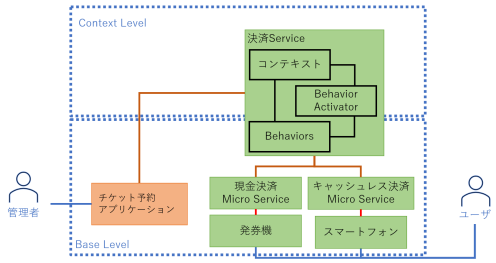

図 5 チケット予約システム Context View

図 6 に Service Oriented View（以下，SO View）を示す．ESB に接続してサービスとなるコンポーネントを明らかにするために作成した．互いにサービスとして扱える関係にあるコンポーネントを抜き出している．本研究では，実行効率を優先して，簡易 ESB では MQTT だけを扱い，パブリッシュ/サブスクライブアーキテクチャは利用しない．マイクロサービスのマッシュアップを行うコンポーネントが，決済 Service であることが明らかとなった．これにより，決済 Service 内部の構造について定義することが可能になった．

図 7 に IoT View を示す．決済 Service においてマッシュアップする対象を整理するために作成した．決済 Service でマッシュアップをするために，各サービスは API を提供する．紫の点線はリファレンスアーキテクチャにおける API ゲートウェイを表し，線の下部分はチケット予約システムアーキテクチャにおける IoT デバイスとしての側面であることを明らかにした．

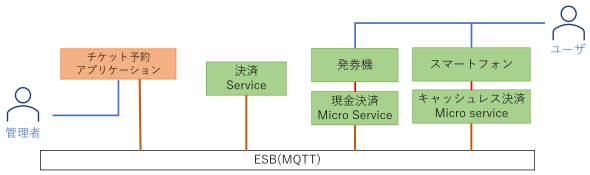

図 6　チケット予約システム SO View

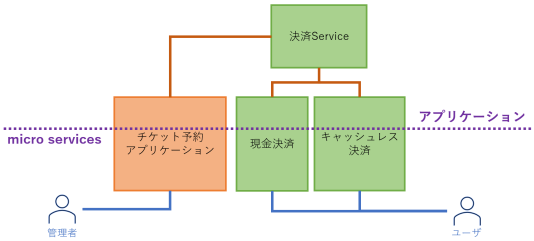

図 7　チケット予約システム IoT View

4　考察

4.1　提案アーキテクチャを用いた試作の概要

事例としたチケット予約システムには，予約待ち，入金要求，発券要求の状態があり，それぞれの状態で動作するサービスの構成を切り替える必要がある．それぞれの状態におけるユースケースと構成図を示す．以下に示すユースケースと構成図は，3.1 節で定義したチケット予約システム具象アーキテクチャに基づいて記述する．

- 予約待ち状態
  決済 Service は，現金またはキャッシュレス決済の履歴管理サービスから発券履歴のコンテキストを得る．チケット予約システムのユーザがチケットの予約操作を行うと，発券履歴データから発券可能枚数を計算する．発券可能な場合，入金要求状態に遷移する．発券不可の場合，チケット予約アプリケーションに対して発券不可のビューを提供するように要求する．予約待ち状態の構成図を図 8 に示す．
- 入金待ち状態
  チケット予約システムのユーザが指定した支払方法で，チケット代金以上の金額を支払うまで，決済 Service は現金またはキャッシュレス決済の入金サービスを起動するよう要求する．ユーザからの支払が終了すると，発券待ち状態に遷移する．入金待ち状態の構成図を図 9 に示す．
- 発券待ち状態
  決済 Service は，チケット予約システムのユーザが指定した発券方法で発券する要求を行う．現金決済の場合は発券機，キャッシュレス決済の場合はスマートフォンでの発券を発券サービスが行う．発券が終了すると，予約待ち状態に遷移する．発券待ち状態の構成図を図 10 に示す．

これらの構成図から，本研究で定義したチケット予約システムは，コンテキスト指向で決済 Service が動作させるサービスの構成を変化できることが明らかとなった．サービス指向により，現金決済とキャッシュレス決済の機能がマイクロサービス化されているので，それぞれの状態で動作させるマイクロサービスと動作させるデバイスを明らかにすることができた．以上のことから，提案アーキテクチャはコンテキスト指向とサービス指向の考え方に沿って統合することができていると考える．

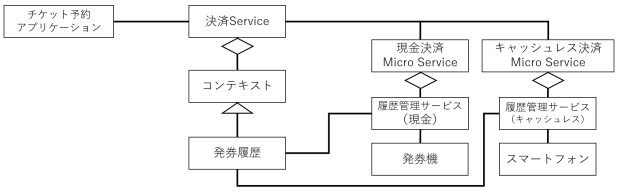

図 8　予約待ち状態の構成図

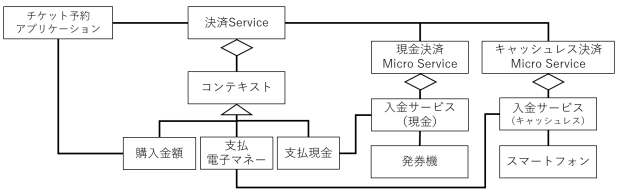

図 9　入金要求状態の構成図

4.2　サービス統合に並行処理記述を利用することについての考察

本研究では，サービス統合のための言語の例として，BPEL を扱う．BPEL を用いることで，異なるグローバル変数にレスポンスが格納され，複数のサービスを同時に起動することができる [5]．BPEL のようなビジネスプロセス言語で定義されている同期記述のための言語要素は，粒度が大きすぎるので，記述が煩雑になり実行効率が低下する問題が起こる．並行処理記述の利用により，実行効率の向上と同期のメカニズム記述の簡素化ができると考え，検証を行う．

図 11 に，BPEL でサービス連携した場合のチケット予約システムのアクティビティ図を示す．チケット予約システムにおいて正常に発券が行われる場合を想定している．BPEL を用いる場合，オブジェクト間の通信で，際どい領域に対する BPEL 固有のプロセス記述が必要となる．このプロセス記述でロック，アンロックによる相互排除を実現している．並行動作するサービスの数だけ際どい領域に対してプロセスを複数記述する必要がある．

図 12 に，並行処理記述を利用してサービス連携を行った場合のチケット予約システムのアクティビティ図を示す．図 13 に，支払時の順路式の一例を示す．オブジェクト指向に基づいて，共有資源を明らかにして順路式を用いれば，同期のメカニズムを簡単に書けることに着目した．これにより，順路式でメソッドの実行順序を指定するだけで相互排除が可能となる．組込みシステムにおいて，一般的に利用される状態遷移モデルの作成と，親和性が高い．

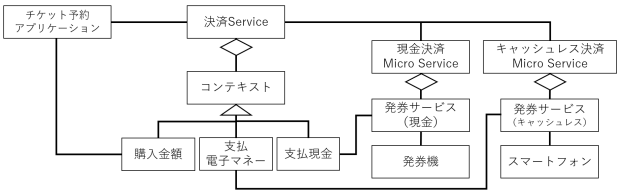

図 10　発券要求状態の構成図

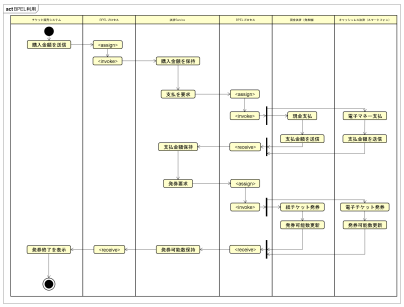

図 11　チケット予約システムのアクティビティ図（BPEL）

4.3　アプリケーション開発プロセスの考察

本研究では，IoT アプリケーション開発において，アーキテクチャを中心にアプリケーション開発をするためのアプリケーション開発プロセスを提案する．提案するアプリケーション開発プロセスを以下に示す．

1. 概念アーキテクチャに基づいて階層構造の具象アーキテクチャを定義
   本研究で提案する概念アーキテクチャに基づいて，階層構造の具象アーキテクチャを定義し，共有資源を特定する．
2. Context View を記述
   コンテキスト指向の視点から，コンテキストに応じて振る舞いが変わるコンポーネントを整理する．
3. SO View を記述
   サービス指向の視点から，サービスとして扱うコンポーネントを整理すし，サービスのマッシュアップを行うコンポーネントを明確にする．
4. IoT View を記述
   IoT の視点から，共有資源においてマッシュアップする対象を整理する．
5. 並行動作するメソッドの実行順序定義
   順路式を利用して相互排除を実現するために，並行動作するメソッドの実行順序を定義する．

提案するこのプロセスは，本研究の図 3 で定義した IoT アプリケーションのための概念アーキテクチャが，含意するプロセスである．

5　おわりに

本研究は，コンテキスト指向，サービス指向，サービスのマッシュアップの技術を統合したソフトウェアアーキテクチャを定義することを目的とした．サービスのマッシュアップにおいては，並行処理記述の利用を試みた．

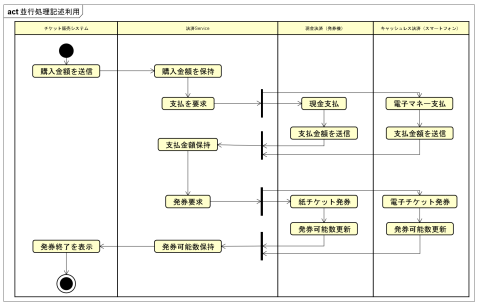

図 12　チケット予約システムのアクティビティ図

```
pay_amount = // pay_request() [cash_pay() / cashless_pay() = pay_amount()] / pay_finish()
```

図 13　支払時の順路式記述例

IoT アプリケーションのための概念アーキテクチャを定義した．アーキテクチャ定義のために，IoT とサービス指向のリファレンスアーキテクチャを統合し，コンテキスト指向を統合した．定義した IoT アプリケーションのための概念アーキテクチャに基づいて，チケット予約システムのための具象アーキテクチャを定義した．コンテキスト指向，サービス指向，サービスのマッシュアップを横断的関心事として，コンポーネントの関連を整理した．

本研究で提案するアーキテクチャと並行処理記述について考察を行った．オブジェクト指向に基づいたアプリケーション設計と並行処理記述を行うことで，実行効率の観点で有用性が高いと考察した．本研究で定義した概念アーキテクチャと，並行処理記述方法を用いるアプリケーション開発プロセスを提案した．今後，管理側面を考慮したアーキテクチャの再設計，再検討を行う．

参考文献

[1] FIWARE, “What is FIWARE? | Let’s FIWARE”, https://www.letsfiware.jp/what-is-fiware/ (Accessed 2023.02.13).

[2] Anthony J. Lattanze（著），橘高 陸夫（監訳），“アーキテクチャ中心設計手法: ソフトウェア主体システム開発のアーキテクチャデザインプロセス”，翔泳社，2011.

[3] Paul Fremantle, “A Reference Architecture For The Internet of Things”, WSO2, 2015.

[4] J. Louis, M Fowler, “Microservices”, martin-Fowler.com, 2014.

[5] Oracle, “Oracle SOA Suite での SOA アプリケーションの開発 | 10 BPEL プロセスでのパラレル・フローの使用”, https://docs.oracle.com/cd/F17744_01/soa/12.2.1.3/develop/using-parallel-flow-bpel-process.html (Accessed 2023.02.13).