

コンシューマ・デバイス論文

ホームゲートウェイ向け機器管理制御フレームワークの開発

宮田 克也^{1,a)} 寺岡 秀敏^{1,b)} 関原 拓也^{2,c)} 芳野 明彦^{2,d)}

受付日 2012年12月14日, 採録日 2013年4月26日

概要: 各家庭には様々な機器があり, それらをホームゲートウェイ上で管理・制御する多様なアプリケーションの登場が期待される. しかし, 各機器のネットワークへの接続方法, 通信プロトコルが様々である等, 制御方法が統一されていないことが課題である. 本稿では, ホームゲートウェイ向けに, 多様な機器を統一的に管理・制御するための機器管理制御フレームワークを提案する. 本フレームワークは, OSGi上に各アプリケーションが共通で必要とする機能を取り込み, 機器制御用の共通 I/F, 使いやすい機器選択機能, 登録すべき機器情報項目を取得するための共通 I/F を備えることで, アプリケーション開発量を削減できる. また, 本フレームワークを利用して, 試作 HEMS 環境を構築し, 本フレームワークの有用性をアプリケーションの開発生産性の観点で評価した.

キーワード: OSGi, ネットワーク家電, ホームネットワークシステム, マルチデバイスシステム, HEMS

Development of HGW Software Framework for Managing and Controlling Multi-devices

KATSUYA MIYATA^{1,a)} HIDETOSHI TERAOKA^{1,b)} TAKUYA SEKIHARA^{2,c)} AKIHIKO YOSHINO^{2,d)}

Received: December 14, 2012, Accepted: April 26, 2013

Abstract: Various applications are expected to be proposed for a home network system (HNS). One of main subjects in developing applications which work on a home-gateway (HGW) in the HNS is that the applications must be available in various home environments, where a lot of types of devices are connected to the HGW and various connection methods and connection protocols are used. This paper proposes a method that constructs an OSGi-based software framework for managing and controlling multi-devices. Since the framework which works on the HGW includes common functions for applications, and provides common APIs for controlling devices, device selection features that is easy to use, and common APIs for acquiring the list of device properties be registered, then it can reduce development cost for the applications. We have implemented the proposed method, and deployed the experimental HEMS. We have also evaluated the effectiveness of the framework.

Keywords: OSGi, networked appliances, home network system, multi-device system, HEMS

1. はじめに

家電やセンサ等の宅内機器とネットワークを連携することで, 機器単体では実現できない高付加価値サービスを

ユーザに提供するホームネットワークシステム (HNS) が注目を集めている. HNS では, 宅内に配置されたホームゲートウェイ (HGW) が, 宅内機器を制御したり, 外部サーバと情報交換したりする. 宅内消費電力の見える化や省電力制御を行うホーム・エネルギー・マネジメント・システム (HEMS), 快適生活を支援するホームオートメーション, 防犯やヘルスケアとしての見守りシステムといった多様な応用が考えられる.

しかしながら, HNS 向けアプリケーション開発の最大の課題は, 制御対象機器の制御方法が統一されていないことである. 伝送メディアや通信プロトコルは機器の種類に

¹ 株式会社日立製作所横浜研究所
Yokohama Research Laboratory, Hitachi Ltd., Yokohama, Kanagawa 244-0817, Japan

² 株式会社日立ソリューションズ
Hitachi Solutions, Ltd., Yokohama, Kanagawa 244-0801, Japan

a) katsuya.miyata.yn@hitachi.com

b) hidetoshi.teraoka.rf@hitachi.com

c) takuya.sekihara.ew@hitachi-solutions.com

d) akihiko.yoshino.ev@hitachi-solutions.com

よっても異なり、同種の機器であってもベンダごとに異なる場合がある。たとえば、伝送メディアとしては、有線/無線 LAN, 802.15.4, Bluetooth 等があり、通信プロトコルとしては、標準的なものだけでも ECHONET Lite, UPnP 等がある。また、家庭ごとに使用する機器が異なり、長期間で見た場合には使用機器が追加/変更されることもある。そのため、統一的な方法で機器制御でき、機器構成の変化にも柔軟に対応できるシステムが望まれている。

一方で、様々なデバイスやサービスに対応するためのホーム ICT 実行基盤として OSGi [1] が注目されている。OSGi は Java ベースのソフトウェアモジュール (バンドル) の動的更新を実現する基盤システムであり、ソフトウェアの部品化を効率的に実現することができる。家庭ごとに使用する機器や、利用するサービスが異なる場合でも、必要なバンドルの組合せで適切なシステムを構築することが可能となる。

そこで本研究では、HNS において、アプリケーションの開発効率の向上を図るため、多様な機器を統一的に管理・制御するための機器管理制御フレームワークを OSGi 上で開発し、その評価を行った。

2. 先行研究

HNS の一般的な構成を図 1 に示す [2]。宅内機器を最終的に制御するのは宅内の HGW である。HGW 内の制御ロジックのみで宅内機器を制御する場合と、外部サーバや外部機器との情報交換に基づいて制御する場合とがある。

HNS において、機器種別、ネットワーク接続形態、通信プロトコルが異なる機器を統一的な方法で制御するという観点で、これまで様々な研究が行われてきた。たとえば、HGW の中に多様なプロトコルを 1 つの共通プロトコルに変換する機能を持たせることで、HGW 外部の機器から仮想的な 1 つのネットワークとして扱えるようにする研究 [3], [4], [5] がある。また、外部アプリケーションによ

るマッシュアップを促すために、HGW 向け機器制御用統合 API の研究 [6] がある。しかし、いずれの従来研究も、HGW の外部アプリケーションの開発を簡易化するためのものであり、HGW の内部アプリケーションの開発を簡易化する方法については十分な検討がなされていない。

HGW の内部アプリケーションの開発簡易化の点では、HGW 内部のアプリケーションと宅内機器制御モジュール間の I/F を共通化する研究 [7] が行われている。しかし、従来研究では、多数の宅内機器の中から制御対象機器を柔軟に選択する方法や、機器種別、ネットワーク接続形態、通信プロトコルが異なる機器の機器情報登録を簡易化する方法について十分な検討がなされていない。

HEMS における電力監視のように常時動作するアプリケーションは、HGW 内部で動作させることが望ましい。よって、本研究では HGW の内部アプリケーションの開発簡易化を目標とする。OSGi 搭載 HGW において、アプリケーションと宅内機器制御モジュール間 I/F の共通化のみにとどまらず、制御機器選択や機器情報登録の観点でのアプリケーションの開発効率向上に寄与する機器管理制御フレームワークを提案する (図 1)。

3. 機器管理制御フレームワークの検討

3.1 要件定義

本稿では、機器管理制御フレームワークの要件として、以下の 3 つを定義した。

要件 R1: 機器に対して制御を行う場合、接続形態、通信プロトコルによらず、同じ API で制御できること。

要件 R2: 任意の制御対象機器を柔軟に選択できること。

要件 R3: 異なる種別の機器を含む複数の機器情報を統一的に管理できること。

すでに述べたとおり、各家庭で利用機器は異なり、機器によって通信プロトコルが異なる。また、機器の接続形態も複雑である。たとえば、2 つ以上のネットワークがプロトコル変換アダプタを介して接続される場合もあれば、複数の機器を中継器でいったん集約して HGW と接続される場合もある。このような状況であっても、たとえば、「宅内の消費電力が所定値を超えた場合に、全エアコンを停止する」というアプリケーションは、どの家庭のどのエアコンに対しても同様に動作する必要がある。要件 R1 を満たすことで、これを実現できる。

また、アプリケーションによって、制御対象機器は様々である。たとえば、節電目的で機器の電源をオフする場合であっても、機器単体制御、「全エアコン」のような機種指定制御、「全リビング機器」のような設置場所指定制御、「宅内の全機器」のような全機器制御等が考えられる。このように柔軟に制御対象機器を選択できることが望ましい。要件 R2 を満たすことで、これを実現できる。

機器の種別/接続形態/通信プロトコルの多様性のため、

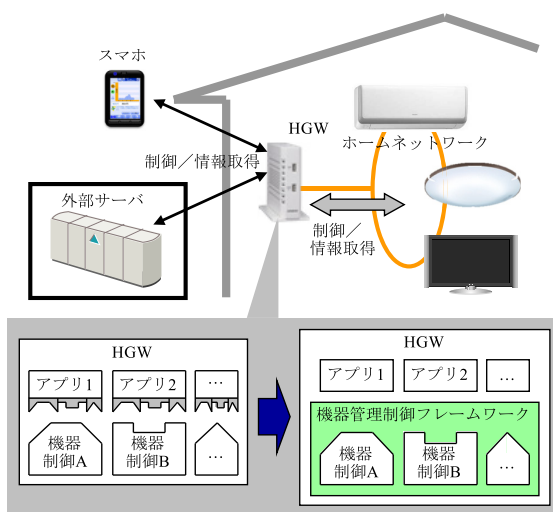


図 1 HNS 構成図

Fig. 1 General architecture of HNS.

1つの機器が保持すべき情報の多様化は避けられない。たとえば、使用するアドレスは通信プロトコルによって異なるし、中継機を介して接続されていれば中継機の動作パラメータ情報が必要となる。そのため、少なくとも一度は、各機器情報を機器管理制御フレームワークに対して入力する必要がある。要件 R1 と同様の観点で、機器登録処理においても、機器の種別/接続形態/通信プロトコルが異なっても、統一的な方法で行えることが望ましい。要件 R3 を満たすことで、これを実現できる。

3.2 基本方針

OSGi の標準サービスの 1 つに Device Access サービス [8] がある。これは、たとえば、機器が動的に追加された場合に、その機器のデバイスドライバを動的にネットワークからダウンロードするといった機能の実現を支援する。今回の機器管理制御フレームワーク開発においては、多種多様な機器と、実際に制御コマンド等を扱う制御モジュールとの対応付けが必要である。よって、Device Access サービスを利用するのが効率的であると考えた。

Device Access サービスは、物理的な機器を表すデバイスクラスと、最適デバイスドライバの検索機能の一部を担うドライバクラスとの対応付けを行う。基本的な動作の流れは次のとおりである。デバイスクラスのオブジェクトがサービス登録 [9] されたことを検出した場合、その時点で登録されているドライバサービス（サービス登録されたドライバクラスのオブジェクト、クラスとサービスの関係については以下でも同様）について適合判定処理を行う。これにより最適なドライバサービスが確定した場合、そのドライバサービスのドライバ確定処理を呼ぶ。

ドライバ確定処理は利用者が実装する必要がある。今回、ドライバ確定処理の処理内容を 3.4 節で検討した。また、3.5 節でデバイスクラスを利用する実現方法を検討した。

3.3 制御インタフェースの抽象化の検討

(1) 機器制御 API の基本クラス構成

要件 R1 に対し、機器制御 API を機器制御 I/F クラス (Java の interface class) と、機器制御実装クラス (Java の implement class) とに切り分けた (図 2)。具体例として、エアコン制御の場合、機器制御 I/F クラスでは電源 ON/OFF、動作モード変更、設定温度変更という抽象的な I/F を規定した。これにより、接続形態や通信プロトコルに非依存となる。一方、機器制御実装クラスは、ECHONET Lite [10] 対応エアコン用、メーカー独自 I/F 対応エアコン用といった単位で別クラスとした。各クラスで通信プロトコル等の違いを考慮し、適切な制御コマンドを生成、送信する処理を実現する。

(2) 多段接続機器の機器制御実装クラスの実装方法

中継器を介して多段接続された機器について、機器制御

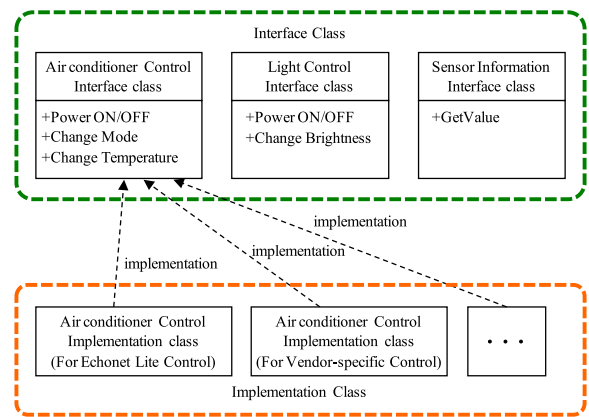


図 2 機器制御用 I/F クラスと実装クラスの関係図

Fig. 2 Interface and implementation class for device control.

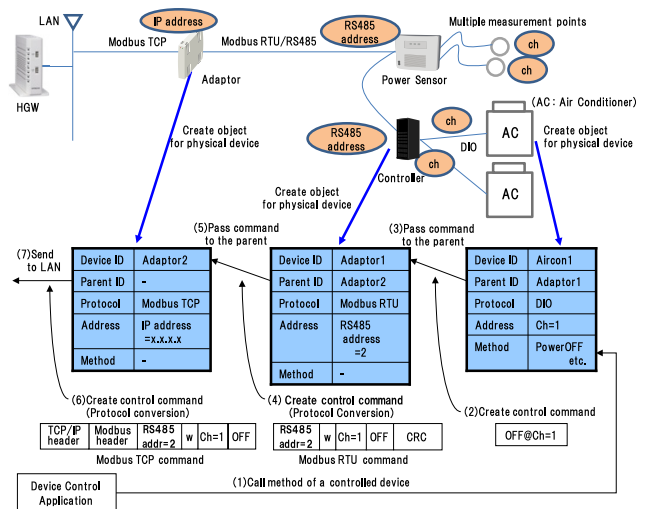


図 3 多段接続機器への制御コマンド生成方法 (案 1)

Fig. 3 Control command creation method for hierarchically connected device (1).

実装クラスの実装方法を検討した。

(方法 1) 親機/子機を個別に表すクラスを設け、それらを組み合わせて 1 つの機器制御実装クラスとして表す方法 (図 3)

個々の中継機に対応するクラスを生成する方法である。各中継機の実装クラスは、自分が受信できるプロトコル、アドレス、親機 ID 等の情報を含む。

HGW 上のアプリケーションは、末端の制御対象機器の制御メソッドを呼ぶ (処理 1)。そのメソッド実体である機器制御実装クラスの中で、プロトコル、アドレス、制御内容に応じたコマンド (指示) を生成する (処理 2)。そのコマンドを親機の機器制御実装クラスに渡す (処理 3)。制御対象機器の親機にあたる中継機クラスは、自分のプロトコル、アドレス、子機から渡されたコマンドに応じて、プロトコル変換を行い、新たなコマンドを生成する (処理 4)。これを繰り返すことで (処理 5, 6), HGW が最終的に送信すべきコマンドを生成する (処理 7)。

以上のように、複数の中継機クラスを連動させることで、

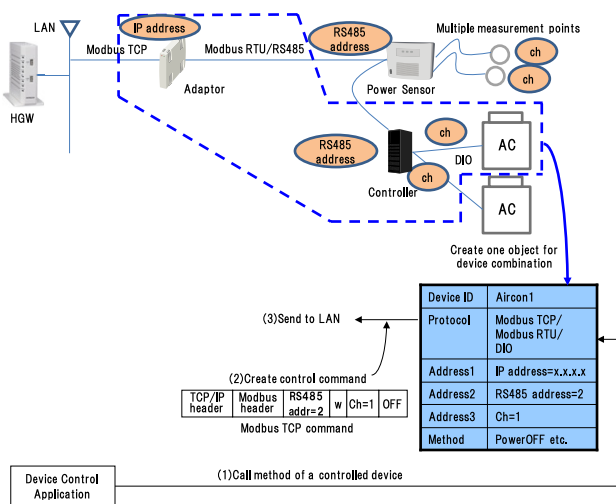


図 4 多段接続機器への制御コマンド生成方法 (案 2)

Fig. 4 Control command creation method for hierarchically connected device (2).

1つの機器制御実装クラスとしての役割をなす。

本方法の長所は、各中継機クラスを一度用意すれば、その組合せを変えても使用できるという拡張性が高い点である。逆に、短所は実装が複雑な点である。

(方法 2) 親機/子機を含めて 1つの機器制御実装クラスとして表す方法 (図 4)

HGWと末端の制御対象機器との間のつながりを、中継器も含めて一括りにして、1つの機器制御実装クラスとして実装する方法である。そのために、本機器制御実装クラスには、途中で経由するプロトコル、階層構造の中の各アドレス等の、最終的な制御コマンドを生成するために必要な情報をすべて含める。

HGW上のアプリケーションは、末端の制御対象機器のメソッドを呼ぶ (処理 1)。そのメソッド実体である機器制御実装クラスは、プロトコルやアドレスの階層関係を考慮して、HGWが最終的に送信すべきコマンドを一度に生成する (処理 2)。HGWはそのコマンドを送信する (処理 3)。

本方法の長所は実装が比較的容易であること、短所は、親機/子機の組合せが 1つでも変わった場合に新たな機器制御実装クラスの作成が必要な点である。

上記 2つの方法を比較した場合、当面は親機/子機の組合せを自由に変えて使用することが少ない点を考慮し、方法 2を採用することとした。なお、将来的に親機/子機の組合せパターンが増加し、方法 1の方が有利になったときには、上記機器制御実装クラスの部分だけの修正で済むと考える。

(3) 機器制御 I/F クラスの階層化

機器制御 I/F クラスを定義するにあたり、どの単位で機器制御 I/F を共通化すべきかを検討した。

基本的には機器の種類ごと (エアコン、照明、テレビ、冷蔵庫等) に制御できる内容が異なる。よって、その単位で

機器制御 I/F クラスを用意した。しかし、電源 ON/OFF 制御はほぼすべての機器で実行可能と考えられる。よって、電源 ON/OFF 制御を行うメソッドを含む基本機器制御 I/F クラスを定義し、それを継承して (Java の extend)、各機器用の制御 I/F クラスを定義した。

同じ種類の機器については、たとえば、普及機と高級機とで機能に違いがあり、両者の機器制御 I/F クラスは異なる。しかし、基本的には高級機は普及機の機能を含むと考えられる。よって、普及機用の機器制御 I/F クラスを継承して高級機用の機器制御 I/F クラスを定義した。これにより、高級機で追加された機能についての機器制御 I/F だけを記述すればよい。

3.4 制御対象機器の選択方法の検討

要件 R2 に対して、3.3 節で検討した機器制御 I/F クラスを OSGi サービスとして登録することによって、機器制御 I/F サービスとして利用できるようにした。

具体的な処理の流れは以下のとおりである。ある機器が機器管理制御フレームワークに登録された場合 (機器登録処理は 3.5 節参照)、3.2 節で述べたように Device Access サービスが、その機器を表すデバイスサービスに最適なドライバサービスを検索し、ドライバ確定処理を行う。この処理の中で、その機器の制御機能を表す機器制御実装クラスのオブジェクトを生成する。それを、OSGi が提供するサービス登録機能を利用し、機器制御 I/F サービスとして登録することとした。その際、そのオブジェクトがサポートするすべての機器制御 I/F クラスをサービス登録することとした (図 5)。たとえば、温度センサ付き高級エアコン用の機器制御実装クラスは、基本機器制御 I/F クラス、基本エアコン制御 I/F クラス、高級エアコン制御 I/F クラス、センサ情報 I/F クラスのすべての実装クラスであるといえる。よって、このオブジェクトが生成された場合、上記すべての機器制御 I/F クラスをサービス登録する。

これにより、ある機器制御 I/F サービスを取得したいアプリケーションは、OSGi が提供するサービス取得機能 [9] を利用する際に、所望の機器制御 I/F サービス名を渡すだけで済む。

このとき、指定した機器制御 I/F サービスとして複数機器分が登録されていれば、それらをすべて取得できる。よって、たとえば、基本エアコン制御 I/F サービスを取得することで、全エアコンの設定温度をいっせいに 1 度下げるといった制御ができる。また、基本機器制御 I/F サービスを取得することで、全機器 (エアコン、照明、テレビ、冷蔵庫等) をいっせいに電源 OFF するという制御もできる。

また、より柔軟に制御対象機器の選択を行うために、機器制御 I/F クラスをサービス登録する際に、デバイスサービス情報をサービスプロパティとして登録するようにした。デバイスサービス情報は、Device Access サービスの

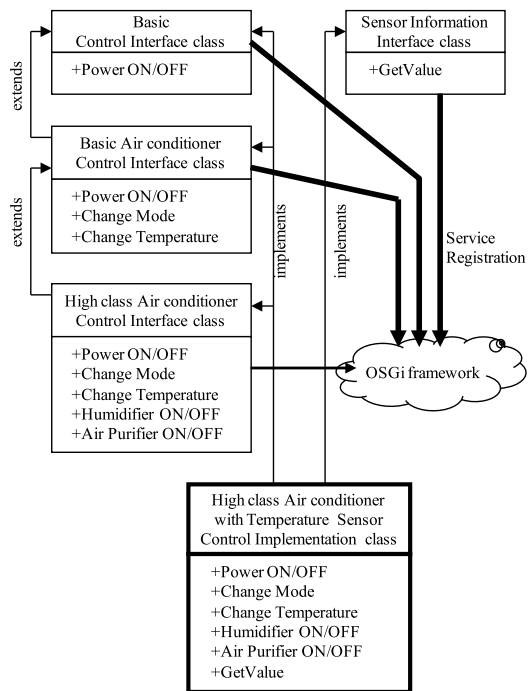


図 5 機器制御 I/F クラスの階層化

Fig. 5 Hierarchical design of control interface class.

ドライバ確定処理でドライバサービスに渡されるので、これを利用する。OSGi のサービス取得機能には強力な検索機能が備わっており、複数の条件を含む検索式で表せる。上記サービスプロパティに機器 ID、機器名称、設置場所、プロトコル種別等を含めることで、サービス取得時には、それらを使った検索式を指定するだけで簡単に所望の機器制御 I/F サービスを取得できる。たとえば、設定場所を指定した機器制御や、機器 ID を指定した機器制御を簡単に行える。

3.5 機器情報登録方法の検討

要件 R3 に対して、機器 ID、機器名称、設置場所、プロトコル種別といった機器情報を表し、各情報へのアクセス API 機能を持つ機器情報クラスと、機器情報クラスの登録/削除/一覧管理を行う機器構成管理クラスを定義した。

機器情報には、機器 ID や機器名称といった制御対象機器自体に属する情報がある。一方、HGW と当該機器との接続方法や機器制御実装クラスの実装方法に依存する情報がある。たとえば、IP アドレスや RS485 局番等のアドレス情報、使用する通信プロトコルによっては書き込みデータサイズ、中継器を介して接続される場合は中継器の動作パラメータである。これらは、機器制御実装クラスで、機器制御コマンド生成処理の中で利用されるため、機器制御実装クラスに属する情報といえる。

そこで、制御対象機器に対応する機器制御実装クラスが一意に決まった場合に、その機器が保持すべき機器情報の項目一覧（これを「ドライバ情報」と呼ぶ）を管理するクラ

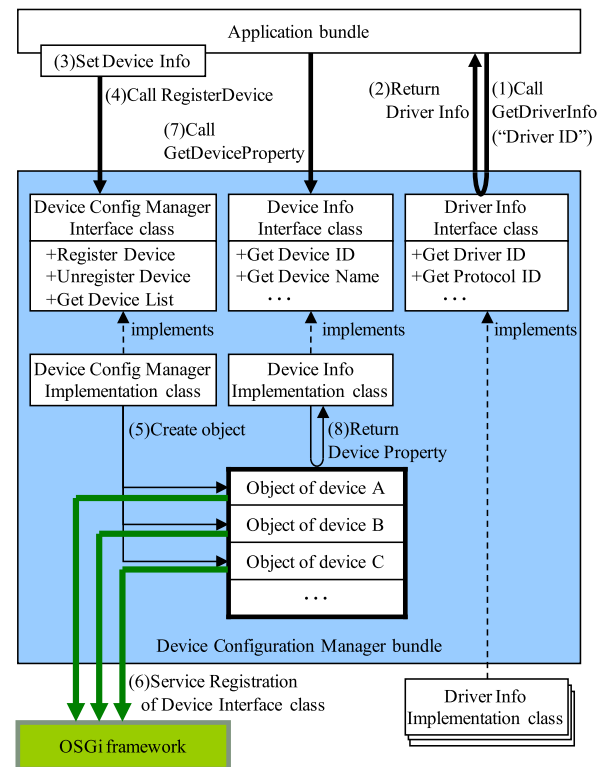


図 6 機器構成管理バンドルの構造

Fig. 6 Structure of device configuration manager bundle.

スを設けた。本ドライバ情報クラスを、ドライバ情報 I/F クラスとドライバ情報実装クラスとに切り分けた。これにより、ドライバ情報にアクセスする API を共通化し、かつ、機器制御実装クラスごとに異なる情報を扱うことができる。

以上の、機器情報クラス、機器構成管理クラス、ドライバ情報 I/F クラスをまとめて機器構成管理バンドルとした。ドライバ情報実装クラスは、前述の機器制御実装クラスとドライバクラスとまとめてドライババンドルとした。動作概要を以下に示す（図 6）。

- (i) 機器登録を行うアプリケーションは、ドライバ情報 I/F クラス中のドライバ情報取得 API を呼び、ドライバ情報を取得する（処理 1, 2）。このとき、登録したい機器に対応するドライババンドルを特定する情報を引数で指定する。
- (ii) アプリケーションは、取得したドライバ情報に対して、必要なパラメータを設定し（処理 3）、機器構成管理クラス中の機器登録 API を呼ぶ（処理 4）。
- (iii) 機器登録 API の処理の中で、機器情報クラスのオブジェクトを生成する（処理 5）。機器情報クラスは、Device Access サービスが認識できるように、デバイスクラスの実装クラスとしても実装し、これを OSGi のサービス取得機能を利用してデバイスサービスとして登録する（処理 6）。
- (iv) 機器構成管理バンドルに登録された機器情報を取得したいアプリケーションは、OSGi のサービス取得機能を用

利用して機器情報サービスを取得する。機器情報サービスが提供する機器情報取得 API を呼ぶ (処理 7)。機器情報取得 API の処理の中で、指定された機器に関する情報を返す (処理 8)。

以上により、対応する機器制御実装クラスによって異なる機器情報を登録する必要があっても、アプリケーションは処理 1、2 で取得したドライバ情報に値を設定すればよい。よって、機器情報登録において、すべての機器を统一的に取り扱うことができる。

4. 試作システムの構築と評価

4.1 試作システム構成

今回設計した機器管理制御フレームワークの効果を確認するため、図 7 に示す試作 HEMS 環境を構築した。機器管理制御フレームワークを動作させる日立製 HGW の主要緒元を表 1 に、各制御対象機器の HGW との接続形態/通信プロトコルを表 2 に示す。HNS においては、一般的に

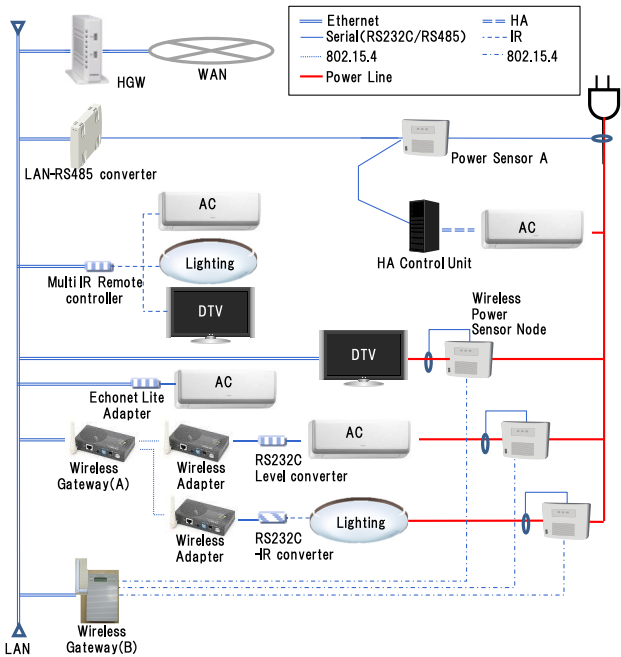


図 7 試作システム構成
Fig. 7 Structure of experimental HEMS.

表 1 評価機仕様

Table 1 Specification of HGW.

Item		Description
CPU		650MHz
Memory		RAM:512MB, ROM:128MB
LAN/WAN interface	Port	RJ-45 4port(LAN), 1port(WAN)
	Supported Standard	10BASE-T/100BASE-TX /1000BASE-T
USB interface		2port
OS		Linux
Application Platform		JavaME(SuperJ Engine)/ OSGi Framework (SuperJ Engine Framework)

有線/無線 LAN, 802.15.4, IR, RS485, HA 端子を利用した機器制御が想定される。今回の評価環境は、これらの要素を含むよう構築したものである。また、エアコン、照明、DTV 以外の機器についても、大部分の機器は今回の接続形態のいずれかを適用できると考える。

今回の試作システムで作成した機器管理制御フレームワークの全体バンドル構成を図 8 に示す。今回の制御対象機器 3 種 (エアコン、照明、DTV) 用と、電力センサ用の I/F バンドルを実装した。このバンドルには、各機器制御 I/F クラスが含まれる。機器ドライババンドルとしては機

表 2 試作システムにおける制御対象機器一覧

Table 2 List of target devices in experimental HEMS.

Device	Description
エアコン 1	・ HA で制御 (電源 ON/OFF のみ) ・ Modbus 中継機経由で接続
エアコン 2	・ 独自プロトコルで制御 ・ 無線中継器 A 経由で接続
エアコン 3	・ ECHONET Lite で制御 ・ 試作 ECHONET Lite アダプタ経由で接続
エアコン 4	・ マルチ IR リモコンで制御 ・ マルチ IR リモコンは LAN 及び独自プロトコルで HGW と接続
照明 1	・ IR で制御 ・ 無線中継器 A とシリアル-IR 変換機経由で接続
照明 2	・ マルチ IR リモコンで制御 ・ マルチ IR リモコンは LAN 及び独自プロトコルで HGW と接続
DTV1	・ 独自プロトコルで制御 ・ LAN 接続
DTV2	・ マルチ IR リモコンで制御 ・ マルチ IR リモコンは LAN 及び独自プロトコルで HGW と接続
電力量計 1	・ Modbus 中継機経由で接続
電力量計 2-4	・ 無線中継器 B 経由で接続

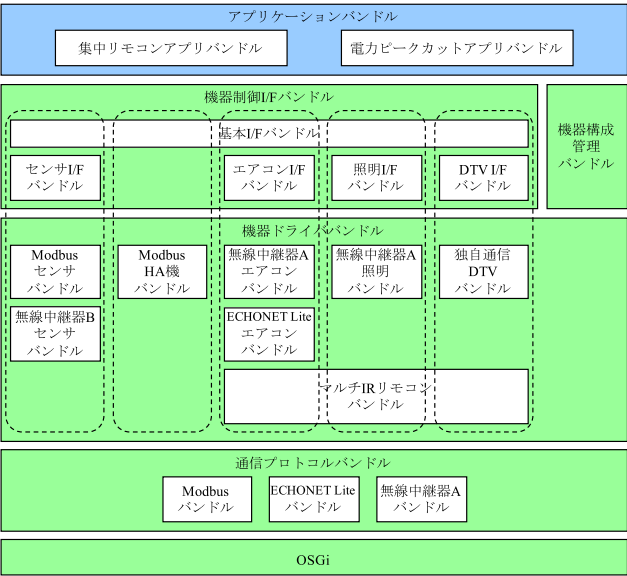


図 8 試作バンドル構成
Fig. 8 Developed framework bundle structure.

器種別と接続方法との組合せにより 8 個のバンドルを実装した。このバンドルには、ドライバクラス、機器制御実装クラス、ドライバ情報実装クラスが含まれる。通信プロトコルバンドルとしては 3 個のバンドルを実装した。このバンドルは、各通信プロトコル処理を行う機能であり、機器制御実装クラスから利用する。また、機器情報クラス、機器構成管理クラス、ドライバ情報 I/F クラスを含む機器構成管理バンドルを実装した。

4.2 要件の満足性に関する評価

今回実装した機器管理制御フレームワークが、各要件を満たすかを評価した。評価に際して、下記シナリオを実現するために、簡易アプリケーションを開発した。

(a) 集中リモコン制御 (図 9 参照)

基本的な制御機能を確認するために、HGW に接続された宅内機器用の操作画面 (GUI) を Servlet として公開する機能を持つアプリケーションを実装した。PC/タブレット/スマホ等のブラウザで上記操作画面を開き、ユーザが操作を行うことで、あらゆる接続機器の遠隔制御が可能となる。

(b) 電力ピークカット制御 (図 10 参照)

電力量計の値を定期的に監視し、あらかじめ設定した最大許容値を超えた場合に、所定の機器を電源 OFF する機能を持つアプリケーションを実装した。所定機器の指定方法を変えて動作の違いを確認した。

なお、機器 1 つずつの情報を GUI から入力させる機器登録アプリケーションを作成した。最初にドライババンドル種別を入力させた後、それ以外に登録すべき機器情報リストを提示し、ユーザに入力を促す。また、登録した機器



図 9 集中リモコンアプリケーションの操作画面
Fig. 9 Snapshot of remote control application.

情報一覧を表示する機能も持たせた。(a), (b) のいずれの場合も、これを使って機器情報を事前に登録した。実際に登録した機器情報の一例を表 3 に示す。

(1) 要件 R1

本評価環境では、接続方法の異なるエアコンが 4 台接続されており、そのうちの 3 台は基本的なエアコン機能の制御が可能である (1 台は HA 制御のため電源 ON/OFF 制御のみ可)。

集中リモコンアプリケーションにおいて、上記 3 台のエアコンに対して、同じエアコン制御 I/F クラスを介して、電源 ON/OFF、モード変更 (冷房/暖房/除湿等)、設定温度変更の制御を行うことができた。

また、図 8 に示す最終的なバンドル構成に至る途中段階で、対応機器を追加するたびに、新たな機器ドライババンドルの追加、機器情報の登録が必要であったが、アプリケーションの変更はまったく不要であった。以上から、本評価環境においては、要件 R1 を満たすことを確認できたと考える。

(2) 要件 R2

電力ピークカットアプリケーションにおいて、表 3 に示す共通機器情報を含む検索式を指定し、基本機器制御 I/F

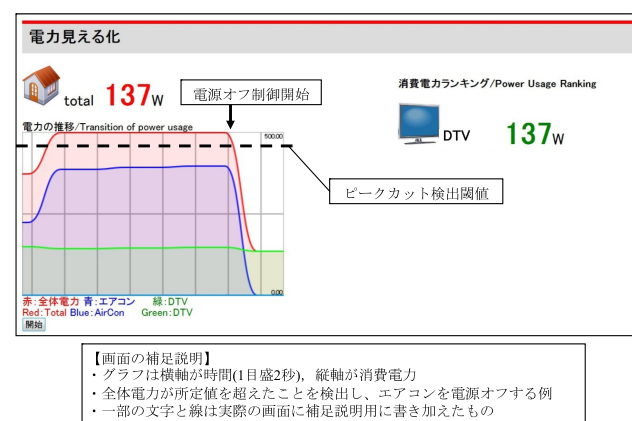


図 10 電力ピークカットアプリケーションの画面
Fig. 10 Snapshot of power peak cut application.

表 3 試作システムにおける機器情報 (抜粋)

Table 3 Example of implemented device information.

種別	内容
共通機器情報	- 機器種別 - ベンダ名 - 機器説明 - ドライバ ID - シリアル番号 - 機器名称 - 通信プロトコル - 設置場所
エアコン 2 ドライバ情報	- 中継器親機 IP アドレス - 中継器子機 MAC アドレス
エアコン 4 ドライバ情報	IP アドレス

サービスを取得するようにした。これにより，“設置場所”や“機器名称”で指定した機器を，目標どおりに電源 OFF 制御できることを確認した。以上から，本評価環境においては，要件 R2 を満たすことを確認できたと考える。

(3) 要件 R3

本評価環境では，表 3 に示すとおり，機器に応じて異なるドライバ情報を実装した。しかし，機器登録アプリケーションとしては，機器にかかわらず同じドライバ情報 I/F クラスを利用することで，機器登録用 GUI を作成できた。また，登録機器情報を一覧表示する機能についても，共通の機器情報 I/F クラスを利用することで実現できた。以上から，本評価環境においては，要件 R3 を満たすことを確認できたと考える。

4.3 アプリケーション開発生産性に関する評価

4.3.1 開発量算出

まず，機器管理制御フレームワークの適用有無に応じた開発量を算出する。

機器管理制御フレームワークを適用する以前に開発した従来アプリケーションの機能構成を図 11 (a) に示す。機器管理制御フレームワークを適用したアプリケーションの機能構成を図 11 (b) に示す。図中には，各機能ブロックの開発ステップ数も示す。なお，従来アプリケーションは，今回開発したアプリケーションに比べ，制御対象機器種別数が少ない，一部の機能を備えていないといった相違がある。これらの条件をそろえた場合の開発量の比較表を表 4 に示す。表中の「-」は当該機能を搭載しないことを示す。なお，GUI および制御アルゴリズムの開発量の差は，外観の作り込み度合いの差であり，本フレームワーク利用の有無による差ではないため，対象外とした。以下で表 4 の算出方法を説明する。

(1) 機器制御処理に関する開発量

機器ごとの制御プロトコル処理等を行う機器制御処理は，従来の 940 ステップに対して，今回は 9,500 ステップと大きく増加している。これは，制御対象機器種別数が 3 から

8 に増えたことが主要要因である。開発量と制御対象機器種別数が比例関係でないのは，制御対象機器種別によって開発量が異なるためであり，たとえば，今回新たに開発した ECHONET Lite 処理の開発量は 3k ステップ程度である。一方で，従来と今回で同一の制御対象機器種別に関する機器制御処理はほぼ同等であった。また，それに加え，ドライバ確定処理やドライバ情報取得処理等の本フレームワーク利用によるオーバーヘッド分がある。これは，1 つの制御対象機器種別あたり 100 ステップずつかかっており，今回は制御対象機器種別数が 8 なので合計で 800 ステップである。

よって，比較条件をそろえるために，従来構成の延長で制御対象機器種別数を 8 に増やした場合を考えると，今回の開発量 (9,500 ステップ) から上記オーバーヘッドを除いた 8,700 ステップとなる。

また，従来は共通制御 I/F がなかったため，制御 I/F を呼ぶ際に，機器に応じた呼び分け処理 (430 ステップ) が必要だった。従来構成のまま対応機器種別数を 8 に増やした場合を考えると，呼び分け処理量が対応機器種別数に比例すると仮定すれば 1,100 ステップかかる。これに対し，今回は共通制御 I/F を呼ぶだけであり，数 10 ステップで実現できた。なお，共通制御 I/F の開発量は 230 ステップだった。

(2) 機器選択/機器登録処理に関する開発量

従来アプリケーションでは，機器選択処理を実装していなかった。実装する場合には，機器と様々な属性値を対応付けて管理する処理が必要となる。たとえば，テーブル管理処理，テーブルサーチ処理等を実装した場合，数 100 ステップはかかると思われる。

これに対し，今回のアプリケーションの機器選択処理は，通常の制御 I/F 呼び出しのステップ数 (数 10 ステップ) に含まれ，増分は 0 で実現できた。

また，従来アプリケーションでは，機器登録処理を実装

表 4 開発量の比較

Table 4 Comparison of development scales.

開発量 [単位: ステップ]	従来構成 (注 1)	フレームワーク 利用時	
	アプリ	アプリ	フレーム ワーク
機器制御	8700	-	9500
制御 API 呼び出し	1100	数 10	-
共通制御 I/F	-	-	230
機器選択	数 100	- (注 1)	-
機器登録	数 100	90	-
機器構成管理	-	-	340
合計	10400	120	10070
		10190	

(注1) 制御 API 呼び出し処理に含まれる。

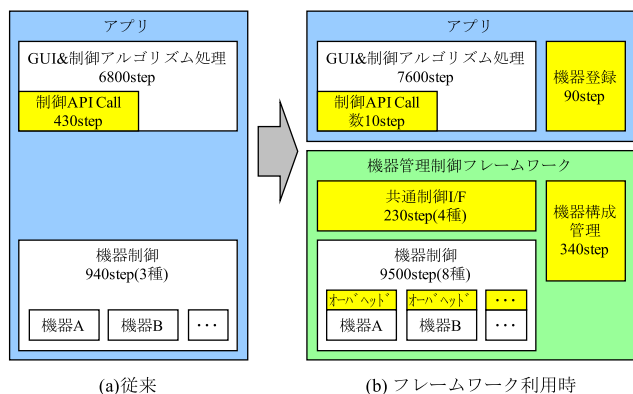


図 11 アプリケーションの機能構成

Fig. 11 Structure of HGW application.

していなかった。実装する場合には、機器ごとに登録事項を管理する処理が必要となる。たとえば、テーブル管理処理、テーブルサーチ処理等を実装した場合、数 100 ステップはかかると思われる。

これに対し、今回のアプリケーションの機器登録処理は 90 ステップで実現できた。一方、フレームワーク側では、機器構成管理処理に 340 ステップかかった。

4.3.2 フレームワーク適用有無による合計開発量比較

前項の結果を基に、機器管理制御フレームワークを利用した場合の、HGW 上のアプリケーションの開発生産性について評価した。

(1) 1 アプリケーションあたりの合計開発量比較

従来アプリケーションの合計開発量は 10,400 ステップ程度であるのに対し、今回のアプリケーションの合計開発量は 120 ステップ程度である。すなわち、1 アプリケーションあたりの開発削減量は約 10k ステップである。これは機器制御処理の削減量が最も大きい。そのため、制御対象機器種別数によって変わると考えられるが、制御対象機器種別数 1 個あたり 1~3k ステップ程度削減できるといえる。

(2) フレームワークを含んだ合計開発量比較

従来アプリケーションの合計開発量は 10,400 ステップ程度であるのに対し、今回のアプリケーションおよびフレームワークの合計開発量は 10,190 ステップ程度である。すなわち、フレームワーク利用のためのオーバーヘッドよりも、フレームワーク利用による削減効果の方が大きいといえる。

ただし、これは一評価環境における結果であり、制御対象機器種別数等によって変わると考えられる。しかしながら、そのような場合であっても、複数アプリケーションを開発することで、フレームワーク利用のためのオーバーヘッドよりもフレームワーク利用による削減効果が有利になると考えられる。

5. おわりに

本研究では、HNS で利用される HGW 用アプリケーションの開発効率の向上を図るため、多様な機器を統一的に管理・制御するための機器管理制御フレームワークを OSGi 上で開発し、その評価を行った。

開発にあたっては、アプリケーションが宅内機器を制御するための共通 I/F を設けるだけでなく、OSGi のサービス検索機能を利用して制御対象機器を柔軟に選択する機能と、機器情報として登録すべき項目一覧を取得する共通 I/F を利用して、様々な機器情報の登録処理を統一に行える機能を設けた。

また、開発した機器管理制御フレームワークを利用して、試作 HEMS システムを構築した。多種多様な機器を制御するアプリケーション開発において、制御対象機器の種別や接続形態を意識せずに実装できることを確認した。1 つ

の制御機器を扱うアプリケーションであれば 1~3k ステップ程度削減できる見込みを得た。

今後は、最大収容可能機器数や処理速度等の性能評価を行う必要がある。また、クラウドサービスとの連携を想定した、HGW 外部からの制御機能の開発や、制御対象機器の拡大を検討していきたい。

参考文献

- [1] OSGi Alliance, available from <http://www.osgi.org/>.
- [2] 丹 康雄：ホームネットワーク (OSGi, ECHONET) モデルに基づく家庭内エネルギーマネジメント, 情報処理学会誌, Vol.51, No.8, pp.959-965 (2010).
- [3] Nakajima, T. et al.: A Virtual Overlay Network for Integrating Home Appliances, *Proc. 2002 Symposium on Applications and the Internet (SAINT '02)* (2002).
- [4] Sumino, H. et al.: Home Appliance Control from Mobile Phones, *4th IEEE Consumer Communications and Networking Conference, 2007, CCNC 2007* (2007).
- [5] Moon, K.-D. et al.: Design of a Universal Middleware Bridge for Device Interoperability in Heterogeneous Home Network Middleware, *IEEE Trans. Consumer Electronics*, Vol.51, No.1 (Feb. 2005).
- [6] 大和ハウス工業株式会社：平成 21 年度スマートハウス実証プロジェクト報告書 第 2 章 テーマ 2-1：マッシュアップを促進するホームサーバ向け統合 API の開発実証およびテーマ 3-1：マルチベンダによる家電・設備機器統合コントロールシステムの開発 (Mar. 2010).
- [7] 福岡祐介ほか：動的バインディング機構を用いたマルチベンダホームネットワークシステムの一実現手法, 信学技報, Vol.107, No.525, pp.295-300 (2008).
- [8] OSGi Alliance: OSGi Service Platform Release4 Device Access Specification Version1.1, available from <http://www.osgi.org/>.
- [9] OSGi Alliance: OSGi Service Platform Core Specification, available from <http://www.osgi.org/>.
- [10] ECHONET コンソーシアム ECHONET Lite 規格書 Ver1.01 (日本語版), 入手先 http://www.echonet.gr.jp/spec/spec.v101_lite.htm.

1 ECHONET は、ECHONET コンソーシアムの登録商標です。

2 OSGi は、米国 OSGi Alliance の登録商標です。

3 Java は、Oracle Corporation およびその子会社、関連会社の米国およびその他の国における登録商標です。

4 Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

5 SuperJ Engine および SuperJ Engine Framework は株式会社日立ソリューションズの登録商標です。

6 Modbus は、Modicon Inc. の登録商標です。

7 Bluetooth は、米国内における Bluetooth SGI Inc. の登録商標または商標です。

8 UPnP は、UPnP Implementors Corporation の登録商標です。



宮田 克也

1976 年生。2000 年京都大学大学院情報学研究科修士課程修了。同年日立製作所入社。空調制御および需要家向け Energy Management System に関連する研究開発に従事。



寺岡 秀敏

1976 年生。2002 年京都大学大学院工学研究科修士課程修了。同年日立製作所入社。サービスゲートウェイおよび需要家向け Energy Management System に関連する研究に従事。



関原 拓也

1978 年生。2003 年静岡大学大学院情報学研究科修士課程修了。同年現、日立ソリューションズ入社。ネットワーク機器の組み込みソフトウェア開発に従事。



芳野 明彦

1977 年生。2002 年愛媛大学大学院理工学研究科修士課程修了。同年現、日立ソリューションズ入社。M2M システムの開発に従事。