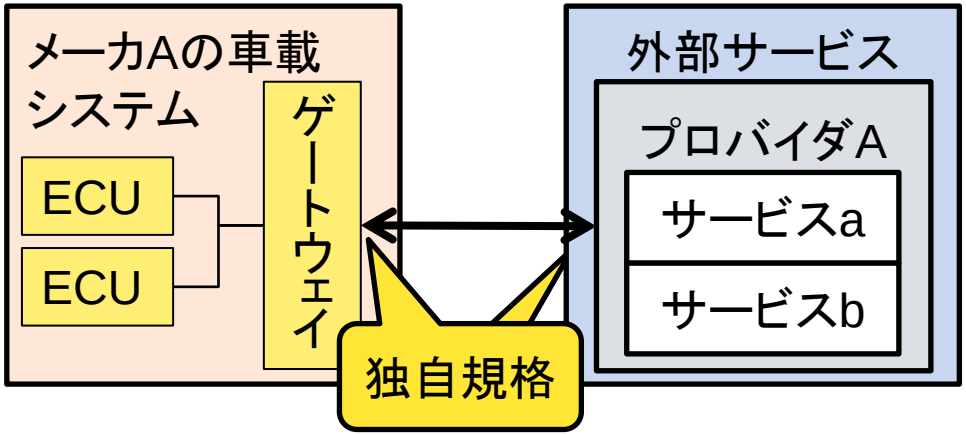
# OSGIを用いたサービスブローカ

2008MI011　朝倉知也
2008MI079　岩井　　大

# 目次

# 1.1 車外サービス連携の問題点(1/2)

- 近年のサービス連携の現状




- 車載システム内のGWがECUを制御，連携を行う
- 各メーカ毎にサービス提供
  メーカとプロバイダが密結合
  →プロトコル，インタフェースの統一

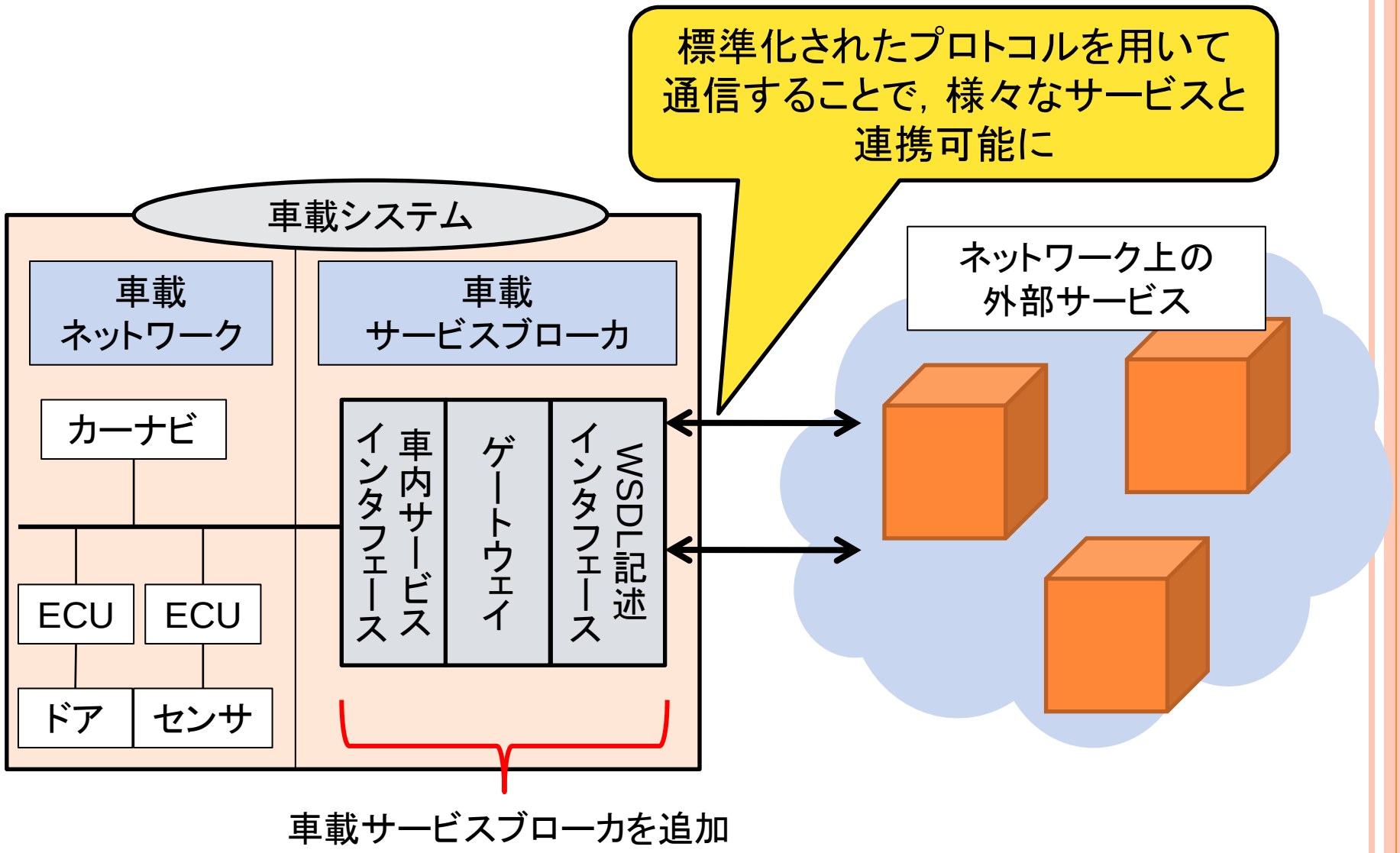
- 問題点

  サードパーティサービスや異なるメーカとのサービス連携が困難
  →解決方法のひとつとして，SOAPやRESTなどを用いて標準化されたメッセージプロトコルの適用が考えられる

  ↓

  メッセージ変換を行うシステムを追加する必要がある
  **⇒サービスブローカの実現によって解決**

# 1.1 車外サービス連携の問題点(2/2)



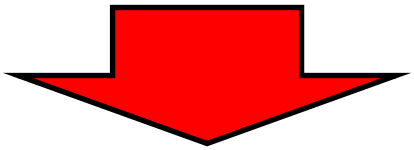
# 1.2 OSGiブローカを用いたモデル(1/2)



## OSGi

OS上のJavaVMのプロセス上で実行され，単一のJavaVM上で複数のBundleを実行可能

| | |
|---|---|
| 特徴 | ・組込みシステムが前提の実装の為，省メモリ<br>・機能の修正・追加が容易(ネットワークから取得)<br>・複数のBundleを相互連携可能 |



OSGiをサービスブローカに用いる

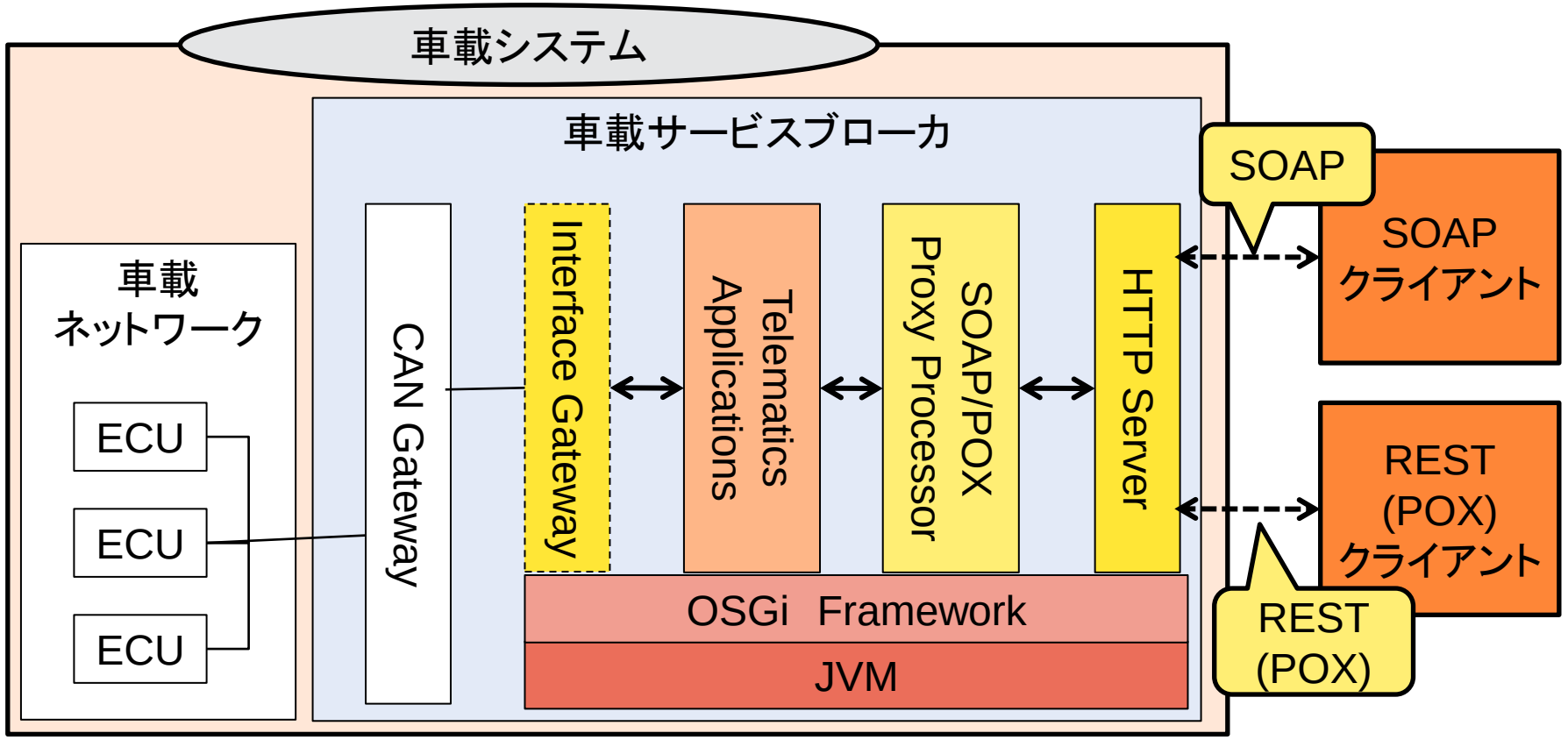
- 新サービスの追加の際，連携に必要となるBundleのみをネットワークからダウンロード，インストールを行うだけでよい
- Webとの通信を行うBundleが提供されている為，外部連携を容易に行える

# 1.2 OSGiブローカを用いたモデル(2/2)

資料から提案された車載サービスブローカアーキテクチャ

↔ ・・・Bundle間連携

Interface　Gateway は今回の実装では用いていない



# 2. OSGi – 実装・プロトタイプモデルの構築(1/9) –



- 実装するサービス

  前述モデルに従い，車外から車内のドアロック状態を取得するサービス.

  → 車外クライアントから車内ECUに接続，状態を取得し表示.

- 実装手順

  1.前述モデルを構成する四つのプログラムを実装.
  (ECU，CANGW，ServiceBundle，Client)

  2.実装したプログラムを異なる四つのPC上で実行，Clientから接続を行い，
  挙動を確認する.

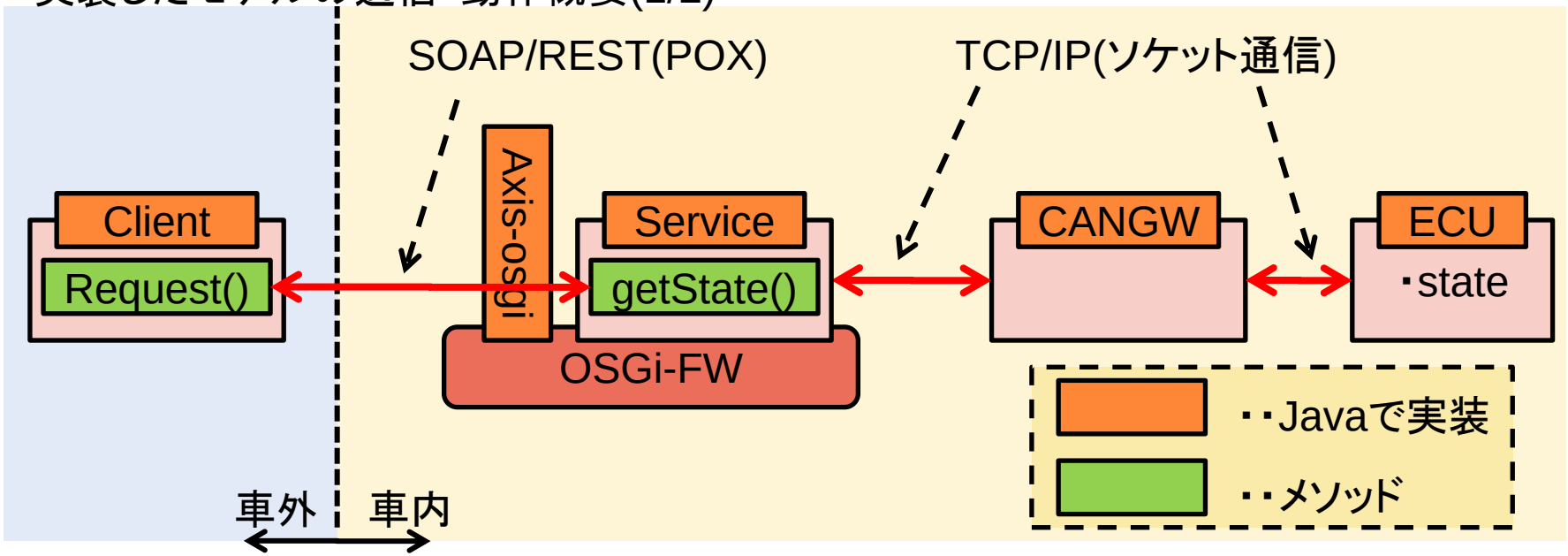
- 備考

  ・外部と内部の通信にはSOAP/REST(POX)の二種類を実装し，実行する.

  ・内部の通信にはTCP/IPを用いる.

# 2. OSGi – 実装・プロトタイプモデルの構築(2/9) –

実装したモデルの通信・動作概要(1/2)

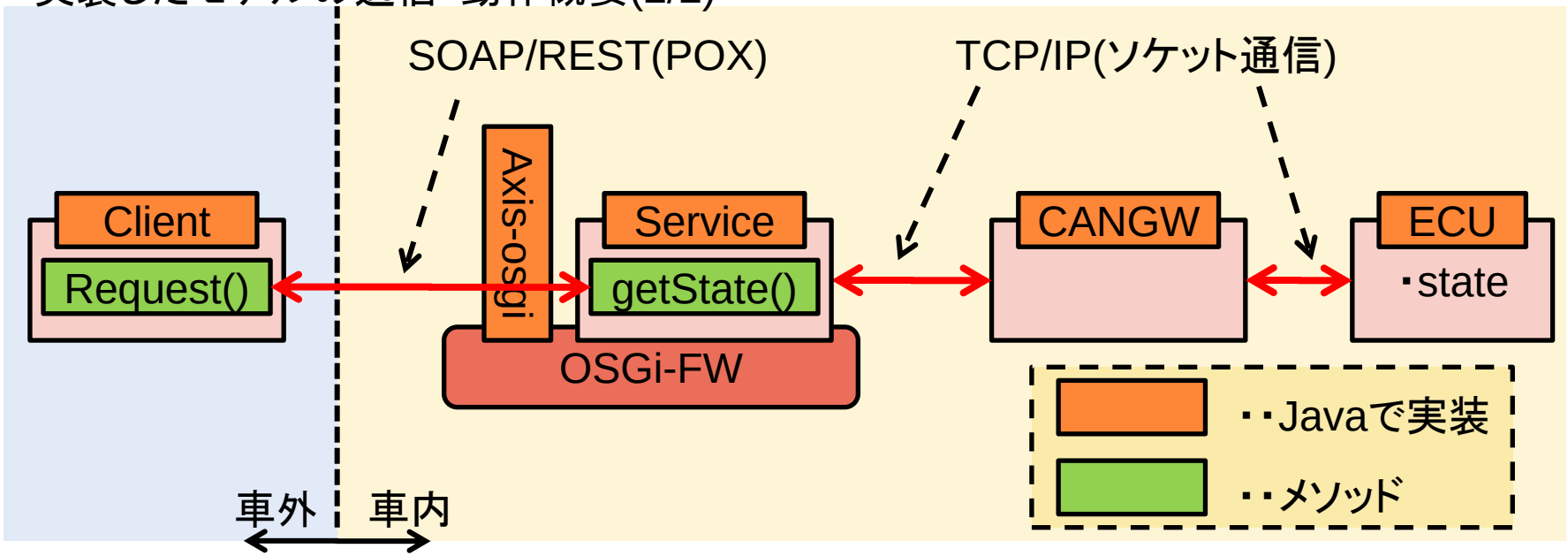



| 装置 | 実装 | 備考 |
|---|---|---|
| ECU | Java | 代替品 |
| CANGW | Java | 代替品 |
| Service | Bundle(Java) | Axis-osgiを用いてサービスを外部に公開 |
| Client | Java | SOAP/REST(POX)の二種類を作成 |

# 2. OSGi – 実装・プロトタイプモデルの構築(3/9) –



実装したモデルの通信・動作概要(2/2)




| 装置 | 動作概要 | 使用PC |
|---|---|---|
| ECU | ドアロック状態(state)を保持し，通信が発生した場合stateを返す | PC1 |
| CANGW | Serviceからの接続に対応してECUに接続，取得したstateを返す | PC2 |
| Service | Axis-osgiを用いてgetState()を外部サービスとして公開する<br>(getState()：CANGWに接続し，ECUのstateを取得し返す) | PC3 |
| Client | 外部サービスとして効果されているgetState()を利用し，ECUの状態を取得する | PC4 |

# 2. OSGi – 実装・プロトタイプモデルの構築(4/9) –

## 1.1 ECU，CANGWの実装

通常のJavaプロジェクトとしてECU，CANGWを作成し，実装する．
内部通信にはTCP/IPを用いる為，あらかじめ接続先PCのIPとポートを確認しておく．

ECUのソースコード(一部)

```
public class OSGi_01_ECU {

    static Socket soc = null;
    static ServerSocket svsoc = null;
    public static final int PORT = 8190;

    public static String state = "lock";
```

接続されるCANGWのポート

ドアロック状態

CANGWのソースコード(一部)

```
public class OSGi_01_CANGW {

    static Socket socE = null;
    static Socket socB = null;
    static ServerSocket svsocB = null;

    public static final int PORTE = 8190;
    public static final int PORTB = 8191;
    public static final String addE = "10.63.6.216";
```

接続するECUのポート

接続されるBundleServiceのポート

接続するECUを動作させるPCのIP

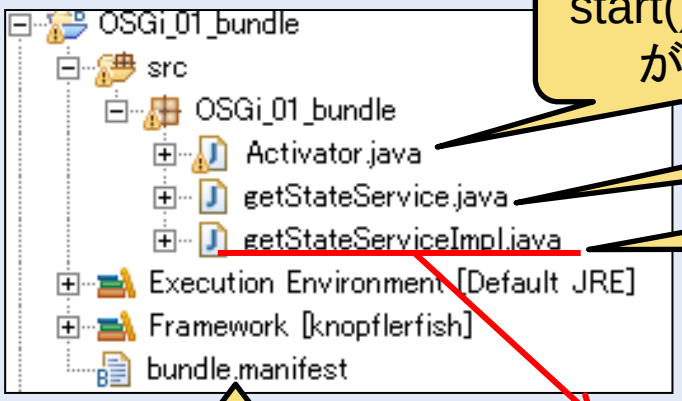
# 2. OSGi – 実装・プロトタイプモデルの構築(5/9) –

## 1.2 BundleServiceの実装

BundleServiceをBundleプロジェクトとして作成する.

ECU，CANGWと同様，接続先のIP/ポートを確認しておく.

プロジェクトの構造




start(), stop()が記述

公開するサービスのインタフェース

公開するサービスの実装

マニフェストファイル

公開サービスのソースコード(一部)

```
public class getStateServiceImpl implements getStateService{

    public String getState(){

        Socket soc = null;
        final int PORT = 8191;
        final String add = "10.63.6.13";
        String resStr = "";
```

接続するCANGWのポート

接続するCANGWのIP

# 2. OSGi – 実装・プロトタイプモデルの構築(6/9) –

## 1.3 Client(SOAP/REST)の実装

SOAP又はREST(POX)でリクエストを行うClientを作成する.

REST(POX)クライアントのソースコード(一部)

```
public class ClientPOX {

    public static void main(String[] args) throws MalformedURLException,
            IOException, ParserConfigurationException, SAXException {

        // URL url = new URL(
        // "http://localhost:8080/axis/services/test02?method=TestRequest");
        URL url = new URL(
                "http://10.63.6.10:8080/axis/services/getStateMethod?method=getState");
        HttpURLConnection urlcon = (HttpURLConnection) url.openConnection();
        urlcon.setRequestMethod("GET");
        urlcon.setInstanceFollowRedirects(false);
        urlcon.setRequestProperty("Accept-Language", "ja;q=0.7,en;q=0.3");
```

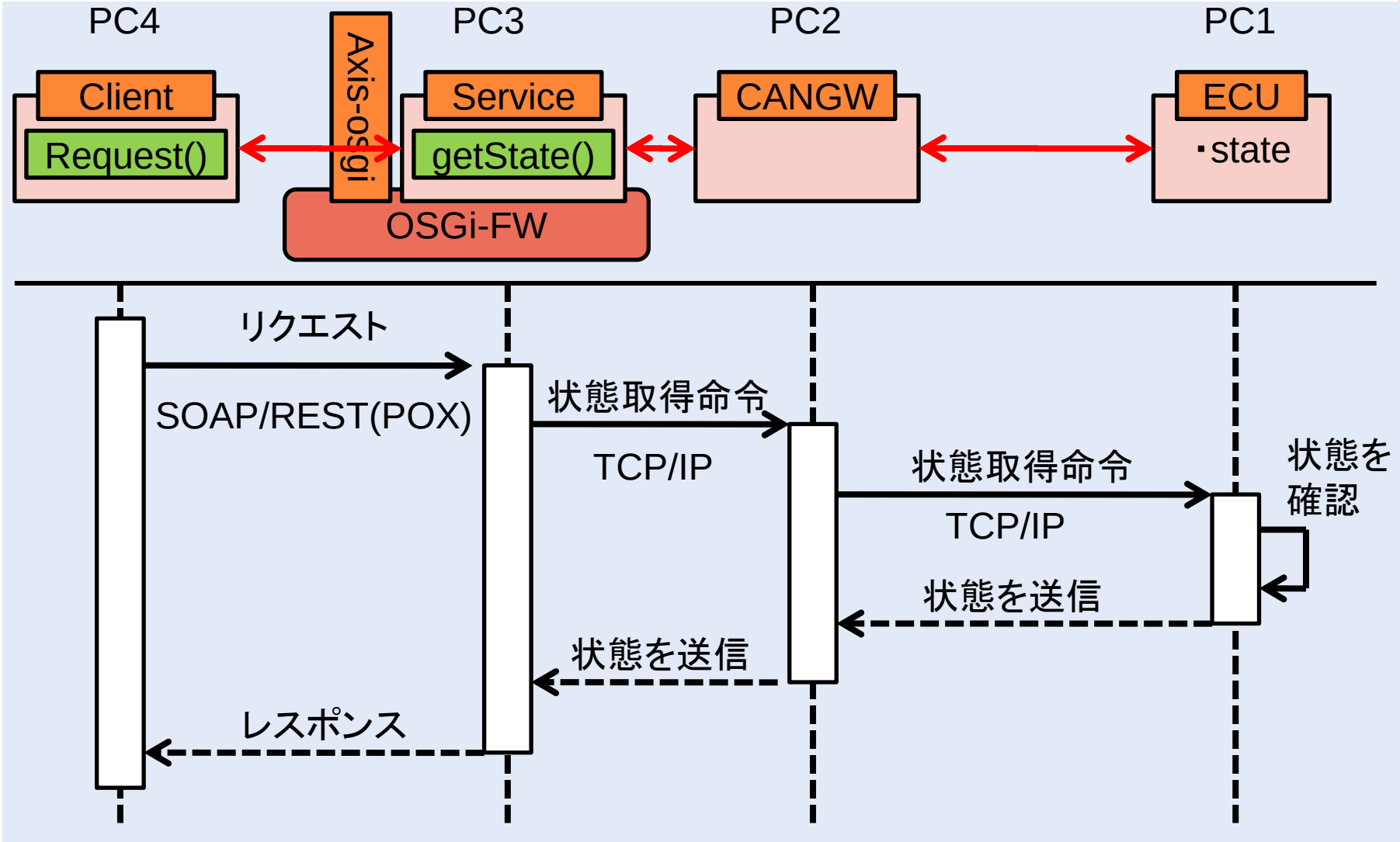
SOAPクライアントのソースコード(一部)

```
public class ClientSOAP {

    public static void main(String[] args) throws Exception{

//      String endpoint = "http://localhost:8080/axis/services/test02";
        String endpoint = "http://10.63.6.10:8080/axis/services/getStateMethod";

        Service service = new Service();
        Call call = (Call)service.createCall();
        call.setTargetEndpointAddress(new java.net.URL(endpoint));
        call.setOperationName(new QName("http://10.63.6.10:8080", "getState"));
        call.setReturnType(XMLType.XSD_STRING);
```

# 2. OSGi – 実装・プロトタイプモデルの構築(7/9) –

## 2.1 実装したプロジェクトをエクスポート

ECU，CANGW，OSGi-FWを実行状態にしておく.



# 2. OSGi – 実装・プロトタイプモデルの構築(8/9) –

## 2.2 Clientからプログラムを実行し，動作を確認

REST(POX)Clientで返信されたレスポンス(XML)

```
ClientPOX Request START.
ClientPOX Request SUCCESS
ResponseMessage :
レスポンスヘッダ
 null: [HTTP/1.1 200 OK]
レスポンスコード[200]レスポンスメッセージ[OK]

---ボディー---
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http
 <soapenv:Body>
  <getStateResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <getStateReturn xsi:type="xsd:string">lock</getStateReturn>
  </getStateResponse>
 </soapenv:Body>
</soapenv:Envelope>
ResponseTime    : 31ms
```

SOAPClientで返信されたレスポンス(レスポンスメッセージのみ)

```
ClientSOAP Request START.
ClientSOAP Request SUCCESS.
ResponseMessage : lock
ResponseTime    : 253ms
```

同じレスポンス

# 2. OSGi – 実装・プロトタイプモデルの構築(9/9) –



- 実装の目的

  提案されたアーキテクチャ，及び実装内容(プログラム)の理解

- 実装の結果，考察
  - 目的(アーキテクチャの概要，プログラム内容)を理解することができた
  - 試行回数が多くはないので確定的ではないが，SOAPとREST(POX)の応答時間の比較を行い，実際にREST(POX)の方が早いことが確認できた

- 実装の課題
  - 他機能の実装
  - 正確な応答時間の測定
  - 課題とされていた点の解決方法の模索

# 3. 課題



- 前回調べたNGTPは主に車外側中心で，今回のOSGiを用いた技術は車内側中心である.

  今後，どちらを軸に研究を進めていくのか，また，

  それらの技術を応用できないかを考えたい.

- NGTP，OSGiについて，資料を読み理解する段階を終え，疑問点や問題点を発見，追求していきたい.

# 4. 参考文献


- IT用語辞典 e-Words(http://e-words.jp/)
- 青山幹雄　ほか：車載ソフトウェアのサービスプラットフォームのモデルアーキテクチャ　自動車技術会　学術講演会前刷集
- 濱千代正弥、片桐雅仁：自動車ネットワークサービスの連携アーキテクチャ