平成 25 年度 修士論文

M2M におけるデバイスへの統一アクセス手法

電気通信大学 大学院情報システム学研究科
情報システム基盤学専攻
1253012　中原 祥吾

指導教員　末田 欣子　客員准教授
多田 好克　教授
本多 弘樹　教授

提出日　　平成 26 年 1 月 27 日

目次

図目次

表目次

ソースコード目次

第 1 章

序論

近年, 有線や無線ネットワークの進歩に伴い M2M(Machine-to-Machine) 通信も普及している. M2M 通信とは, 人間の手を介在せず, デバイス同士を通信させる仕組みやコンセプトのことである. M2M 通信を用いることで機器の自動化制御やスマートハウス構築などといった M2M アプリケーションを開発することが可能となる. 例えば, 自動車のブレーキが踏まれた位置を収集し, よく踏まれている場所では, 速度を自動で制限するシステムが研究されている [1]. またこのシステムでは, 収集した情報を地図上で利用することにより, 急カーブの場所や見えにくい信号機などといった危険な場所を早期発見するサービスを提供することもできる. このように, M2M アプリケーションは, M2M 通信を行うことで発生したデータを用いることにより, 新しいサービスやビジネスに繋がっていく可能性があるため, 期待されている.

しかし, M2M アプリケーションを構築するためには, 通信方式や制御方式が異なる様々なデバイスを使用することを考慮しなければならない. そのため, M2M アプリケーション開発時には, デバイス毎に処理を作成する必要がある. したがって M2M アプリケーションが肥大化し, 複雑となる要因となるため, 開発時のネックとなっている. そこで本研究では, 通信方式や制御方式が異なる様々なデバイスに対して, 共通の方式でデバイスにアクセスするため手法を検討した. 提案手法では, ホーム ICT 基盤などでデバイス管理のために利用されている OSGi(Open Services Gateway initiative) というフレームワーク上に REST(REpresentational State Transfer) を用いてデバイスにアクセスできる機構を実装した. REST とは, Web サービスにアクセスする際に利用されているアーキテクチャである. REST をデバイス操作に割り当てることにより, Web を操作する感覚で通信方式や制御方式が異なる様々なデバイスの操作が可能となる. 評価のために, 家庭エアコンを自動制御するサンプル M2M アプリケーションを構築し, 本システムを介した場合と介しない場合との M2M アプリケーションのコード量を調査した. その結果, 本システムを介した場合, 必要となるコードは, REST 処理だけで済み, 介しない場合と比較して大幅に削減できることを示した. 更に, 本システムを介した場合の通信ボトルネックを調査するために大量のリクエストを送信し, CPU とメモリ使用量を調査した. その結果, ホーム ICT 基盤などで用いられているパフォーマンスの低いマシン上でも動作可能であるこ

とを示した.

本論文の構成を以下に示す. 2 章で, 近年のインターネットと M2M の関係について述べる. 3 章で, M2M アプリケーションの開発時の課題と解決するための要求条件について述べる. 4 章で, 提案手法として使用する REST と OSGi について述べる. 5 章で, 提案システムの概要と実装時に重要となる Bundle 間通信及び各 Bundle の動作について述べる. 6 章で, 実装として Bundle の開発方法, アノテーションの作成方法, アノテーションの解析方法について述べる. 7 章で, 提案システムを利用したケーススタディとして家庭用エアコンの自動制御を行う M2M アプリケーションについて述べる. 8 章で, 評価実験として 7 章であげたケーススタディのソースコードをもとにして提案システムを介す場合と介しない場合のソースコード量の比較, 提案システムを介すことで発生する遅延の調査, 及びパフォーマンス測定について述べる. 9 章で, 関連研究および関連サービス及び本研究の差分について述べる. 最後に, 今後の予定と課題について述べる.

第 2 章

研究背景

この章では, 通信技術やデバイスの進歩に伴う M2M 通信とそれを活用した M2M アプリケーションの普及について述べる.

図 2.1 で示すように近年, ネットワークに接続されたデバイスが急増している. Cisco IBSG（Internet Business Solutions Group）の調査によれば, 2003 年では世界人口 63 億人に対し, インターネットに接続されるデバイス数は 5 億台であり, 1 人あたりのデバイス数はわずか 0.08 台であった. しかし, 2008 年から 2009 年頃には 1 人あたりのデバイス数は, 1.0 を超え, 2010 年には 68 億人の人口に対し, 125 億台のデバイスが接続されるようになった. これは, 2003 年から 2010 年の 7 年間で 1 人あたりのデバイス数は 23 倍になったことを示している. そして, この傾向は今後も継続し, 2020 年の接続デバイス数は 500 億台に達し, 1 人あたり 6.58 台になると予測されている [2].

こうしたデバイスの増加により, インターネットの新しい活用方法が検討されている. 図 2.2 は, 新しいインターネット形態を示したものであり, 以下に示す通信方式が存在する.

- 「人と人」(H2H: Human to Human)
- 「人と機械」(H2M: Human to Machine)
- 「機械と機械」(M2M: Machine to Machine)

このように H2H, H2M, M2M が混在したインターネットは, IoT(Internet of Things) と呼ばれている. 従来のインターネットは H2H と H2M でのやり取りが主流であり, 利用者が主体的に操作することで情報のやり取りが行われてきた. これに加えて IoT では, 新しく M2M 通信を行うことにより, 人間の手を介在せず, 機械同士を通信させることで, 「機器の自動化や制御」「情報取集の自動化」などが行えるようになる. これにより, 「スマートシティ」や「ヘルスケア」,「オートモティブ」などを行うための M2M アプリケーションが開発可能になった.

また, M2M アプリケーションは, 図 2.3 のように, 様々なデバイスから情報を収集し, 蓄積したデータを外部に公開することで, 新しいサービスやビジネスに繋がっていくことも期待されている [3]. しかしながら, 多くの研究では, M2M アプリケーションの事例な

どは多く取り上げているが M2M アプリケーションの開発に関する研究はあまり行われていない．そこで本研究では, M2M アプリケーションの開発時に起きうる問題点などを調査し, M2M アプリケーションの開発を円滑に行うための仕組みを検討したので報告する．

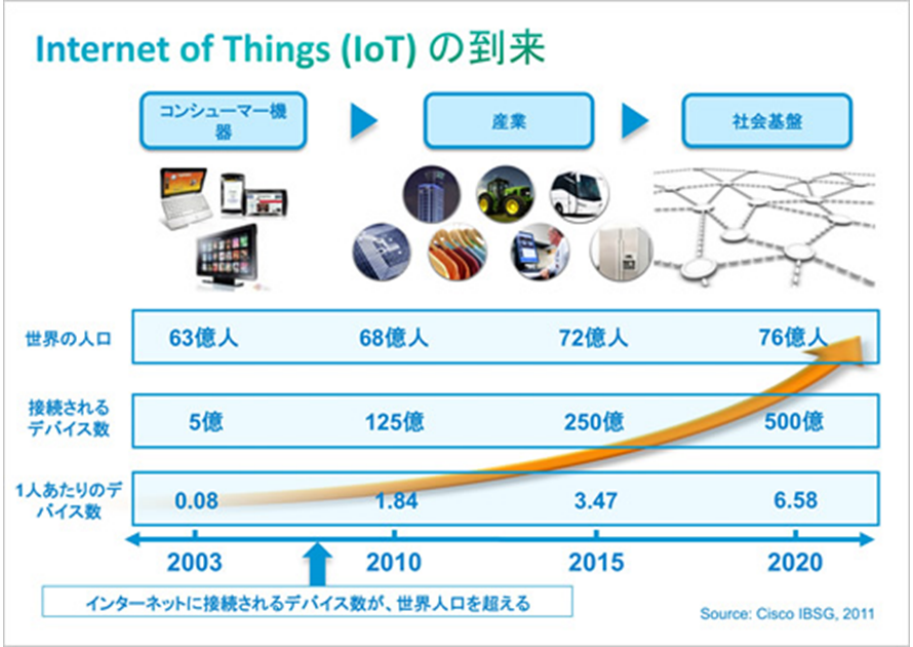

図 2.1　通信可能なデバイスの増加傾向 (文献 [2] より抜粋)

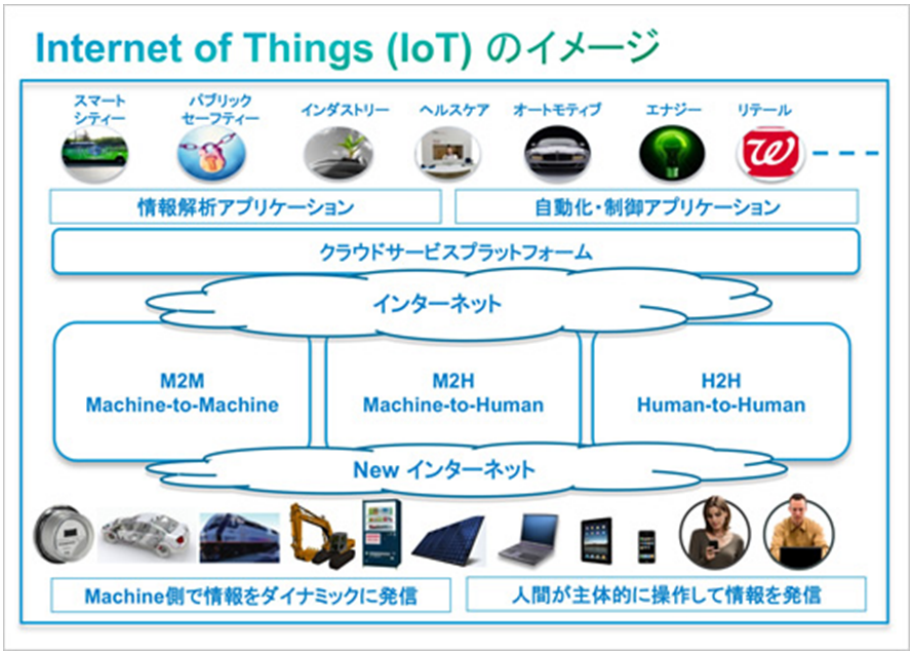

図 2.2　IoT のイメージ (文献 [2] より抜粋)

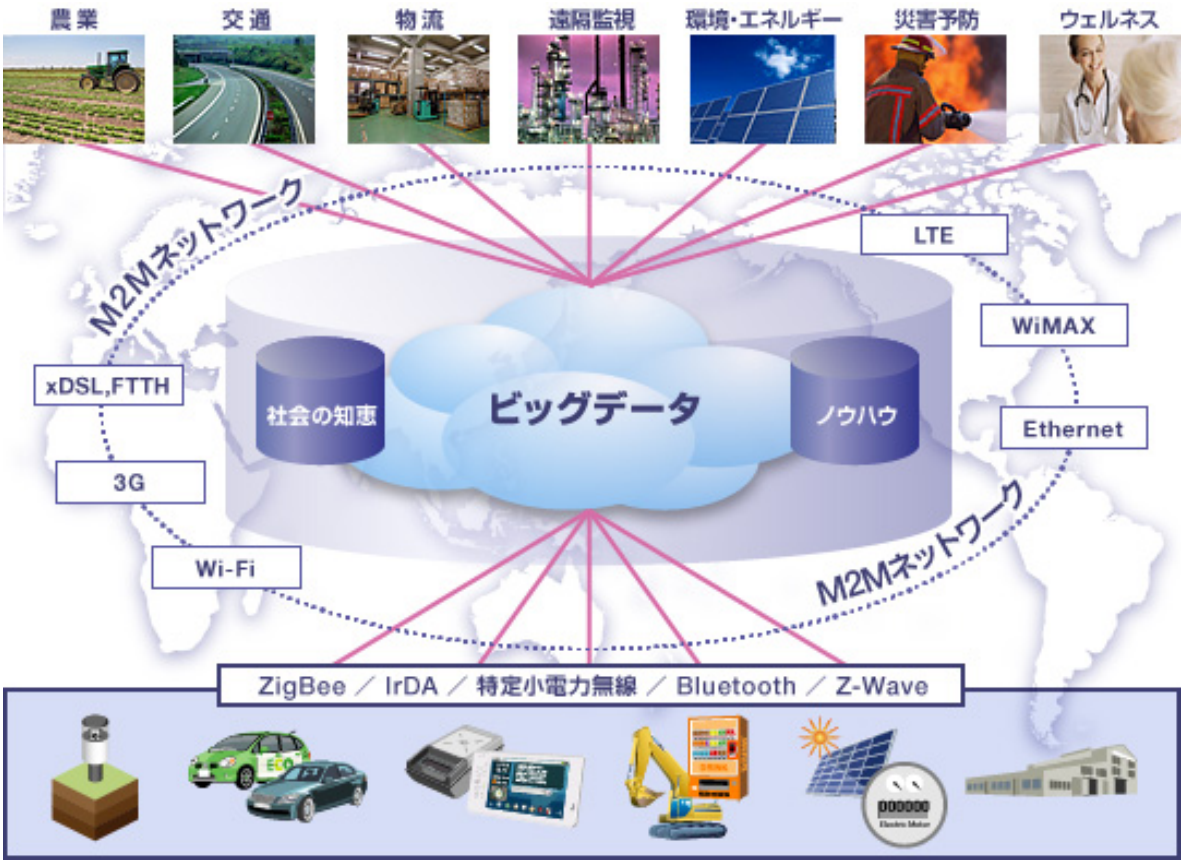

図 2.3　M2M のイメージ (文献 [3] より抜粋)

第 3 章

課題と要求条件

この章では, M2M アプリケーション開発における課題とそれを解決するための提案について述べる. その後, 提案の要求条件について述べる.

3.1 M2M アプリケーション開発における課題

M2M アプリケーション開発において, センシングしたい事象やデータの活用方法は, 開発する M2M アプリケーション要件により異なるため, M2M アプリケーション開発者は, 要件に合った様々なデバイスを自由に制御できることが望ましいと考えられる [4]. そのため, M2M アプリケーションを開発するためには, 以下に示す処理が必要となる.

- **デバイス固有の処理**
  1 つのデバイスを操作したり, データを取得するために必要な処理
  - **通信処理**
    デバイスが参加しているネットワークへのアクセスやプロトコル処理など
  - **制御処理**
    デバイスを制御するための電文構造の作成や解析処理など
- **M2M アプリケーション固有の処理**
  複数のデバイス固有の処理の組み合わせによるデバイスの制御ルール

これにより, 1 つのデバイスを操作するために必要な通信処理と制御処理といったデバイス固有の処理が必要である. また, デバイス固有の処理を組み合わて, それぞれのデバイスを制御するためのルールである M2M アプリケーション固有の処理が必要である. しかし, 以下で示す要因が M2M アプリケーションの開発を困難にしている.

- 図 3.1 のようにデバイス固有の処理を全て M2M アプリケーションに実装した場合, M2M アプリケーションのソースコード量が肥大化する可能性がある.
- 異なるデバイスを複数使用する場合, デバイス固有の処理もデバイス毎に存在するため, これらを組み合わせた M2M アプリケーション固有の処理も複雑になる可能性がある.

3.2 研究の目的

本研究では, M2M アプリケーションの肥大化とデバイス固有の処理を組み合わせた M2M アプリケーション固有の処理が複雑になることを緩和するために図 3.2 のような様々なデバイスを共通のインタフェースで操作できる仕組みを提案する.

これにより, 以下で示すように M2M アプリケーションの開発における課題を解決することができ, M2M アプリケーションの開発を容易にすることが可能であると考えられる.

- デバイス固有の処理は, 共通のインタフェースに置き換えられる. そのため, デバイス固有の処理を M2M アプリケーションに実装する必要が無くなり, ソースコード量の肥大化を抑えることが可能である.
- M2M アプリケーション固有の処理を実装するためには, 共通のインタフェースのみで実装可能になる. そのため, 開発者は複数のデバイス固有の処理を組み合わせる必要がなくなり, M2M アプリケーション固有の処理の複雑さを緩和することが可能である.

3.3 要求条件

インタフェースを検討するにあたり, 要求条件を以下に示す.

(1) **様々なデバイスへの対応**

M2M アプリケーション開発では, 様々な通信方法や制御方法が異なる複数のデバイスを操作する必要がある. また, 似たようなデバイスでも機能が異なるデバイスが存在する. 例えば, 温度センサでもロガーのように蓄積された値を参照しなければならないデバイスや定期的に値を取得するデバイスなどが存在する.

そのため, 検討するインタフェースでは, どのようなデバイスが使用されても共通のやり取りでデバイスを操作できることが望ましいと考えられる.

(2) **開発効率への対応**

検討したインタフェースが, デバイス固有の処理よりも複雑になる場合, インタフェースの実装にコストがかかることになる. また, インタフェースを使用した M2M アプリケーション固有の処理も複雑になる可能性がある. そのため, 検討したインタフェースを使用する場合, 開発の効率が低下する恐れがあり, 望ましくないと考えられる. したがって, 検討するインタフェースは, やり取りがシンプルで, 実装が容易なものが望ましいと考えられる.

(3) 仕様変更への対応

M2M アプリケーション事例では, 新しいデバイスを追加したり, 不要になったデバイスを除去することなどで仕様変更が起こる可能性がある [1]. それにより, インタフェースは, その都度, デバイスの変更に対応する必要がある. また, M2M アプリケーションもインタフェースの変更に対応していく必要がある [6].

そのため, インタフェースをデバイス固有の処理に依存して定義していた場合, 仕様変更時には, インタフェースが大幅に変わる可能性がある. それにより, M2M アプリケーションも大幅に変更する必要があるため, 仕様変更のコストがかかる.

したがって, 検討するインタフェースは, デバイスの追加や除去に柔軟に対応できると同時にデバイス固有の処理への依存性を抑えて, 仕様変更時には, M2M アプリケーションに与える影響を小さくできることが望ましいと考えられる.

(4) リアルタイム性への対応

開発する M2M アプリケーションの事例によっては, 定められた時間内に処理を完了させたり, 逐次デバイスからセンシングする必要がある事例が存在する [5].

例えば, 工場の稼働率を確認し, 稼働状況やログ状況と過去の故障時のデータなどの分析を行い, 機器のメンテナンス時期を予想するシステムでは, 全工場の機器データを逐次蓄積し続ける必要がある [1].

そのため, デバイスにアクセスする際にインタフェースを介すことで発生する遅延が, M2M アプリケーションの動作に影響を与えることは望ましくないと考えられる. したがって, デバイスにアクセスする際にインタフェースを介す時間を短縮したり, ストリーミングサービスのように逐次データ転送に耐えれるようなインタフェースを設計する必要がある.

(5) 低パフォーマンス環境への対応

インタフェースは, デバイスを管理している計算機を使用することで容易に提供できると考えられる.

しかし, M2M アプリケーション事例 [1] によれば, 人間が入っていけないような危険な場所や屋内のように非常に限定された場所にあるデバイスのみを使用する場合がある. このような場所では, 比較的パフォーマンスの低い計算機を使用して, デバイスを管理していることが多い. 例えば, ホーム ICT では, モバイル端末のような Home Gate Way(HGW) Server を使用している.

そのため, 様々な M2M アプリケーション事例に対応可能にするためには, 限られた場所でデバイスを管理するために使用されている比較的パフォーマンスの低い計算機でインタフェースを提供することが可能であることが望ましい.

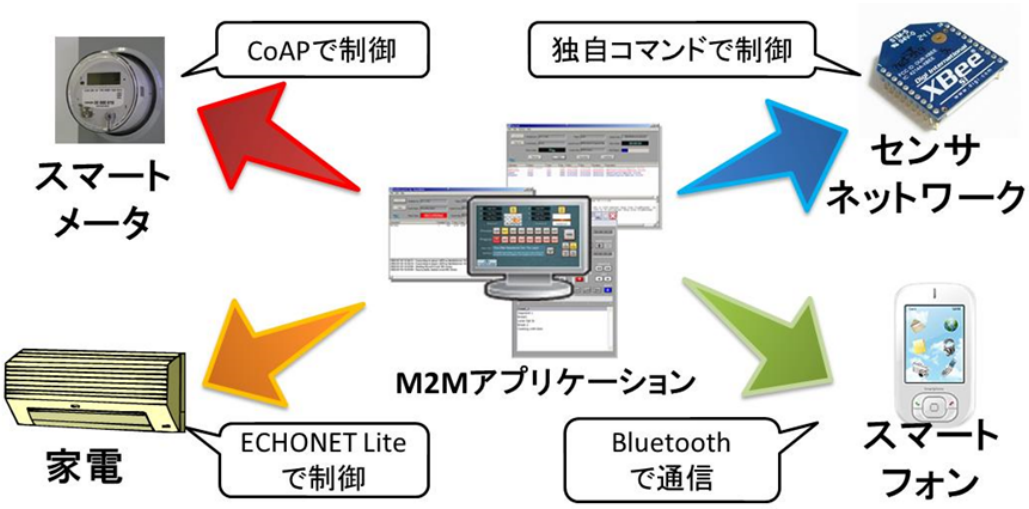

図 3.1　M2M アプリケーション開発時の課題

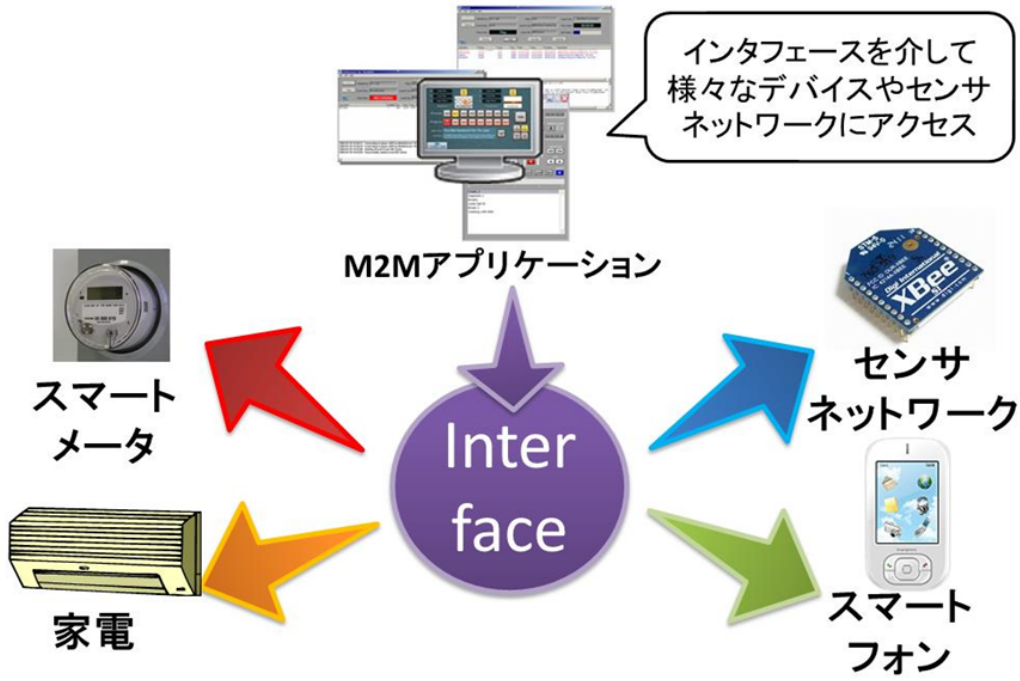

図 3.2　統一アクセス手法

第 4 章

提案方式

この章では, 要求条件を満たすための手法について説明する.

4.1 REST アーキテクチャの適応検討

要求条件 (1)(2)(3) を満たすために, REST(Representational State Transfer) アーキテクチャを使用してデバイスを操作することを検討する.

4.1.1 REST アーキテクチャ

REST とは, Web のような分散システムにおいて複数のソフトウェアを連携させるのに適した設計方法のことである [7]. REST では, HTTP の GET/POST/PUT/DELETE などといった CRUD(Create/Read/Update/Delete) に結び付けることができるメソッドを使用し, Web Service が提供しているリソース (Web Service が提供している情報) を操作することにより, そのリソースの状態を取得するといった通信設計を行う [8]. 最近では, Twitter や DropBox といった Web Service にアクセスするための API として活用されている.

図 4.1 は, 特定の人物の情報を操作するための REST サービス例である. Web Client は, Web サービス側が用意している「User/Profile/nakahara」というリソースに対して「GET」リクエストを送信する. その結果, 「nakahara」という人物の情報を取得することができる. また, 「POST」リクエストは人物情報の編集, 「DELETE」リクエストは人物情報の削除を行う操作に対応付け可能である. 更に「User/Profile/murai」というリソースに対して「PUT」リクエストを送信すると「murai」という人物情報を新しく作成する操作に対応付け可能である.

また, REST は, RPC(Remote Procedure Call) や SOAP(Simple Object Access Protocol) とよく比較される. RPC や SOAP では, サービス提供側が独自に定義したインタフェースをクライアントに提供することで, サービスのやり取りを行う. 図 4.2 は, RPC の例である. 図 4.2 では, クライアント側スタブで Profile メソッドを実行するとサーバに引数が送信される. サーバ側スタブは, 引数を受信すると, 対応するメソッドを実行してク

ライアント側スタブに返信され, あたかもクライアントで Profile メソッドが実行されたかのように見える仕組みである. このとき, サーバ側スタブでメソッドの変更があった場合, クライアント側スタブのメソッドも変更する必要がある. 例えば, Profile メソッドの名前を GetUesrData などに変更した場合, クライアントも変更する必要がある. SOAP についても RPC と同様にクライアントに変更が生まれる可能性がある. しかし, REST ならば, GET/POST/PUT/DELETE などの固定されたメソッドを使用するため, RPC や SOAP よりもクライアントに与える影響を小さくすることが可能である.

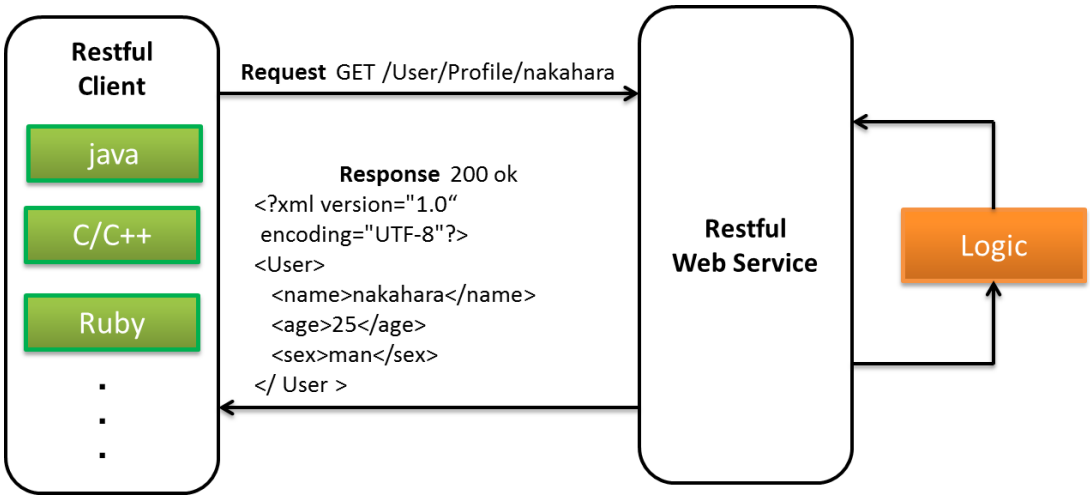

図 4.1　REST アーキテクチャ

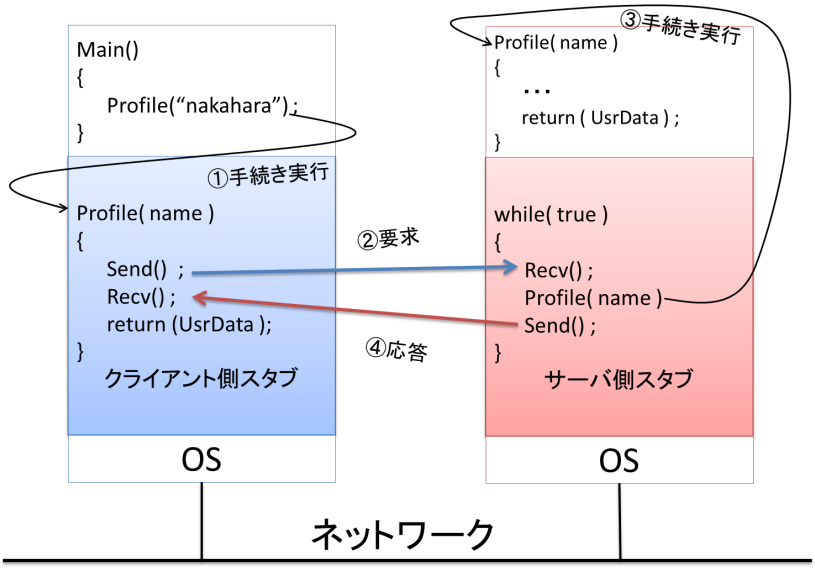

図 4.2　RPC の例

4.1.2 JAX-RS

REST サービスを作成する手段として JAX-RS(Java API for RESTful Web Services) がある [9]. JAX-RS は, Java の仕様であり, 表 4.1 に示すアノテーションを class や method, filed に付加することにより, 簡単に REST サービスの作成を可能にするものである [10].

表 4.1 アノテーション一覧

アノテーション	概要
@GET/POST/PUT/DELETE	リソースに対する操作を設定
@PATH	リソースパスを設定
@ENCODED	エンコードタイプを設定
@PRODUCES	レスポンスの形式を設定
@CONSUMES	ユーザから送られてくるリクエストに対するパラメータ形式の設定
@PATH_PARAM	リソースパスに含まれるパラメータを取得する
@QUERY_PARAM	リクエストストリングに対するパラメータを取得
@FORM_PARAM	ボディリクエストに対するパラメータを取得
@MATRX_PARAM	リソースパスに含まれるマトリクスパラメータを取得

4.1.2.1 @GET/@POST/@PUT/@DELETE

@GET/@POST/@PUT/@DELETE はリソースに対する操作を表現したアノテーションであり, 主にメソッドに付加する. これにより, リクエスト受信時には対応したアノテーションが付加されているメソッドが動的に呼び出される. ソースコードは, @GET/@POST/@PUT/@DELETE アノテーションの付加例である. なお, リソースの操作には@HEADER や@LINK などのアノテーションも存在するが基本的に CRUD に対応しやすい@GET/@POST/@PUT/@DELETE がよく使用される.

ソースコード 4.1 @GET/@POST/@PUT/@DELETE アノテーション付加例

```
1 @GET
2 public void method1(){}
3 @POST
4 public void method2(){}
5 @PUT
6 public void method3(){}
7 @DELETE
8 public void method4(){}
```

4.1.2.2 @PATH

@PATH は, リソースパスを記述するためのアノテーションであり, 主にクラスとメソッドに付加する.

クラスに付加された@PATH アノテーションは, クラスを呼び出すために使用され, メソッドに付加された@PATH アノテーションは, メソッドを呼び出すために使用される. クラスに付加された@PATH アノテーションの値はベースパス, メソッドはサブパスと呼ばれる.

リソースパスを表現するためは, ベースパスとサブパスが必要なため, @PATH アノテーションは, クラスとメソッドの 2 つに付加する必要がある.

ソースコードは, @PATH アノテーションを付加した例である. JAX-RS を使用して method1() を呼び出すためには「rest/sample/service1」, method2() を呼び出すためには「rest/sample/test/service」リソースパスを指定する.

ソースコード 4.2 @PATH アノテーション付加例

```
1 @PATH( "rest/sample/" )
2 public class SampleRestService
3 {
4   @PATH( "service1" )
5   public void method1(){}
6
7   @PATH( "test/service" )
8   public void method2(){}
9 }
```

4.1.2.3 @ENCODED

@ENCODED アノテーションは, ユーザからのリクエストやレスポンスの形式を指定するために使用し, クラスに付加する. ソースコードは, @ENCODED アノテーションの付加例である. アノテーションの値には, 文字コード名を指定する.

ソースコード 4.3 @ENCODED アノテーション付加例

```
1 @ENCODED( UTF-8 )
2 public class SampleRestService
3 {
4    . . .
5 }
```

4.1.2.4 @PRODUCES

@PRODUCES アノテーションは, ユーザに返答するレスポンスの形式を指定するために使用し, メソッドに付加する. アノテーションの値には, MIME タイプ名を指定する. なお, @PRODUCES アノテーションを付加しなかった場合, 「application/x-www-form-urlencoded」形式に設定される. ソースコードは, @PRODUCES アノテーションの付加例である. metohd1() の返値は, XML 形式に変換され, method2() の返値は, JSON 形式に変換される.

ソースコード 4.4 @PRODUCES アノテーション付加例

```
1 @PRODUCES( XML )
2 public void method1(){}
3
4 @PRODUCES(JSON)
5 public void method2(){}
```

4.1.2.5 @PATH_PARAM

@PATH_PARAM アノテーションは, リソースパスに含まれる値を取得するために使用し, メソッドの引数に付加される. リソースパスに含まれる値を作成するためには@PATH アノテーションを使用して作成する. ソースコードは, @PATH_PARAM アノテーションの付加例を示したものである.

ソースコードでのリソースパスは「rest/sample/service/{num}」となる. このとき, {} で囲んだ箇所が@PATH_PARAM アノテーションで取得可能な値である.

このソースコードに対して「rest/sample/service/123」というパスでリクエスト要求した場合, 「123」が引数「iNum」に代入される.

なお, 引数型をプリミティブな型に指定しておくことで, 値を自動でキャスト可能だがキャストが失敗した場合, レスポンスコード 404 が返される.

ソースコード 4.5　@PATH_PARAM アノテーション付加例

```
1  @PATH( "rest/sample" )
2  public class SampleRestService
3  {
4    @PRODUCES( XML )
5    @PATH( "service/{num}" )
6    public CCResponse method1( @PATH_PARAM( "num" ) int iNum )
7    {
8       . . .
9    }
10 }
```

4.1.2.6　@QUERY_PARAM

@QUERY_PARAM アノテーションは, クエリストリングの値を取得するために使用し, メソッドの引数に付加される. ソースコードは, @QUERY_PARAM アノテーションの付加例である.

このソースコードに対して「rest/sample/service?key1=123&key2=ABC」というクエリストリングをリソースパスに付加してリクエスト要求した場合, 値「123」が「iValue」に代入され, 値「ABC」が「strValue」に代入される.

なお, クエリストリングに対応するメソッドがない場合は, レスポンスコード 404 が返される.

ソースコード 4.6　@QUERY_PARAM アノテーション付加例

```
1  @PATH( "rest/sample" )
2  public class SampleRestService
3  {
4    @PRODUCES( XML )
5    @PATH( "service" )
6    public CCResponse method1( @QUERY_PARAM( "key1" ) int iValue,
7                    @QUERY_PARAM( "key2" ) String strValue)
8    {
9       . . .
10   }
11 }
```

4.1.2.7　@FORM_PARAM/@CONSUMES

@FORM_PARAM アノテーションは, リクエスト要求時にユーザから送られてくるパラメータを取得するために使用され, メソッドの引数に付加される. また, @CONSUMES アノテーションは, 受け取れるパラメータの形式を指定するために使用され, メソッドに付加される. @CONSUMES アノテーションの値は, MIME タイプ名を指定する. なお, @CONSUMES アノテーションを付加しなかった場合, 「application/x-www-form-urlencoded」形式に設定される.

ソースコードは, @FORM_PARAM アノテーションの付加例である. ボディメッセージに「key1=456&key2=XYZ」を付加して「rest/sample/service」リソースパスに対し

てリクエスト要求した場合, 値「456」が「iValue」に代入され, 値「XYZ」が「strValue」に代入される.

なお, ボディメッセージの値に対応するメソッドがない場合は, レスポンスコード 404 が返される.

ソースコード 4.7 @FORM_PARAM アノテーション付加例

```
1  @PATH( "rest/sample" )
2  public class SampleRestService
3  {
4    @PRODUCES( XML )
5    @CONSUMES( application/x-www-form-urlencoded )
6    @PATH( "service" )
7    public CCResponse method1( @FORM_PARAM( "key1" ) int iValue,
8                    @FORM_PARAM( "key2" ) String strValue)
9    {
10     . . .
11   }
12 }
```

4.1.2.8 @MATRX_PARAM

@MATRX_PARAM アノテーションは, リソースパスに含まれるマトリクス値を取得するために使用し, メソッドの引数に付加される. ソースコードは, @MATRX_PARAM アノテーションの付加例である.

ソースコードに対して「rest/sample;num=123/service」というマトリクス値「num=123」を含むリソースパスにリクエスト要求した場合, 値「123」が引数「iValue」に代入される.

なお, 対応するメソッドが存在しない場合やキャストが失敗した場合, レスポンスコード 404 が返される.

ソースコード 4.8 @MATRX_PARAM アノテーション付加例

```
1  @PATH( "rest/sample" )
2  public class SampleRestService
3  {
4    @PRODUCES( XML )
5    @PATH( "service" )
6    public CCResponse method1( @MATRX_PARAM( "num" ) int iValue )
7    {
8      . . .
9    }
10 }
```

4.1.3 JAX-RS を使用したデバイスの REST サービス例

ソースコードは, 温度センサがあることを仮定して JAX-RS で REST サービスを作成した例である. ソースコードでは, osgi/sensor/temperature リソースを GET メソッドで操作することにより, センサから温度を取得することができる. また, PUT と DELETE

メソッドの場合, センサの電源を ON/OFF にする操作に対応付けを行っている. 更に POST メソッドの場合, 温度の取得周期やタイマ設定といったセンサの設定を変更する操作に対応付けしている.

このように JAX-RS を用いることで簡単にデバイスを REST サービス化できる.

ソースコード 4.9 JAX-RS を使用したデバイスの REST サービス例

```
1  @PATH( "osgi/sensor/" )
2  @ENCODED( CCEncoded .UTF8 )
3  public class CCSensorRes tService
4  {
5    @GET
6    @PRODUCES( CCMediaType.XML )
7    @PATH( "temperature" )
8    public String GetTemperature ()
9    {
10     // 温度センサ用の通信ミドルウェアを
11       呼び出して温度の取得
12   }
13
14   @POST
15   @PRODUCES( CCMediaType.XML )
16   @CONSUMES( CCMediaType.XML )
17   @PATH( " temperature " )
18   public String SetupTemperature(
19     @FORMPARAM( "usr_input" ) String strInputValue )
20   {
21     // 温度センサ用の通信ミドルウェアを
22     呼び出してセンサの設定
23   }
24
25   @PUT
26   @PRODUCES( CCMediaType.XML )
27   @PATH( "temperature" )
28   public String TemperatureSwitchON()
29   {
30     // 温度センサ用の通信ミドルウェアを
31     呼び出して電源ON
32     }
33
34   @DELETE
35   @PRODUCES( CCMediaType.XML )
36   @PATH( "temperature" )
37   public St r ing TemperatureSwitchOFF()
38   {
39     // 温度センサ用の通信ミドルウェアを
40     呼び出して電源OFF
41   }
42    . . .
43 }
```

4.2 Chunked Transfer Coding の適応検討

要求条件 (2)(4) を満たすために, Chunked Transfer Coding を利用したストリーミングインタフェースも検討している [14].

Chunked Transfer Coding とは, RFC2068[15] に定義されている HTTP1.1 からの仕様である. 図 4.3 のようにサイズの大きいレスポンスデータを送信する際に, データを分

割し, 分割データごとにサイズ (16 進数表記) を記してデータを送っていき, 最後に 0byte を送信することで, レスポンス送信完了とする手法である.

そこで, Chunked Transfer Coding を使用してセンシングデータを送信する仕組みを検討した. 図 4.4 は, Chunked Transfer Coding の例であり, 以下はその流れである.

(1) HTTP で Chunked Transfer Coding 通信を確立させる.
(2) (1) で確立した Chunk 通信を使用して受け取りたいセンシングデータを別の HTTP リクエストで要求する. 図 4.4 では, 温度と照度を取得するためのリソースである「streaming/sensor/Temperature」や「streaming/sensor/iLLuminance」にリクエストを送っている.
(3) (2) で要求したリソースに変化があった場合, (1) で確立した Chunk 通信を使用して図 4.3 のように変化後の値を取得可能である.

このように, Chunked Transfer Coding を利用することで定期的にセンシングデータを受け取ることが可能である. また HTTP のみを利用して Chunked Transfer Coding による通信が行えるため, REST と同様に実装も容易であると考えられる.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Transfer-Encoding: chunked

42
{device:”Temperature”, value:27.5}
40
{device:”Illuminance”, value:35}
42
{device:”Temperature”, value:27.4}
0
```

図 4.3 Chunked Transfer Coding 例

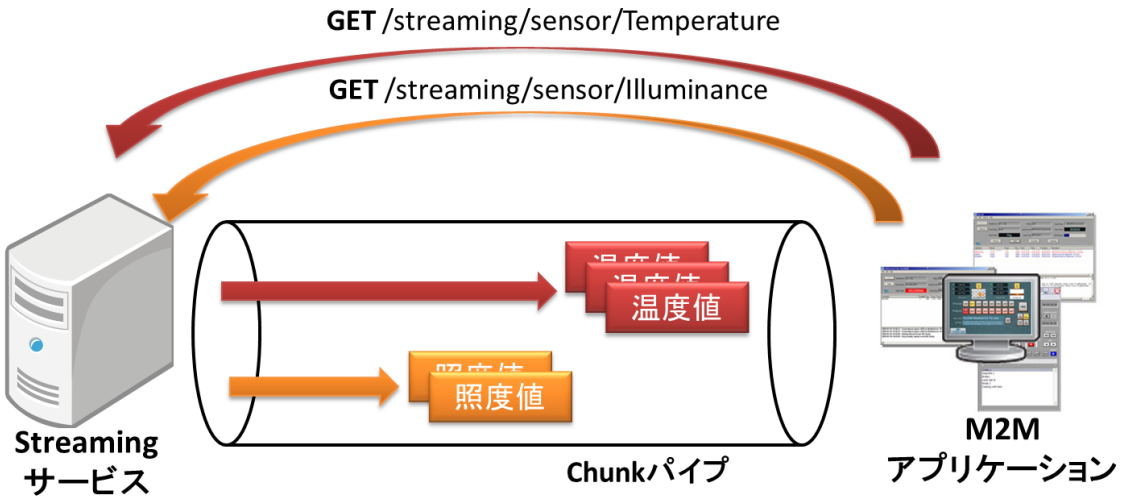

図 4.4　Streaming Service 例

4.3　OSGi の適応検討

要求条件 (3)(5) を満たすために OSGi(Open Services Gateway initiative) を利用することを検討する．OSGi とは, Java モジュールの動的追加や実行を管理するための基盤システム仕様であり, 以下に示すような機能がある [11].

- 図 4.5 のように Bundle と呼ばれるモジュールを動的に追加や削除することが可能である．また, OSGi を遠隔操作することも可能である．
  なお, Bundle の実体は, class や OSGi 用のマニフェストファイルを JAR ファイルにまとめたものである
- Bundle は, OSGi 上でアプリケーションとして動作し, 他の Bundle と通信が可能である．
- Bundle は, 他の OSGi 上でも動作可能である．

また, OSGi は, よくホーム ICT 基盤の HGW(HomeGateWay) Server として様々な家電やセンサネットワークなどを管理するためにも使用されている [12][13].

そこで HGW Server のようなデバイスを管理している OSGi 上に REST アーキテクチャの機能を付加することを検討する．OSGi ならば, ソースコード 4.9 のような JAX-RS アノテーションを使用して作成した REST サービスを Bundle 化することで必要に応じて OSGi 上に追加や削除することが可能である．そのため, M2M アプリケーションの仕様変更が起こった場合でも, 柔軟に対処することが可能であると考えられる．

また, Bundle 同士での通信も可能なため, JAX-RS の機能を提供する Bundle とソースコード 4.9 のような JAX-RS アノテーションを使用して作成した REST サービスを分離することが可能である. そのため, JAX-RS の機能を提供する Bundle を再利用することで, 他の OSGi にも REST アーキテクチャの機能を付加することが可能である.

したがって, OSGi 上で管理されているデバイスに対して REST アーキテクチャで操作が可能になる.

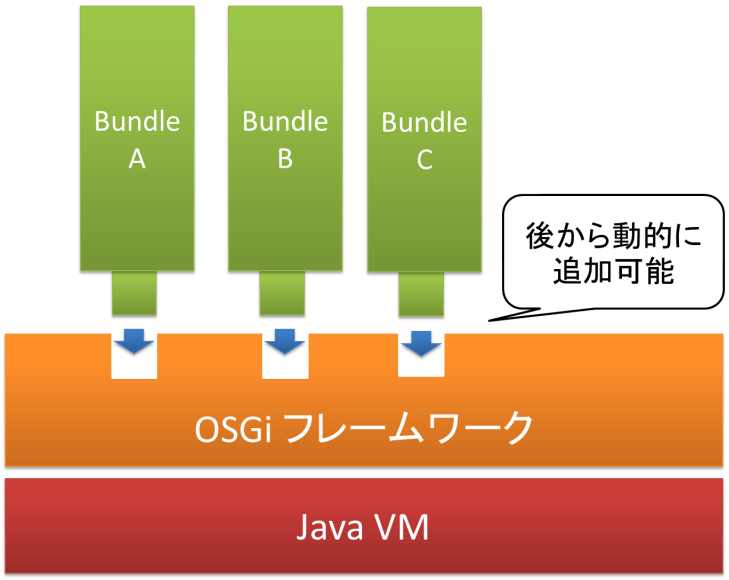

図 4.5 OSGi の仕組み

第 5 章

設計

この章では, 提案システムについて述べる. また, 提案システムを構成する Bundle やユーザからのリクエスト要求があった場合の動作についても述べる.

5.1 提案システム

本研究は, OSGi 上で動作し, REST アーキテクチャの機能を提供する Bundle を実装することで, HGW Server のようなデバイス管理などを行っている OSGi に対して REST アーキテクチャを使用してデバイスを操作できるようにする. 図 5.1 は, 提案システムを示したものである. 提案システムでは, 以下に示す Bundle 群が互いに通信し合うことで構成され, OSGi の種類を問わず動作可能である.

- **REST Protocol Bundle**
  REST アーキテクチャに対応することができるプロトコルを処理する Bundle
- **JAX-RS Bundle**
  REST Protocol Bundle からの内容をもとに REST Service Bundle を呼び出すための Bundle
- **REST Service Bundle**
  JAX-RS アノテーションを利用して作成された REST サービスを含んでいる Bundle
- **通信ミドルウェア Bundle**
  デバイス固有の処理をまとめた Bundle

このうち, 提案システムは, REST Protocol Bundle と JAX-RS Bundle を提供する. したがって, 提案システムを構成するためには, REST Service Bundle, 通信ミドルウェア Bundle 及び M2M アプリケーションの開発が必要になる. そこで, これらを分担して開発するモデルを提案する.

表 5.1 は, REST サービスの追加/修正/削除時における REST Service Bundle, 通信ミドルウェア Bundle 及び M2M アプリケーションの開発対象者を示したものである. これ

により, M2M アプリケーション開発者は, HTTP 処理を実装するだけである. そのため, デバイス固有の処理を知る必要はなく, 比較的開発は容易である. しかし, REST Service Bundle 開発者と通信ミドルウェア Bundle 開発者は, Bundle 開発の知識が必要なため, 開発は比較的困難になる. ただし, REST Service Bundle 開発者は, アノテーションを付加したコードを作成するだけなため, デバイス固有の処理を作成しなければならない通信ミドルウェア Bundle 開発者より, 開発は容易である. このとき, REST Service Bundle 開発者が, アノテーションを適切に付けるためには, REST によるデバイスのリソースやそれを操作するためのメソッドを定義しておく必要がある. この作業を円滑に行うためには, デバイスの機能を知っている通信ミドルウェア Bundle 開発者が機能抽出を行っておくことが望ましいと考えられる.

このように開発の難易度や開発者のスキルにより, REST Service Bundle, 通信ミドルウェア Bundle 及び M2M アプリケーションの開発者を分離することが可能である.

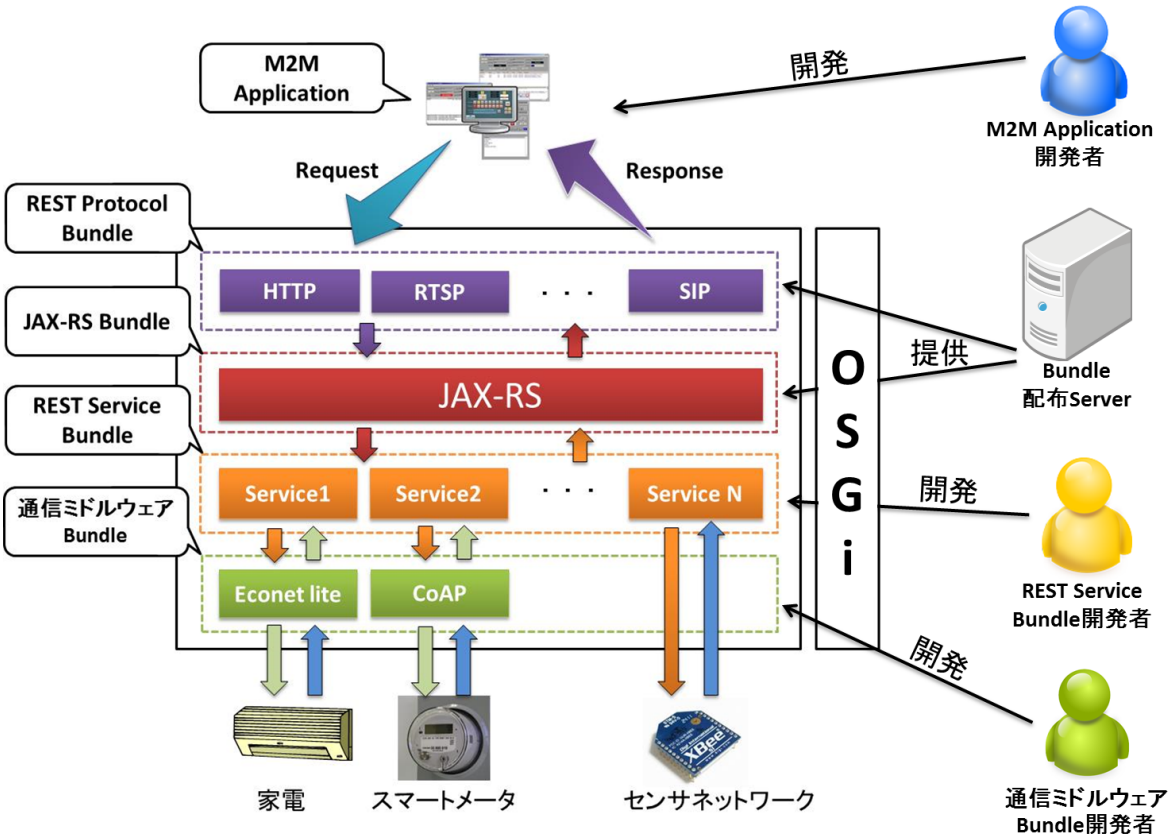

図 5.1 提案システム

表 5.1 提案システムを使用したときの開発対象

開発者	M2M アプリケーション開発者	REST Service Bundle 開発者	通信ミドルウェア Bundle 開発者
REST サービスの追加	HTTP の GET や POST などでリソースに要求を出す処理を作成	アノテーションを付加したソースコードを Bundle としてインストール	デバイス固有の処理と Bundle を作成してインストール
REST サービスの修正	HTTP の GET や POST などでリソースに要求を出す処理を修正	アノテーションを修正して Bundle を更新	デバイス固有の処理を修正し, Bundle を更新
REST サービスの削除	HTTP の GET や POST などでリソースに要求を出す処理を削除	アノテーションの削除または Bundle をアンインストール	Bundle をアンインストール
開発の難易度	HTTP 通信を実装するだけなため比較的容易	アノテーションを付加したコードの作成は容易だが Bundle の知識が必要である. また REST による通信設計の知識が必要である.	デバイス固有の処理と Bundle の知識が必要なため比較的困難である. また REST リソース定義のためデバイスの機能を抽出する必要がある.

5.2 Bundle 間通信

各 Bundle は, OSGi の Bundle 間通信によって接続される. Bundle 間通信とは, 各 Bundle が提供する機能 (以降, OSGi サービスと呼ぶ) を OSGi が持つサービス管理表に登録しておき, この表を参照することで, 他の Bundle が OSGi サービスを利用することができる OSGi の仕組みのことである.

なお, 以降は OSGi のサービス管理表に OSGi サービスを登録して, 自身のサービスを他の Bundle に提供する Bundle のことを OSGi サービス提供側 Bundle と呼び, 他の Bundle のサービスを利用する Bundle のことを OSGi サービス利用側 Bundle と呼ぶ.

このとき, Bundle 間通信を行う方法として, OSGi サービス利用側 Bundle が, OSGi のサービス管理表から必要なサービスを検索して取得する方法と, OSGi サービス提供側 Bundle が, OSGi サービス管理表に OSGi サービスを登録したタイミングで他の Bundle に通知する方法があるので説明する.

まず, 図 5.2 は, 必要に応じて OSGi サービス利用側の Bundle が OSGi サービスを検索する動作を示したものであり, 以下はその流れである.

(1) OSGi サービス提供側の Bundle をインストールする
(2) OSGi サービス提供側の Bundle が OSGi のサービス管理表にサービスを登録する
(3) OSGi サービス利用側の Bundle をインストールする
(4) OSGi サービス利用側の Bundle が OSGi のサービス管理表を参照して, 必要な OSGi サービスを検索する
(5) OSGi サービス利用側が必要な OSGi サービスを取得する

次に, 図 5.3 は, OSGi サービス提供側 Bundle が他の Bundle に通知を行う動作を示したものであり, 以下はその流れである.

(1) OSGi サービス利用側の Bundle をインストールする
(2) OSGi サービス提供側の Bundle をインストールする.
(3) OSGi サービス提供側の Bundle が OSGi のサービス管理表にサービスを登録する
(4) OSGi が他の Bundle にサービスが登録されたことを通知する
(5) OSGi サービス利用側が必要な OSGi サービスを取得する

この 2 つの仕組みを用いることにより, OSGi では Bundle がどのような順番でインストールされても必要なサービスを利用することが可能である.

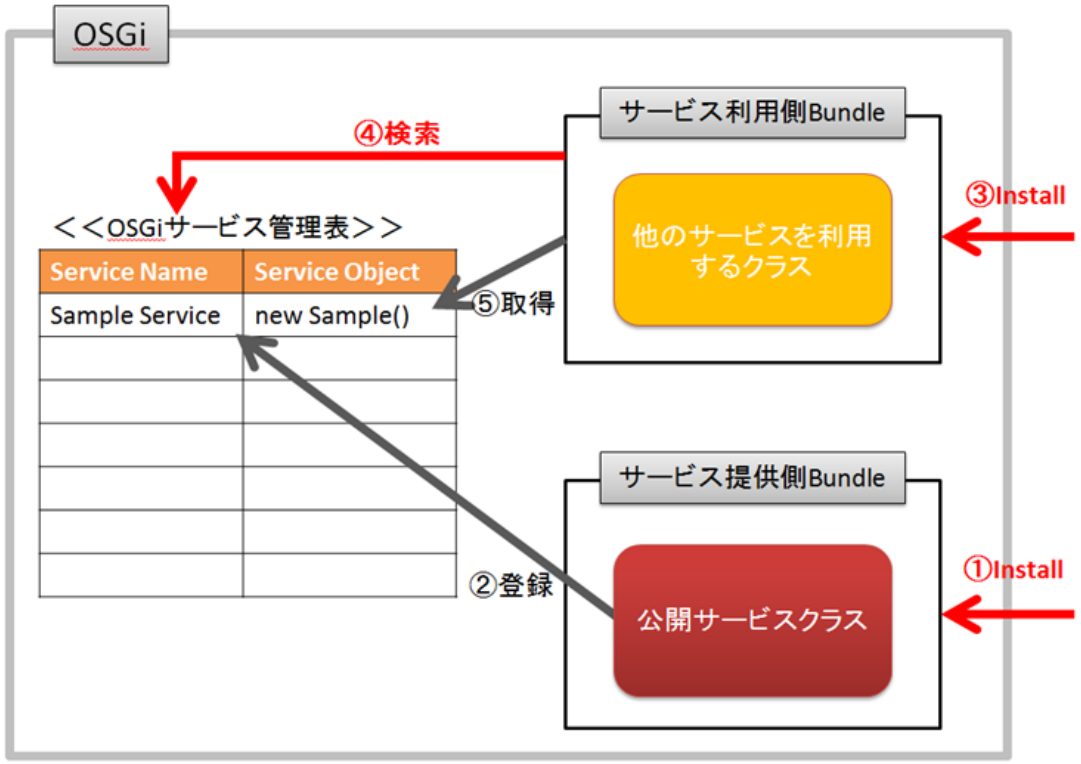

図 5.2　検索により OSGi サービスの取得

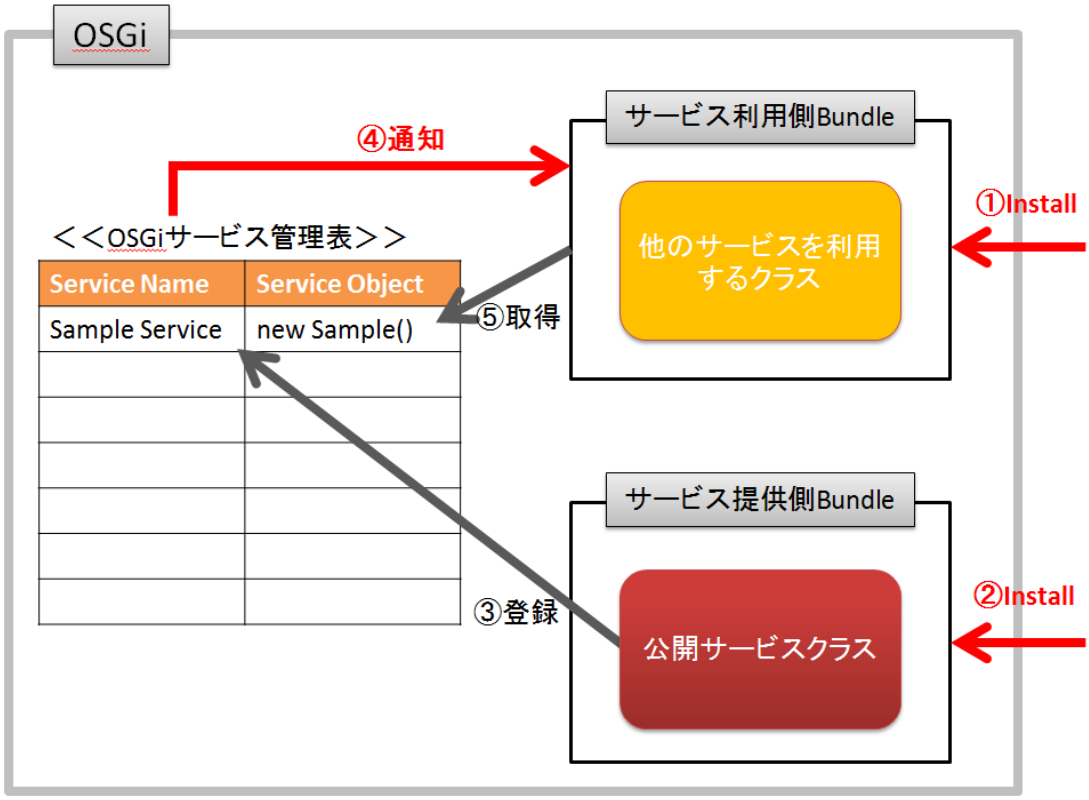

図 5.3　通知により OSGi サービスの取得

5.3 各 Bundle の動作

Bundle 間通信を使用して, REST Protocol Bundle, JAX-RS Bundle, REST Service Bundle の動作について説明する.

5.3.1 REST Protocol Bundle

REST Protocol Bundle は, REST アーキテクチャに対応することができるプロトコルを処理する Bundle である. REST に対応できるプロトコルには, HTTP や RTSP(Real Time Streaming Protocol), SIP(Session Initiation Protocol) などが挙げられる. 今回は, HTTP で一般的な REST アーキテクチャを行う. そのため, 以後は HTTP Bundle と呼ぶ.

図 5.4 は, JAX-RS Bundle がインストールされていなかった状態で, 事前に HTTP Bundle をインストールした場合の動作を示したものであり, 以下はその流れである.

(1) OSGi に HTTP Bundle をインストールする. このとき, JAX-RS Bundle がインストールされるまで待機する.
(2) OSGi に JAX-RS Bundle をインストールする.
(3) JAX-RS Bundle が OSGi サービス管理表に自信の OSGi サービスを登録する.
(4) OSGi が JAX-RS Bundle がサービスを登録したことを HTTP Bundle に通知する.
(5) HTTP Bundle が JAX-RS Bundle のサービスを取得する.

図 5.5 は, JAX-RS Bundle がインストールされていた状態で, HTTP Bundle を事後にインストールした場合の動作を示したものであり, 以下はその流れである.

(1) OSGi に JAX-RS Bundle をインストールする.
(2) JAX-RS Bundle が OSGi サービス管理表に自信の OSGi サービスを登録する.
(3) OSGi に HTTP Bundle をインストールする.
(4) HTTP Bundle が JAX-RS Bundle が登録されているか OSGi サービス管理表に問い合わせを行う
(5) HTTP Bundle が JAX-RS Bundle のサービスを取得する.

これにより, HTTP Bundle と JAX-RS Bundle の通信が開始される.

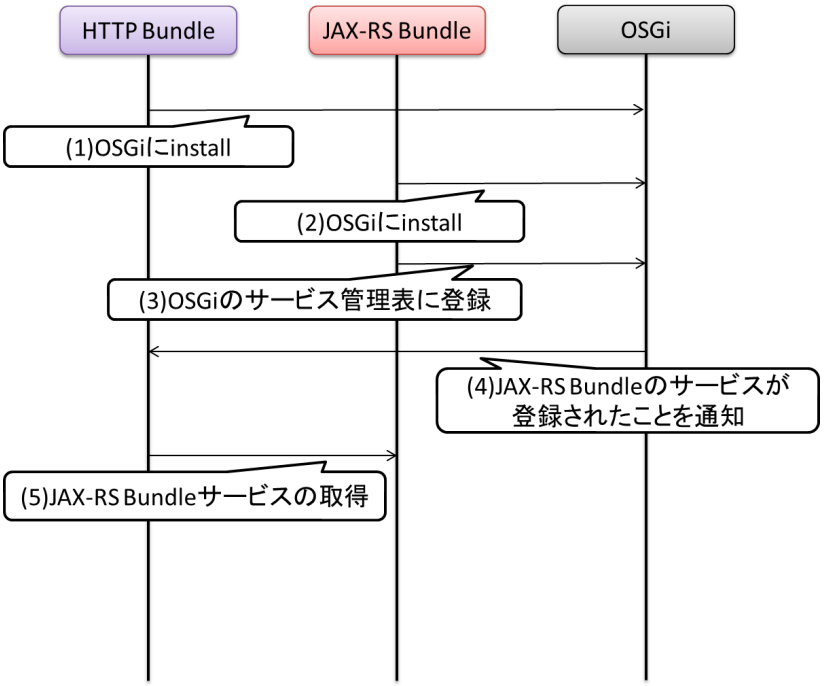

図 5.4　HTTP Bundle 事前インストール時の動作シーケンス図

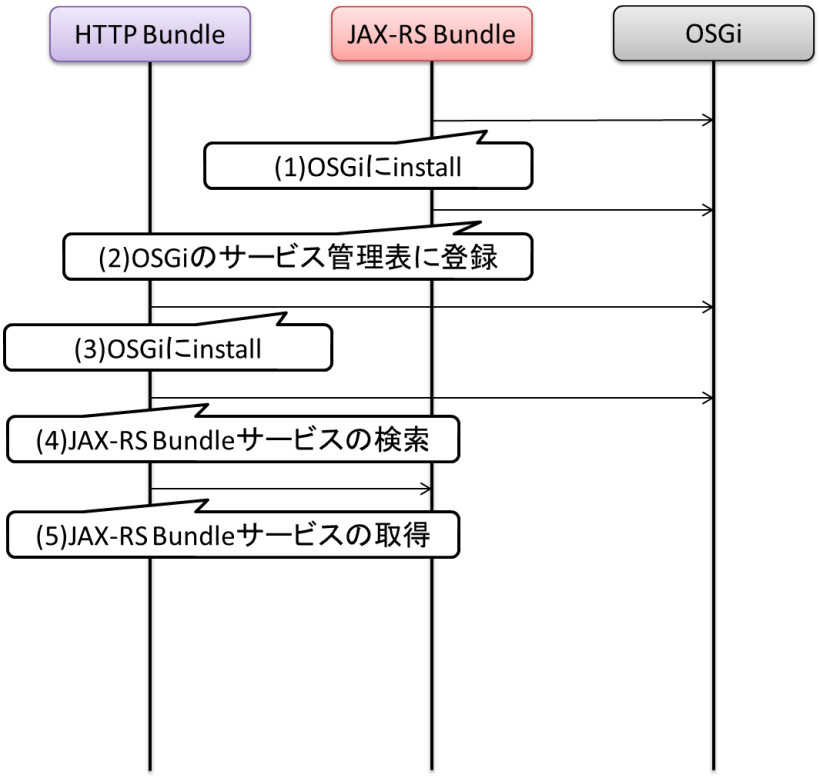

図 5.5　HTTP Bundle 事後インストール時の動作シーケンス図

5.3.2 JAX-RS Bundle

JAX-RS Bundle は, REST Protocol Bundle からの内容をもとに REST Service Bundle を呼び出すための Bundle である.

JAX-RS Bundle は, ソースコードのようなインタフェースを OSGi に登録し, 自身にアクセスする仕組みを REST Protocol Bundle に提供している. インタフェースには GET/POST/PUT/DELETE メソッドがあり, 共通の第一引数にはリソースパスを指定し, POST/PUT/DELETE メソッドの第二引数には, リクエスト実行時に必要なパラメタを指定する.

このインタフェースを使用することで HTTP だけでなく, RTSP や SIP といった REST に対応することができるプロトコルを Bundle 化することが可能である. また, このインタフェースを使用することで, Bundle として M2M アプリケーションを OSGi 上に実装することも可能である.

ソースコード 5.1 JAX-RS Bundle のインタフェース

```
1  package RESTInterface;
2
3  import java.util.HashMap;
4
5  public interface IFRESTInterface
6  {
7    public CCRESTResponse GET( String strResourcePath ) ;
8    public CCRESTResponse POST( String strResourcePath,
9        HashMap<String, String> Parms ) ;
10   public CCRESTResponse PUT( String strResourcePath,
11       HashMap<String, String> Parms ) ;
12   public CCRESTResponse DELETE( String strResourcePath,
13       HashMap<String, String> Parms ) ;
14 }
```

5.3.3 REST Service Bundle

REST Service Bundle は, JAX-RS のアノテーションを利用して作成された REST サービスを含んでいる Bundle である. ユーザからの REST リクエストをもとに JAX-RS Bundle から呼び出され, サービス内容に応じて適切な通信ミドルウェア Bundle を呼び出す役目がある.

図 5.6 は, JAX-RS Bundle がインストールされていない状態で, REST Service Bundle を事前にインストールした場合の動作を示したものであり, 以下はその流れである.

(1) OSGi に REST Service Bundle をインストールする.
(2) REST Service Bundle が OSGi にサービスを登録する.
(3) OSGi に JAX-RS Bundle をインストールする.

(4) JAX-RS Bundle が OSGi サービス管理表に現在登録されている全てのサービスを検索する.
(5) 全てのサービスを取得する.
(6) 全てのサービスのクラスに付加されている@PATH アノテーションだけを解析する.
(7) @PATH アノテーションが付加されているクラスを発見した場合, リクエスト要求時にサービスの呼び出しを高速に行えるように, OSGi のサービス管理表にリソースパス列を追加して, @PATH アノテーションの値を設定する.

図 5.7 は, JAX-RS Bundle がインストールされていた状態で, REST Service Bundle を事後にインストールした場合の動作を示したものであり, 以下はその流れである.

(1) OSGi に JAX-RS Bundle をインストールする.
(2) OSGi に REST Service Bundle をインストールする.
(3) REST Service Bundle が OSGi にサービスを登録する.
(4) OSGi が JAX-RS Bundle に対して新しいサービスが登録されたことを通知する.
(5) JAX-RS Bundle が新しく登録されたサービスを取得する.
(6) 取得したサービスのクラスに付加されている@PATH アノテーションだけを解析する.
(7) @PATH アノテーションが付加されているクラスを発見した場合, リクエスト要求時にサービスの呼び出しを高速に行えるように, OSGi のサービス管理表にリソースパス列を追加して, @PATH アノテーションの値を設定する.

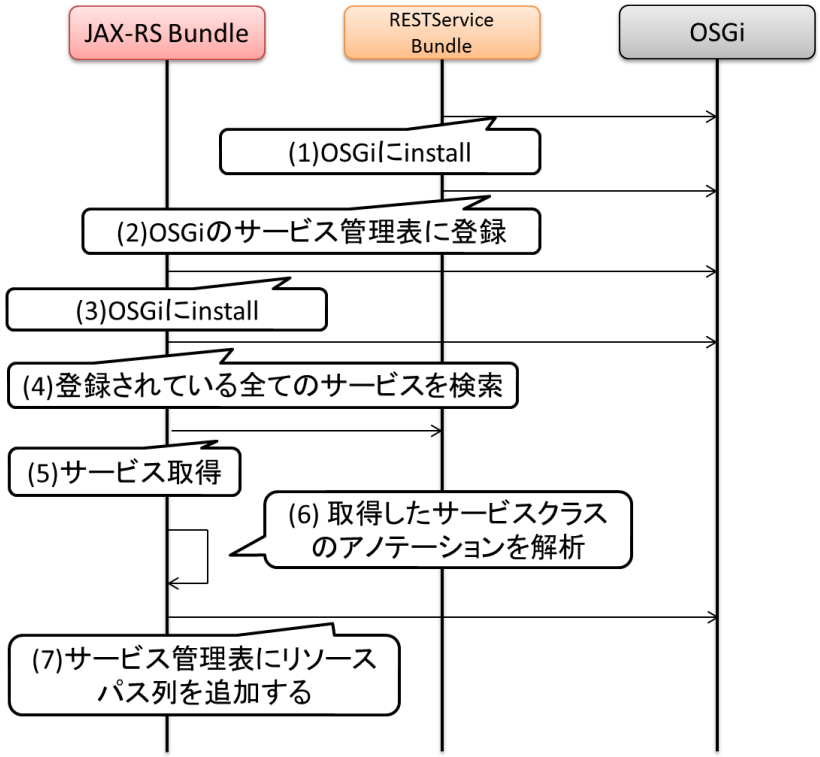

図 5.6　REST Service Bundle 事前インストール時の動作シーケンス図

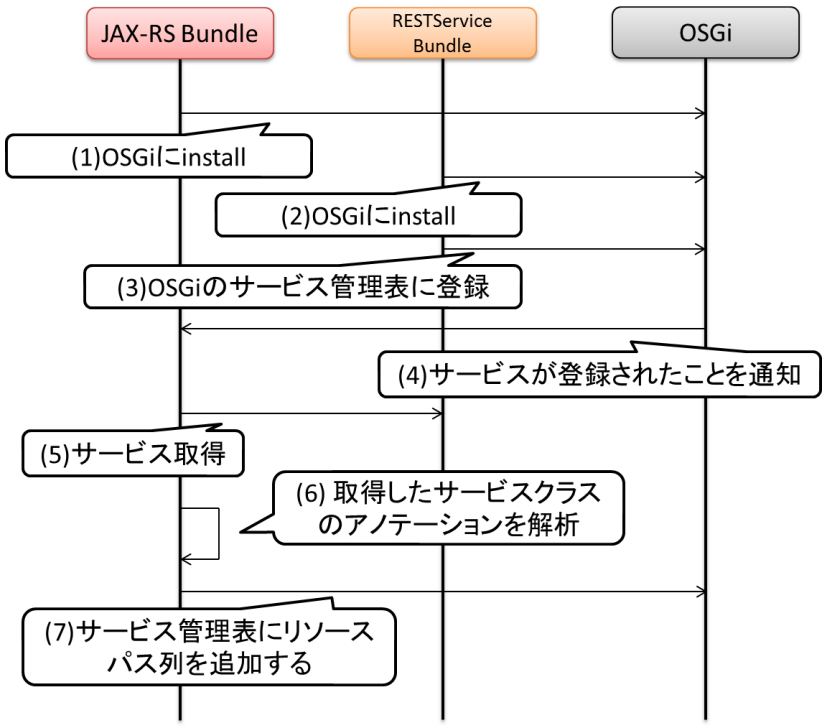

図 5.7　REST Service Bundle 事後インストール時の動作シーケンス図

5.4 リクエスト時の動作

図 5.8 は, ユーザから REST リクエストが来た場合の動作を示したものであり, 以下は, その流れを示したものである.

(1) M2M アプリケーションが HTTP リクエストを送信
(2) HTTP Bundle が JAX-RS Bundle に HTTP リクエストを送信
(3) HTTP リクエストから REST メッセージ (HTTP メソッドやリソースパスなど) を取り出す.
(4) リソースパスを使用して OSGi のサービス管理表から REST サービスを検索する.
(5) REST サービスを取得する.
(6) (3) の REST メッセージを使用して, 取得した REST サービスクラスのメソッドに付加されたアノテーションを解析してメソッドを取得する. もしメソッドが見つからなかった場合, レスポンスコード 404 を M2M アプリケーションに返答する.
(7) 取得したメソッドを実行する.
(8) REST Service Bundle がメソッドを実行するために必要な通信ミドルウェア Bundle を OSGi から検索する.
(9) メソッド実行に必要なサービスを通信ミドルウェア Bundle から取得する.
(10) メソッド実行結果を JAX-RS Bundle が受け取る.
(11) JAX-RS Bundle が HTTP Bundle にメソッド実行結果を送信する.
(12) メソッド実行結果をもとに HTTP レスポンスを作成する.
(13) M2M アプリケーションに HTTP レスポンスを返答する.

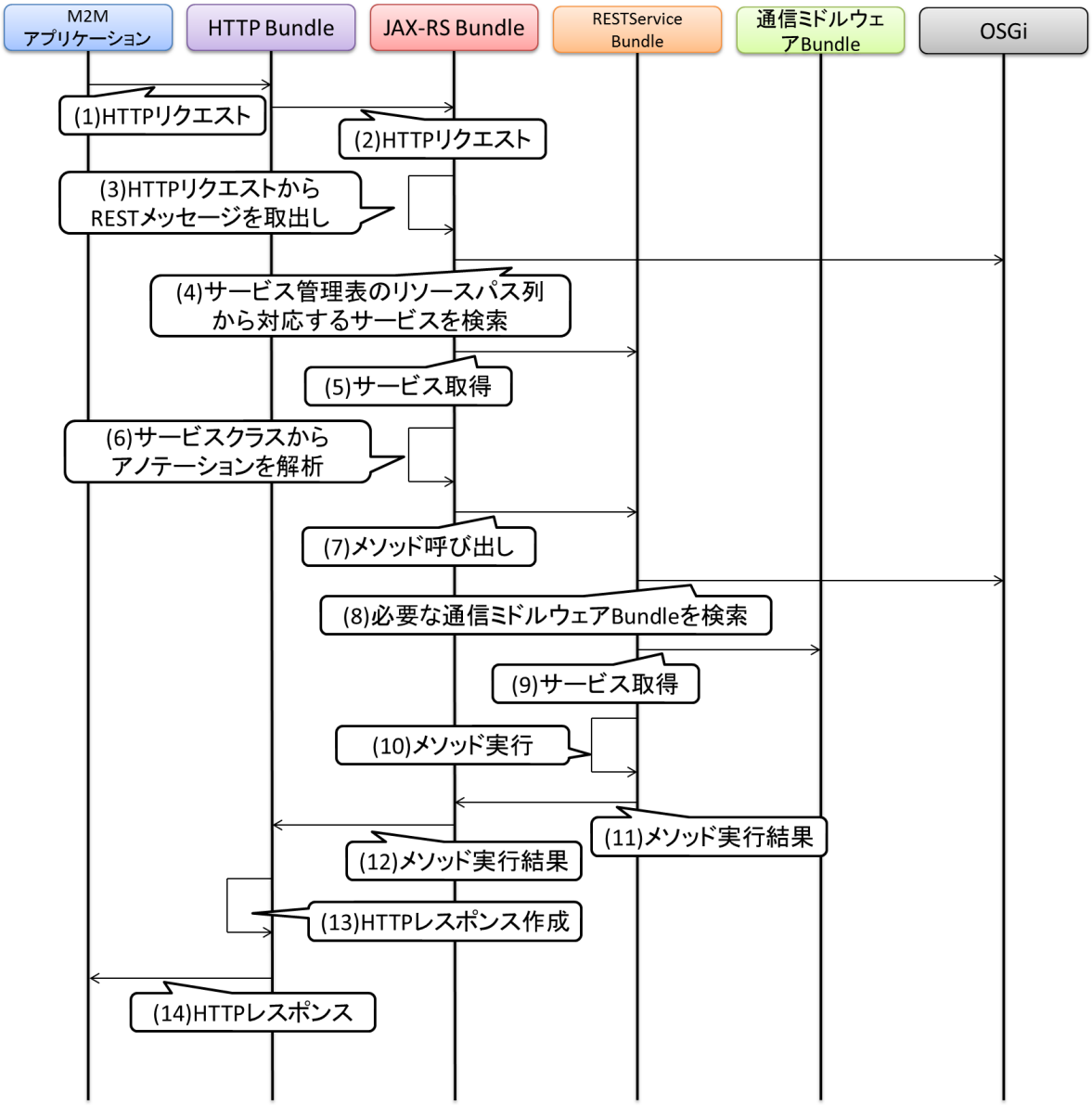

図 5.8　リクエスト時の動作シーケンス図

5.5　ストリーミング時の動作

ストリーミングを行うために, JAX-RS のアノテーションをもとに作成された REST サービスがあり, 表 5.2 に示すリソースを持っている.

このサービスでは,「/osgi/rest/stream」リソースに対して PUT メソッドを実行するとストリームが作成され, StreamID が返される. 以後は, この ID を使用してストリームを操作する. ストリームを停止させるには,「/osgi/rest/stream」リソースに ID を付加して DELETE メソッドを実行する.

表 5.2 Streaming サービスリソース

メソッド	リソース	概要
PUT	/osgi/rest/stream	ストリームを作成し, StreamID を返答する
DELETE	/osgi/rest/stream/{StreamID}	StreamID で指定したストリームを停止する

図 5.9 は, ストリーミングを行うための Chunk 接続確率までの動作を示したものであり, 以下は, その流れを示したものである.

(1) M2M アプリケーションが Chunk リクエストを送信する.
(2) HTTP Bundle が JAX-RS Bundle に REST メッセージを送信する.
(3) REST メッセージをもとに OSGi サービス管理表から Streaming サービスを検索する.
(4) Streaming サービスを取得する.
(5) アノテーションを解析して Stream 登録メソッドを取得する.
(6) Stream 登録メソッドを呼び出す.
(7) Stream 登録処理を実行する.
(8) 実行結果として StrreamID が生成され, JAX-RS Bundle に渡される.
(9) JAX-RS Bundle が StreamID を HTTP Bundle に送信する.
(10) Body に StreamID を含む HTTP レスポンスを作成する.
(11) M2M アプリケーションに StreamID を含む HTTP レスポンスが返される.

ストリームの作成が完了した場合, ストリームで流したいリソースを実行する. 実行するためには, 表 5.3 で示すようにリソースパスに StreamID を付加して GET リクエストを実行する. このとき, ストリームで流すことができるリソースは, GET メソッドで実行できるリソースのみに限られる. また, ストリームに流すことを停止したい場合は, DELETE メソッドを実行する.

表 5.3　Streaming リクエストリソース例

メソッド	リソース	概要
GET	/sensor/temperature/{StreamID}	ストリームで温度値を送信
GET	/sensor/humidity/{StreamID}	ストリームで湿度値を送信
GET	/sensor/illuminance/{StreamID}	ストリームで照度値を送信
DELETE	/sensor/temperature/{StreamID}	温度値の送信を停止する
DELETE	/sensor/humidity/{StreamID}	湿度値の送信を停止する
DELETE	/sensor/illuminance/{StreamID}	照度値の送信を停止する

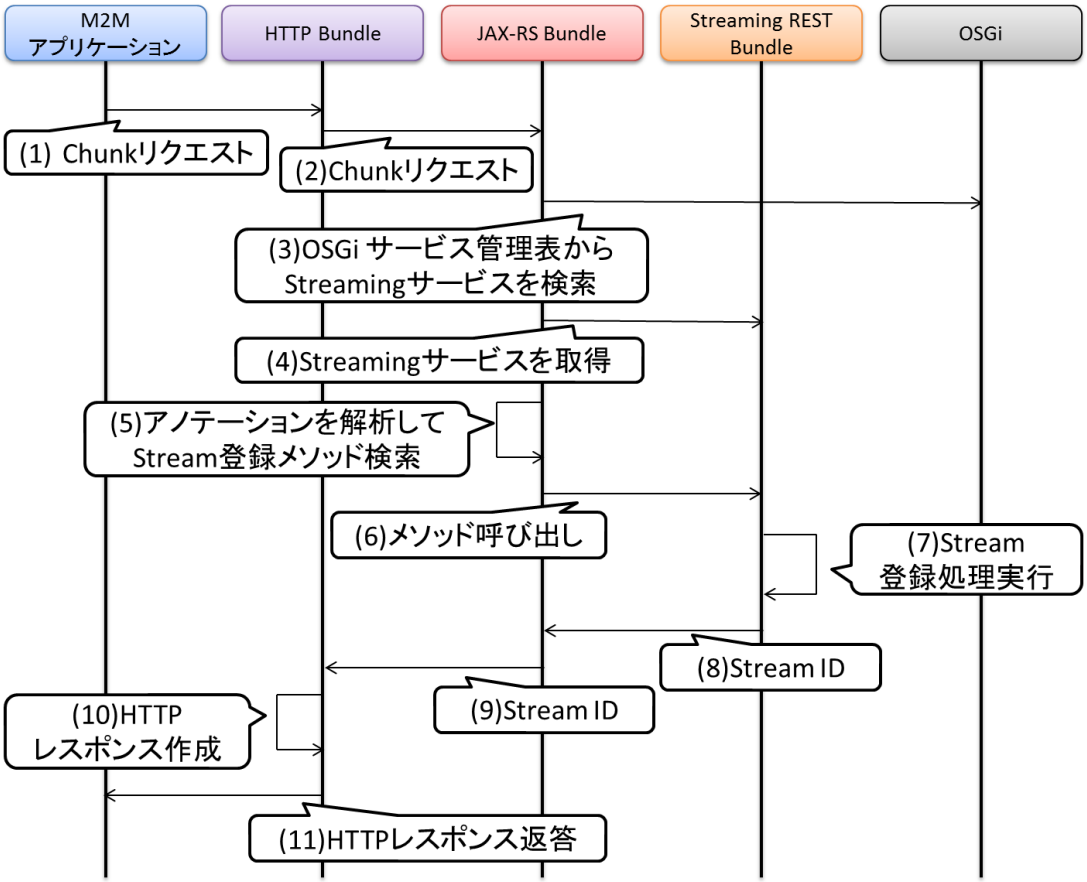

図 5.9　ストリーミング時の動作シーケンス図

第 6 章

実装

この章では, 提案システムの実装について説明する. まず, Bundle の開発方法について述べる. その後, アノテーションの作成方法とアノテーションを解析するためのリフレクション API について述べる.

6.1 開発環境

表 6.1 は, 開発環境を示したものである. OSGi や JAX-RS は Java 仕様であるため, 開発言語は Java を使用した. また, OSGi の Bundle 開発を行い易くするために Eclipse IDE を使用した. 動作確認のための OSGi として Apache Felix を使用した.

表 6.1 開発環境

言語	Java SE 7
OS	Windows 7 64bit
IDE	Eclipse IDE 4.3.2
OSGi	Apache Felix 4.2.1

6.2 OSGi の準備

6.2.1 OSGi インストール

OSGi には, Equinox[16], Apache Felix[17], Knoplersh[18] などといったオープンソースの実装があり, 誰でも使用することができる. 今回は, 利用が簡単な OSGi である Apache Felix の利用方法について説明する [19]. まず, 導入するためには Apache Felix のパッケージを「http://felix.apache.org/」からダウンロードする必要がある.

次にダウンロードが完了したら任意のディレクトリに展開し, Felix Shell の起動を起動する. なお Felix Shell を起動するためには felix のホームディレクトリに移動し, java -jar コマンドより実行すると図 6.1 のような Felix Shell が起動する.

```
$ cd /opt/Java/felix-4.2.1
$ java -jar bin/felix.jar

Welcome to Felix.
=================
->
```

図 6.1　Felix Shell

6.2.2　コマンド一覧の表示

OSGi のコマンドを知るためには,「help」コマンドを実行する. Felix Shell が起動している状態で「help」を入力すると図のように現在使用可能であるコマンドの説明が表示される. 図 6.2 は「help」コマンドを実行した例である.

```
-> help
bundlelevel <level> <id> ... — <id> - set or get bundle start level.
cd [<base-URL>] - change or display base URL.
headers [<id> ...] - display bundle header properties.
help - display impl commands.
install <URL> [<URL> ...] - install bundle(s).
obr help - OSGi bundle repository.
packages [<id> ...] - list exported packages.
ps [-l — -s — -u] - list installed bundles.
refresh [<id> ...] - refresh packages.
resolve [<id> ...] - attempt to resolve the specified bundles.
services [-u] [-a] [<id> ...] - list registered or used services.
shutdown - shutdown framework.
start <id> [<id> <URL> ...] - start bundle(s).
startlevel [<level>] - get or set framework start level.
stop <id> [<id> ...] - stop bundle(s).
uninstall <id> [<id> ...] - uninstall bundle(s).
update <id> [<URL>] - update bundle.
version - display version of framework.
```

図 6.2　OSGi help コマンド

6.2.3 Bundle のインストールとアンインストール

Apache Felix に Bundle をインストールするためには,「install」コマンドを実行する.

インストールコマンドの引数には「file://」または「http://」を使用して Bundle アーカイブのパスを指定する. インストールが成功すると Bundle ID が振り分けられ, 以後は, この ID を使用して Bundle を操作する.

Apache Felix から Bundle をアンインストールするためには「uninsatll」コマンドを実行する.「uninstall」コマンドの引数には, install 時に取得できる Bundle ID を指定する.

図 6.3 は「install」と「uninsatll」コマンドの実行例である.

```
-> install file:C:/Users/Desktop/plugins/HTTP_Bundle.jar
Bundle ID: 5


-> uninstall 5
HTTP Bundle uninstall
```

図 6.3 OSGi install と uninstall コマンド

6.2.4 Bundle 一覧の取得

現在, Apache Felix にインストールされている Bundle の一覧を取得するためには「lb」コマンドを実行する. 図 6.4 は「lb」コマンドを実行した例である.「lb」コマンドを使用することで Bundle ID と各 Bundle の状態を取得することが可能である.

```
-> lb
START LEVEL 1
ID State Level Name
< 0> <Active > < 0> System Bundle (1.4.1)
< 1> <Active > < 1> Apache Felix Shell Service (1.0.2)
< 2> <Active > < 1> Apache Felix Shell TUI (1.0.2)
< 3> <Active > < 1> Apache Felix Bundle Repository (1.2.1)
< 5> <Installed > < 1> HTTP Bundle (1.0.0)
```

図 6.4 OSGi lb コマンド

6.2.5 Bundle の開始と停止

Bundle を開始するためには,「start」コマンドを実行する.「start」コマンドの引数には,「install」コマンド実行時に取得することができる Bundle ID を指定する. なお, イ

ンストールされたときの Bundle の状態は「Installed」になっており, 開始するためには「start」コマンドを実行する必要がある. Bundle を停止させる場合は「stop」コマンドを実行する. 図 6.5 は, 「start」「stop」コマンドを実行した例である. 「stop」コマンドの引数も, 「start」コマンドを同様に Bundle ID を指定する. なお, 引数を「0」に指定すると Felix Shell を終了することが可能である.

```
-> start 5
HTTP Bundle start

-> stop 5
HTTP Bundle stop
```

図 6.5 OSGi start と stop コマンド

6.2.6 Bundle の更新とリフレッシュ

Bundle の更新が必要になった場合, 「update」コマンドを実行する. Bundle を更新するためには, 「install」コマンドを実行したときに指定した Bundle のパスに新しい Bundle アーカイブを設置する必要がある. 引数に Bundle ID を指定することで新しい Bundle に置き換わる. また, Bundle パスを変更したい場合は, パスを直接入力することで可能である.

Bundle を再起動したい場合は, 「refresh」コマンドを実行する. 「refresh」コマンドは, 引数に Bundle ID を指定することで, リフレッシュすることが可能である.

図 6.6 は, 「update」と「refresh」コマンドを実行した例である.

```
-> update 5
HTTP Bundle UPDATE

-> update 5 htpp://127.0.0.1/example1/target/HTTP_Bundle.jar
HTTP Bundle UPDATE

-> refresh 5
HTTP Bundle refresh
```

図 6.6 OSGi update コマンド

6.3 Bundle 開発

6.3.1 Eclipse SDK のインストール

Eclipse IDE を使用して OSGi の Bundle を開発するためには, Eclipse SDK をインストールする必要がある. 以下は, インストールの手順である.

(1) Eclipse IDE を起動する.
(2) Menu バーから「ヘルプ」を選択する.
(3) 「新規ソフトウェアのインストール」を選択する.
(4) 図 6.7 のような画面が表示されるので作業対象欄に
「http://download.eclipse.org/eclipse/updates/{Version}」を入力する.
(5) リストボックス内に Eclipse SDK が表示されるのでチェックボックスにチェックを入れる.
(6) 「次へ」のボタンを押下してインストールする.

これにより, Eclipse IDE を使用して OSGi の Bundle を開発するための準備が完了する.

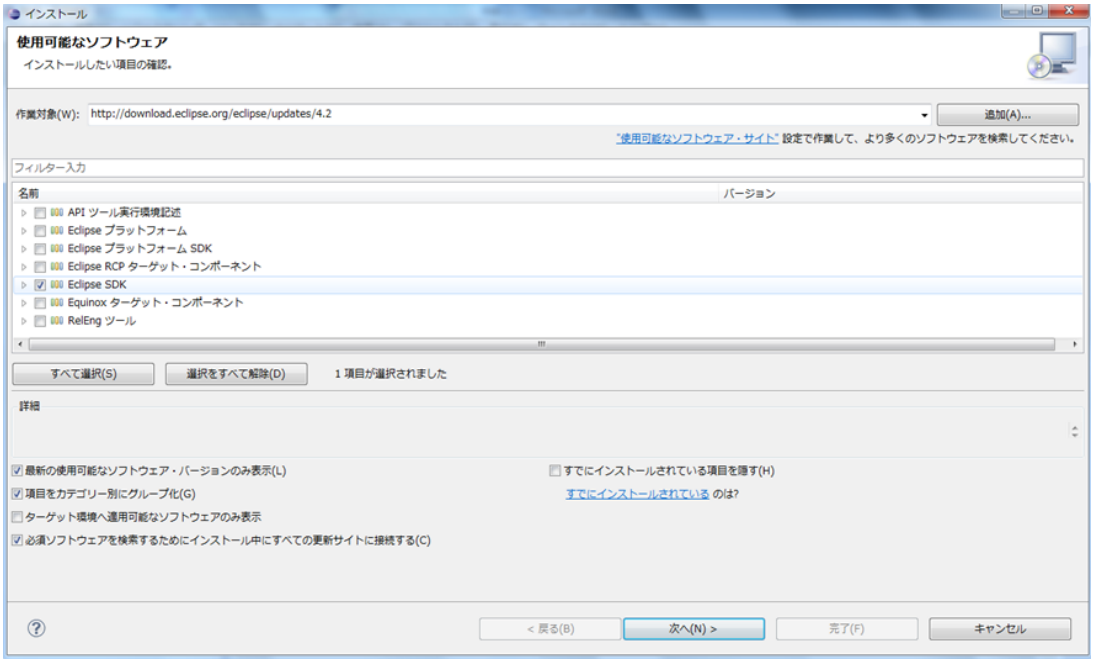

図 6.7 Eclipse SDK インストール画面

6.3.2 Plugin プロジェクトの作成

Eclipse SDK のインストールが完了した場合, Plugin プロジェクトを作成する. 以下は Plugin プロジェクト作成の流れである.

(1) Eclipse の Menu バーより「ファイル」を選択する.
(2) 「新規」を選択する.
(3) 「プロジェクト」を選択する.
(4) 図 6.8 のような新規プロジェクトウィザードが表示されるので「プラグイン・プロジェクト」を選択して「次へ」のボタンを押下する.
(5) 図 6.9 の画面でプロジェクト名を入力して, ターゲットプラットフォームの項で OSGi フレームワークを選択し,「次へ」のボタンを押下する.
(6) 図 6.10 のようなテンプレート画面で「Hello OSGi バンドル」を選択し, 「完了」ボタンを押下すると Plugin プロジェクトが作成される.

Plugin プロジェクトの作成が完了した場合, プロジェクト内にソースコード 6.1 のような Activator.java が自動で作成される. これは Bundle のエントリ・ポイントコードであり, 以降 Bundle で行う処理は, Activator.java に記述していくことになる.

Activator.java には, start メソッドと stop メソッドがあらかじめ実装されている. start メソッドは, OSGi に Bundle がインストールまたは開始コマンドが実行された場合に呼び出される. stop メソッドは, OSGi から Bundle がアンインストールまたは停止コマンドが実行された場合に呼び出される.

ソースコード 6.1 Bundle エントリ・ポイント (Activator.java)

```
1  import org.osgi.framework.BundleActivator;
2  import org.osgi.framework.BundleContext;
3
4  public class Activator implements BundleActivator
5  {
6    public void start(BundleContext context) throws Exception
7    {
8      System.out.println( "Bundle開始" ) ;
9    }
10   public void stop(BundleContext context) throws Exception
11   {
12     System.out.println( "Bundle停止" ) ;
13   }
14 }
```

6.3.3 OSGi サービスの作成と登録

Plugin プロジェクトを使用して Activator.java の生成が完了した場合, Bundle が行う処理を記述する. このとき, Bundle が行う処理を他の Bundle に提供したい場合, OSGi

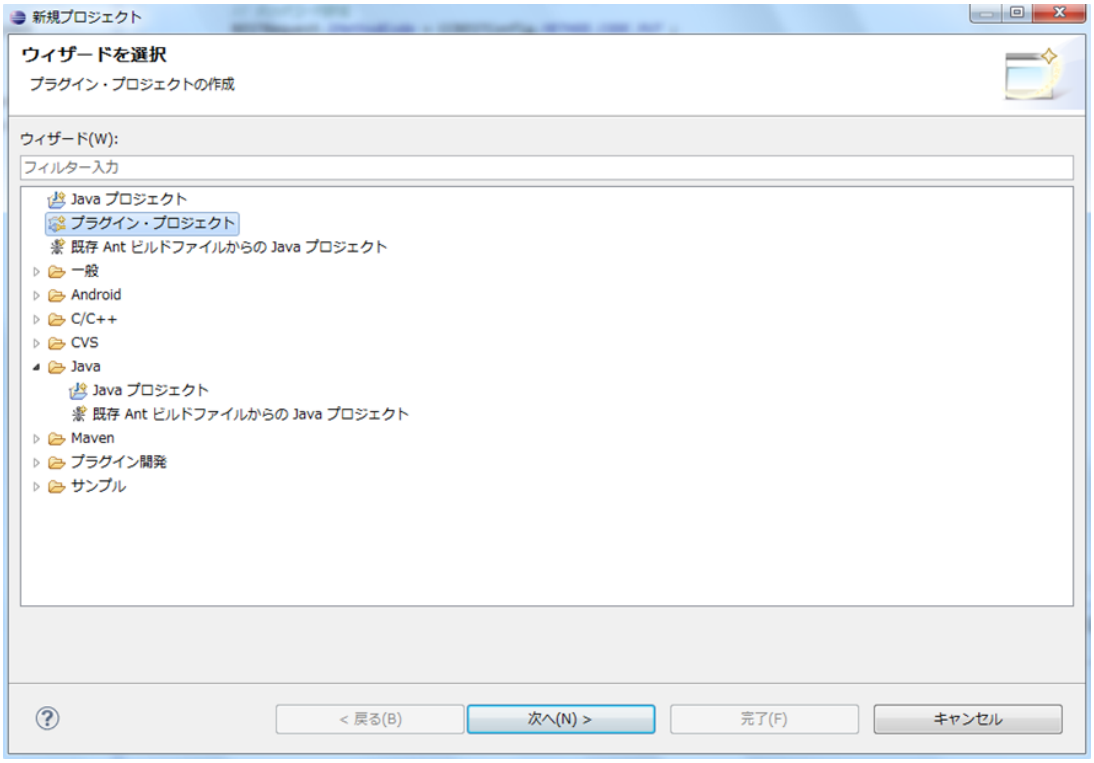

図 6.8　新規プロジェクト作成画面

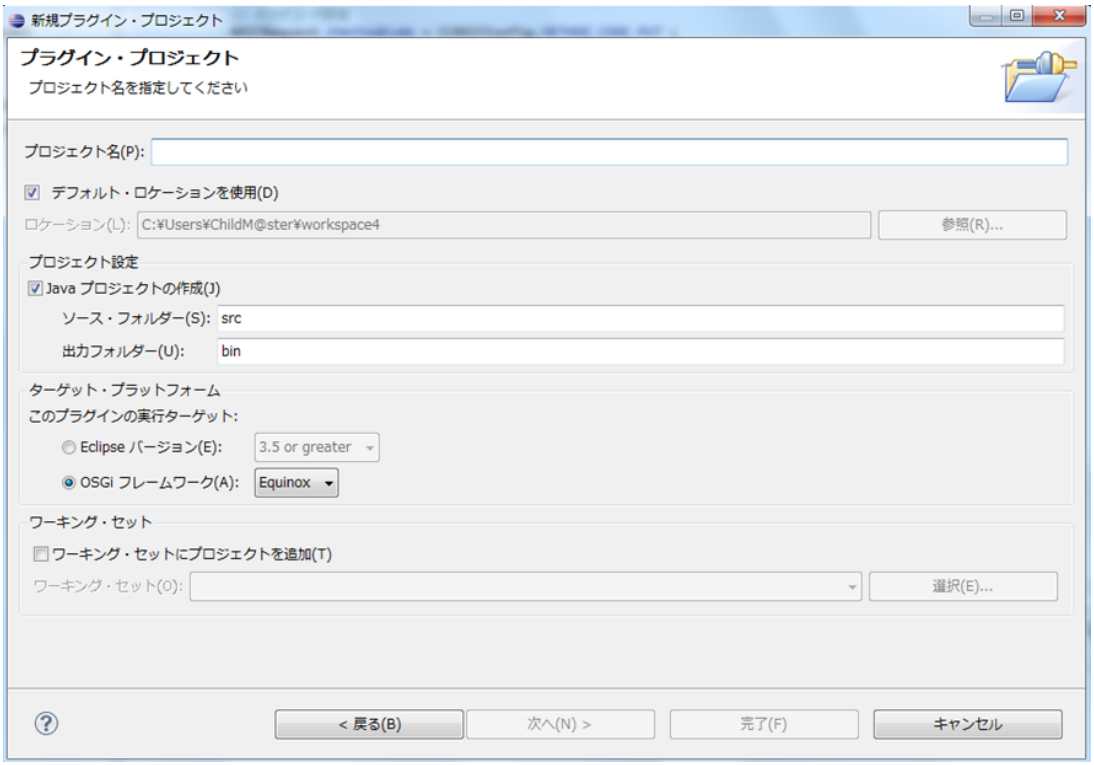

図 6.9　プラグインプロジェクト作成画面

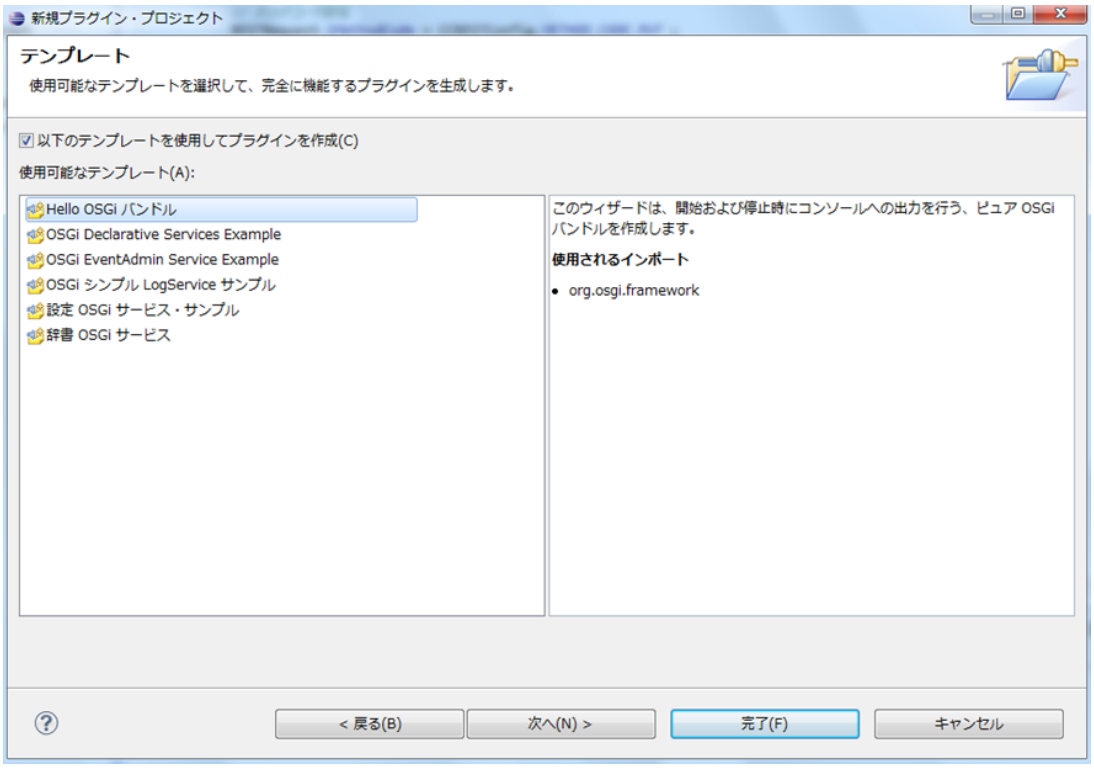

図 6.10　プラグインテンプレート選択画面

用のサービスの作成と登録を行う処理も作成する.

サービスは, インタフェースとそれを実装したクラスから成る. ソースコード 6.2 と 6.3 は, 加算を行うサービスのインタフェースとそれを実装したクラス例である. サービス提供側の Bundle は, インタフェースを OSGi に登録し, サービス利用側の Bundle は, OSGi からインタフェースを受け取る. そして, サービス利用側の Bundle が, インタフェースを利用して Add メソッドを呼び出すとサービス提供側の Bundle で処理が実行される.

ソースコード 6.2　OSGi サービスインタフェース例

```
1 package test.osgi.bundle;
2 public interface IFSampleService
3 {
4   public int Add( int a, int b ) ;
5 }
```

ソースコード 6.3　OSGi サービス例

```
1  package test.osgi.bundle;
2  public class CCSampleService implements IFSampleService
3  {
4    @Override
5    public int Add(int a, int b)
6    {
7      return ( a + b ) ;
8    }
9  }
```

サービスの作成が完了した場合, サービス提供側の Bundle は, OSGi にサービスを登録する．ソースコード 6.4 は, OSGi にサービスを登録する例である．登録を行うためには, Activator.java の start や stop メソッドの引数にある BundleContext を使用する．BundleContext は, registerService メソッドを持っており, このメソッドを利用することで OSGi にサービスを登録することが可能である．registerService の第一引数は, サービスの型を指定する．以後, サービス利用側の Bundle は, ここで指定されているサービスの型を指定して登録したサービスを取得する．第二引数には, サービスの実体を登録する．この実体は, OSGi 内部で管理され, サービス利用側の Bundle がサービスを呼び出した場合, 実効される．第三引数には, サービス利用側の Bundle が必要なサービスを検索する際の条件を登録する．通常, 何も条件を付加しない場合は null を指定しておく必要がある．これにより, OSGi にサービスを登録することができ, サービス利用側の Bundle がサービスを利用できる状態となる．

ソースコード 6.4　OSGi サービス登録例

```
1  package test.osgi.bundle;
2
3  import org.osgi.framework.BundleActivator;
4  import org.osgi.framework.BundleContext;
5
6  public class Activator implements BundleActivator
7  {
8    // サンプルサービス
9    IFSampleService mSampleService = null ;
10
11   public void start(BundleContext context) throws Exception
12   {
13     System.out.println("Bundle開始");
14     // サンプルサービス作成
15     mSampleService = new CCSampleService() ;
16     // OSGiにサンプルサービスを登録する
17     context.registerService( IFSampleService.class, mSampleService, null ) ;
18   }
19
20   public void stop(BundleContext context) throws Exception
21   {
22     System.out.println("Bundle停止");
23   }
24 }
```

次に,　サービス利用側の Bundle で, サービスを取得する処理を記述する必要がある．ソースコード 6.5 は, 利用したいサービスを検索して取得する例である．サービスを

検索するためには, BundleContext の getServiceReference メソッドを使用を使用する. getServiceReference メソッドの第一引数には, 利用したいサービスの型名を指定する. もし, サービスが見つからなかった場合, null が返される.

そして, 利用したいサービスが見つかった場合, BundleContext の getService メソッドを使用してサービスを取得する. getService メソッドの第一引数には, getServiceReference メソッドで返ってきた値を指定する. なお, getService メソッドで取得できたサービスクラスはキャストする必要がある. これにより, 利用したいサービスを取得することが可能である.

ただし, この方法だけでは, サービス提供側の Bundle がサービスを登録するまで, getServiceReference メソッドを繰り返し実行する必要がある. そこで, サービス提供側の Bundle がサービスを登録するとサービス利用側の Bundle に通知する仕組みを使用する. ソースコード 6.6 は, 利用したいサービスの登録通知を受け取り, サービスを取得する例である. まず, BundleContext の addServiceListener メソッドを使用して, サービスの監視するリスナー登録を行う. このとき, ServiceListener インタフェースと serviceChanged メソッドを実装する必要がある. サービスが登録された場合や解除された場合, serviceChanged メソッドが呼び出される. また, サービスが登録されたか解除されたかの判定は, ServiceEvent の getType メソッドで判定可能である. もし, サービスが登録された場合, ソースコード 6.5 と同様にサービスを取得する.

ソースコード 6.5　検索によるサービスの取得例

```
1  package test.osgi.bundle;
2
3  import org.osgi.framework.BundleActivator;
4  import org.osgi.framework.BundleContext;
5  import org.osgi.framework.ServiceReference;
6
7  public class Activator implements BundleActivator
8  {
9    // サンプルサービス
10   private IFSampleService mSampleService = null ;
11
12   public void start(BundleContext context) throws Exception
13   {
14     System.out.println("Bundle開始");
15     // サンプルサービスの取得
16     GetService( context ) ;
17   }
18
19   public void stop(BundleContext context) throws Exception
20   {
21     System.out.println("Bundle停止");
22   }
23
24   // サンプルサービスの取得
25   public Boolean GetService( BundleContext context )
26   {
27     // Streamingサービスが存在する場合
28     if( mSampleService != null )
29     {
30       // 成功
31       return ( true ) ;
32     }
33
34     // PUSHサービスを参照する
35     ServiceReference<?> PUSHServiceReference =
36         context.getServiceReference(IFSampleService.class) ;
37     // 参照失敗時
38     if( PUSHServiceReference == null )
39     {
40       // 失敗
41       return ( false ) ;
42     }
43     // PUSHサービスを取得する
44     mSampleService = (IFSampleService)context.getService( PUSHServiceReference ) ;
45
46     // 成功
47     return ( true ) ;
48   }
49 }
```

ソースコード 6.6　通知によるサービスの取得例

```
1  package test.osgi.bundle;
2
3  import org.osgi.framework.BundleActivator;
4  import org.osgi.framework.BundleContext;
5  import org.osgi.framework.ServiceEvent;
6  import org.osgi.framework.ServiceListener;
7  import org.osgi.framework.ServiceReference;
8
9  public class Activator implements BundleActivator, ServiceListener
10 {
11   // サンプルサービス
12   private IFSampleService mSampleService = null ;
13   private BundleContext mBundleContext = null ;
14
15   public void start(BundleContext context) throws Exception
16   {
17     mBundleContext = context ;
18     System.out.println("Hello World!!");
19     // サービス監視リスナ登録
20     context.addServiceListener(this) ;
21   }
22
23   public void stop(BundleContext context) throws Exception
24   {
25     System.out.println("Goodbye World!!");
26   }
27
28   @Override
29   public void serviceChanged(ServiceEvent event)
30   {
31     // 状態分岐
32     switch(event.getType())
33     {
34     // サービス登録時
35     case ServiceEvent.REGISTERED:
36       // 検索と同様にサービスを取得する
37       GetService() ;
38       // 終了
39       break ;
40
41     // サービス解除時
42     case ServiceEvent.UNREGISTERING:
43       // 終了
44       break ;
45
46     // 上記以外
47     default:
48       break ;
49     }
50   }
51 }
```

6.3.4　Bundle の生成

Bundle 処理の記述が完了した場合, 最後に Bundle の生成を行う. 図 6.11 は, Bundle 生成画面である. この画面は, OSGi 用の MANIFEST.MF の編集を行い易くするものである. なお直接, 編集したい場合は, 生成ページの下位タブにある「MANIFEST.MF」を選択することで編集可能である.

MANIFEST.MF の編集では, 主にサービス利用側との依存関係パッケージやビルドに

必要なライブラリのパスなどを設定する．まず，「依存関係」タブで，サービス利用側の Bundle と共有するパッケージを指定する．次に，「ランタイム」タブで，サービス利用側の Bundle に提供する OSGi サービスのインタフェースを含んでいるパッケージを指定する．最後に，「ビルド」タブで，ライブラリのクラスパスなどを指定する．

設定が完了した場合，生成画面内の「エクスポート」欄より「エクスポート・ウィザード」を選択することで，Bundle 用の jar ファイルが生成される．

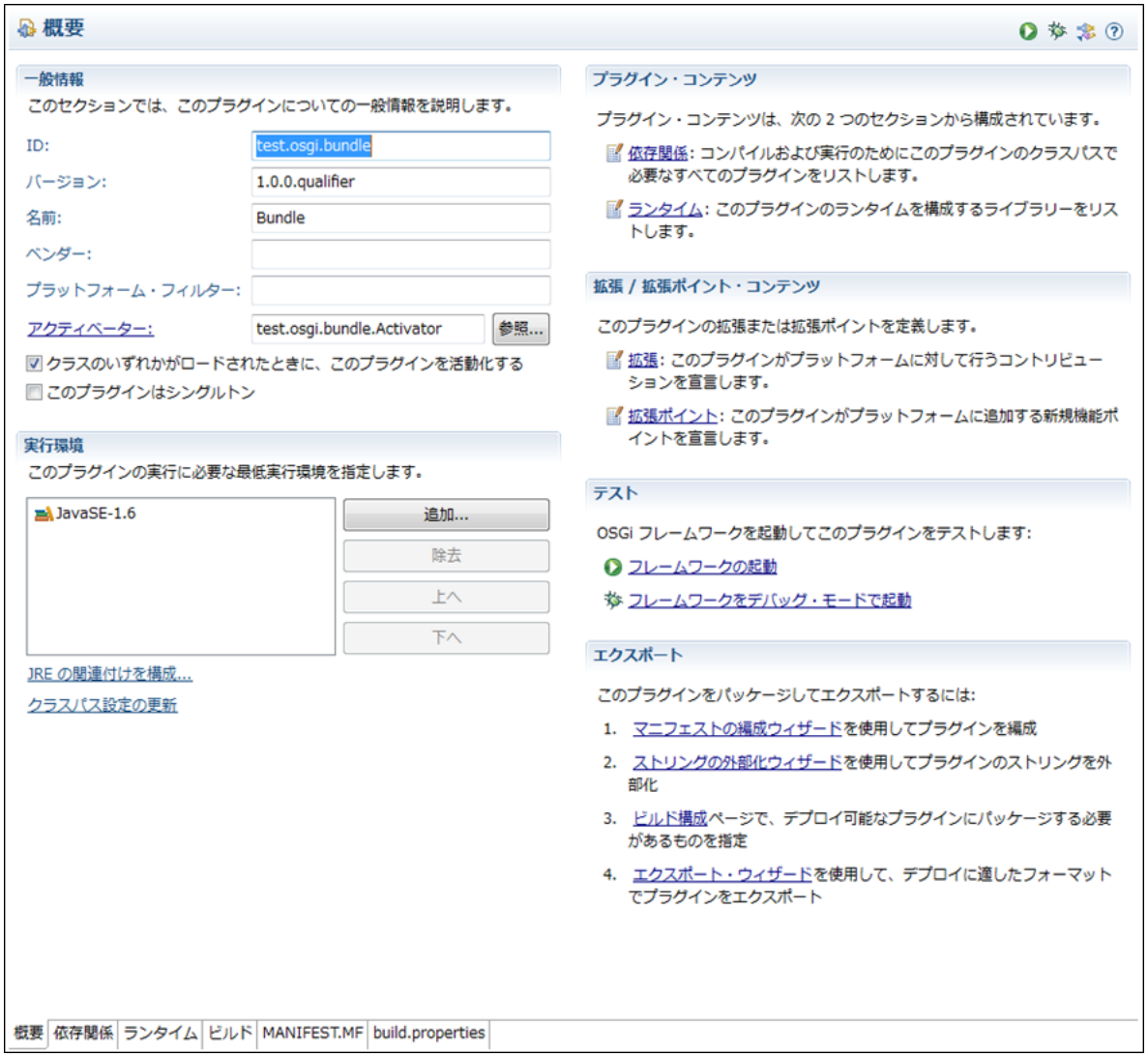

図 6.11　Bundle 生成画面

6.4 JAX-RS の実装

6.4.1 アノテーション作成

6.4.1.1 アノテーション型クラスの作成

まず, JAX-RS を実装するためには, アノテーションを作成する必要がある [20][21]. アノテーション（annotation, 注釈）とは, プログラムのソースコード内に埋め込まれたプログラムのメタデータである. Java でアノテーションを独自に定義するためには, ソースコード 6.7 のようなアノテーション型のクラスを作成する必要がある.

ソースコード 6.7 アノテーション型のクラス

```
@interface Annotation名
{
    キーの型 キーの名前();
    ...
}
```

アノテーション型は, インタフェース型に近い形式をしているが, 以下のような制約がある. ソースコード 6.8 は, 制約を踏まえたうえで定義したアノテーション例である.

- extends 節を書けない
- メソッドは引数と throws 節を持てない
- メソッドの戻り値で使えるのはプリミティブ型, String 型, Class 型, enum 定数, アノテーション型, それらの配列のみ
- ジェネリクスは使えない

ソースコード 6.8 アノテーション定義例

```
// Profileアノテーション
public @interface Profile
{
  String UserName();  // 氏名
  int Age();       // 年齢
}

// GenderType列挙型
public enum GenderType
{
  MAN,         // 男性
  WOMAN,         // 女性
  OTHER        // その他
}

// Genderアノテーション
@interface Gender
{
  GenderType value(); // 性別
}
```

ソースコード 6.8 では, 名前と年齢の値を持った@Profile アノテーションと性別の値を持った@Gender アノテーションがプログラム内で使用可能になる. これらのアノテーションを利用した例をソースコード 6.9 に示す. このとき, @Gender アノテーションのように, キーが一つしか存在しない場合, そのキーの名前を value とすることによってキー名を省略できるようになる.

ソースコード 6.9 では, @Gender アノテーションを使用して, 各クラスに性別情報を付加している. また, クラス内の各メソッドには, @Profile アノテーションが付加されており, メソッド毎に人物情報が付加されている.

ソースコード 6.9　アノテーション使用例

```
1  @Gender( GenderType.MAN )
2  public class ManClass
3  {
4    @Profile( UserName = "nakahara", Age = 25 )
5    public void Profile_NAKAHARA()
6    {
7      ...省略
8    }
9
10   @Profile( UserName = "murai", Age = 24 )
11   public void Profile_MURAI()
12   {
13     ...省略
14   }
15 }
16
17 @Gender( GenderType.WOMAN )
18 public class WomanClass
19 {
20   @Profile( UserName = "satou", Age = 24 )
21   public void Profile_SATOU()
22   {
23     ...省略
24   }
25 }
```

6.4.1.2　付加可能な構文要素制限

ソースコード 6.9 では, アノテーションを付加できる構文要素に制限は設けられていない. したがって, @Gender アノテーションをメソッドに付加したり, @Profile アノテーションをクラスに付加することが可能である. しかし, ソースコード 6.9 のような形式にした場合, アノテーションを付加できる構文要素を制限すべきである.

アノテーションを付加できる構文要素を制限したい場合は, メタアノテーション型 Target を付与する. このとき, 制限箇所を指定するためには, メタアノテーション型 Target で ElementType を指定する. ElementType には, 表 6.2 のようなものが挙げられる. なお, メタアノテーション型 Target を付与していない場合は, 全ての要素に対してアノテーションを付加することが可能である.

ソースコード 6.10 は, 付加できる構文要素を制限した例である. これにより, @Profile

アノテーションは, メソッド以外には付加できなくなり, @Gender アノテーションは, クラス以外には付加できなくなる.

表 6.2　ElementType 一覧

定義名	構文要素
ElementType.ANNOTATION_TYPE	アノテーション型宣言
ElementType.CONSTRUCTOR	コンストラクタ
ElementType.FIELD	フィールド宣言, enum 宣言
ElementType.LOCAL_VARIABLE	ローカル変数宣言
ElementType.METHOD	メソッド宣言
ElementType.PACKAGE	パッケージ宣言
ElementType.PARAMETER	引数宣言
ElementType.TYPE	クラス宣言, インタフェース宣言, アノテーション型宣言, enum 宣言

ソースコード 6.10　アノテーションの付加制限

```
1  // Profileアノテーション
2  @Target( ElementType.METHOD )
3  public @interface Profile
4  {
5    String UserName();  // 氏名
6    int Age();       // 年齢
7  }
8
9  // Genderアノテーション
10 @Target( ElementType.TYPE )
11 @interface Gender
12 {
13   GenderType value(); // 性別
14 }
```

6.4.1.3　アノテーションの有効範囲設定

最後にアノテーションの有効範囲を設定する. 有効範囲を指定するためには, メタアノテーション型 Retention を付加し, RetentionPolicy を指定する. 表 6.3 は, RetentionPolicy の一覧を示したものである. もし, メタアノテーション型 Retention を付加しなかった場合, RetentionPolicy.CLASS が自動的に設定される. なお, 今回はリフレクション API を使用して, 動的にアノテーションを解析する必要があるため, RetentionPolicy.RUNTIME を使用する.

ソースコード 6.11 は, メタアノテーション型 Retention を付加した例である. これにより, @Profile アノテーションと@Gender アノテーションは, クラス・ファイルに出力し,

実行時 JVM にもロードされる．したがって, リフレクション API を使用することで, プログラム内でアノテーションの動的解析が可能である．

表 6.3　RetentionPolicy 一覧

定義名	構文要素
RetentionPolicy.CLASS	アノテーションの情報をクラス・ファイルに出力するが, 実行時 JVM にロードされない．省略された場合のデフォルト値
RetentionPolicy.RUNTIME	アノテーションの情報をクラス・ファイルに出力し, 実行時 JVM にもロードされる．その為, リフレクション API 経由でその情報にアクセスできる
RetentionPolicy.SOURCE	アノテーションの情報はソース・コード上のみに保持され, コンパイル時によって破棄される

ソースコード 6.11　アノテーションの有効範囲設定

```
1  // Profileアノテーション
2  @Target( ElementType.METHOD )
3  @Retention( RetentionPolicy.RUNTIME )
4  public @interface Profile
5  {
6    String UserName();  // 氏名
7    int Age();       // 年齢
8  }
9
10 // Genderアノテーション
11 @Target( ElementType.TYPE )
12 @Retention( RetentionPolicy.RUNTIME )
13 @interface Gender
14 {
15   GenderType value(); // 性別
16 }
```

6.4.1.4　JAX-RS アノテーションの定義例

これまでのアノテーション作成方法を踏まえ, ソースコード 6.12 に JAX-RS アノテーションの定義例を示す．ソースコード 6.12 では, @GET/@POST/@PATH/@PATH_PARAM アノテーションが定義されている．@GET/@POST アノテーションは, メソッドのみに付加することが可能である．@PATH アノテーションは, メソッドとクラスに付加することが可能である．@PATH_PARAM アノテーションは, メソッドの引数のみに付加することが可能である．また, 全てのアノテーションは, リフレクション API を使用して, 動

的に解析を行えるようにするために RetentionPolicy.RUNTIME が指定されたメタアノテーション型 Retention が付加されている.

ソースコード 6.12　JAX-RS アノテーション定義例

```
1  // GETアノテーション
2  @Retention(RetentionPolicy.RUNTIME)
3  @Target({ElementType.METHOD})
4  public @interface GET{}
5
6  // POSTアノテーション
7  @Retention(RetentionPolicy.RUNTIME)
8  @Target({ElementType.METHOD})
9  public @interface POST{}
10
11 // PATHアノテーション
12 @Retention(RetentionPolicy.RUNTIME)
13 @Target({ElementType.METHOD,ElementType.TYPE})
14 public @interface PATH
15 {
16   String value() ;
17 }
18
19 // PATH_PARAMアノテーション
20 @Retention(RetentionPolicy.RUNTIME)
21 @Target({ElementType.PARAMETER})
22 public @interface PATH_PARAM
23 {
24   String value() ;
25 }
26
27 ・・・省略
```

6.4.2　リフレクション API

Java では, クラス生成やメソッド呼び出しをソースコード上に直接書いてコンパイル時に決定されるだけでなく, 文字列 (クラス名) を使ってクラスを生成したり, メソッド名の文字列を使ってメソッドを呼び出したりすることが可能なリフレクションと呼ばれる仕組みがある [22]. そこで, JAX-RS を実装するためにリフレクション API を使用する.

6.4.2.1　解析対象クラスの作成

JAX-RS を実装するにあたり, ソースコード 6.12 で定義したアノテーションを使用して, ソースコード 6.13 のようなアノテーション解析対象クラスを作成した. 以降は, ソースコード 6.13 を使用して, アノテーションの解析とメソッドの呼び出しについて説明する.

ソースコード 6.13 では, クラスに@PATH(“osgi/rest”) というアノテーションが付加されている. また, 実装されているメソッドには, Method1 と Method2 があり, それぞれ, @GET と @POST アノテーションと@PATH(”/service”) アノテーションが付加されている. したがって, このクラスは,「osgi/rest/service」リソースに対して GET リクエ

ストが来た場合, Method1 メソッドを呼び出し, POST リクエストが来た場合, Method2 メソッドが呼び出されるという意味になる. 更に, Method2 メソッドでは, リソースパスに付加された {num} の値を取得するため, 引数に@PATH_PARAM アノテーションが付加されている.

ソースコード 6.13　アノテーション解析対象クラス

```
1  @PATH( "osgi/rest" )
2  class CCRESTService
3  {
4    @GET
5    @PATH( "/service" )
6    public String Method1()
7    {
8      // 処理
9    }
10
11   @POST
12   @PATH( "/service/{num}" )
13   public String Method2( @PATH_PARAM( “num” ) int iNum )
14   {
15     // 処理
16   }
17    ・・・省略
18 }
```

6.4.2.2　アノテーションの解析

まず, クラスのアノテーション解析方法について説明する. ソースコード 6.14 は, OSGi からサービスクラスを取得し, クラスに付加されている@PATH アノテーションを取得して解析する例である. アノテーションを取得するには, クラス情報を取得するための java.lang.Class 型が保持している getAnnotation メソッドを呼び出す.

ソースコード 6.14 では, OSGi から取得したサービスクラスから getClass メソッドを使用してクラス型を取得し, 更に getAnnotation メソッドを使用してアノテーションを取得している. このとき, アノテーションが取得できなかった場合は, null が返される. アノテーションが取得できた場合, PathAnnotation が保持している value メソッドを呼び出すことで, @PATH アノテーションの値を取得できる.

このようにして, クラスに付加されているアノテーションの取得と解析を行い, リクエストに合ったサービスクラスを選択する.

ソースコード 6.14　アノテーション解析例

```
1  // OSGiからアノテーションを含んでいるRESTサービスクラスを取得する
2  CCRESTService mRESTService = getService() ;
3
4  // PATHアノテーション取得
5  PATH PathAnnotation = ServiceObject.getClass().getAnnotation( PATH.class ) ;
6
7  // 取得できなかった場合
8  if( PathAnnotation == null )
9  {
10   // 失敗
11   return ( false ) ;
12 }
13
14 // PATHアノテーションの値を表示
15 System.out.println( PathAnnotation.value() ) ;
```

6.4.2.3　実行対象メソッドの取得

リクエストに合ったサービスクラスを選択が完了した場合, サービスクラス内のメソッドを取得する必要がある．ソースコード 6.15 は, 「osgi/rest/service」リソースに対して POST リクエストが来た場合, Method2 メソッドを取得するための例である. クラスが保持しているメソッド一覧を取得するためには, java.lang.Class 型が保持している getMethods メソッドを実行する．これにより, 保持しているメソッド一覧が java.lang.reflect.Method 型の配列で取得可能である.

次に取得したメソッド一覧の中から, 値「/service」を持っている@PATH アノテーションと@POST アノテーションが付加されているメソッドを取得する必要がある．メソッドからアノテーションを取得するためには, java.lang.reflect.Method 型が保持している getAnnotation メソッドを実行する．ソースコード 6.15 では, これを利用してメソッドに付加されてる@PATH と@POST アノテーションを取得して, 発見したメソッド番号を iTargetMethodIdx 変数に格納している．これは, MethodList[iTargetMethodIdx] が目的のメソッドであることを示している.

これにより, リクエストの条件に合うメソッドを取得することが可能である.

ソースコード 6.15　メソッドの取得例

```
1  // メソッドの取得
2  Method[] MethodList = ServiceObject.getClass().getMethods();
3
4  // 実行対象メソッド番号
5  int iTargetMethodIdx = -1 ;
6
7  // メソッドの個数だけ繰り返し
8  for( int i = 0; i < MethodList.size(); i++ )
9  {
10   // メソッドに付加されている PATHアノテーション取得
11   PATH PathAnnotation = MethodList[i].getAnnotation( PATH.class ) ;
12   // 取得失敗時
13   if( PathAnnotation == null )
14   {
15     // 次のメソッドへ
16     continue ;
17   }
18
19   // メソッドに付加されている POSTアノテーション取得
20   POST PostAnnotation = MethodList[i].getAnnotation( POST.class ) ;
21   // 取得失敗時
22   if( PostAnnotation == null )
23   {
24     // 次のメソッドへ
25     continue ;
26   }
27
28   //実行対象メソッド番号を記録
29   iTargetMethodIdx = i ;
30
31   // メソッド取得終了
32   break ;
33 }
34
35 // 移行は MethodList[iTargetMethodIdx]の実行準備処理へ
```

6.4.2.4 実行対象メソッドの引数作成

リクエストに合ったメソッドの取得に成功した場合, メソッドの引数リストを作成する.

ソースコード 6.16 は, 引数リストの作成例である. 引数リストを作成するためには, 引数の型と引数が持っているアノテーションの一覧を取得する必要がある. これらは, java.lang.reflect.Method 型が保持している getParameterTypes メソッドと getParameterAnnotations メソッドを使用することで取得可能である. 一覧の取得が完了した場合, @PATH_PARAM アノテーションが付加されている引数を検索し, その値を取得する.

次に取得した@PATH_PARAM アノテーションの値を利用して, リクエストパスから該当する値を取得する. このとき, @PATH_PARAM の値は,「num」であり, 定義リソースパスが「osgi/rest/service/{num}」になっているため, {num} に該当する箇所をリクエストパスから取得する. 例えば, リクエストパスが「osgi/rest/service/123」だった場合, 値「123」を取得する.

続いて取得した値を引数型に合わせる. ソースコード 6.13 から@PATH_PARAM アノテーションが付加されている引数の型は, int 型であるため Integer 型を使用して int 型に変換している.

最後にメソッドを実行する際に引数リストは, Object 配列の形式で指定してやる必要があるため, Object 型の配列に変換した値を格納して引数リストが完成する.

ソースコード 6.16　引数の作成例

```
1  private Object[] CreateArgumentList()
2  {
3    // 実行対象メソッド取得
4    Method TargetMethod = MethodList[iTargetMethodIdx] ;
5
6    // メソッドの持つ引数型を取得する
7    Class<?>[] ParmeterTypes
8      = TargetMethod.getParameterTypes() ;
9    // 引数アノテーションを取得する
10   Annotation[][] ParameterAnnotations
11     = TargetMethod.getParameterAnnotations() ;
12   // 引数リスト
13   Object[] ArgumentList = new Object[GetParmeters.length] ;
14
15   // 引数の数だけ繰り返し
16   for( int i = 0; i < ParmeterTypes.size(); i++ )
17   {
18     // PATH_PARAMアノテーションが存在しない場合
19     if( ParamAnnotations[i].annotationType() != PATH_PARAM.class )
20     {
21       // 次の引数へ
22       continue ;
23     }
24
25     // PATH_PARAMアノテーション取得
26     PATH_PARAM PathAnntation = (QUERY_PARAM)ParamAnnotations[i] ;
27
28     // PATH_PARAMアノテーション値を取得(numという値を取得)
29     String strParamKey = PathAnntation.value() ;
30     // 取得失敗時
31     if( strParamKey == null )
32     {
33       // 終了
34       break ;
35     }
36
37     // リクエストパスからパラメタ値を取得({num}の箇所を取得)
38     String strParmValue = GetPathParam( strParamKey ) ;
39     // 取得失敗
40     if( strParmValue == ArgumentList )
41     {
42       // 終了
43       break ;
44     }
45
46     // 引数型が「int」だった場合
47     // 引数型が「float」「double」・・・だった場合もキャスト処理を記述
48     if( ParamAnnotations[i].getName().equals( "int" ) )
49     {
50       // int型にして引数リストに格納
51       ArgumentList[i] = new Integer( strParmValue ) ;
52       // 次の引数へ
53       continue ;
54     }
55
56   }
57
58   // 引数リストを返す
59   return ( ArgumentList ) ;
60 }
```

6.4.2.5 メソッドの実行

最後に, リクエストに合ったメソッドの取得に成功した場合, そのメソッドを実行する. メソッドを実行するためには, java.lang.reflect.Method 型が保持している invoke メソッドを呼び出す. これは, メソッドを実行するためのメソッドである.

ソースコード 6.17 は, 取得したメソッドの実行例である. invoke メソッドの第一引数には, 既存のインスタンスを設定する. もし null を指定した場合, static メソッドとして実行される. 第二引数には, メソッドを実行する際に必要となる引数リストを指定する. このとき, 引数リストは, Object 配列の形式で指定する. もし, 引数がないメソッドを実行する場合は, null を指定する.

これにより, 指定されたメソッドを実行することが可能である. なお, 引数に合わない値が代入された場合 IllegalArgumentException, public ではないメソッドやメンバにアクセスしようとした場合, IllegalAccessException, 実行したメソッド内で例外が発生した場合, InvocationTargetException 例外が発生するため, 各例外を処理する必要がある. また, 返り値はオブジェクトとして返されるため, 必要に応じてキャストが必要である.

ソースコード 6.17 メソッド実行例

```
1  // 実行対象メソッド取得
2  Method TargetMethod = MethodList[iTargetMethodIdx] ;
3
4  // メソッド実行結果
5  Object Result = null ;
6
7  // 引数リストを取得する
8  Object[] ArgumentList = CreateArgumentList() ;
9
10 try
11 {
12   // メソッドの実行
13   Result = Target Method.invoke( ServiceObject, ArgumentList );
14 }
15 catch (IllegalArgumentException e)
16 {
17   // 引数に合わない値が代入された場合
18   throw new RuntimeException(e);
19 }
20 catch (IllegalAccessException e)
21 {
22   // 可視性限定子が許可しないメソッドやメンバーにアクセスした場合
23   throw new RuntimeException(e);
24 }
25 catch (InvocationTargetException e)
26 {
27   // 実行したメソッド内で例外が発生した場合
28   throw new RuntimeException(e);
29 }
```

第 7 章

ケーススタディ

この章では, 提案システムを使用してサンプル M2M アプリケーション例を検討したので紹介する. また, サンプル M2M アプリケーションの構築環境や構築方法についても述べる.

7.1 サンプル M2M アプリケーション

ケースステディでは, 温度, 湿度, 照度, 加速度, GPS などといったコンテキスト情報を使用を使用して, 家庭用エアコンを自動制御するサンプル M2M アプリケーションを作成する. 以下は, サンプル M2M アプリケーションの動作例である.

- 緯度/経度値をもとにして, 自宅に近づいているか遠ざかっているかを判断し, 電源の ON/OFF を行う.
- 温度/湿度値をもとにして, 設定温度変更や運転切り替えを行う.
- 照度/加速度値をもとにして, 就寝中などの判断を行い, タイマ設定を行う.

また, サンプル M2M アプリケーションでは, コンテキスト情報の取得を行うにあたり, 以下のデバイスを使用する.

- リモコンの赤外線信号を学習し, 学習した信号を発行可能かつ API を持っている学習リモコン
- 温度/湿度/照度値が取得可能なセンサ
- GPS/加速度値が取得可能なスマートフォン

図 7.1 は, 提案システムとこれらのデバイスを使用して家庭用エアコンの自動制御方法を示したものであり, 以下は, その流れである.

(1) エアコンのリモコンを学習リモコンに学習させる.
(2) M2M アプリケーションが, REST リクエストを使用して, センサやスマートフォンからコンテキスト情報を取得する.
(3) M2M アプリケーションが, (2) で取得したコンテキスト情報を解析し, 学習リモコン

を操作する REST リクエストを送る.

(4) 学習リモコンがエアコンを制御する.

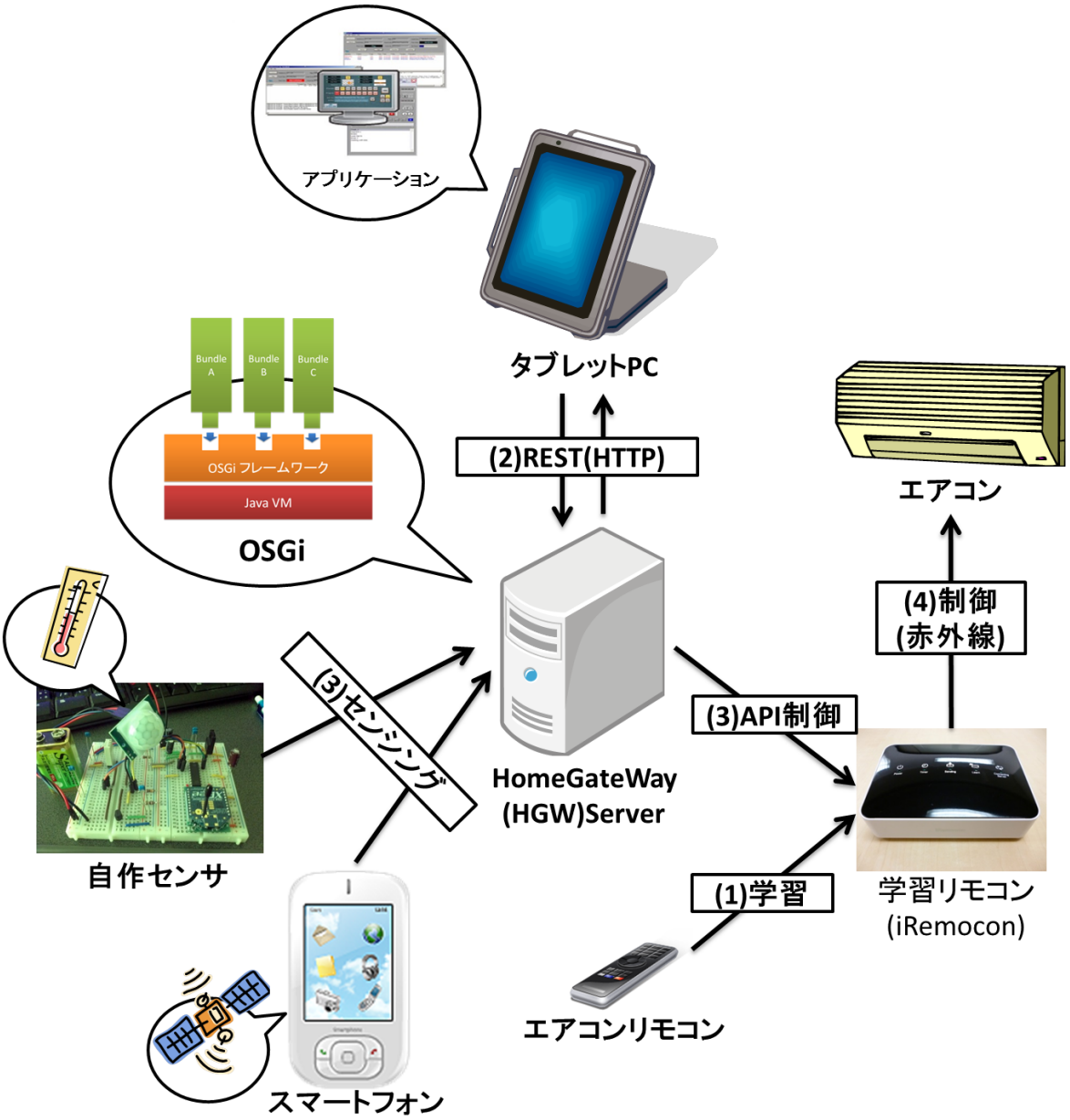

図 7.1　サンプル M2M アプリケーション

7.2 使用デバイス

7.2.1 学習リモコン

家庭用エアコンのリモコンが発行する赤外線信号を学習し, 学習した内容を発行可能な iRemocon と呼ばれる学習リモコンがある (図 7.2).

赤外線信号を学習を学習させるためには, iRemocon に対してリモコンで赤外線信号を発信するだけである. また, iRemocon には, 信号を学習させたり, 発信させるための API が存在する [23]. iRemocon API を利用するには, 図 7.3 に示す電文構造に対して表 7.1 で示すコマンドとパラメータを TCP を使用して送信することで可能である.

しかし, この API は, iRemocon 固有の API であるため, 他のデバイスでは使用することはできない. そのため, M2M アプリケーションに直接 API を使用した処理を記述すると他の処理と混雑し, M2M アプリケーションが複雑になる可能性がある. そこで, iRemocon API による処理を Bundle 化し, REST を使用して iRemocon の操作を行えるようにした.

図 7.2 iRemocon(文献 [23] より抜粋)

*XX;yyyy\r\n

* : コマンドバッファクリア文字列
XX : コマンド部
; : パラメータ指定記号
yyyy : パラメータ部
\r\n : コマンド実行文字列

図 7.3 iRemocon API の構造

表 7.1 iRemocon API コマンド一覧

コマンド部	パラメータ部	備考
au	—	接続確認用
is	;1〜1500	赤外線発信
ic	;1〜1500	リモコン学習開始
cc	—	リモコン学習停止
tm	;1〜1500 ; 現在時刻 +60〜4102444800 ;0〜31536000	タイマーセット
tl	—	タイマー一覧取得
td	;1〜1500	タイマー解除
ts	;1293775200〜4102444800	現在時刻設定
tg	—	現在時刻取得
vr	—	ファームバージョン番号取得

7.2.2 自作センサ

温度, 湿度, 照度値といったコンテキスト情報を取得するために図 7.4 のようなセンサを作成した. このセンサでは, 温度, 湿度, 照度値を Zegbee 経由で他の計算機に送信することが可能である. なお, Zegbee 通信を行うために Xbee モジュールを使用した. 計算機側では Xbee モジュールを RS232C で接続する.

操作を行うためには, CoAP(Constrained Application Protocol) を使用する. CoAP とは, 現在 IETF が策定中の M2M デバイス用のプロトコルである [24]. 図 7.5 は CoAP の電文構造である. code に GET/POST/PUT/DELETE といったメソッドを記述し, option 内でリソースパスを指定する. CoAP も REST の考え方を使用してデバイスを操

作するためのプロトコルであるが, 通常の HTTP のプロトコルと併用して使用することができない. そこで, CoAP を HTTP に変換し, 自作センサを操作するために Bundle 化を行った.

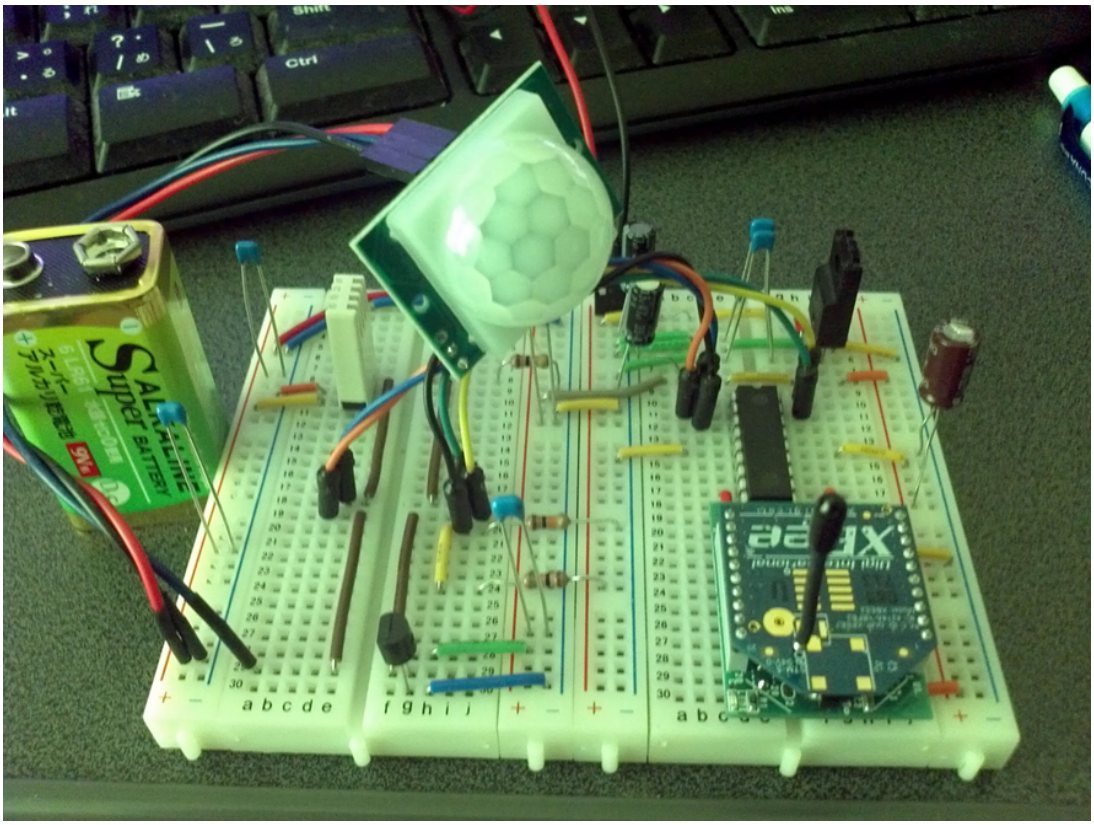

図 7.4　自作センサ

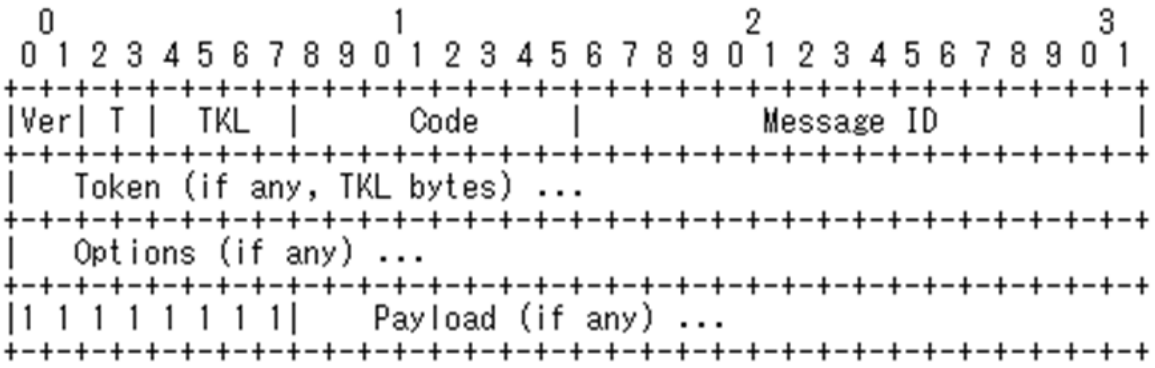

図 7.5　CoAP 電文構造 (文献 [24] より抜粋)

7.2.3 スマートフォン

位置情報や加速度値といったコンテキスト情報を取得するために, スマートフォンを使用する. 今回, 使用したのは, ISW11HT と呼ばれる一般的なスマートフォンである (図 7.6[25]). このスマートフォンの OS は, Android2.2.3 であり, センサにアクセスするための API がある.

図 7.6 使用スマートフォン (文献 [25] より抜粋)

ソースコード 7.1 は, Android API を使用して GPS を利用した位置情報取得例である. まず, 位置情報を取得するためには, LocationListener インタフェースと onLocationChanged メソッドを実装する必要がある. onLocationChanged メソッドは, 位置情報を取得するたびに呼び出されるメソッドであり, 引数の Location 型に位置情報が格納される.

次に GPS 装置の動作を設定する必要がある. GPS 装置の動作を設定するためには, LocationManager を使用する. ソースコード 7.1 では, Android アプリケーション画面が作成されたタイミングで呼び出される onCreate メソッド内で LocationManager を用い

て GPS 装置の設定を行っている．LocationManager には, requestLocationUpdates メソッドを持っており, このメソッドで GPS 装置の動作を設定できる．ソースコード 7.1 では, 1 秒おきに 10ms 以上の変化があった場合, onLocationChanged メソッドを呼び出すという設定を行っている．

また, GPS 装置を停止させたい場合, LocationManager の removeUpdates メソッドを呼び出す．ソースコード 7.1 では, アプリケーションが停止したときに呼び出される onStop メソッド内で GPS 装置を停止させている．

ソースコード 7.1　Andorid 端末による GPS 取得例

```
1  import android.app.Activity;
2  import android.location.Location;
3  import android.location.LocationListener;
4  import android.location.LocationManager;
5  import android.location.LocationProvider;
6  import android.os.Bundle;
7  import android.util.Log;
8
9  public class CCSampleAndroidApp1 extends Activity implements LocationListener
10 {
11   // GPS装置の操作クラス作成
12     private LocationManager mLocationManager = null ;
13
14     // Android画面が作成されたときに呼び出される
15     @Override public void onCreate(Bundle savedInstanceState)
16   {
17         super.onCreate(savedInstanceState);
18         // ページレイアウト設定
19         setContentView(R.layout.main);
20         // GPS装置の操作クラス作成
21         mLocationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
22         // GPS装置の動作設定
23         mLocationManager.requestLocationUpdates(
24           LocationManager.GPS_PROVIDER,
25           LocationManager.NETWORK_PROVIDER,
26           1000, // 位置情報を取得する最小時間(ms)
27           10,   // 位置情報を取得する最小距離(m)
28           this ) ;
29     }
30
31     // アプリケーション終了時に呼び出される
32   @Override public void onStop()
33   {
34     super.onStop();
35
36     // 位置情報の更新を止める
37     mLocationManager.removeUpdates(this);
38   }
39
40   // 現在位置が更新された場合,  呼び出される
41     @Override public void onLocationChanged(Location location)
42   {
43         Log.v("----------", "----------");
44         Log.v("Latitude", String.valueOf(location.getLatitude()));
45         Log.v("Longitude", String.valueOf(location.getLongitude()));
46         Log.v("Accuracy", String.valueOf(location.getAccuracy()));
47         Log.v("Altitude", String.valueOf(location.getAltitude()));
48         Log.v("Time", String.valueOf(location.getTime()));
49         Log.v("Speed", String.valueOf(location.getSpeed()));
50         Log.v("Bearing", String.valueOf(location.getBearing()));
51     }
52 }
```

ソースコード 7.2 は, Android API を使用して加速度センサの値を取得する例である．加速度センサから値を取得するためには, SensorEventListener インタフェースと onSensorChanged メソッドを実装する必要がある．onSensorChanged メソッドは, センサの状態が変化した場合, 呼び出されるメソッドである．変化したセンサの値は, onSensorChanged メソッドの引数にある SensorEvent の values 配列を参照することで取得可能である．

次に, 加速度センサの値が変化した場合, onSensorChanged メソッドを呼び出す設定を行う必要がある．ソースコード 7.2 では, Android アプリケーション画面が作成されたタイミングで呼び出される onCreate メソッド内で加速度センサの設定を行っている．設定を行うためには, 端末に加速度センサが存在するか確認し, 通知リスナに登録する必要がある．これらの処理は, SensorManager クラスを使用して行うことが可能である．ソースコード 7.2 では, 端末に加速度センサが存在するか確認するために getSensorList メソッドを使用し, 通知リスナに登録するために registerListener メソッドを使用している．

なお, 他のセンサを利用する場合も同様の処理を行う必要がある．そのため, onSensorChanged メソッドは, 他のセンサの値が変化しても呼び出される．したがって, どのセンサの値が変化したのか判断する処理が必要である．どのセンサの値が変化したのか判断を行うためには, onSensorChanged メソッドの引数にある SensorEvent の getType メソッドを利用することで可能である．

ソースコード 7.2　Andorid 端末による加速度値取得例

```
1  import java.util.List;
2  import android.app.Activity;
3  import android.hardware.Sensor;
4  import android.hardware.SensorEvent;
5  import android.hardware.SensorEventListener;
6  import android.hardware.SensorManager;
7  import android.os.Bundle;
8  import android.widget.TextView;
9
10 public class CCSampleAndroidApp2 extends Activity implements SensorEventListener
11 {
12   // センサ管理クラス
13   private SensorManager mSensorManager = null ;
14
15   // Andoroid画面が作成されたときに呼び出される
16   @Override public void onCreate(Bundle savedInstanceState)
17   {
18     super.onCreate(savedInstanceState);
19     // 画面レイアウト設定
20     setContentView(R.layout.main);
21
22     // センサ管理クラス作成
23     mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE) ;
24
25     // 加速度センサを取得する
26     List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ACCELEROMETER);
27     // 端末に加速度センサがある場合
28     if( sensors.size() <= 0 )
29     {
30       // 加速度センサを取得
31       Sensor s = sensors.get( 0 );
32       // 加速度センサに変化があった場合, 通知するリスナーを設定
33       manager.registerListener(this, s, SensorManager.SENSOR_DELAY_UI);
34     }
35   }
36
37   // アプリケーション終了時に呼び出される
38   @Override protected void onStop()
39   {
40     super.onStop();
41     // Listenerの登録解除
42     mSensorManager.unregisterListener(this);
43   }
44
45   // センサに変化がった場合, 呼び出される
46   @Override public void onSensorChanged(SensorEvent event)
47   {
48     // 加速度センサではない場合
49     if(event.sensor.getType() != Sensor.TYPE_ACCELEROMETER)
50     {
51       // 終了
52       return ;
53     }
54
55     // 加速度センサの値を取得する
56     String str = "加速度センサー値:"
57           + "\nX軸:" + event.values[SensorManager.DATA_X]
58           + "\nY軸:" + event.values[SensorManager.DATA_Y]
59           + "\nZ軸:" + event.values[SensorManager.DATA_Z] ;
60     ・・・省略
61   }
62 }
```

ソースコード 7.1 と 7.2 により, 現在位置情報と加速度センサの値が取得可能である. しかし, ソースコードより, 同じ端末でも装置によって, 取得方法が異なるものがあるため, Android API を使用して直接 M2M アプリケーションの作成は行わず, 取得したコンテキスト情報を OSGi に送信するアプリケーションを作成した. OSGi では, 受け取ったコンテキスト情報を REST サービスによって M2M アプリケーションに提供する.

図 7.7 は, Android 端末のコンテキスト情報を OSGi に送信するアプリケーションの設定画面である. このアプリケーションは, Android サービスとして動作し, 新しいコンテキスト情報を取得するたびに OSGi に送信する.

また, 起動時に端末にある全てのセンサ一覧を取得し, チェックボックスを自動生成する. したがって, このアプリケーションをインストールだけで, 他の Android 端末でも OSGi にコンテキスト情報を送信することが可能である.

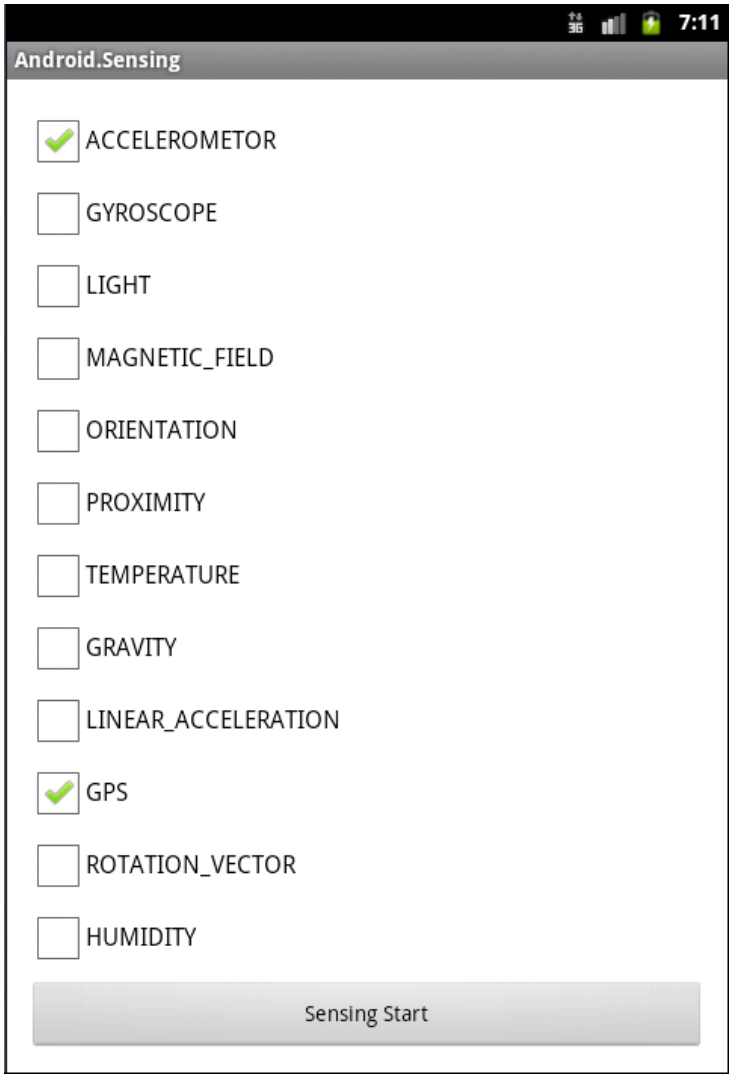

図 7.7　Sensing アプリケーション設定画面

7.3 サンプル M2M アプリケーションの作成

7.3.1 構築環境

提案システムは, HomeGateWay(HGW) Server のような比較的パフォーマンスの低い計算機上でも動作させることを検討している. そこで, 提案システムを構築するにあたり, 表 7.2 のように OSGi を実行する計算機は, 一般的な HGW Server と同程度のパフォーマンスを持つ計算機を選択した.

また, サンプル M2M アプリケーションは, 表 7.3 に示すタブレット PC 上に実装した. このタブレットは, 普段電源を落とすことなく, 自宅内のみで使用している Android タブレット PC である. サンプル M2M アプリケーションは, このタブレット PC 上に Android アプリケーションとして実装する.

表 7.2 提案システム構築環境

	HGW で使用されている計算機	OSGi を実行する計算機
OS	Windows8 64bit	Windows7 64bit
CPU	AMD デュアルコア E1-1200 APU (1.4GHz)	Celeron (1.2GHz)
メモリ	4GB	4GB

表 7.3 M2M アプリケーション構築環境

形状	タブレット
OS	Android 4.0
CPU	NVIDIA Tegra 2 (1GHz)
メモリ	1GB
無線 LAN	IEEE802.11 a/b/g/n

7.3.2 リソースの定義

サンプル M2M アプリケーションが自作センサ, iRemocon, スマートフォンを REST により操作可能にするためには, リソースを定義する必要がある. そこで表 7.4 のように, 各デバイスのリソースを定義した. このように定義したリソースをもとにして, JAX-RS により REST サービスクラスを作成し, Bundle として OSGi にインストールを行った.

表 7.4　リソース定義例

メソッド	リソース	概要
GET	/sensor/temperature	温度取得
PUT	/sensor/temperature	温度センサ ON
DELETE	/sensor/temperature	温度センサ OFF
GET	/sensor/humidity	湿度取得
PUT	/sensor/humidity	湿度センサ ON
DELETE	/sensor/humidity	湿度センサ OFF
GET	/sensor/illuminance	照度取得
PUT	/sensor/illuminance	照度センサ ON
DELETE	/sensor/illuminance	照度センサ OFF
PUT	/iRemocon	iRemocon ON
DELETE	/iRemocon	iRemocon OFF
GET	/iRemocon/signal/{num}	指定された番号の信号を発行
PUT	/iRemocon/signal/{num}	指定された番号に信号を学習
DELETE	/iRemocon/signal/{num}	指定された番号の信号を除去
PUT	/andorid/gps	GPS ON
DELETE	/android/gps	GPS OFF
GET	/andorid/gps	緯度/経度値の取得
PUT	/android/accelerometor	加速度センサ ON
DELETE	/android/accelerometor	加速度センサ OFF
GET	/android/accelerometor/	指定数の加速度値を取得
POST	/android/accelerometor/{num}	蓄積する加速度値数を設定

7.3.3　Bundle の構成

提案システムの Bundle 構成を図 7.8 に示す.

最下層は, 各デバイス固有の処理をまとめた Bundle である. iRemocon API を使用して iRemocon を操作したり, Zigbee/CoAP 通信や Sensing アプリケーションから送られてくるコンテキスト情報を取得する役目がある. これらの Bundle は, リソース定義をもとに作成された各デバイス毎の REST サービス Bundle により呼び出される.

REST サービス Bundle は, アノテーションが付加されているクラスで構成され, JAX-RS Bundle により, 呼び出される. このとき, JAX-RS Bundle は, アプリケーションから

のリクエストをもとに, REST サービス Bundle 内に存在するクラスのアノテーションを解析し, サービスメソッドを呼び出す.

また, JAX-RS Bundle 上には, HTTP Bundle が存在する. これは, アプリケーションと OSGi 間の橋渡しを行う役目があり, アプリケーションは, HTTP を使用して OSGi にリクエストを送信することが可能である.

図 7.9 は, 動作確認のために Web ブラウザを使用して,「/sensor/temperature」リソースに GET リクエストを送信した結果である. これにより, HTTP によるデバイス操作のための REST アーキテクチャを利用して, M2M アプリケーションを作成する準備が完了する.

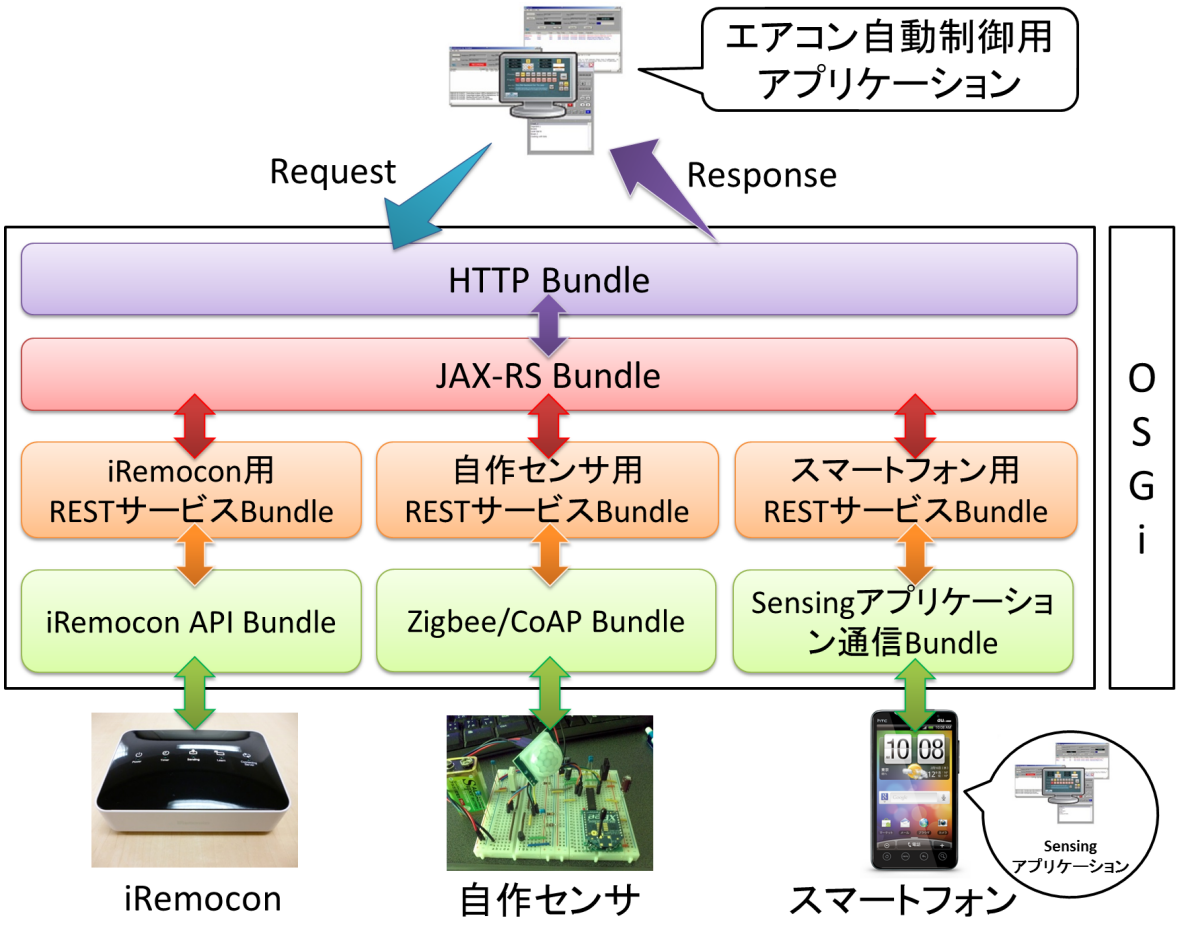

図 7.8　Bundle 構成

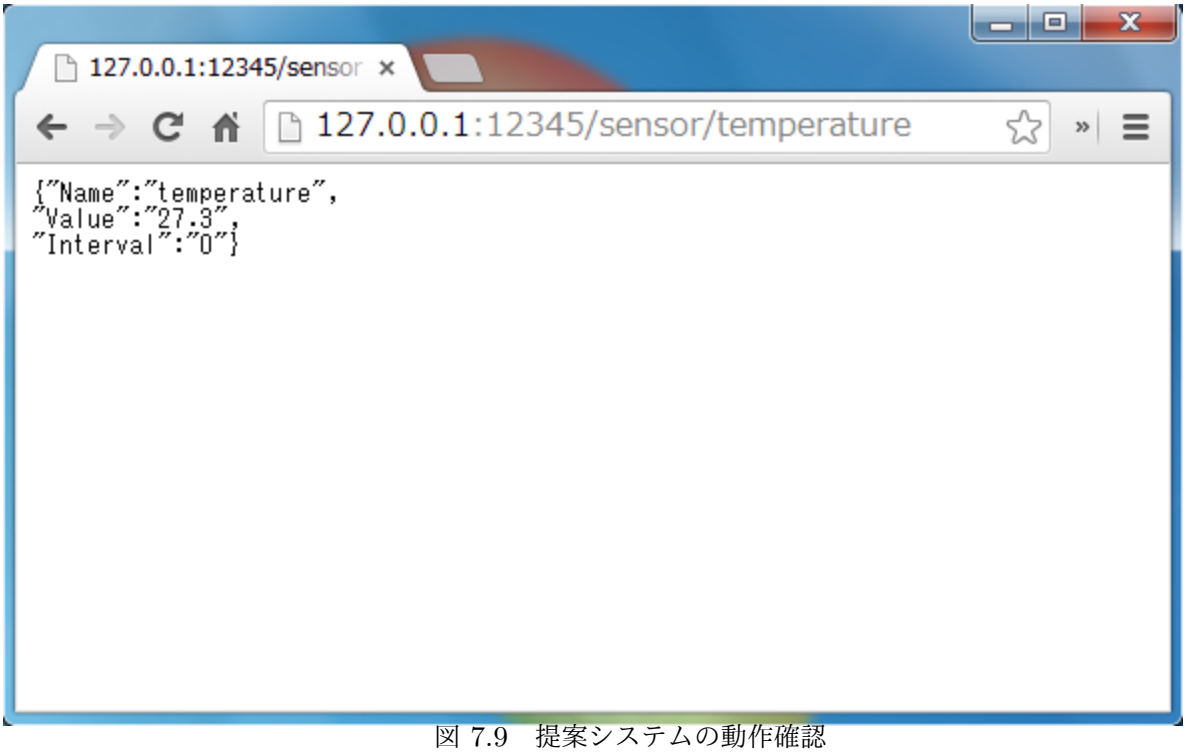

図 7.9　提案システムの動作確認

7.3.4　REST によるコーディング

提案システムを介して定義したリソースにアクセスできるようになった場合, M2M アプリケーションの開発を行う. M2M アプリケーションは, Android タブレット PC 上で動作するアプリケーションとして開発するため, Java で作成した.

ソースコード 7.3 は, M2M アプリケーションの例である. ソースコード 7.3 では, 独自に作成した HTTP クラスを使用している. HTTP クラスには, GET/-POST/PUT/DELETE メソッドが実装されており, これらのメソッドを使用して定義されたリソースにアクセスすると図 7.9 で確認したようなレスポンスを受け取ることが可能である.

これによりソースコード 7.3 では, iRemocon と温度センサの電源を ON にして, エアコンの温度が指定温度以下になるまで iRemocon の赤外線信号を発信している. 最後に目的の温度に到達した場合, iRemocon と温度センサの電源を OFF にする動作を行っている.

このように REST 処理のみで各デバイスの操作を記述することが可能である.

ソースコード 7.3　提案システムを介した場合の M2M アプリケーション例

```
1  // iRemoconON
2  HTTP.PUT( "/iRemocon" ) ;
3  // 温度センサON
4  HTTP.PUT( "/sensor/temperature" ) ;
5  // 温度値の取得
6  iTempValue = HTTP.GET( "/sensor/temperature" ) ;
7
8  // 温度確認
9  while( true )
10 {
11   if( iTempValueが指定温度以下だった場合 )
12   {
13     // 温度確認終了
14     break ;
15   }
16
17   // iRemoconで温度を下げる信号を発行
18   HTTP.GET( "/iRemocon/signal/1" ) ;
19   // 一定時間待機
20   WAIT() ;
21 }
22
23 // iRemoconOFF
24 HTTP.DELETE( "/iRemocon" ) ;
25 // 温度センサOFF
26 HTTP.DELETE( "/sensor/temperature" ) ;
27 // 終了
28 return ;
```

第 8 章

実験と評価

この章では, 提案システムを介した M2M アプリケーション開発における実験と評価について説明する.

8.1 評価項目

本研究の目的は, M2M アプリケーションの開発を容易にすることである. そのため, 要求条件を満たしているか, 調査する必要があるため, 以下に示す項目を調査した. なお, 要求条件 (1) については, ケーススタディのサンプル M2M アプリケーションを構築する際, 独自で定義したリソースを REST サービス化することが可能だったように, 正確にリソースを定義できていれば, 満たすことが可能である.

- **開発効率の調査 (要求条件 (2)(3) への対応)**
  提案システムを介すことで M2M アプリケーションの開発が容易になったか調査する必要がある. そのため, ケーススタディでサンプル M2M アプリケーションを使用して, 提案システムを介す場合と介さない場合とのソースコード量を比較した. また, OSGi に Bundle として REST サービスを追加/削除した際の利点について, 定性評価を行った.
- **通信ボトルネックの調査 (要求条件 (4) への対応)**
  リアルタイム性を重視するような M2M アプリケーションを REST のみで開発する場合, 大量のリクエストが要求される場合がある. そのため, 提案システムを介すことで発生する通信ボトルネックが, M2M アプリケーション開発時に影響しないか調査する必要がある. そこで, 大量のリクエストを要求し, それぞれのリクエストの応答時間の平均から提案システムを介すことによる遅延を調査した.
- **パフォーマンスの調査 (要求条件 (5) への対応)**
  提案システムは HomeGateWay(HGW) Server のような比較的パフォーマンスの低い計算機上でも動作させることを検討している. しかし, 上記でも述べたように大量のリクエストが要求されることで, 負荷がかかる可能性がある. そこで, パフォーマンスの低い計算機上でも動作するか調査する必要があるため, 通信ボトル

ネックの調査と同時に, OSGi を実行している計算機のメモリ使用量と CPU 使用率を調査した.

8.2 開発効率の調査

M2M アプリケーション開発における容易性を調査するため, 7 章で紹介したケーススタディのソースコードを用いて, 提案システムを介した場合と介しない場合のソースコード量を比較した. なお, ソースコードは, 一般的なコーディング規約をもとに Java で作成した.

まず, サンプル M2M アプリケーションのソースコードをもとに提案システムを介す場合に必要となるソースコードと介さない場合でも必要となるソースコードに分類した.

表 8.1 と 8.2 は, 分類した場合のソースコード比率を示したものである. 表 8.1 により, 提案システムを介す場合に必要となるソースコード量は全体の約 12.1 %であり, 表 8.2 により, 提案システムを介しない場合に必要となるソースコード量は全体の約 87.9 %である.

表 8.1 提案システムを介す場合に必要なソースコード量

概要	コード割合
REST リクエスト送信処理 (HTTP 通信)	5.4 %
REST レスポンス解析処理 (JSON 解析)	6.7 %

表 8.2 提案システムを介さない場合なソースコード量

概要	コード割合
iRemocon 用の通信処理 (TCP)	13.1 %
iRemocon 用の操作処理 (iRemocon API)	9.9 %
自作センサ用の通信処理 (Zigbee/RS232C)	4.3 %
自作センサ用の操作処理 (CoAP)	40.2 %
Android Sensing アプリケーション通信処理	20.4 %

表 8.2 により, 提案システムを介さない場合, 必要な処理を全て M2M アプリケーションに実装する必要がある. また, 新しいデバイスを使用するたびに M2M アプリケーションに処理を追加しなければならない可能性がある. 更に, 使用するデバイスによってソースコード量も変化する. 例えば, 自作センサ用の操作処理は, 他の処理に比べてソースコード量が多くなっている. これは他の処理と比較して複雑な処理であることを示している.

このように使用するデバイスによっては, 複雑な処理を実装する必要があり, M2M アプリケーションが肥大化する可能性があることを考慮しなければならない.

そのため，我々は表 8.2 に示す各デバイス固有の処理を, 図 7.8 のように OSGi 上に Bundle として作成した．これにより，表 8.1 に示すような OSGi に対して REST リクエストを送信する処理と REST レスポンスを解析する処理が必要となった．REST リクエストを送信するためには, HTTP 通信を実装すればよく, デバイスごとに固有の処理を用意する必要はない．しかし, REST レスポンス解析処理は, レスポンスの形式により, コード量が肥大化する可能性がある．そこで, 今回作成した REST サービスは, JAX-RS の JSON Parser を使用して単純な JSON 形式に変換したものをレスポンスとしている．したがって, JAX-RS の JSON Parser を使用する場合にのみ, 共通の JSON 形式を解析することになるため, REST レスポンス解析処理を使い回すことが可能である．これにより, M2M アプリケーションは, 表 8.1 に示す処理のみで実装可能である.

次に, 新しいデバイスを追加したときの変化について考える．図 8.1 は, Android 端末を使用せず, 提案システムを介す場合と介さない場合のソースコード割合を示したものである．これにより, 提案システムを介さず, Android 端末を使用する場合, 図 8.2 で示すように M2M アプリケーションで新たに 23.5 %の処理を実装する必要があった．しかし， 提案システムを介す場合, M2M アプリケーションは, 図 8.1 の提案システムを介す場合と介さない場合の共通割合である 15.4 %の処理のみで実装可能である．したがって, M2M アプリケーションには, 新しいデバイスが追加されても, 新たに処理を追加する必要はない.

また, 提案システムを介す場合, 図 8.2 の処理は, OSGi 上で処理を管理しているため, 仕様変更にも柔軟に対処することが可能である．例えば, 自作センサ用の通信ミドルウェア Bundle に変更があったと仮定する．もし, 提案システムを介さない場合, 自作センサを利用している全ての M2M アプリケーションの修正が必要となる．しかし, 提案システムを介す場合, 自作センサ用の REST Service Bundle で定義されている REST リソースに合わせて自作センサ用の通信ミドルウェア Bundle を修正することで, 全ての M2M アプリケーションに修正を反映させることが可能である．したがって, 仕様変更時でも M2M アプリケーションに与える影響を小さくすることができる.

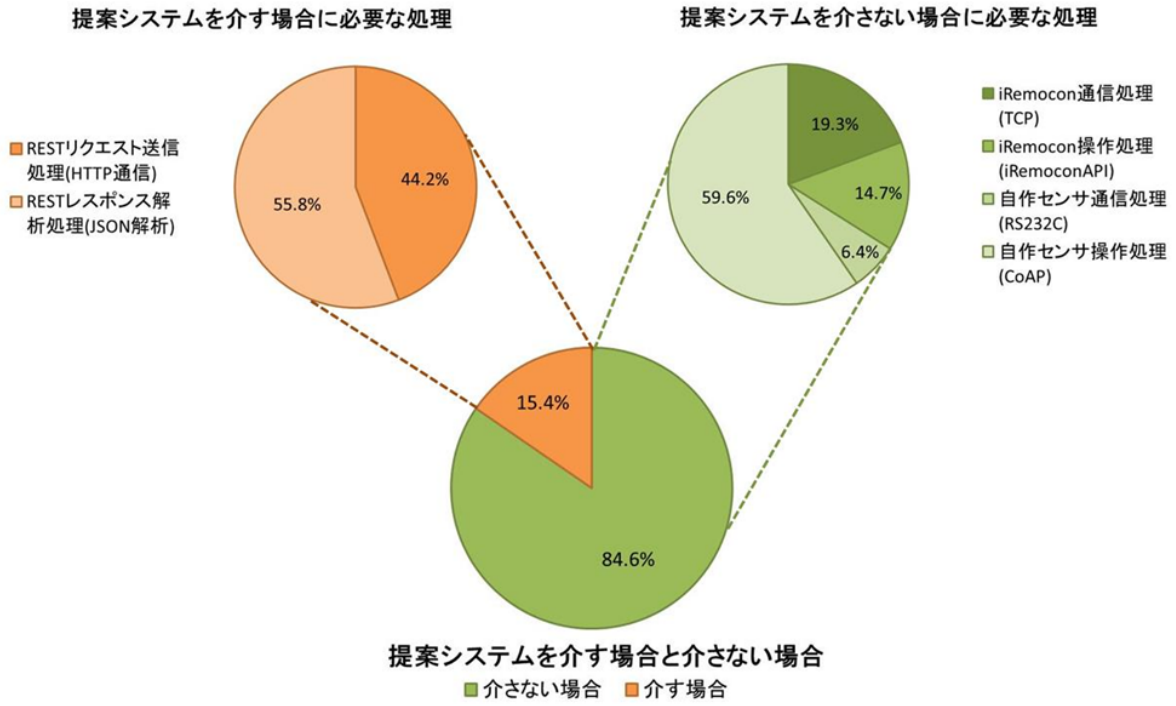

図 8.1　各コードの割合

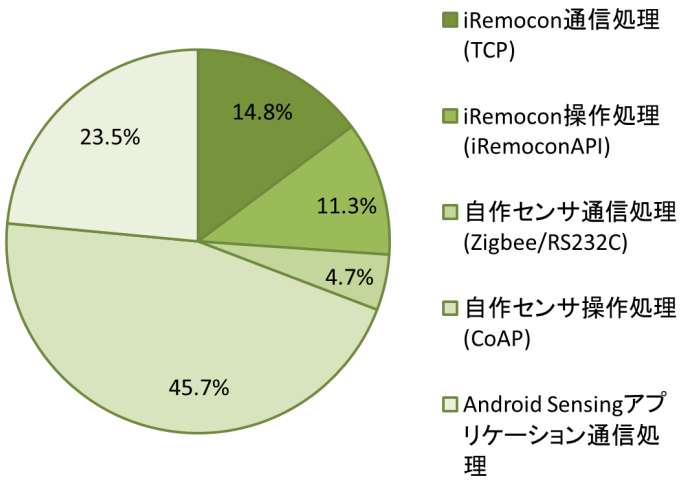

図 8.2　Android 端末追加時のソースコード割合

8.3 通信ボトルネックの調査

提案システムを介すことによる通信ボトルネックが M2M アプリケーション開発時に影響を与えないか調査するため, JMeter を用いてテストを行った. JMeter とは, パフォーマンス計測用の Java アプリケーションである [26]. Web アプリケーションなどに対して負荷テストを行う際などに利用されている.

8.3.1 負荷テストの準備

負荷テストでは, JMeter を用いて OSGi に対して 1 秒当たり 100 回, 200 回, 300 回, 400 回, 500 回のリクエストを約 1 時間送信し, その応答速度を調査した. 図 8.3 は, JMeter の設定画面である. 負荷テストを行うに当たり, JMeter では以下の項目を設定する必要がある. なおこの項目は「スレッドグループ」の項目から設定可能である.

- スレッド数
- Ramp-UP 期間 (秒)
- ループ回数

スレッド数とは, JMeter を実行する端末から発行するリクエストのスレッド数を示している. 実際には「ユーザの数」と同等の意味になる. Ramp-UP 期間とは, 指定したスレッド数を生成する時間を示している. 実際には, アクセス期間と同等の意味になる. なお 0 を指定すると同時アクセスするという意味になる. ループ回数とは, 1 スレッドがリクエストを発行する回数を示している. 実際には連続アクセス回数と同等の意味になる. これにより, どれだけのユーザ (スレッド数) がどれくらいの期間 (Ramp-UP 期間) にどの程度アクセス (ループ回数) してくるのかを設定する.

次に, JMeter を使用してどこにリクエストを送信するのか設定する必要がある. 図 8.4 は, JMeter の HTTP リクエスト設定画面である. この画面で, サーバアドレスやアクセスするリソースなどを設定する.

これを踏まえ, 表 8.3 の設定項目で, 1 秒あたり「100」「200」「300」「400」「500」のリクエストを送信するテストを各 1 時間ずつ行った.

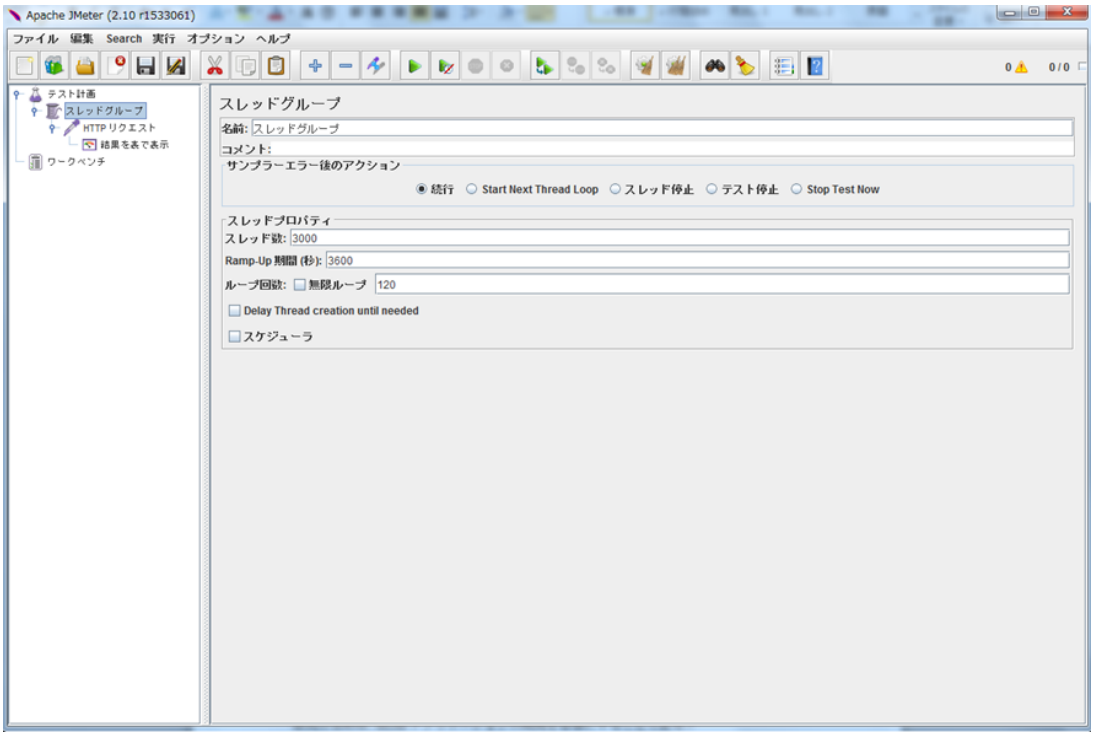

図 8.3　JMeter の動作設定画面

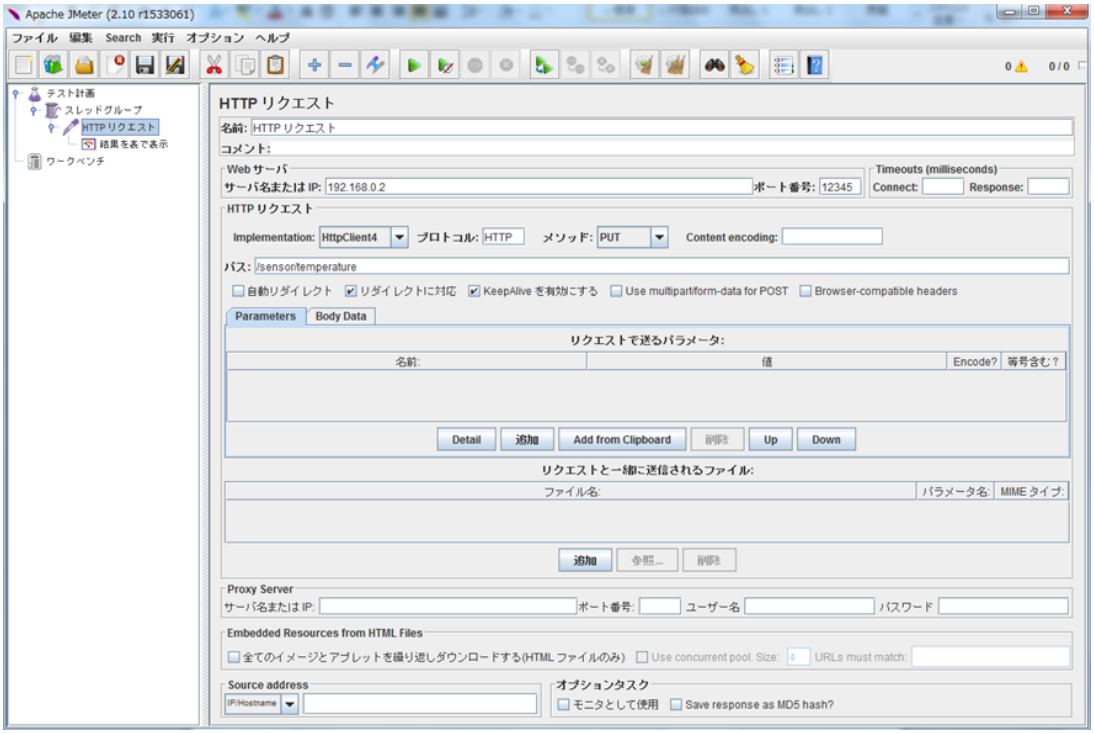

図 8.4　JMeter の HTTP リクエスト設定画面

表 8.3　テスト設定

スレッド数	3000
Ramp-UP 期間 (秒)	3600
1 秒当たり 100 リクエストの場合のループ回数	120
1 秒当たり 200 リクエストの場合のループ回数	240
1 秒当たり 300 リクエストの場合のループ回数	360
1 秒当たり 400 リクエストの場合のループ回数	480
1 秒当たり 500 リクエストの場合のループ回数	600

8.3.2　負荷テストの結果

図 8.5, 8.6, 8.7, 8.8, 8.9 は, 1 秒あたり「100」「200」「300」「400」「500」のリクエストを処理させた際の応答速度を示したものであり, 表 8.4 は, そのときの平均応答速度である. これにより, 1 秒当たり 500 リクエストの場合, 急激に応答速度が低下している. したがって, 1 秒当たり 400 リクエスト程度が通信面での提案システムの限界値であると考えられる.

表 8.4　平均応答速度

1 秒当たり 100 リクエストの場合	8ms
1 秒当たり 200 リクエストの場合	10ms
1 秒当たり 300 リクエストの場合	13ms
1 秒当たり 400 リクエストの場合	18ms
1 秒当たり 500 リクエストの場合	197ms

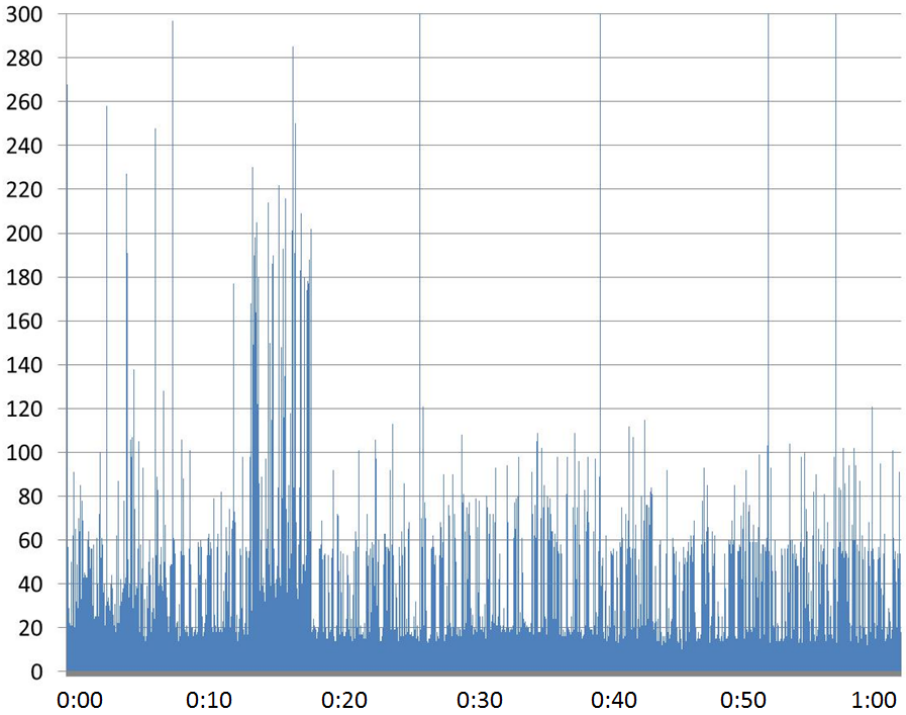

図 8.5　1 秒当たり 100 リクエスト時の応答速度

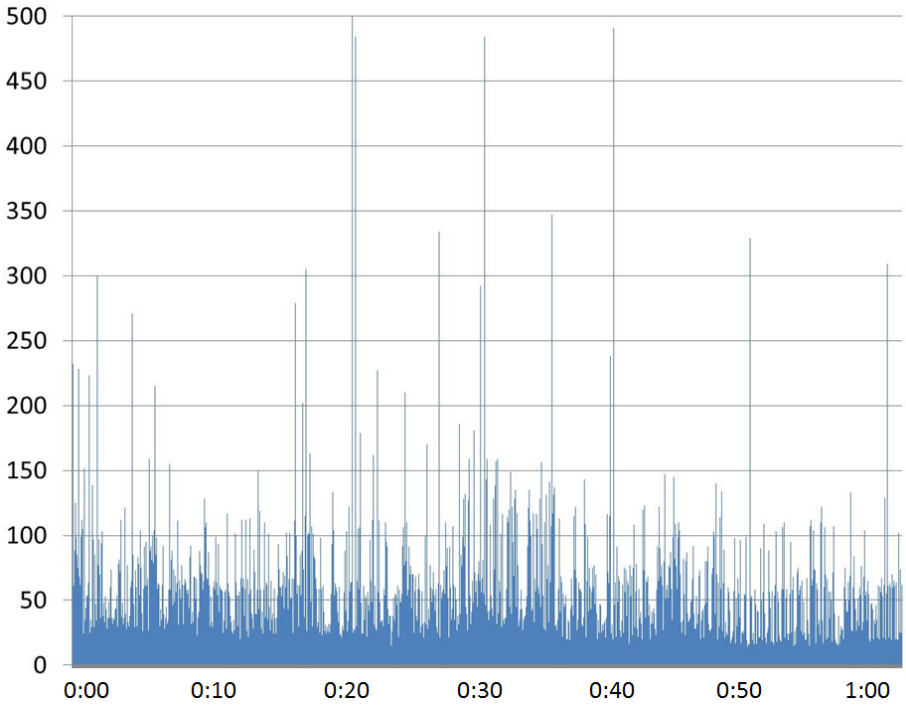

図 8.6　1 秒当たり 200 リクエスト時の応答速度

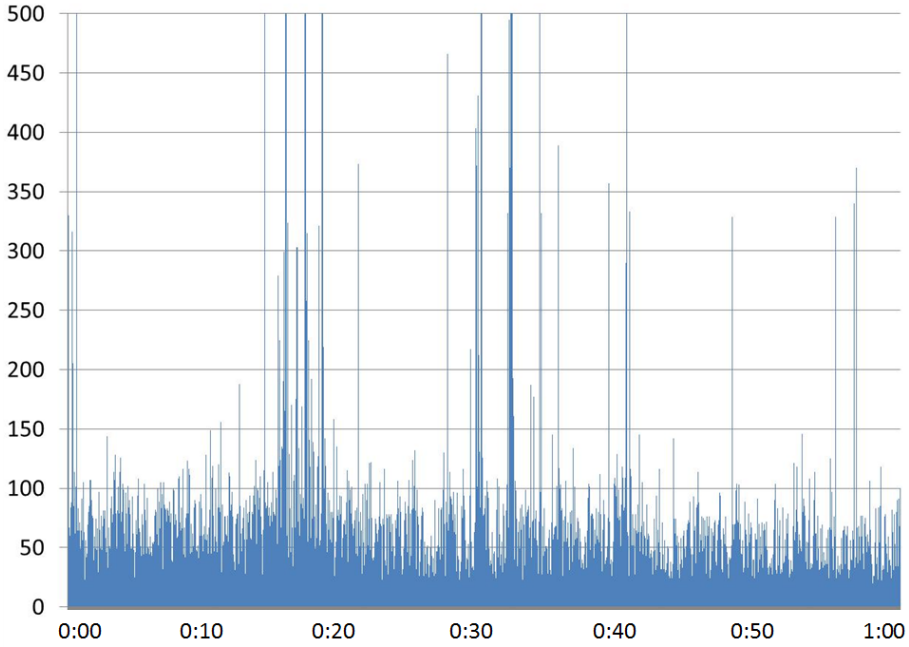

図 8.7　1 秒当たり 300 リクエスト時の応答速度

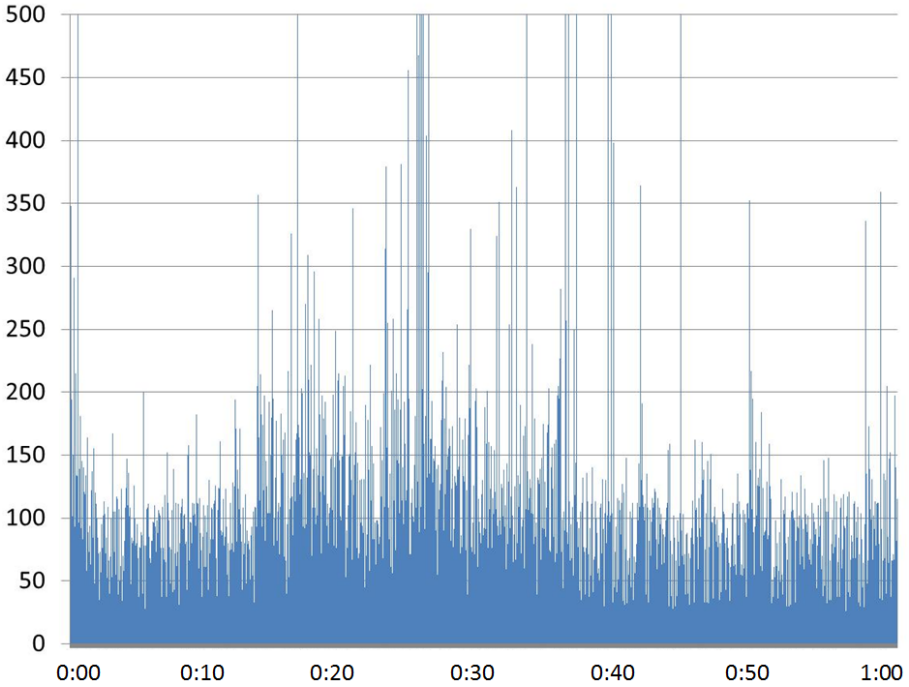

図 8.8　1 秒当たり 400 リクエスト時の応答速度

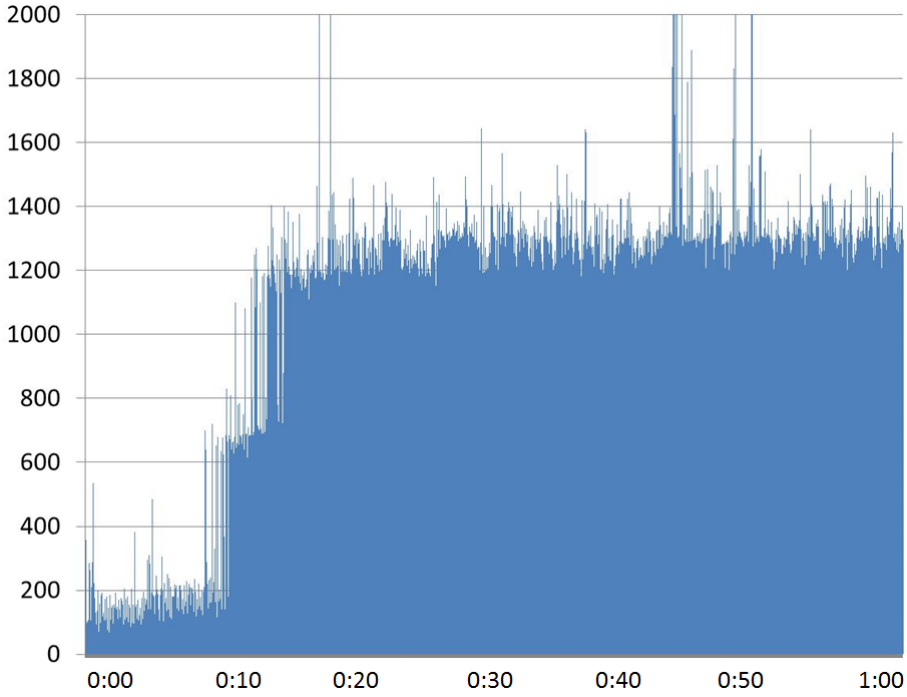

図 8.9　1 秒当たり 500 リクエスト時の応答速度

8.4　パフォーマンスの調査

提案システムは, HGW Server のような比較的パフォーマンスの低い計算機上でも動作させることを検討している. そこで, 通信ボトルネックテストと同時に, OSGi を実行する計算機上で, Java アプリケーションの CPU 使用率やヒープメモリ使用量を測定するためのツールである JConsole を用いてメモリ使用量と CPU 使用率の調査を行った. なお, HGW ではヒープサイズは 50MB に設定されているが, 今回は提案システムの限界値や各リクエスト毎に理想のヒープサイズを知るために OSGi を実行する計算機上の JVM は最大ヒープサイズは 1GB で測定を行った.

図 8.10, 8.12, 8.14, 8.16, 8.18 は, 1 秒あたり「100」「200」「300」「400」「500」のリクエストを処理させた際のメモリ使用量を示したものであり, 図 8.11, 8.13, 8.15, 8.17, 8.19 は, 1 秒あたり「100」「200」「300」「400」「500」のリクエストを処理させた際の CPU 使用率を示したものである.

1 秒あたりのリクエスト数「100」の場合, 受信バッファが蓄積されていき, メモリ使用量が上昇しているが, ガベージコレクションが動作して最大 60MB 程度で収まっている. また CPU 使用率は, 10 %から 15 %程度で安定した動作を見せている. 1 秒あたりのリクエスト数「200」の場合, メモリ使用量は, 最大 70MB 程度だが, その後, 50MB 程度で安

定した．CPU 使用率は, 25 %から 32 %程度であった．1 秒あたりのリクエスト数「300」の場合, メモリ使用量は, 最大 110MB 程度使用する箇所が見られるが, 50MB 程度で安定した動作を見せている．CPU 使用率は, 35 %から 50 %程度であった．1 秒あたりのリクエスト数「400」の場合, メモリ使用量は, 最大 350MB 程度使用した．安定した箇所に着目しても平均 50MB から 80MB 程度使用している．CPU 使用率は, 50 %から 70 %程度であった．1 秒あたりのリクエスト数「500」の場合, メモリ使用量は, 最大 380MB 程度で頻繁にガベージコレクションが動作している．CPU 使用率は, 60 %から 70 %程度であった．リクエストの失敗も見られたため, パフォーマンス面での提案システムの限界値であると考えられる．

この結果を踏まえ, 表 8.5 に各テスト別に理想ヒープサイズを示す．考察として HGW Server と同様にヒープサイズを 50MB にした場合, 1 秒間に「400」から「500」リクエスト時には, 頻繁にガベージコレクションが動作する可能性があり, それにより CPU 使用率も上昇すると考えられる．また, それにより, 通信面にも影響すると考えられる．しかし, 1 秒間に「100」から「300」リクエスト時には, ガベージコレクションの動作回数が上昇する可能性はあるが, 最大メモリ使用量や平均使用量を見る感じ, 処理可能範囲だと考えられる．

表 8.5　理想のヒープサイズ設定

1 秒当たり 100 リクエストの場合のループ回数	60MB
1 秒当たり 200 リクエストの場合のループ回数	70MB
1 秒当たり 300 リクエストの場合のループ回数	110MB
1 秒当たり 400 リクエストの場合のループ回数	350MB
1 秒当たり 500 リクエストの場合のループ回数	370MB

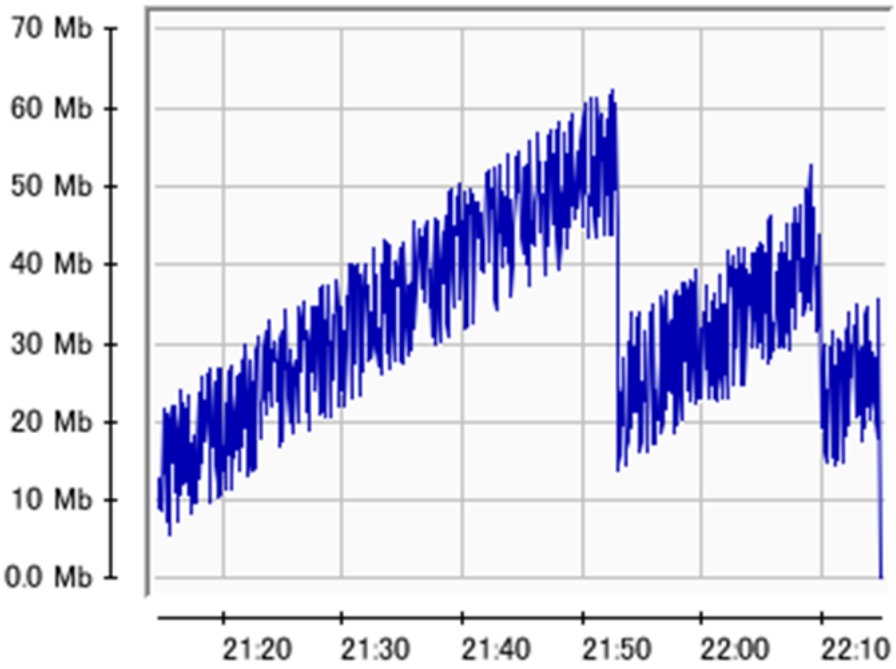

図 8.10　1 秒当たり 100 リクエスト時のメモリ使用量

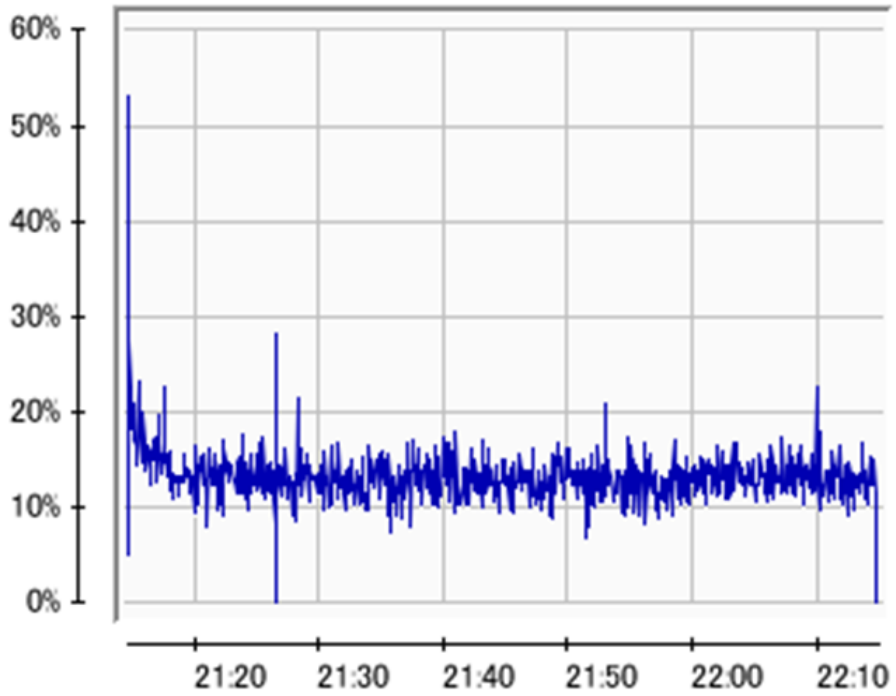

図 8.11　1 秒当たり 100 リクエスト時の CPU 使用率

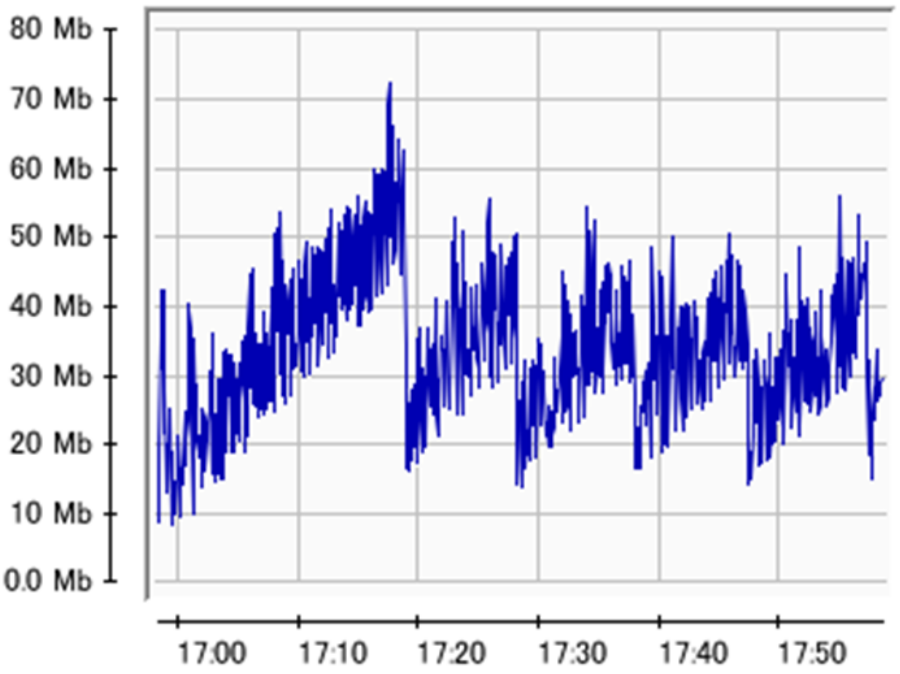

図 8.12　1 秒当たり 200 リクエスト時のメモリ使用量

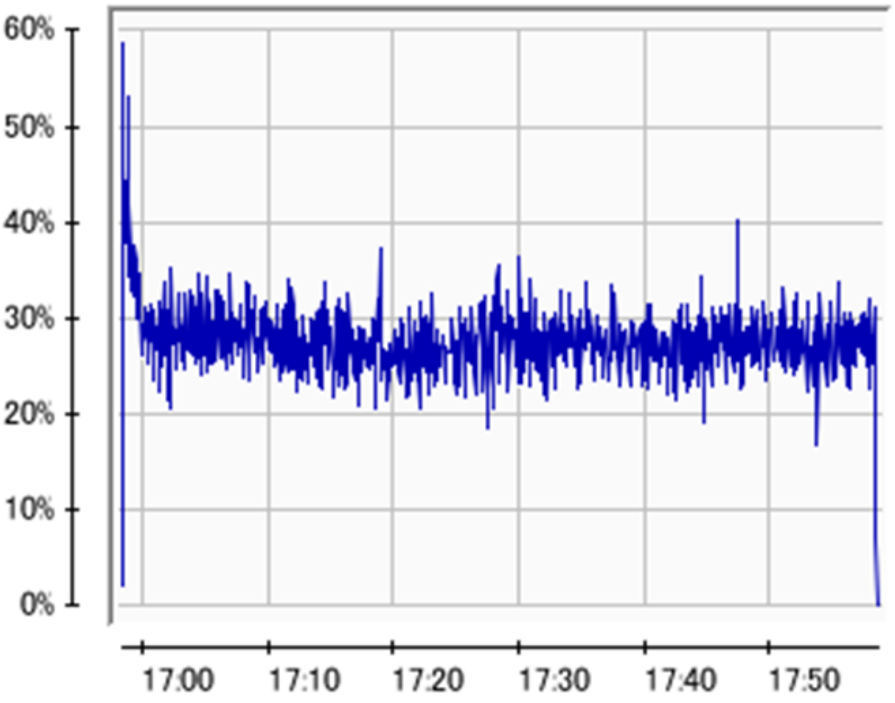

図 8.13　1 秒当たり 200 リクエスト時の CPU 使用率

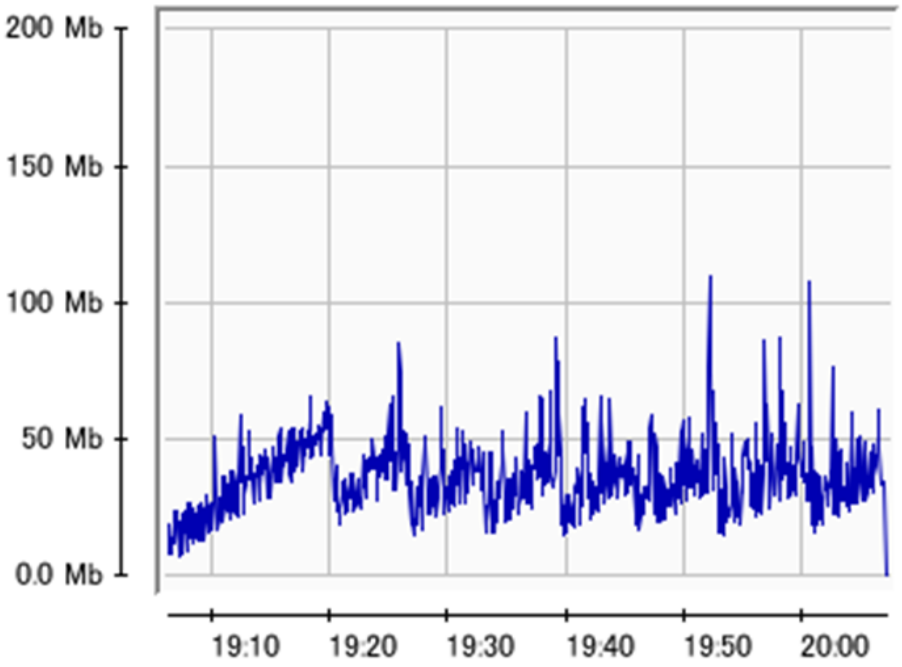

図 8.14　1 秒当たり 300 リクエスト時のメモリ使用量

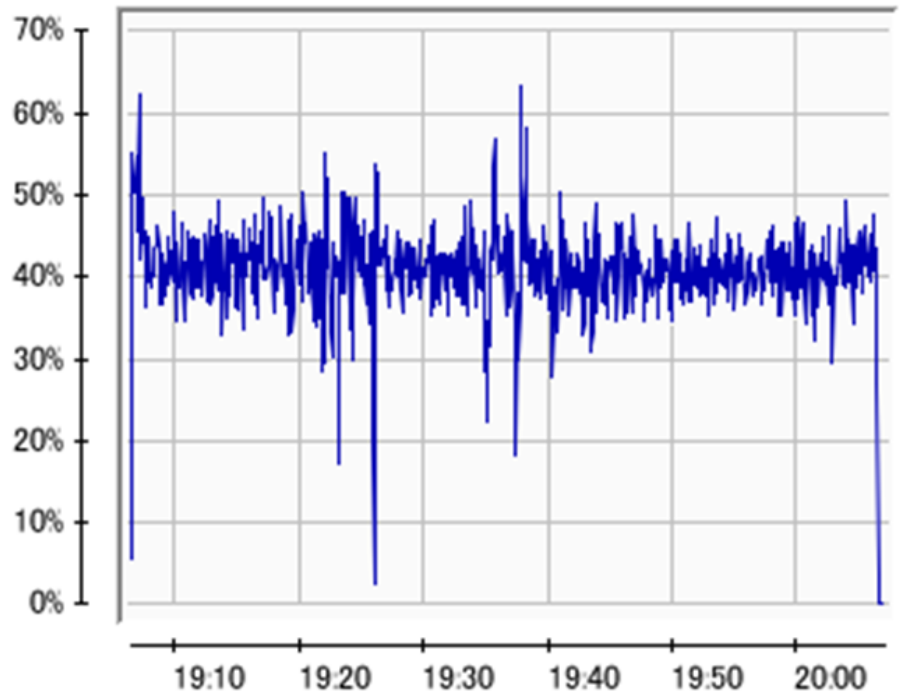

図 8.15　1 秒当たり 300 リクエスト時の CPU 使用率

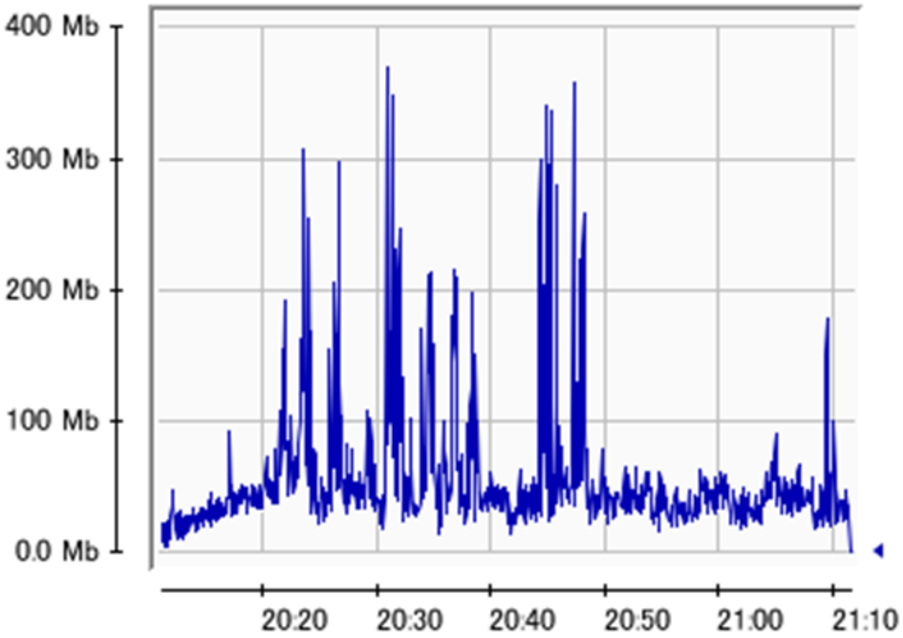

図 8.16　1 秒当たり 400 リクエスト時のメモリ使用量

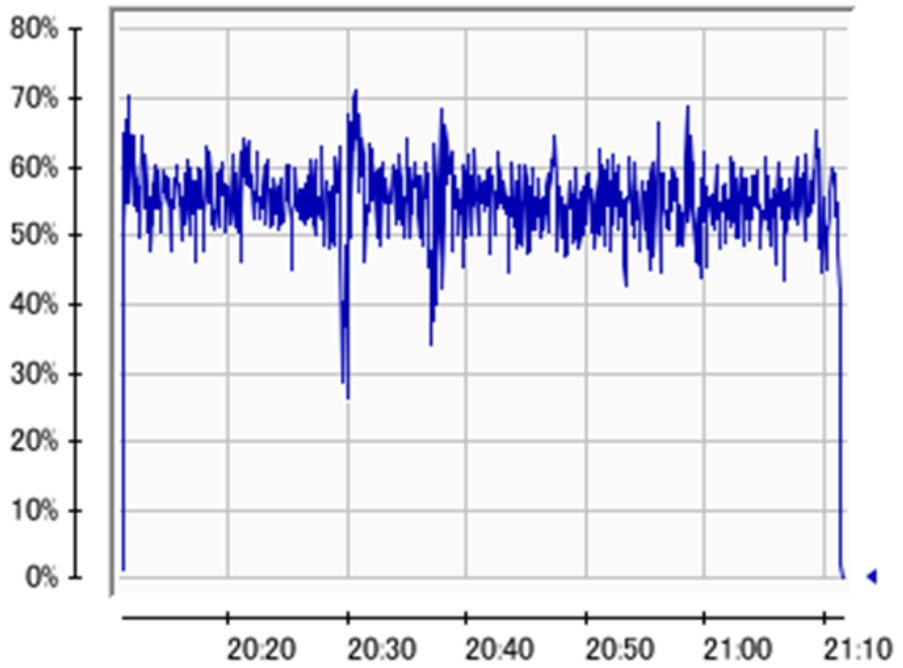

図 8.17　1 秒当たり 400 リクエスト時の CPU 使用率

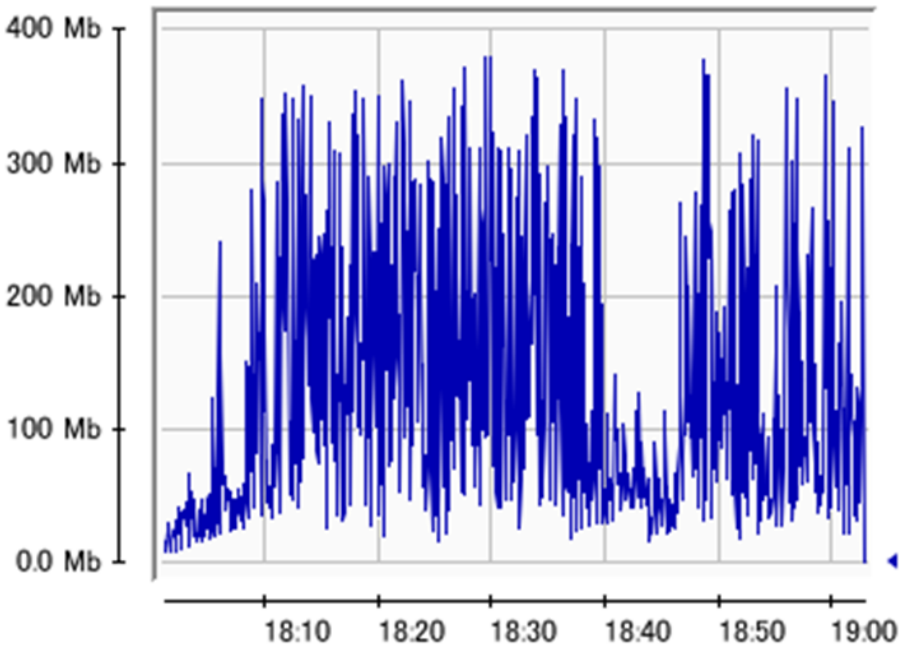

図 8.18　1 秒当たり 500 リクエスト時のメモリ使用量

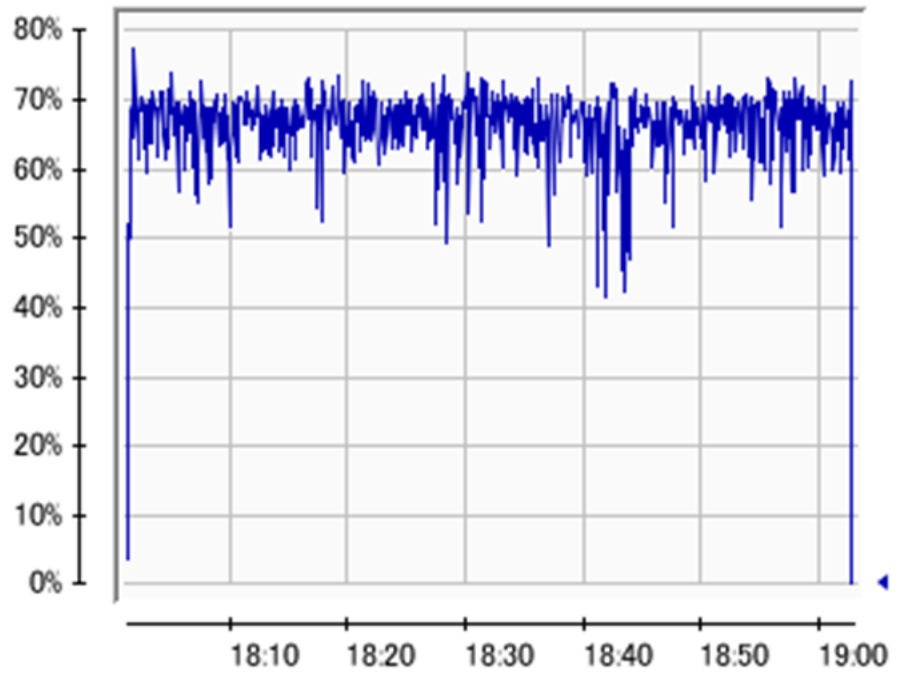

図 8.19　1 秒当たり 500 リクエスト時の CPU 使用率

8.5 評価考察

評価結果より以下に考察を述べる.

- **要求条件 (1) への対応**
  ケーススタディで使用したデバイスを REST サービスにするために, リソースの定義を行った. これにより, 今回使用した全てのデバイスおよびデバイスの操作は, 全てリソース定義可能だった. そのため, 定義したリソースをもとに JAX-RS で REST サービスを作成することができ, それにより, REST を使用してデバイスの操作も可能である. したがって, 様々なデバイスに対応可能である.
- **要求条件 (2)(3) への対応**
  提案システムを介すことで, REST 処理のみで M2M アプリケーションの開発が可能である. そのため, 開発者はデバイス固有の処理を知らなくとも, 実装が行うことができ, デバイス固有の処理を全て実装するよりもソースコード量は, 少なくなった. また新しいデバイスが追加されても特に新しい処理を追加する必要はない. さらに, REST サービスは OSGi の Bundle として付け外し可能であるため, デバイスの追加や除去にも柔軟に対応可能である.
- **要求条件 (4)(5) への対応**
  通信ボトルネックの調査より, 1 秒あたり 500 リクエスト時に平均応答速度が急激に低下した. またパフォーマンスの調査より, CPU は安定動作を見せているがメモリは 1 秒あたり 400 リクエスト時にまれに 350MB 程度を使用し始め, 1 秒あたり 500 リクエスト時には, 頻繁に使用し始める結果となった. そのため, 1 秒あたり 400 と 500 リクエスト時では, HGW Server のような比較的パフォーマンスの低い計算機上での動作は, 困難であると考えられる. したがって, HGW Server のような比較的パフォーマンスの低い計算機上で動作させることを考えると, 1 秒あたり 300 リクエスト程度が目安であると考えられる. ただし, 1 秒あたり 300 リクエストでは, 平均応答時間が 13ms であるため, 頻繁に起こるわけではないが 100ms〜500ms 程度の遅延が発生する可能性を考慮して M2M アプリケーションを実装する必要があると考えられる.

第 9 章

関連研究及びサービス

この章では, 関連研究およびサービスについて説明する.

9.1 デバイスの操作に関する研究

9.1.1 住宅 API

住宅 API は, 大和ハウス工業株式会社のスマートハウスが提供している住宅内の家電や設備機器を統合的にコントロールするアプリを開発するためのコマンドセット（命令集）である [27]. 住宅 API では, 図 9.1 に示す URL に対して, 表 9.1 で示すパラメータを付加してリクエストを送信することで図のような XML の結果を得ることが可能である.

http://[HGW IP]/smart/rest/request?deviceid=echonet.HomeAirConditioner_X

図 9.1 住宅 API 用 URL

ソースコード 9.1 住宅 API のレスポンス

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resultset type="map">
3   <result type="value">true</result>
4   <data type="map">
5     <result type="value">true</result>
6   </data>
7 </result>
```

このように住宅内の家電や設備機器を REST のように操作可能である. しかし, 住宅 API は, パラメータに動作を指定して GET リクエストを送信することで制御を行っているため, 厳密には REST を使用していない. そのため, 住宅 API が仕様変更になった場合, アプリケーションも住宅 API に合わせて変更しなければならなくなる可能性がある.

また, 住宅 API は, OSGi を使用して, 家電や設備機器を管理しているが, ユーザが自由に Bundle を追加したり, 削除したりすることは不可能である.

表 9.1　住宅 API 例

キー		値		
名称	reqkey	名称	reqvalue	URL
エアコン電源状態設定	Operation Status	オン	ON	&reqkey=OperationStatus &reqvalue=ON
		オフ	OFF	&reqkey=OperationStatus &reqvalue=OFF
運転モード設定	Mode	自動	Auto	&reqkey=Mode&reqvalue=Auto
		冷房	Cooling	&reqkey=Mode&reqvalue= Cooling
		暖房	Heating	&reqkey=Mode&reqvalue= Heating
		除湿	Dehumidifying	&reqkey=Mode&reqvalue= Dehumidifying
湿度設定	Humidity Setting	除湿低	Low	&reqkey=HumiditySetting& reqvalue=Low
		除湿	Middle	&reqkey=HumiditySetting& reqvalue=Middle
		除湿高	High	&reqkey=HumiditySetting& reqvalue=High
温度設定	TemperatureSetting	温度	温度値	&reqkey=TemperatureSetting& reqvalue=17
風量設定	Wind Volume	自動	Auto	&reqkey=WindVolume&reqvalue=Auto
		微風	Low	&reqkey=WindVolume&reqvalue=Low
		弱風	Middle	&reqkey=WindVolume&reqvalue=Middle
		強風	High	&reqkey=WindVolume&reqvalue=High
空気清浄機能状態設定	AirPurifier	オン	ON	&reqkey=AirPurifier&reqvalue=ON
		オフ	OFF	&reqkey=AirPurifier&reqvalue=OFF

9.1.2 Sensing Web

ユビキタスセンサネットワーク (Ubiquitous Sensor Network : USN) は，ごく近い将来に社会全体に普及し，世界各地に様々な用途の USN が多数設置される．そこで, 図 9.2 のように USN から得られるセンサ情報を Web のように誰もが自由に利用できる仕組みが研究されている [28].

しかし, Sensing Web の研究では, 誰でも自由にセンサ情報を使用する仕組みが検討されているが取得したセンサ情報の活用方法については現時点では, あまり検討されていない．そのため, 今後様々なセンサから得られるセンサ情報を活用したシステムの検討が行われていくことが予測されるため, これらを容易に構築できる仕組みが必要となると考えられる.

図 9.2 Sensing Web(文献 [28] より抜粋)

9.1.3 SENCHA と REST

生形らの研究では, SENCHA(Smart Environment for Controlling Home Appliances) と呼ばれるホームコンピューティングミドルウェアに REST を導入することを提案して

いる [29].

図 9.3 は, SENCHA の概要を示したものである. SENCHA では, ユーザが持っているパーソナル端末 (ここでは PAC と呼ぶ) にゲートウェイを設置して, PCA が移動しても RDF(Resource Description Framework) をもとにユーザがよく使用しているようなデバイスを周辺より検索する. デバイスが発見されると HTML による UI を生成する. ユーザは生成された UI(User Interface) を使用することで, デバイスより WSDL(Web Services Description Language) を取得し, SOAP で操作可能となる. このとき, 生形らは, SOAP によるデバイスの操作は, 通信負荷がかかり SENCHA のようなミドルウェアでは非効率なため, REST によりデバイスを操作可能な仕組みを提案している. 評価として SOAP と REST による通信比較やメモリ使用量を調査しており, REST の方が有効であることを示している.

しかし, この提案では, デバイスにも PAC 同様に SENCHA が必要であり, 導入にコストがかかる. また, デバイス間で連携を取るために図 9.4 で示すようなイベント通知接続を使用している. これは, HTTP の SUBSCRIBE メソッドを使用してデバイスが指定した状態になったとき, 他のデバイスの操作を行うといったパラメータを送信しておき, デバイスが指定した状態になったとき, そのデバイスが他のデバイスに新しいリクエストを送信する仕組みである. しかし, この通信方式では, 全てのデバイスに M2M アプリケーションを実装していかなければならないことになる. また, 複数のデバイスの連携も困難であると考えられる.

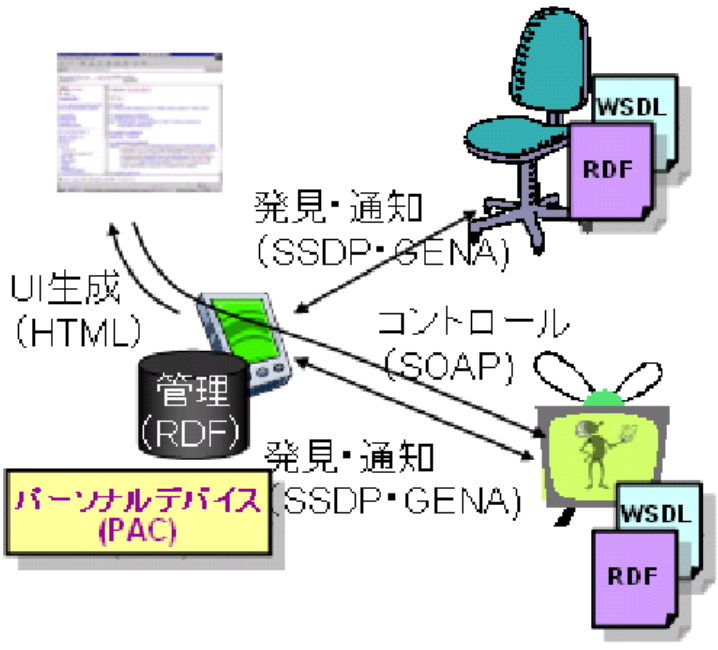

図 9.3　SENCHA の概要 (文献 [29] より抜粋)

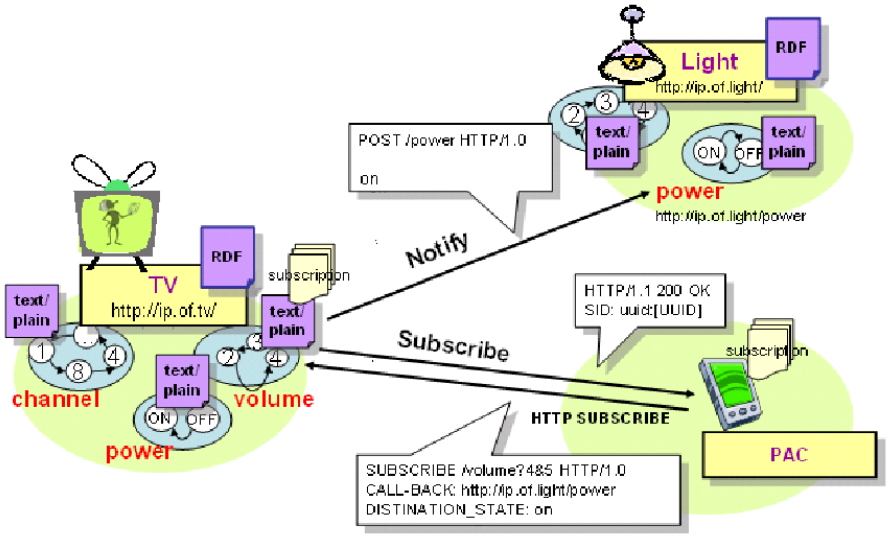

図 9.4　SENCHA のイベント通知接続 (文献 [29] より抜粋)

9.1.4　Apache AXIS

福田らの研究では, Apache AXIS を用いて家電を制御するための Web サービスを提案している [30].

Apache AXIS とは, Java と XML 技術に基づいた Web サービスのフレームワークである. SOAP サーバの実装, Web サービスを生成するためのツール, Web サービスをデプロイ (配備) するためのツールなどから構成されている.

福田らの研究では, 図 9.5 に示すように, 指定した時間に指定した家電を動作させたり, 停止させたりするために, ホームネットワークシステム内を監視し, 家電の API を実行するリアルタイム実行基盤として Apache AXIS を用いている.

ユーザは, リアルタイム実行基盤に対して家電をどのように動作させるのを記したサービス定義書を REST を用いて実行や登録, 取り消しなどといった操作を行うことが可能である. また, 外部のアプリケーションからも REST を使用してスケジュールの編集が可能である.

しかし, リアルタイム実行基盤とホームネットワーク間では, REST は使用させていない. そのため, ホームネットワークの構成が変化したり, 各家電の API が変化した場合, リアルタイム実行基盤の修正が必要となる. また, サービス定義書は, 「機器名」「操作名」「操作に必要なパラメータ」を含んでいる, 福田らが独自に定義したデータ構造を HTTP で送信している. したがって, 厳密には REST を使用しておらず, リアルタイム実行基盤の仕様変更が起こった場合, アプリケーションに与える影響は, 大きいと考えられる.

本研究では, 外部アプリケーションとホームネットワークが REST でやり取りされるため, リアルタイム実行基盤もユーザが自由に構築することが可能である. また, OSGi と REST を使用してデバイスを操作するため, 仕様変更時にアプリケーションに与える影響を小さくすることが可能である.

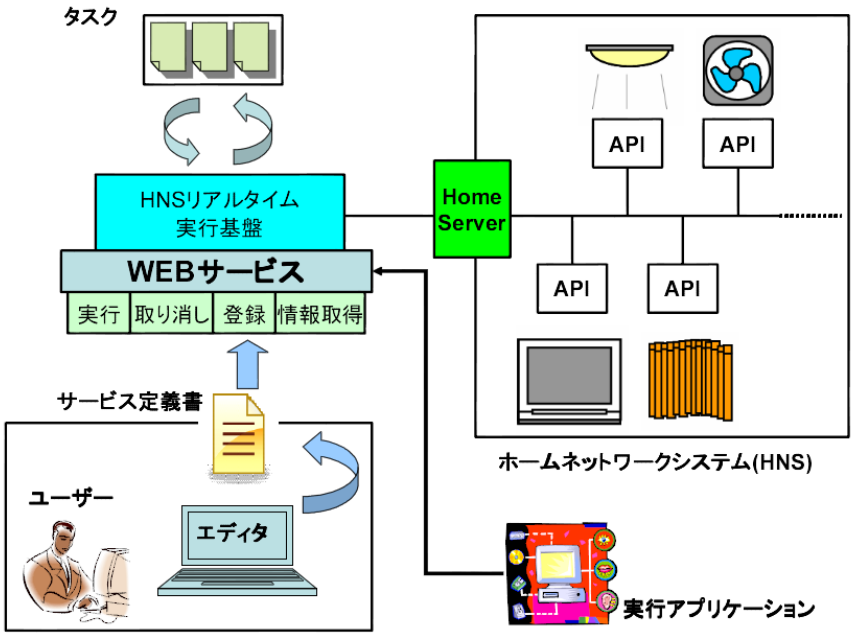

図 9.5　家電リアルタイム制御システム (文献 [30] より抜粋)

9.2　M2M に関する研究

9.2.1　Android と M2M

Matko らの研究では, Android 端末からセンシングしたデータを収集する M2M アプリケーションを開発することを提案している [6]. Matko らは, Android 端末によって実装されているセンサの種類が異なるため, 図 9.6 のように OSGi を Android 上に実装し, センサを Bundle で管理することを提案している. また, Android 端末からセンシングしたデータを収集する計算機もセンサの種類に応じて, 処理を柔軟に追加や削除可能にするために OSGi を使用している. これにより, 機能の追加やデバイスの変更時は, Bundle のインストール/アンインストールを行うことで対処可能である.

評価として, センシングするアプリケーションを Android アプリケーションと Android 上で動作している OSGi の Bundle アプリケーションとして実装を行い, メモリの使用量や消費電力, アプリケーションの起動時間を比較している. これにより, 通常の Android アプリケーションとして実装するよりも OSGi の Bundle アプリケーションとして実装を

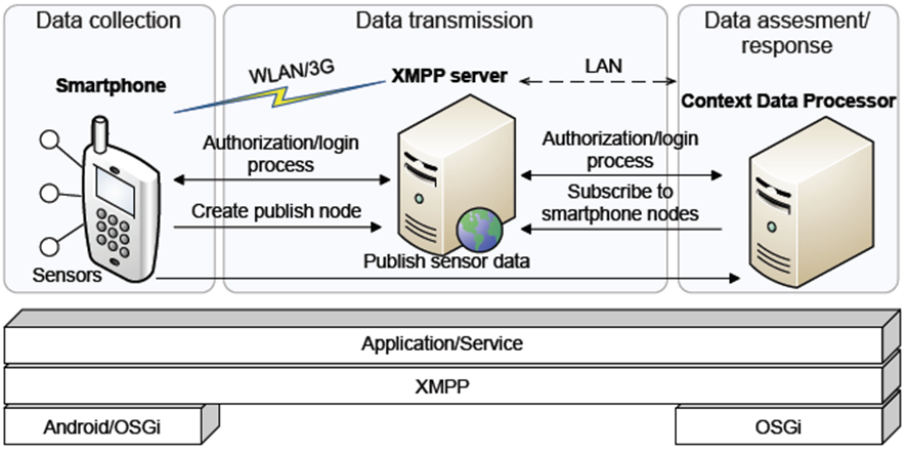

図 9.6　Android と OSGi による M2M アプリケーション (文献 [6] より抜粋)

することでアプリケーションのパフォーマンス面でも有効であるとしている.

しかし, このシステムでは, Android 端末と情報収集計算機の間で, XMPP(eXtensible Messaging and Presence Protocol) と呼ばれるプレゼンテーションプロトコルを利用している. そのため, Android 端末と情報収集計算機の仲介役を務める XMPP Server が必要である. また, このシステムでは, Android 端末と情報収集計算機の両方に OSGi が必要となる. したがって, Android 端末と情報収集計算機の両方に OSGi の Bundle として, アプリケーションを開発する必要があり, 開発にコストがかかる. デバイスとアプリケーション間に HGW Server のような仲介役を設置し, その上で OSGi を動作させるほうが, 様々なデバイスの違いを吸収して, アプリケーションの開発を容易にすることができると考えられる.

9.2.2　OpenMTC と FOKUS Broker

Elmangoush らは, OpenMTC と呼ばれるミドルウェアを使用して構築された M2M ネットワークに対して FOKUS Broker と呼ばれる SDP(Service Delivery Platform) を使用して, M2M ネットワーク内にあるデバイスにアクセスできる仕組みを検討している [31].

OpenMTC は, 図 9.7 のように IMS(IP Multimedia Subsystem) と同様にデバイスを IP 化し, M2M ネットワークを構築する. このとき, Elmangoush らは, 構築された M2M ネットワークの操作やアクセス権限の設定, 管理されているデバイスの操作やデータを取得するために REST による API を定義している.

また, 定義した API を使用して図 9.8 のように FOKUS Broker により M2M ネットワークにアクセスすることを可能にしている. これにより, サードパーティ製のアプリ

ケーションは, OpenMTC の仕組みを知らなくても, M2M ネットワーク内にあるデバイスを操作することが可能になる.

しかし, この API を動作させるためには OpenMTC によって構築された M2M ネットワークと FOKUS Broker の両方が必要であり, 導入にコストがかかる. また, 定義されている API は, OpenMTC により構築したネットワークのためだけに定義されたものであり, 他のネットワークでは, 使用することはできない.

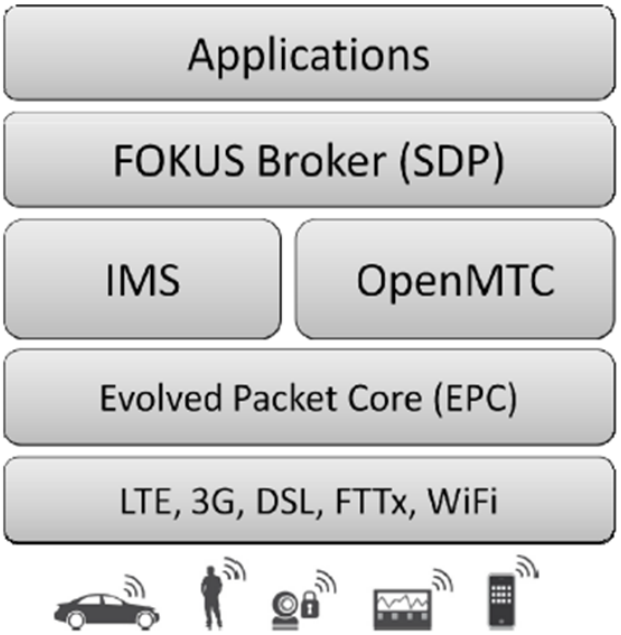

図 9.7　OpenMTC(文献 [31] より抜粋)

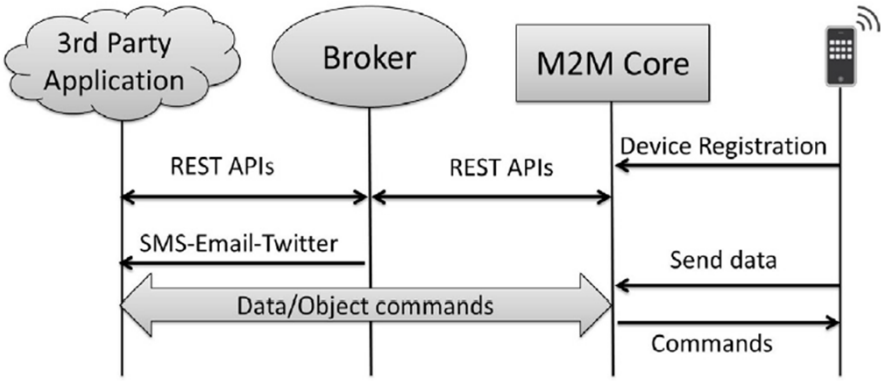

図 9.8　FOKUS Broker(文献 [31] より抜粋)

9.2.3 SOA アーキテクチャとセンサネットワーク

Busemann らは, SOA(Service-Oriented Architecture) アーキテクチャを利用してセンサネットワークを制御する研究を行っている [32]. SOA アーキテクチャとは, アプリケーションなどをコンポーネント化 (部品化) し, それらを組み合わせてシステムを作る設計手法である. Busemann らは, センサネットワークにアクセスするための処理を SOA アーキテクチャを利用してコンポーネント化して, REST や SOAP でアクセスできる仕組みを提供している.

この研究では, SOA アーキテクチャを利用するために ServiceMix を利用している. 図 9.9 の構成を示したものである. ServiceMix の Gateway の箇所には OSGi が使用されており, センサネットワークにアクセスするためのミドルウェアを Bundle として追加/削除可能である. 次に, Gateway でセンサネットワークから取得したデータは, Backend に送られる. Backend では, センサネットワークからのデータを蓄積したり, Frontend に受け渡すデータを作成する役目がある. Frontend では, Backend よりデータを取得し, それをもとにアプリケーションを動作させる役目がある. 今回の構成では Frontend は, Web サーバとして構築されている. これにより, アプリケーションはセンサネットワークの仕様を知らなくとも実装が可能になる.

しかし, ServiceMix を使用してセンサネットワークにアクセスするためのサービスを作成するためには OSGi だけでなく, Backend と Frontend を導入する必要があり, 導入にコストがかかる. また, ServiceMix 用に作成された Bundle は, ServiceMix でしか動作させることができない.

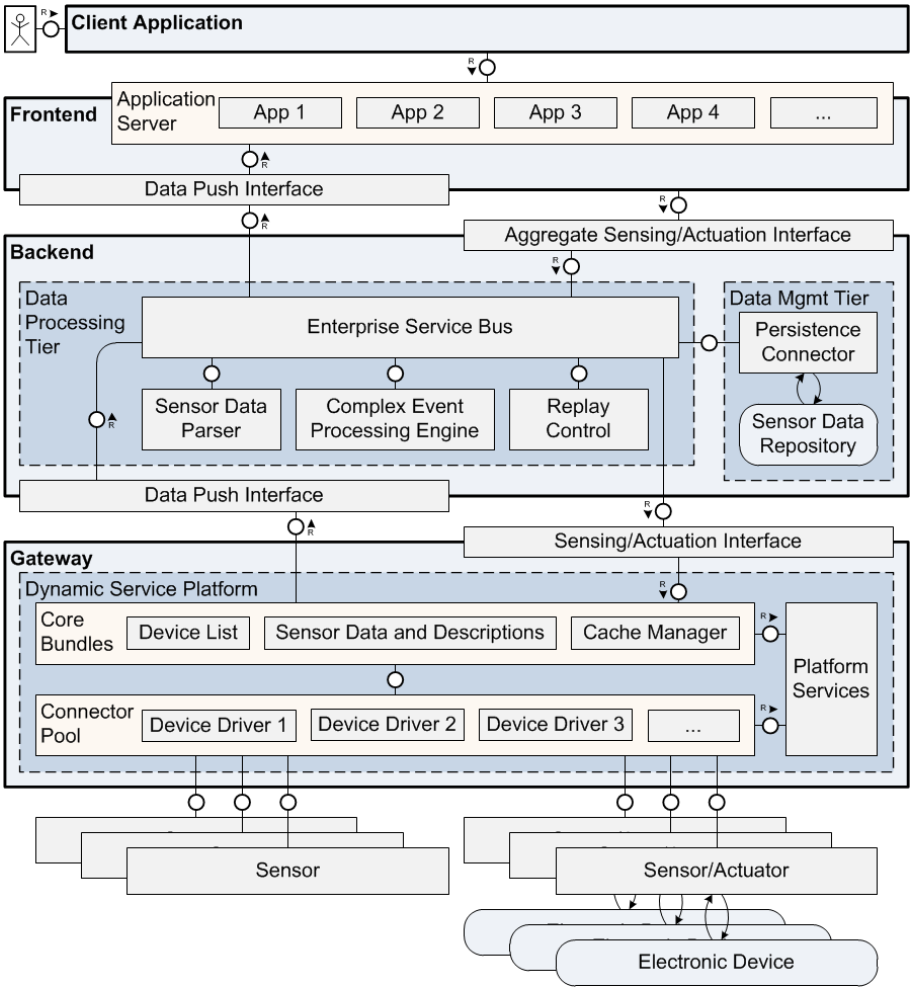

図 9.9　ServiceMix(文献 [32] より抜粋)

9.2.4　物理/仮想統合センサメッセージングシステム

類似研究として, 慶応義塾大学の小澤らが, センサの仮想化を行うことにより, センサを使用したプログラミングを容易にする研究を行っている [4].

図 9.10 は, 小澤らの提案システムである. このシステムでは, 物理センサがネットワークに参加するために UPnP(Universal Plug and Play) を使用している. しかし, UPnP はセンサをコントロールポイントに集約可能だが, それだけではアプリケーションなどにデータを公開できない. そのため, モノを識別 (個体識別) するための番号体系の標準化である EPC(ElectronicProductCode) を使用してセンサを一意に識別し, アクセスを可能にしている. これにより, 「http://x.x.x.x/urn:epc:id:sgtin:457122707.100.1」のような HTTP リクエストを物理センサに送信することで 1 分間に 1 度, センサ値を取得する. 仮想センサは, HTTP リクエストを物理センサに送信し, センサ値を取得することで構築さ

れる. また仮想センサにも EPC が付加され, 物理センサと同様に HTTP リクエストでアクセス可能になる.

この小澤らは, この仕組みを使用して図 9.11 のような照明消し忘れ通知ケーススタディを構築している. ケーススタディでは, 人感センサと照明センサを組み合わせた照明消し忘れを通知する仮想センサと更に別の箇所に存在する人感センサを組み合わせた仮想センサおよび音声合成アクチュエータから成る. これにより, 仮想センサから消し忘れ通知があった場合, 音声にてユーザに通知する仕組みである.

この研究は, 本研究と非常に近いため, 表 9.2 に差分を示す. まず違いとしては, 物理センサと仮想センサの管理のためにゲートウェイが 2 つ必要であるが, 本研究では, OSGi のみで管理可能である. また, デバイスをネットワークに参加させるために, UPnP を使用しているが, 本研究では, OSGi に Bundle をインストール/アンインストールすることでデバイスの参加/離脱が可能である. 更に, デバイスを管理するために EPC を使用しているが, 本研究では, JAX-RS を用いた REST サービスクラスで可能である. そして, デバイスを操作する場合, 必要な値を購読するための登録に HTTP を使用しているが, 本研究では, REST を用いてデバイスの操作を行えるようにしてる. したがって, デバイスの操作は, ユーザが詳細に作成する必要があるが, 本研究では REST を用いることで容易に記述することが可能である.

表 9.2 物理/仮想統合センサメッセージングシステムとの比較

	小澤らの提案	本研究の提案
ゲートウェイの数	2(物理センサと仮想センサの管理に必要)	OSGi のみ
デバイスの参加	UPnP	OSGi Bundle のインストール
デバイスの管理	EPC	JAX-RS
デバイスの操作	HTTP で値を購読するかどうかの登録を行う	REST

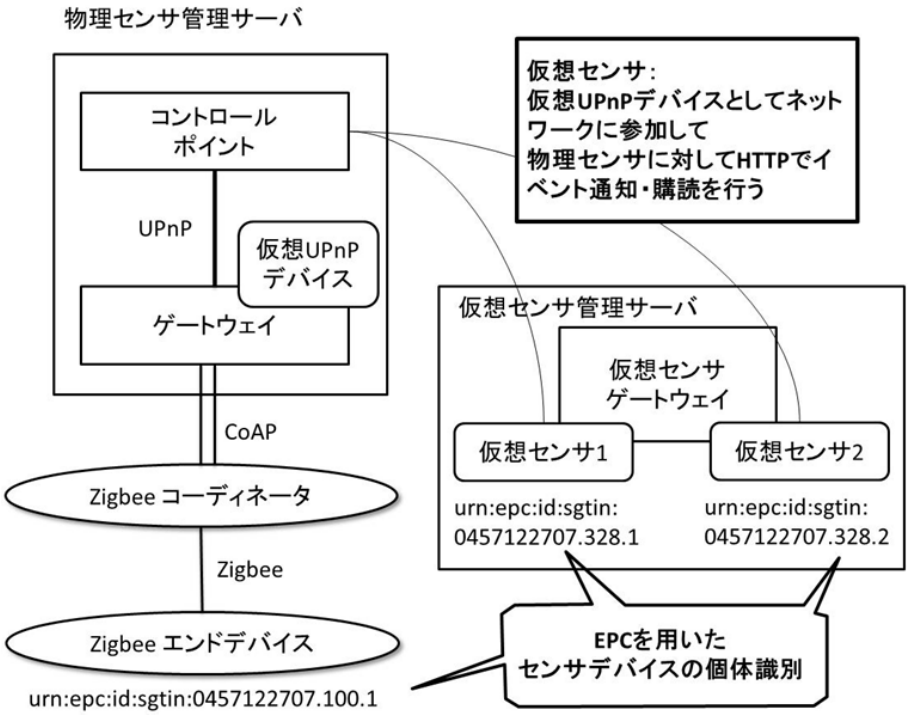

図 9.10　物理/仮想統合センサメッセージングシステム (文献 [4] より抜粋)

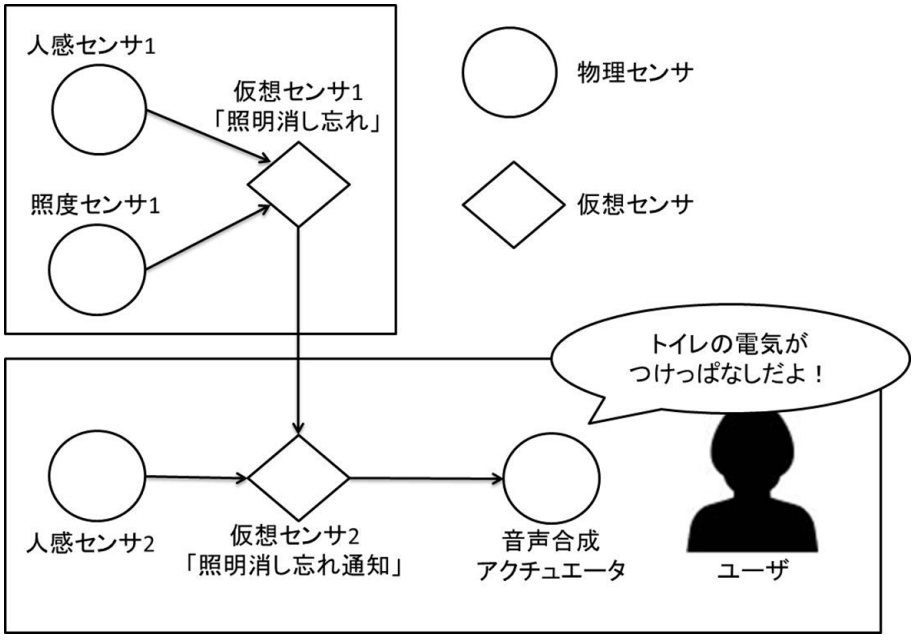

図 9.11　照明消し忘れ通知システム (文献 [4] より抜粋)

9.3 関連研究との差異

表 9.3 は, 要求条件の観点から本研究と関連研究との差異を示したものである.

表 9.3 関連研究との差異

	要求条件 (1)	要求条件 (2)	要求条件 (3)	要求条件 (4)	要求条件 (5)
住宅 API	×	○	×	△	○
SENCHA	×	○	△	×	○
AXIS	○	×	×	○	×
Android OSGi	×	×	○	○	○
OpenMTC	×	○	○	△	×
ServiceMix	○	×	×	○	×
本研究	△	○	○	△	○

住宅 API は, OSGi を使用しているがデバイスを個人が自由に追加することが不可能であるため, 要求条件 (1) を満たしていない. また, 厳密には REST を使用していないため, 仕様変更が起こるとアプリケーションにも影響を与える可能性がある. したがって, 要求条件 (3) を満たしていない.

SENCHA は, 使用する全てのデバイスに導入しなければならないため, 未導入のデバイスでは使用することはできない. したがって, 要求条件 (1) を満たしていない. また, REST を使用しているが, デバイス間でやり取りを行うためにイベント通知接続を行っている. デバイスの構成が変わるとイベント通知接続も再構成が必要になる可能性がある.

AXIS は, デバイス公開者が, デバイス動作のスケジュールを作成するためのサービスを公開しているものである. したがって, デバイス公開者しか, デバイスを追加することができない. また, デバイスの細かい制御を REST を使用して行えないため, 開発効率を重視したものではない. 更に, HGW のような場所での動作も難しいと考えられる.

Android OSGi は, アプリケーションも Bundle として構築される. また, Android のセンサを管理することしか想定していない. したがって要求条件 (1)(2) を満たしていない.

OpenMTC は, それを使用して構築したネットワークでしか使用できない. そのため, 要求条件 (1) を満たしていない. また, ネットワークにアクセスするためには, FOKUS Broker が必要となるが HGW のような場所での動作も難しいと考えられる. したがって, 要求条件 (5) を満たしていない.

ServiceMix は, 複数のインタフェースを公開しており, 場合によっては, アプリケー

ション開発や仕様変更による影響を与える可能性がある．そのため要求条件 (2)(3) を満たしていない．また, ServiceMix も HGW のような場所での動作も難しいと考えられる．したがって要求条件 (5) を満たしていない．

本研究は, REST を使用することでアプリケーションの開発を容易にすることが可能である．また, OSGi のみでデバイスの REST サービスを提供することが可能である．しかし, 様々なデバイスの REST サービスを作成するためには, リソースの定義が必要である．また, HTTP のみによる REST では, システムの限界を考慮した通信を行う必要がある．

第 10 章

結論と今後の予定

この章では，本研究の結論及び今後の課題を述べる．

10.1 結論

近年, 通信可能なデバイスの増加により, M2M 通信も増加している．M2M 通信を利用することで, 情報の自動収集や機器の自動化制御といった M2M アプリケーションが期待されている．しかし, M2M アプリケーションを構築するためには, デバイス毎に異なる通信処理や制御処理を作成する必要がある．そのため, これらを全て M2M アプリケーションに実装するとソースコードが肥大化し, 複雑になる要因となる．

そこで本研究では, REST アーキテクチャを使用して, 様々なデバイスに共通の方法で操作できる仕組みを検討した．この仕組みを使用することで, Web を操作する感覚でデバイスを操作することが可能である．また, M2M アプリケーションでは, REST を使用してデバイスを操作するためのルールを記述するだけで実装可能であり, M2M アプリケーションに直接デバイス固有の処理を記述するよりもソースコード量を軽減できることを示した．さらに, この仕組みを HomeGateWay(HGW)Server などでデバイス管理のために使用されている OSGi に実装することでデバイスや機能の変更といった仕様の変化があったとしてもソースコード量が変化することがないことを示した．

そして, M2M アプリケーションは, 事例によってリアルタイム性が要求されるものもあるため, 提案システムを介す場合に発生する遅延がアプリケーション開発時に影響しないか通信ボトルネック調査を行った．その結果, 1 秒あたり 300 リクエストを 1 時間, 送信し続けても, 発生する遅延は, 13ms 程度あった．したがって, アプリケーション開発時にもさほど影響しないと考えられる．また, HTTP の Chunk 通信によりデバイスからのデータをストリーミングするための機能も検討した．

さらに, HGW Server のような比較的低パフォーマンスな計算機上での動作を検討しているため, 通信ボトルネック調査と同時に, OSGi を実行している計算機の CPU 使用率とメモリ使用量を調査した．その結果, 1 秒あたり 300 リクエストを 1 時間, 送信し続けた場合, メモリ使用量は平均約 50MB で CPU 使用率は, 約 35 %〜50 %程度であった．これ

により, 比較的低パフォーマンスな計算機上でも動作可能であることを示した.

また, 本提案システムは, OSGi 上に特定の Bundle をインストールするだけで OSGi に REST アーキテクチャの機能を付加することが可能である. したがって, 関連研究と比較した場合, 導入するためのコストも小さいと考えられる.

10.2 今後の予定

今後の課題として, 認証について検討する必要がある. 現在の提案システムでは, OSGi にアクセスすることができれば, 誰でも REST を使用して, デバイスの操作が可能になっている. しかし, M2M アプリケーションの事例によっては, デバイスにアクセスできるユーザを制限するための仕組みが必要になる場合がある. そこで, 今後の予定として, リソースに対して認証を設けることで, アクセスを制限するための仕組みを検討する予定である. なお, この仕組みは, OSGi の機能を使用することにより, 実装可能であると考えている.

また, OSGi は, Bundle を再利用して, 他の OSGi でも使い回せることが売りの一つである. したがって, 別の OSGi 上で同様のデバイスを使用する場合, 作成した Bundle がどの程度, 再利用可能か調査したいと考えている.

謝辞

本研究を遂行するにあたっては, いろいろな方々にお世話になりました.

まず, 指導教員の末田欣子先生には, 日頃から熱心なご指導, そしてご鞭撻を賜わりました. また，末田先生のみならず多田好克先生にも，ご多忙中にもかかわらず論文の草稿を丁寧に読んで下さり，大変貴重なご助言をいただきました．ここに厚く御礼申し上げます.

そして，本研究が行なえたことは，研究方針や方法論について議論をし，共に研究生活をおくってきた基盤ソフトウェア学講座の学生諸氏のおかげでもあります．また，同末田研究室の鈴木駿介や OB の西尾信吾にも助言や実験に協力して頂きました. 最後に，これらの皆さんに感謝いたします.

参考文献

[1] David Boswarthick, Omar Elloumi and Olivier Hersent, “M2M Communications: A Systems Approach,” WILEY, 2012.

[2] “いよいよ本格化する Internet of Things”, CISCO inSpire ISSUE 10, Jan.2013, http://cisco-inspire.jp/issues/0010/cover_story.html, 2013.

[3] “なぜ今 M2M ネットワークなのか？、注目を集める 2 つの理由”, http://eetimes.jp/ee/articles/1109/20/news067.html, 2013.

[4] 小澤みゆき, 三次仁 “物理/仮想統合センサメッセージングシステム,” 電子情報通信学会 ネットワークシステム (NS:Network System) 研究会, pp.125–131, 2013.

[5] Shih-Hao Hung, Chun-Han Chen, Chia-Heng Tu, “Performance Evaluation of Machine-to-Machine Systems with Virtual Machines,” Wireless Personal Multimedia Communications (WPMC), pp.159–163, 2012.

[6] Matko Kuna, Hrvoje Kolaric, Iva Bojic, Mario Kusek and Gordan Jezic, “Android/OSGi-based Machine-to-Machine Context-Aware System,” 11th International Conference, pp.95–102, 2011.

[7] “RESTful Web サービス”, オライリージャパン, Leonard Ricbardson and Sam Ruby, 2007.

[8] “Web を支える技術 -HTTP、URI、HTML、そして REST”, 技術評論社, 山本 陽平, 2010.

[9] “第 1 回 いまさら始める Java で RESTful Web サービス構築”, http://www.opentone.co.jp/news/release/article04/article0401.html, 2013.

[10] “JAX-RS REST で使用するアノテーション”, http://d.hatena.ne.jp/tanakakns/20120426/1335436944, 2013.

[11] “Eclipse や Spring で使われている基盤技術 OSGi とは”, http://www.atmarkit.co.jp/fjava/special/osgi/osgi_1.html, 2013.

[12] “スマートハウスの基盤技術としての OSGi”, http://thinkit.co.jp/story/2010/08/11/1708, 2013.

[13] “ホーム ICT 分野での OSGi 標準化活動の最新動向”, 山崎 毅文, 山崎 育生, 近藤 重邦, NTT 技術ジャーナル, 2012.2.

[14] “Transfer-Encoding: chunked について”, “http://d.hatena.ne.jp/kiririmode/20100606/p1”, 2013.

[15] “RFC2068”, “http://www.ietf.org/rfc/rfc2068.txt”, 2013.

[16] “Equinox - Eclipse”, http://www.eclipse.org/equinox/, 2013.

[17] “Welcome to Apache Felix”, http://felix.apache.org/, 2013.
[18] “Knopflerfish OSGi”, http://www.knopflerfish.org/, 2013.
[19] “Apache Felix（OSGi）− Apache Felix の使い方”,
http://d.hatena.ne.jp/IkeT/20090325/1237945826, 2013.
[20] “Java のアノテーションを独自で定義する”,
http://d.hatena.ne.jp/vaice/20091227/p1, 2013.
[21] “アノテーション作成方法”,
http://www.ne.jp/asahi/hishidama/home/tech/java/annotation.html, 2013.
[22] ‘リフレクション API”,
http://www.ne.jp/asahi/hishidama/home/tech/java/reflection.html, 2013.
[23] “iRemocon API”,
http://i-remocon.com/development/, 2013.
[24] “Constrained Application Protocol (CoAP)”,
“http://tools.ietf.org/html/draft-ietf-core-coap-13”, 2013.
[25] “ISW11HT”,
http://www.htc.com/jp/smartphones/htc-evo-wimax/, 2013.
[26] “Apache JMeter”, http://jmeter.apache.org/, 2013.
[27] “住宅 API ｜総合技術研究所｜ダイワハウス - 大和ハウス工業”,
http://www.daiwahouse.co.jp/lab/HousingAPI/, 2013.
[28] “Sensing Web — センサ情報の社会利用のためのコンテンツ化”,
http://www.mm.media.kyoto-u.ac.jp/sweb/, 2013.
[29] “REST を用いた家電制御システム構築へのアプローチ”, 早稲田大学大学院理工学研究科 情報ネットワーク専攻 修士論文, 生形裕貴, 2004.
[30] “ホームネットワークシステムにおけるリアルタイムな家電制御サービスの実現”, 神戸大学工学部情報知能工学科 卒業論文, 福田将之, 2007.
[31] Asma Elmangoush, Thomas Magedanz, Alexander Blotny, Niklas Blum, “Design of RESTful APIs for M2M Services,” 16th International Conference on Intelligence in Next Generation Networks, pp.50–56, 2012.
[32] C. Busemann, V. Gazis, R. Gold, P. Kikiras, A. Kovacevic, A. Leonardi, J. Mirkovic, M. Walther, H. Ziekow, “Enabling the Usage of Sensor Networks with Service-Oriented Architectures,” 7th International Workshop on Middleware Tools, 2012.