Matter SDK への新しい Cluster の追加方法の調査

2024/7/18

細川 健真

1　はじめに

工場における Matter を活用した IoT システムの提案にあたって，工場における Matter デバイスのプロトタイプを作成する．工場でのインタビューを実施した結果，工場の広大な敷地内で人員の位置を特定する必要性があることがわかった．そこで，本研究では BLE を用いた位置推定システムに使用できる Matter デバイスを設計し，実装する．本資料では，Linux で動作する Matter アプリケーションを実装するため，connectedhomeip [1] への新しい Cluster の追加方法を調査する．

2　新しい Cluster の開発手順

以下に新しい Cluster の開発手順を示す．

(1) Cluster の定義ファイルの作成
(2) Cluster の実装の作成
(3) Cluster の SDK への組み込み

Matter 公式に工場向け DeviceType を認証してもらう必要はないため，認証テスト等の手順は省略する．この資料では，SDK での Cluster と DeviceType の実装に重点を置く．以降では sample-mei-cluster を例として，各工程について説明を行う．

3　Cluster の定義ファイルの作成

この章では，Cluster の定義ファイルを作成する方法について説明する．sample-mei-cluster.xml を例に説明する．XML 定義ファイルは以下の 4 つの部分に分けられる．

(1) Header
(2) General properties
(3) Attributes
(4) Command

上記の各項目について以降で説明する．

Header

Header では，XML ファイル全体の設定を行うための基本的な構造と情報を定義する．Header

の例をリスト 1 に示す．<?xml version="1.0"?>タグを用いて，XML バージョン 1.0 として定義する．その他に，ルート要素として <configurator> タグ，CHIP へのドメインの指定を行う<domain>タグが含まれる．

リスト 1　Header の例

```
1  <?xml version="1.0"?>
2  <configurator>
3  <domain name="CHIP"/>
```

General properties

General properties では，Cluster の全般的な定義を行う．General properties の例をリスト 2 に示す．<name>タグおよび<define>タグを用いて，新しい Cluster の名前を定義する．connectedhomeip 内のいくつかのスクリプトは，ここで定義された命名規則に基づいてソースコードを生成する．<client tick>タグ もしくは <server tick> タグを用いて，スケジューラからの定期的な呼び出しをスケジュールする．これにより，Cluster が時間指定または遅延操作を実行できる．サンプル Cluster ではこの機能に関する定義はされていない．<code>タグを用いて，ベンダ ID と Cluster ID を定義する．リスト 2 では，ベンダ ID (0xFFF1，この場合はテスト VID) と Cluster ID (0xFC20) を連結した値を指定している．初期の概念実証を超えて製品を開発する場合は，これらの値を独自の本番 VID と選択した Cluster ID に変更しなければならない<description>タグを用いて，Cluster の説明を記述する．

リスト 2　General properties の例

```
1  <cluster>
2  <domain>General</domain>
3  <name>Sample MEI</name>
4  <code>0xFFF1FC20</code>
5  <define>SAMPLE_MEI_CLUSTER</define>
6  <description>The Sample MEI cluster showcases cluster manufacturer extensions</
       description>
```

Attribute

Attribute では，新しい Cluster の Attribute を定義する．Attribute の例をリスト 3 に示す．Attribute の定義は <attribute>タグを用いて行われる．<attribute>タグの各属性について以下で説明する．

side　server もしくは client から選択する．

code　Attribute ID を定義する．

define　Attribute の名前を定義する．

type　保持する値の型を指定する．

writable　書き込み可能かどうかを定義する．

default　Attribute の初期値を定義する．

属性 define は，Screaming Snake Case 形式で記述する必要がある．また，<attribute>タグ内の XML コンテンツには，Pascal ケースで一致する名前を記述する．

リスト 3　Attribute の例

```
<!-- Attributes -->
<!-- A simple boolean attribute that flips or flops -->
<attribute side="server" code="0x0000" define="FLIP_FLOP" type="BOOLEAN" writable="
    true" default="false" optional="false">FlipFlop</attribute>
```

Command

Command では，新しい Cluster の Command を定義する．Command の例をリスト 4 に示す．Command の定義は <command>タグを用いて行われる．<command>タグの各属性について以下で説明する．

code　Command ID を定義する

name　Command の名前を定義する

response　Command Response を指定する

Command の引数の定義は<arg>を用いて行う．<arg>タグの各属性について以下で説明する．

name　引数の名前を定義する

type　引数の型を指定する

リスト 4　Command の例

```
<!-- Command Responses -->
<command source="server" code="0x01" name="AddArgumentsResponse" optional="false"
    disableDefaultResponse="true">
<description>
   Response for AddArguments that returns the sum.
</description>
<arg name="returnValue" type="INT8U"/>
</command>

<!-- Commands -->
<command source="client" code="0x02" name="AddArguments" response="
    AddArgumentsResponse" optional="false">
<description>
   Command that takes two uint8 arguments and returns their sum.
</description>
<arg name="arg1" type="INT8U"/>
<arg name="arg2" type="INT8U"/>
</command>
```

4　Cluster の実装

この章では，Cluster を実装する方法について説明する．Cluster Server の実装として，AttributeAccessInterface と CommandHandlerInterface を介した cpp 実装を作成する．Cluster のソースコードは src/app/clusters/<cluster-name> に配置され，デバイスの .zap/.matter ファイルに Cluster が含まれている場合にビルドされる．Cluster は任意の方法で実装できるが，sample-mei-cluster はその一例を示している．sample-mei-cluster の実装について，以降で説明する．

ヘッダーファイル

Attribute に関するソースコードをリスト 5 に示す．XML ファイルで定義した Attribute と同じ

フィールドを持つクラスが宣言されている．sample-mei-cluster は bool 型の flipflop Attribute のみ保持する．

リスト 5　Cluster ヘッダーファイル内の Attribute に関する例

```
  class SampleMeiContent{
public:
    EndpointId endpoint;

    // Attribute List
    bool flipflop; /* Attributes::FlipFlop::Id */

    SampleMeiContent(EndpointId endpoint);
    SampleMeiContent();
};
```

Server Cluster のメインクラスに関するソースコードをリスト 6 に示す．Server Cluster のメインクラスでは， AttributeAccessInterface と CommandHandlerInterface を継承し，Read(), Write() および InvokeCommand() メソッドをオーバーライドして Attribute の読み書きと，コマンド受信時の処理を行う必要がある．

リスト 6　Cluster ヘッダーファイル内の Server Cluster のメインクラスに関する例

```
class SampleMeiServer : public AttributeAccessInterface, public CommandHandlerInterface
{
public:
    (...)
    // Attributes
    CHIP_ERROR Read(const ConcreteReadAttributePath & aPath, AttributeValueEncoder &
        aEncoder) override;
    CHIP_ERROR Write(const ConcreteDataAttributePath & aPath, AttributeValueDecoder &
        aDecoder) override;

    // Commands
    void InvokeCommand(HandlerContext & ctx) override;

    // Attribute storage
#if SAMPLE_MEI_NUM_SUPPORTED_ENDPOINTS > 0
    SampleMeiContent content[kNumSupportedEndpoints];
#else
    SampleMeiContent* content = nullptr;
#endif

    (...)
};
```

cpp ファイル

各 Command に関するソースコードをリスト 7 に示す．Cluster 内の各 Command は，InvokeCommand() メソッド内の switch 文での分岐処理にて処理される．

リスト 7　Cluster cpp ファイル 内の 各 Command に関するソースコード

```
void SampleMeiServer::InvokeCommand(HandlerContext & ctxt)
{
    (...)

    switch (ctxt.mRequestPath.mCommandId)
    {
```

```
 7      case Commands::Ping::Id:
 8          HandleCommand<Commands::Ping::DecodableType>(ctxt, [endpoint](HandlerContext &
                ctx, const auto & req) {
 9              ChipLogProgress(Zcl,"Ping Command on endpoint %d", endpoint);
10              ctx.mCommandHandler.AddStatus(ctx.mRequestPath, Protocols::InteractionModel::
                    Status::Success);
11          });
12          return;
13
14          (...)
15
16      }
17  }
```

Read()，Write() メソッド に関するソースコードをリスト 8 に示す．Read()，Write() メソッドにおいても，他のクラスメソッドと共に定義する必要がある．

リスト 8　Cluster cpp ファイル 内の Read，Write メソッドに関する例

```
 1   CHIP_ERROR SampleMeiServer::Read(const ConcreteReadAttributePath & aPath,
         AttributeValueEncoder & aEncoder)
 2   {
 3       (...)
 4
 5       switch (aPath.mAttributeId)
 6       {
 7           case Attributes::FlipFlop::Id:
 8               ChipLogProgress(Zcl,"Read Attribute flip-flop from endpoint %d index %zu
                     value %d", endpoint, endpointIndex, content[endpointIndex].flipflop);
 9               err = aEncoder.Encode(content[endpointIndex].flipflop);
10               break;
11           default:
12           break;
13       }
14
15       return err;
16   }
17
18   CHIP_ERROR SampleMeiServer::Write(const ConcreteDataAttributePath & aPath,
         AttributeValueDecoder & aDecoder)
19   {
20       (...)
21
22       switch (aPath.mAttributeId)
23       {
24           case Attributes::FlipFlop::Id:
25               {
26                   auto oldValue = content[endpointIndex].flipflop;
27                   ReturnErrorOnFailure(aDecoder.Decode(content[endpointIndex].flipflop));
28                   ChipLogProgress(Zcl,"Write Attribute flip-flop on endpoint %d index %zu
                         newValue %d oldValue %d", endpoint, endpointIndex, content[
                         endpointIndex].flipflop, oldValue);
29               }
30               break;
31           default:
32           break;
33       }
34
35       return err;
36   }
```

5　定義した Cluster を SDK に組み込む手順

以下に定義した Cluster を SDK に組み込む手順を示す.

(1) 以下のディレクトリに定義した Cluster の XML ファイルを追加する
`src/app/zap-templates/zcl/data-model/chip`

(2) 以下のファイルに定義した Cluster の XML ファイルへの参照を追加する
（A）`.github/workflows/tests.yaml`
（B）`scripts/rules.matterlint`
（C）`src/app/common/templates/config-data.yaml`
（D）`src/app/zap-templates/zcl/data-model/all.xml`
（E）`src/app/zap-templates/zcl/zcl-with-test-extensions.json`
（F）`src/app/zap-templates/zcl/zcl.json`
（G）`src/controller/python/chip/clusters/__init__.py`

(3) 追加する Cluster が派生 Cluster の場合は基本 Cluster 定義 XML に参照を追加する
基本 Cluster が Mode Base Cluster の場合の説明を行う．Mode Base Cluster の定義 XML に新しい Cluster の cluster code を追加する.

```
1  <struct name="ModeOptionStruct">
2   <cluster code="0x0051"/> <!-- Laundry Washer Mode -->
3  + <cluster code="YOUR NEW CLUSTER ID"/>
4  </struct>
```

(4) `src/controller/data_model/controller-clusters.zap` の Python および Android Client で新しい Clulster を有効にする
ZAP ファイルを編集するには，ZAP ツール [2] が必要である．ツールがインストールされていない場合は， リリースページ [3] から入手する．ZAP ツールを用いた編集手順を以下に示す.
（A）コマンドラインから，controller-clusters.zap を含むディレクトリに移動する

```
1  $ cd connectedhomeip/src/controller/data_model
```

（B）ZAP ツールを起動する

```
1  $ ../../../scripts/tools/zap/run_zaptool.sh controller-clusters.zap./scripts/
       tools/zap/run_zaptool.sh
```

（C）左側のパネルから Endpoint-1 を選択する
（D）追加する Cluster のグループを展開し，追加する Cluster の項目を探す
（E）「Enable」カラムにて，ドロップダウンボックスから Client を選択する
（F）File -¿ Save を実行し，設定を保存する
（G）GUI を閉じる

(5) `src/app/zap_cluster_list.json` の ClientDirectories セクションにエントリを追加する

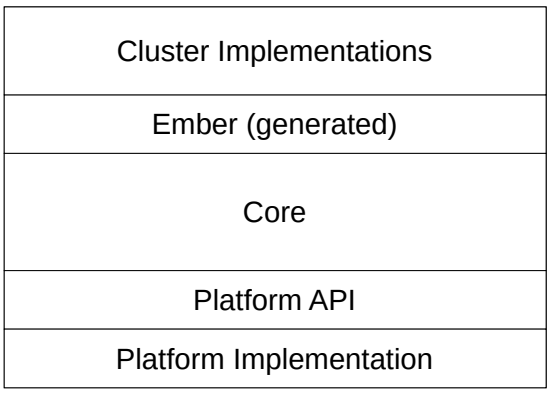

図 1　Matter SDK のアーキテクチャ

```
1   {
2     "ClientDirectories": {
3     + "SAMPLE_MEI_CLUSTER": [],
```

(6) zap ファイルからコードを生成する

```
1 $ ./scripts/tools/zap_regen_all.py
```

(7) chip-tool を再構築する

```
1 $ ./scripts/examples/gn_build_example.sh examples/chip-tool SOME-PATH/
```

6　Matter SDK のアーキテクチャ

図 1 に Matter SDK のアーキテクチャを示し，以下で説明する．

Platform レイヤ

ネットワークスタックとベース OS への接続を実装する．メッセージはネットワークから Platform レイヤーに流れ込み，そこで Platform API にルーティングされ，Matter スタックによって処理される．

Platform API

Core と対話するための共通レイヤを定義する．

Core

基礎となる通信プロトコルのすべてを含む，Matter 仕様の大部分を網羅する．Core は，Cluster と関連する Endpoint 情報を示す有効なメッセージを Ember レイヤに配信する．

Ember レイヤ

1 つの特定のデバイスの構成を実装する．各メッセージを調べて，デバイスに 指定された Cluster，Attribute，Command 実装されているかどうかを判断し，実装とアクセス制御に基づいてブロックまたはルーティングを行う．有効なリクエストは処理のために Cluster

Implementations に転送され，無効なリクエストはエラーとともに送り返される．Ember レイヤの大部分は，.zap ファイル を使用して静的に生成される．

Cluster Implementations

Ember レイヤからメッセージを受信して，Cluster でのデータモデル操作 (読み取り/書き込み/コマンド呼び出し) を行う処理を実装する．また，イベント生成と Attribute 変更レポートも担当する．

7 コード生成

7.1 コード内容

Matter のコードは，Cluster 固有のデータタイプとコールバックのためにコード生成に依存している．コードの内容は以下の 2 つに分けられる．

(1) 構造体/リスト/コマンドのデータシリアライゼーション
クライアント側およびサーバー側の両方の構造とオブジェクトに適用される．

(2) Ember ベースのフレームワークを使用したコールバック設定
主にサーバー側の処理に適用され，特定の Command が受信されたときや Attribute が読み取られたときにどの処理を行うか，Attribute を保存するためにどのメモリを割り当てるかを定義する．

7.2 matter ファイルと zap ファイル

コード生成はアプリケーションが必要とする Cluster に依存する．各アプリケーションは，サポートする DeviceType に基づいて必要な Endpoint と Cluster の特定のセットを構成する．サポートする Cluster と Attribute の選択は，.zap ファイルに保存されている．

.zap ファイルは大きな json ファイルであり，人間には読みやすくない．これは，.zap ファイルが ZigBee との後方互換性を持つように設計されているためである．

一方，Matter SDK は.zap ファイルと共に同等の.matter ファイルも保持している．これには，.zap ファイルと同じデータが含まれており，Matter 固有のデータのみを含むように設計されている．現在，.matter ファイルはアプリケーション固有のコード生成中に.zap ファイルから生成される．これらの.zap ファイルと.matter ファイルはコード生成技術の実験として存在している．

.matter ファイルは人間および機械が読み取り可能である．.matter ファイルを処理できるコードは scripts/py_matter_idl および scripts/codegen.py である．.matter ベースのコード生成には以下の特徴がある．

(1) zap ベースよりもサードパーティの依存関係が少ない (zap は多数の npm パッケージをインストールする)

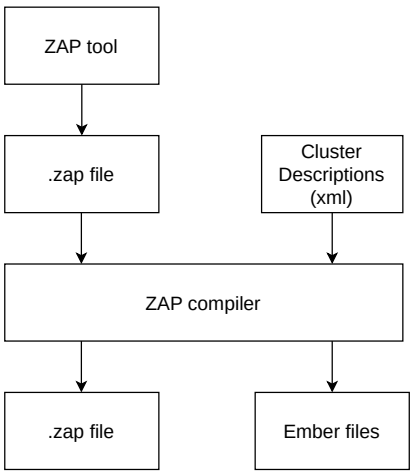

図 2　.zap ファイルが使用される流れ

(2) zap ベースよりもはるかに高速で動作する
(3) 人間が読みやすく，編集しやすい.matter ファイルを入力とする
(4) 複雑さが低く，単体テストが行われており，広範な型指定を使用する

しかし，.matter ベースのコード生成は開発途中であり，現在は.zap ベースのコード生成が主流である．

7.3　zap ファイルが使用される流れ

図 2 に.zap ファイルが使用される流れを示す．.zap ファイルは，Cluster 定義ファイルとともに ZAP コンパイラによって使用され，Ember レイヤを生成する．これはビルドプロセスの一部として自動的に行われ，Ember レイヤはファームウェアにコンパイルされる．Ember レイヤは，Zigbee スタックの基盤を形成するソフトウェアレイヤである．

7.4　コード作成の実行方法

すべてのアプリケーションのコードを再生成するには /scripts/tools/zap_regen_all.py を用いる．

```
1 $ ./scripts/tools/zap_regen_all.py
```

また，個別のアプリケーションのコードを再生成するには，/scripts/tools/zap/generate.py を用いる．

```
1 $ ./scripts/tools/zap/generate.py examples/bridge-app/bridge-common/bridge-app.zap
```

上記の方法は<app>.zap ファイルの横に<app>.matter ファイルを生成する．これはアプリケーションのために更新が必要な唯一のファイルである．その他のものをコード生成するには，generate.py に-t/--templates 引数を渡す．その場合，-o/--output-dir 引数を使用して出力ディレクトリを指

定する必要がある場合がある．

8　おわりに

本資料では，Matter SDK connectedhomeip への新しい Cluster の追加方法を調査した．

参考文献

[1] Connectivity Standards Alliance: project-chip/connectedhomeip: Matter (formerly Project CHIP) creates more connections between more objects, simplifying development for manufacturers and increasing compatibility for consumers, guided by the Connectivity Standards Alliance., GitHub (online), available from ⟨https://github.com/project-chip/connectedhomeip⟩ (accessed 2024-04-10).

[2] Connectivity Standards Alliance: project-chip/zap: ZAP stands for ZCL Advanced Platform. It is used to configure clusters, attributes and other entities for Matter and ZigbeePro applications., GitHub (online), available from ⟨https://github.com/project-chip/zap⟩ (accessed 2024-06-12).

[3] Connectivity Standards Alliance: Releases · project-chip/zap, GitHub (online), available from ⟨https://github.com/project-chip/zap/releases⟩ (accessed 2024-06-12).