NAIST-IS-MT0651105

修士論文

ネットワーク家電と標準インタフェースとの動的結び付けによるマルチベンダホームネットワークシステム構築手法

福岡 佑介

2008年2月7日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

福岡 佑介

審査委員：

松本 健一 教授	（主指導教員）
杉本 謙二 教授	（副指導教員）
門田 暁人 准教授	（副指導教員）
中村 匡秀 准教授	（神戸大学）

ネットワーク家電と標準インタフェースとの動的結び付けによるマルチベンダホームネットワークシステム構築手法*

福岡 佑介

内容梗概

本論文では，複数のベンダのネットワーク家電を収容可能なホームネットワークシステム (HNS) を構築する手法を提案する．まず，ネットワーク家電のベンダに依存しない標準家電サービスを構築し，HNS アプリケーションに向けて公開する．次に，各標準家電サービスとネットワーク家電を動的に結び付ける機構を構築する．これにより，様々なベンダのネットワーク家電に対応できる HNS アプリケーションが開発できるとともに，HNS や HNS アプリケーションを再構築することなく，HNS に収容されているネットワーク家電を置き換えることができる．また，本論文では提案手法に基づいた HNS フレームワークを実装するとともに，そのフレームワークを用いて複数ベンダのネットワーク家電からなる HNS を構築し，提案手法の有用性を確認した．

キーワード

ホームネットワークシステム, ネットワーク家電, マルチベンダシステム, ホームネットワークサービス, 動的バインディング

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT0651105, 2008 年 2 月 7 日.

Constructing Multi-Vendor Home Network System with Dynamic Binding of Networked Appliances and Vendor-Neutral Services *

Yusuke Fukuoka

Abstract

This thesis presents a method that constructs the home network system (HNS) with multi-vendor networked home appliances. The proposed method first prepares vendor-neutral appliance services, and exhibits the services to the HNS applications. We then prepare a mechanism, which dynamically binds each vendor-neutral service and concrete appliances during runtime. Thus, the HNS service provider can build a common HNS application for various combinations of multi-vendor appliances. Moreover, it is unnecessary to rebuild the vendor-neutral services as well as the HNS applications even when any appliance is replaced with another. We have implemented the proposed method, and deployed the system in an actual HNS. The experimental evaluation showed that the implementation worked well with sufficiently small overhead for the practical settings.

Keywords:

home network system, networked appliances, multi-vendor system, home network services, dynamic binding

* Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0651105, February 7, 2008.

目 次

図 目 次

表 目 次

1. はじめに

家電やセンサを家庭内のネットワークに接続することで，家電単体では実現できない，より便利な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) の研究開発が進められており，既に商品化されたものも存在する [20][22][24][25]．また，政府機関なども新たな HNS サービスの開発支援や普及促進を行っている [21]．一般的に，HNS に接続される家電は**ネットワーク家電**と呼ばれ，ネットワークインタフェースやプロセッサ，ストレージを持ち，さらに外部のソフトウェアから家電を制御するための API (**家電 API** と呼ぶ) を備えている．HNS サービスは，これらの家電 API を決められた手順で呼び出すソフトウェア (**HNS アプリケーション**と呼ぶ) によって実現される．

しかしながら，現在提供されている HNS アプリケーションは，単一ベンダ製のネットワーク家電にしか対応していないものが大半である．その原因として，同種のネットワーク家電であっても，そのベンダによって家電 API の仕様が異なることが挙げられる．そのため，HNS サービスを各家庭に提供する事業者 (HNS サービス提供者と呼ぶ) は様々なベンダのネットワーク家電に対応した HNS アプリケーションを開発することができず，またユーザも，好みのベンダのネットワーク家電を任意に選択し，HNS に収容させることができない．

この問題を解決するために，本論文では，複数のベンダのネットワーク家電を収容可能な HNS (**マルチベンダ HNS** と呼ぶ) を構築する手法を提案する．

まず，家電の種類ごとに，任意のベンダのネットワーク家電を制御可能な API を持つ**標準家電サービス**を準備し，HNS アプリケーションに向けて公開する．次に，これらの標準家電サービスと，HNS に収容するネットワーク家電との結び付けを行う．その際，標準家電サービスとネットワーク家電を静的に結び付ける (標準家電サービスの API が呼び出された際に行う家電 API 呼び出し処理を静的に定義する) と，HNS に収容されているネットワーク家電の置き換えが発生する度に，標準家電サービスを再構築する必要が生じる．そこで，標準家電サービスとネットワーク家電との結び付け定義を外部化し，標準家電サービス実行時に，その定義に従って標準家電サービスとネットワーク家電を結び付ける機構を構築する．これにより，HNS に任意のベンダのネットワーク家電を収容することができ

るとともに，ネットワーク家電の置き換えが発生した場合でも，標準家電サービスとネットワーク家電との結び付け定義を変更するだけで対応することができる．

また，本論文では，提案手法に基づいて HNS フレームワークを実装した．さらに，実装したフレームワークを用いて複数ベンダのネットワーク家電からなる HNS を構築し，それらのネットワーク家電を連携動作させる HNS アプリケーションを実装した．その結果，提案手法を用いて複数ベンダのネットワーク家電を収容した HNS を構築できること，また標準家電サービスを利用して HNS アプリケーションを構築できること，さらに HNS や HNS アプリケーションを再構築することなく，HNS に収容されているネットワーク家電を置き換えられることが示された．

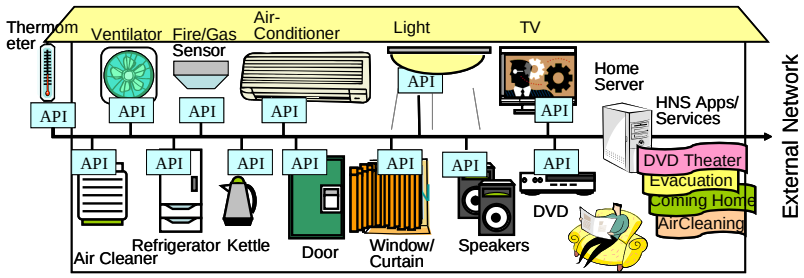

図 1　ホームネットワークシステム (HNS)

2. 準備

2.1 ホームネットワークシステム (HNS)

ホームネットワークシステム (HNS) は，複数のネットワーク家電 (ネットワークに接続する機能を持った家電) と，ホームサーバで構成されており，それらは家庭内の LAN に接続されている．図 1 は HNS を表した図である．各ネットワーク家電には制御用の API (**家電 API**) が備えられており，外部のソフトウェアは家電 API を通してそのネットワーク家電を制御することができる．ホームサーバは，ユーザに提供する様々な HNS サービスを管理するアプリケーションサーバ，ならびに外部ネットワークへのゲートウェイとしての役割を持つサーバである．各 HNS サービスは，あらかじめ決められた手順で各ネットワーク家電の API 呼び出し処理を行うソフトウェア (**HNS アプリケーション**) によって実現される．

HNS サービスの一例として，ここではシアターサービスを挙げる．シアターサービスとは，リビングの照明・カーテン・テレビを連携動作させて，ユーザが映画館のような環境でテレビ番組を視聴できるようにする HNS サービスである．ユーザがシアターサービスを呼び出すと，照明を暗くする，カーテンを閉じる，テレビの電源を入れて最適な音量に設定する，といった一連の操作が自動的に行われる．

なお，家電が持つ API は，いくつかのレイヤに分類することができる．例えば，ネットワークレイヤの API は，アドレス設定，メッセージや信号の形式，通信プロトコルなど，機器がネットワークを介して通信するための処理を扱う．ネッ

トワークレイヤの標準化を図る規格として，DLNA [2]，UPnP [18]，ECHONET [3]，X10 [16]，HomePlug [6] などが提唱されている．

一方，アプリケーションレイヤの API は，低レイヤのミドルウェアやネットワークプロトコルを隠蔽して，ネットワーク家電の論理的な機能を容易に呼び出せるようにしている．例えば，カーテンが持つアプリケーションレイヤの API としては，`open()` や `close()` などが考えられる．通常，これらの API は，そのネットワーク家電のベンダによって開発され，HNS アプリケーションに向けて公開される．このようなアプリケーションレイヤの API を構築するためのフレームワークとして，OSGi フレームワーク [14] が挙げられる．また，ネットワーク家電のアプリケーションレイヤの API を，Web サービス [19] を用いて公開する研究もなされている [7][12].

本論文では，ネットワークレイヤのような低レイヤの家電 API ではなく，アプリケーションレイヤの家電 API について言及する．

2.2 HNS の問題点

現在商品化されている HNS アプリケーションは，単一ベンダ製のネットワーク家電にのみ対応しているものがほとんどである [20][22][24][25]．その要因として，家電 API の仕様がネットワーク家電のベンダによって異なることが挙げられる．

図 2 は，2.1 で述べたシアターサービスを実現する HNS アプリケーションの実装を，模式図で表したものである．(a) の `TheaterService_for_A` は A 社製ネットワーク家電用の HNS アプリケーション，(b) の `TheaterService_for_B` は B 社製ネットワーク家電用の HNS アプリケーションである．また，それぞれの内部に描かれている図形は，各ベンダの家電 API の仕様に合わせた呼び出し処理を表している．`TheaterService_for_A` は，照明・カーテン・テレビの家電 API を順に呼び出すことによって，シアターサービスを実現する．一方，`TheaterService_for_B` も，同じく照明・カーテン・テレビの家電 API を順に呼び出すことによって，`TheaterService_for_A` と同様にシアターサービスを実現するが，それぞれのネットワーク家電が持つ家電 API の仕様 (図中の図形) が，A 社製ネットワーク家電のものと B 社製ネットワーク家電のものとで異なるため，`TheaterService_for_B`

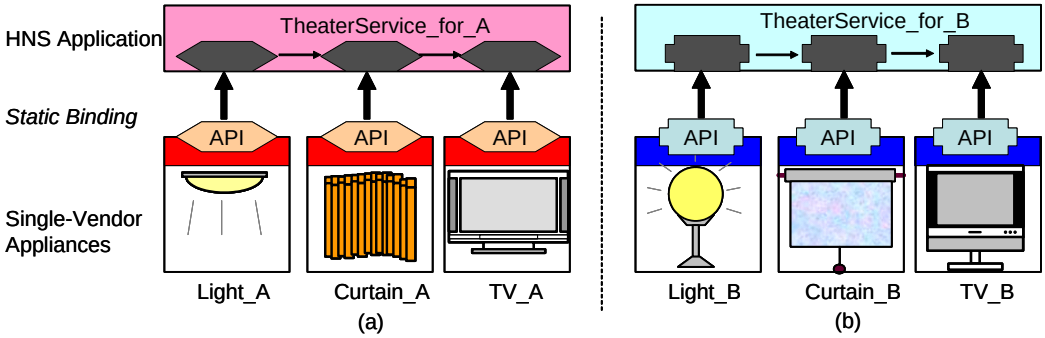

図 2 従来の HNS におけるネットワーク家電と HNS アプリケーションの関係

の実装は `TheaterService_for_A` の実装と全く異なっている．従って，例えば (a) において A 社製の照明を B 社製の照明に置き換えることはできず，各 HNS アプリケーションは単一ベンダ製のネットワーク家電にしか対応できない．

現在の HNS が持つこのような制約は，HNS の普及促進を図る上で大きな障害となる．HNS アプリケーションが対応可能なネットワーク家電の組み合わせが非常に限定されているため，ユーザは自分の好きなベンダのネットワーク家電を選択し，HNS に収容させることができない．また，家電 API の仕様はそのネットワーク家電のベンダが決定するため，それらの API を呼び出す HNS アプリケーションもまた同一のベンダによって提供されることが多く，サードパーティの HNS サービス提供者の参入が困難である．

特定のベンダに依存せず，様々なベンダのネットワーク家電に対応可能な**マルチベンダ HNS** を構築できるようにすることが，HNS の次なる目標である．

2.3 先行研究: 複数ベンダ製従来家電の HNS への適応

HNS はネットワーク家電を対象としたものであり，ネットワーク接続機能を持たず，赤外線リモコンで操作する従来の家電をそのまま HNS に収容することはできない．このような従来の家電を HNS に適応させる手法として，PC から制御できるリモコンを用いることが考えられる．これらのリモコンには，リモコンの初期化・信号種のセット・送信開始・送信停止など，リモコンを制御するための API が備えられている．しかしながら，同じ種類の家電の，同じの機能を呼び出

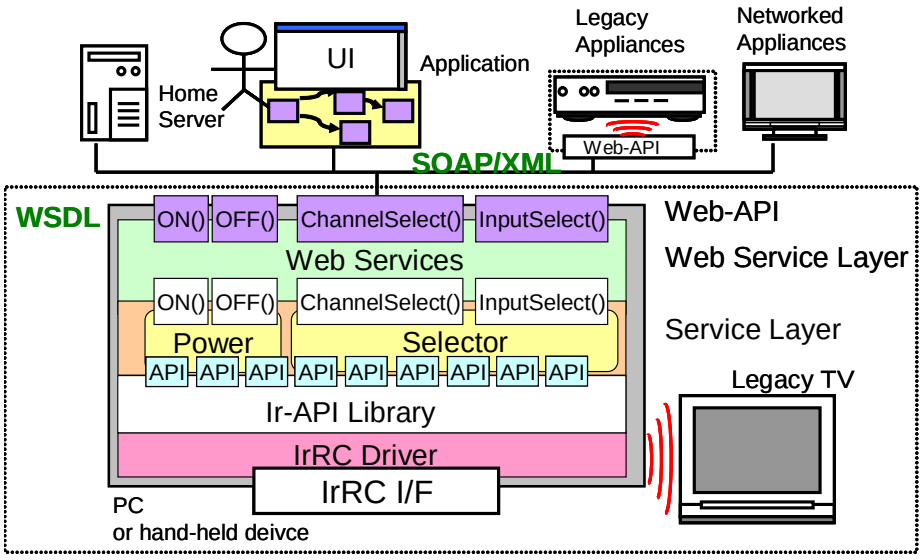

図 3　SOA に基づいた従来家電の HNS への適応

す場合であっても，送信すべき信号種や送信手順は家電のベンダごとに異なるため，各リモコン制御用 API を直接 HNS に公開するのは適当ではない．

この問題を解決するため，田中らはサービス指向アーキテクチャ(SOA) [15] の概念に基づいて従来の家電を HNS に適応させる手法を提案している [11][23]．図 3 に赤外線リモコンで操作するテレビへの適用例を示す．この手法では，リモコン制御用 API の呼び出し処理を家電の論理的な機能単位で纏め，それらを家電 Web サービスの API として HNS アプリケーションに向けて公開する．

これにより，家電のベンダによって異なる信号種や送信手順は，全て各家電 Web サービス内に隠蔽されるため，HNS アプリケーションはこれらの違いを意識することなく，各家電を制御することができる．さらに，複数のベンダ製の家電を組み合わせて連携動作させる HNS サービスを容易に実現することができる．

しかし，田中らの手法はリモコンを用いた従来の家電の制御に特化したアーキテクチャとなっているため，この手法をそのまま各種ネットワーク家電に適用することはできない．さらに，各家電サービスの API と，家電のベンダに応じた学習リモコン制御用 API の呼び出し処理との結び付けが静的に行われているため，HNS に収容されている家電が置き換わる度に家電 Web サービスを再構築する必

要がある．

2.4 家電 API 仕様の標準化

マルチベンダ HNS を構築するためのシンプルな手法として，家電 API の仕様を標準化し，全てのネットワーク家電ベンダの間で統一することが考えられる．しかしながら，これは現実的とはいえない．

家電 API の仕様を厳格に決めてしまうと，結果的に各ネットワーク家電ベンダは他社と同じような製品を生産しなければならず，他社との差別化を図ることができなくなる．また，全てのネットワーク家電ベンダが納得できるような形で家電 API を標準化することは非常に困難である．自社製品の独自性を保つために，他社製品と同じ家電 API 仕様とすることを拒否するネットワーク家電ベンダが存在する可能性もある．

2.5 本論文における仮定

本論文では，以下の 3 つの仮定に基づいて議論を行う．

仮定 A1: 各家電 API は，そのネットワーク家電のベンダが準備する．

仮定 A2: 各家電 API の仕様は，そのネットワーク家電のベンダが決定する．

仮定 A3: 各家電 API の仕様は，ユーザやサードパーティの HNS サービス提供者に公開されている．

3. 提案手法

3.1 要求定義

本論文では，以下の 3 つの要求を満たす，複数ベンダ製のネットワーク家電を収容可能なマルチベンダ HNS を構築するためのフレームワークを提案する．

要求 R1: ベンダを問わず，任意のネットワーク家電を HNS に収容できること．

要求 R2: HNS，ならびに HNS アプリケーションを再構築することなく，HNS に収容されているネットワーク家電を置き換えられること．

要求 R3: ネットワーク家電のベンダに限らず，サードパーティの HNS サービス提供者でもユーザに HNS サービスを提供できること．

一般的に，家庭内には様々なベンダの家電が設置され，また家電のベンダの組み合わせは家庭ごとに異なる．要求 R1 を満たすことにより，HNS はあらゆるベンダの組み合わせに対応できるようになるため，様々な家庭にこの HNS を導入できるようになる．

ユーザがネットワーク家電を買い換えた際は，HNS に収容されているネットワーク家電の置き換えを行う必要がある．また，将来的には音楽プレーヤーや小型テレビなど，携帯可能な家電を，帰宅時に HNS に接続し，外出時に HNS から切断するといった利用形態も予想される．HNS が要求 R2 を満たすことにより，このような収容されているネットワーク家電の変化に柔軟に対応できるようになる．

さらに，HNS フレームワークが要求 R3 を満たすことにより，HNS サービス事業に参入する事業者が増加し，これまでに無かった新しいサービスや，より低コストで高品質なサービスがユーザに提供されるようになることが期待できる．

3.2 提案手法の概要

従来の HNS アーキテクチャでは，HNS アプリケーションとネットワーク家電が，家電 API を通して静的に結び付いているため，2.2 で述べたような問題が発

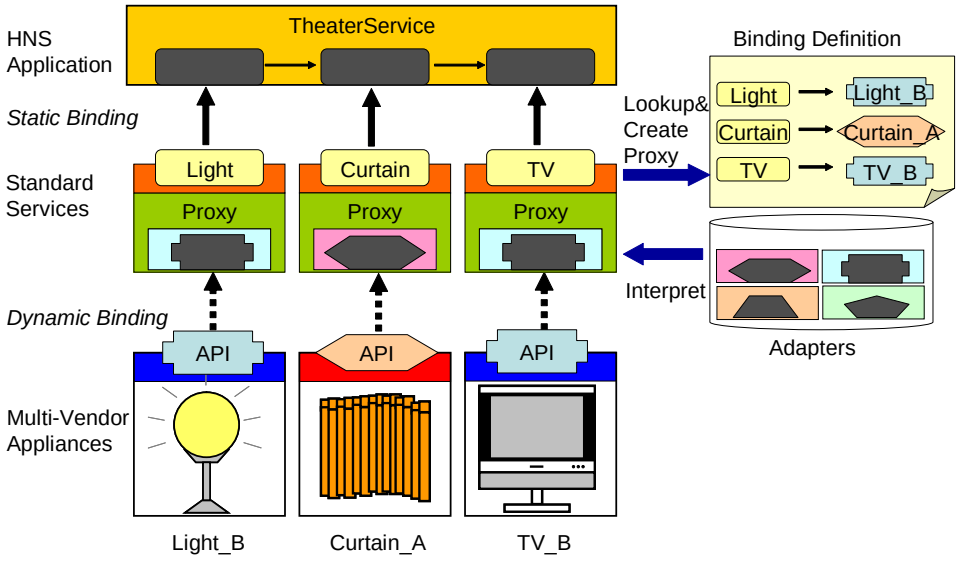

図 4　提案手法のアーキテクチャ図

生している．そこで，本論文では HNS アプリケーションとネットワーク家電との結び付けを動的に行うことにより，要求 R1 から R3 を満たしたマルチベンダ HNS を構築する手法を提案する．

図 4 は，提案手法のアーキテクチャを，2.1 のシアターサービスを例として模式図で表したものである．提案手法は，2 つの大きなステップから構成される．

まず，家電の種類ごとに，ネットワーク家電のベンダに依存しない**標準家電サービス**を準備し，これらを HNS アプリケーションに向けて公開する．HNS アプリケーションは，標準家電サービスを通して HNS に収容されているネットワーク家電を制御することができる．

次に，標準家電サービスとネットワーク家電との結び付け定義を記述した**バインディング定義**と，標準家電サービスの各 API が呼び出された際に行うべき家電 API 呼び出し処理を記述した各ネットワーク家電用の**アダプタ**，さらにこれらを用いて標準家電サービスとネットワーク家電との結び付けを動的に行う，**動的サービスバインディング機構**を構築する．動的サービスバインディング機構は，標準家電サービスの各 API が呼び出された際にバインディング定義を参照し，そこで結び付け先として指定されたネットワーク家電用のアダプタに記述された処

表 1 HNS 構成要素を準備する主体

	従来手法	提案手法
ネットワーク家電	ネットワーク家電ベンダ	ネットワーク家電ベンダ
家電 API	ネットワーク家電ベンダ	ネットワーク家電ベンダ
HNS アプリケーション	ネットワーク家電ベンダ	HNS サービス提供者
標準家電サービス，アダプタ	–	HNS サービス提供者
バインディング定義	–	ユーザ

理を解釈し，家電 API 呼び出し処理を実行する．

表 1 は，提案手法における HNS の各構成要素を準備する主体について示したものである．仮定 A1 ならびに仮定 A2 の通り，各家電 API の仕様はそのネットワーク家電のベンダが決定し，実装する．これは従来手法と同様である．しかし，HNS アプリケーションは，ネットワーク家電のベンダではなく，サードパーティを含めた HNS サービス提供者が準備する．なお，ネットワーク家電のベンダが HNS サービス提供者となることも当然可能である．また，各標準家電サービス，ならびに各ベンダ製品用のアダプタは HNS サービス提供者が準備し，バインディング定義はユーザが準備することを想定している．

以下に，提案手法を構成する 2 つのステップについて詳しく述べる．

3.3 標準家電サービスインタフェースの構築

家電 API の仕様はそのベンダによって異なっており，このことが HNS アプリケーションとネットワーク家電が静的に結合している原因となっている．そこで，ベンダを問わず，標準的な API でネットワーク家電を制御可能な**標準家電サービス**を家電の種類ごとに構築し，これらを HNS アプリケーションに向けて公開する．

標準家電サービスの構築にあたり，まず標準家電サービスのインタフェースを定義する．

3.3.1 標準的な機能の抽出

家電 API の仕様だけでなく，ネットワーク家電に搭載されている機能もそのベンダによって異なっている．そのため，標準家電サービスのインタフェースを定義する前に，まず各標準家電サービスが対象とするネットワーク家電の機能 (**標準的な機能**と呼ぶ) を定義する必要がある．なお，この定義は標準家電サービスを準備する HNS サービス提供者の裁量で行うことができるが，どのベンダの製品にも搭載されており，かつ日常的に使用する機能を標準的な機能と定義することが望ましい．

例えば，各ベンダ製のテレビに搭載されている機能を調査し，表 2 のような結果が得られたとする．これによると，電源・音量調節・消音・副音声・チャンネル変更・チャンネル表示・入力切換・オフタイマー・チャンネル設定・映像設定の 10 機能は，どのベンダの製品も搭載している．一方，その他の機能は，搭載している製品と搭載していない製品が存在する．ここで，BS 放送視聴機能をテレビの標準的な機能に含めてしまうと，標準テレビサービスは BS 放送視聴機能を搭載していない A 社・C 社・D 社製のテレビに対応できなくなり，ベンダに依存することになってしまう．また，一度設定した後は使用する機会の少ない，チャンネル設定・映像設定の 2 機能を呼び出す API を HNS アプリケーションに向けて公開するメリットは非常に小さい．従って，この例の場合は，電源・音量調節・消音・副音声・チャンネル変更・チャンネル表示・入力切換・オフタイマーの 8 機能を，テレビの標準的な機能と定義することが望ましい．

3.3.2 標準家電サービス API の定義

各標準家電サービスについて，3.3.1 で定義した標準的な機能を呼び出すための API と，現在の家電の状態を取得するための API を定義する．

図 5 は，表 2 の結果を基にした標準テレビサービスのインタフェースの定義例を擬似コードで示したものである．標準テレビサービスは，先に定義した 8 つの標準的な機能を呼び出すための API と，現在のテレビの状態を取得するための API を持つ．`getStatus()` API は，現在のテレビの状態を図 6 のようなテレビ状態クラスの形で HNS アプリケーションに通知する．テレビ状態クラスは，テレビ

表 2 各ベンダ製テレビの搭載機能の例

	A 社	B 社	C 社	D 社	E 社
電源	✓	✓	✓	✓	✓
音量調節	✓	✓	✓	✓	✓
消音	✓	✓	✓	✓	✓
副音声	✓	✓	✓	✓	✓
チャンネル変更	✓	✓	✓	✓	✓
チャンネル表示	✓	✓	✓	✓	✓
入力切換	✓	✓	✓	✓	✓
BS 放送視聴	–	✓	–	–	✓
オフタイマー	✓	✓	✓	✓	✓
オンタイマー	✓	–	–	–	–
電力量調整	–	✓	–	✓	–
センサー節電	–	✓	–	–	–
字幕	✓	✓	–	–	✓
番組表	–	✓	–	–	✓
番組内容	–	✓	–	–	✓
チャンネル設定	✓	✓	✓	✓	✓
映像設定	✓	✓	✓	✓	✓
音声設定	✓	✓	–	✓	✓
時刻設定	✓	✓	–	–	–

の標準的な機能を使用することによって取り得る，機器の状態を示すフィールドを持つ．電源などの 2 つの値で表現される状態は真偽型のフィールドで示し，入力などの 3 つ以上の値で表現される状態，ならびに音量などの数値で表現される状態は整数型のフィールドで示す．

なお，標準的な機能の定義と同様に，各標準家電サービスの API 仕様も HNS サービス提供者の裁量で定義することができる．ただし，パラメータを必要とする API に関しては，そのパラメータの仕様についてもネットワーク家電のベンダに依存しないよう配慮すべきである．

例として，標準テレビサービスの `changeVolume()` API について考える．この API は，目的の音量をパラメータとして取る必要があるが，一般的にテレビの最大音量は機種によって異なるため，指定された音量が最大音量以下であるかどうかの判定が困難である．この問題を解決するために，図 5 の `changeVolume()` API の定義では，最小音量を 0，最大音量を 100 としてパラメータを取り，HNS に収容されているテレビの最大音量に合わせて実際の設定音量を変換する仕様と

```
TVInterface {  // 標準テレビサービスのインタフェース
    // 各 API は成功時に 0 を，失敗時にその他の値を返す.
    int on();                              // 電源入
    int off();                             // 電源切
    int changeVolume(int volume);          // 音量調節 (0: 最小，100: 最大)
    int mute();                            // 消音
    int unmute();                          // 消音解除
    int changeVoiceMode(int voiceMode);    // 副音声 (0: 主，1: 副，3: 主+副)
    int changeChannel(int channel);        // チャンネル変更
    int showChannelIndicator();            // チャンネル表示
    int hideChannelIndicator();            // チャンネル表示解除
    int changeInput(int input);            // 入力切替 (0: テレビ，1: HDD レコーダー，2: ゲーム機，...)
    int startOffTimer(int hour);           // オフタイマー
    int cancelOffTimer();                  // オフタイマー解除

    TVStatus getStatus();                  // 状態取得
}
```

図 5 標準テレビサービス API の定義例

```
TVStatus {  // テレビ状態クラス
    boolean power;             // 電源 (true: 入，false: 切)
    int     volume;            // 音量 (0: 最小，100: 最大)
    boolean mute;              // 消音
    int     voiceMode;         // 副音声 (0: 主，1: 副，3: 主+副)
    int     channel;           // チャンネル
    boolean channelIndivator;  // チャンネル表示
    int     input;             // 入力 (0: テレビ，1: HDD レコーダー，2: ゲーム機，...)
    int     offTimer;          // オフタイマー残り時間 (0: オフタイマー停止中)
}
```

図 6 テレビ状態クラスの定義例

している．この場合，例えば最大音量が 50 のテレビが収容されている HNS において `changeVolume(10)` を呼び出すと，そのテレビの音量は 5 に設定されることになる．

3.4 標準家電サービスとネットワーク家電の動的結び付け

HNS アプリケーションが標準家電サービスを通してネットワーク家電を制御できるようにするためには，3.3.2 で定義した標準家電サービスの各 API ごとに家電 API を呼び出す処理を実装し，標準家電サービスとネットワーク家電を結び付ける必要がある．ところが，その際に標準家電サービスとネットワーク家電を静的に結び付けてしまうと，ネットワーク家電の置き換えが発生する度に標準家電サービスを再構築しなければならなくなり，HNS フレームワークが要求 R2 を満たすことができなくなる．そこで，標準家電サービスとネットワーク家電との結び付け定義を外部化し，標準家電サービス実行時に，その定義に従って標準家電サービスとネットワーク家電を動的に結び付ける機構 (**動的バインディング機構**) を構築する．

まず，各標準家電サービスとネットワーク家電との**バインディング定義**と，標準家電サービスの各 API が呼び出された際に行うべき家電 API 呼び出し処理を記述した，各ベンダのネットワーク家電用の**アダプタ**を準備する．図 7 はバインディング定義の例である．バインディング定義では，各標準家電サービスについて，結び付けるネットワーク家電を指定する．また，図 8 はアダプタの記述例である．仮定 A3 により，各家電 API の仕様は公開されているため，HNS サービス提供者は各ベンダのネットワーク家電の API 仕様に合わせたアダプタを準備し，ユーザに提供することができる．

次に，各標準家電サービスについて，標準家電サービスとネットワーク家電との結び付けを動的に行うダイナミックプロキシを実装する．このダイナミックプロキシは，標準家電サービスの API が呼び出された際にバインディング定義を参照し，そこで指定されたネットワーク家電用のアダプタを解釈して，家電 API の呼び出し処理を実行する．

```
BindingDefinition {
    // 標準照明サービスと B 社製照明を結び付ける.
    StandardLightService -> Light_B;

    // 標準カーテンサービスと A 社製カーテンを結び付ける.
    StandardCurtainService -> Curtain_A;

    // 標準テレビサービスと B 社製テレビを結び付ける.
    StandardTVService -> TV_B;

    ...
}
```

図 7 バインディング定義の例

```
adapter for TV_B {  // B 社製テレビ用アダプタ
    if StandardTV.on() is called {
        // テレビの電源を入れるために必要な家電 API 呼び出し処理
        ret = tv_B.power(true);
        return ret;
    }

    if StanderdTV.off() is called {
        // テレビの電源を切るために必要な家電 API 呼び出し処理
        ret = tv_B.power(false);
        return ret;
    }

    ...
}
```

図 8 アダプタの記述例

3.5 提案フレームワークの設計

標準テレビサービスを例として，これまでに述べた提案フレームワークをクラス図で表したものを図 9 に示す．また，HNS アプリケーションが B 社製のテレビを制御する際のシーケンス図を図 10 に示す．

TVInterface は，3.3 で定義した，標準テレビサービスのインタフェースである．標準テレビサービス TVService は，TVInterface を実装して，HNS アプリケーションに向けて公開する．

まず，標準テレビサービスはバインディング定義 BindingDefinition を参照し，そこで結び付け先として指定されている B 社製テレビ用のプロキシオブジェクト TVProxyForB を ProxyFactory に生成させる．TVProxyForB は，TVService と同様に TVInterface を実装しており，各 API が呼び出された際に B 社製テレビの家電 API を呼び出す処理を行う．

HNS アプリケーション TVClient が標準家電サービスの on() API を呼び出すと，標準家電サービスは TVProxyForB の on() メソッドを呼び出す．TVProxyForB は B 社製テレビ用のアダプタ TVAdapter_B に記述された，電源を入れるために必要な家電 API 呼び出し処理を解釈し，実行する．その結果，B 社製テレビの電源が入る．なお，HNS アプリケーションが他の標準家電サービスの API を呼び出した場合も，同様の流れとなる．

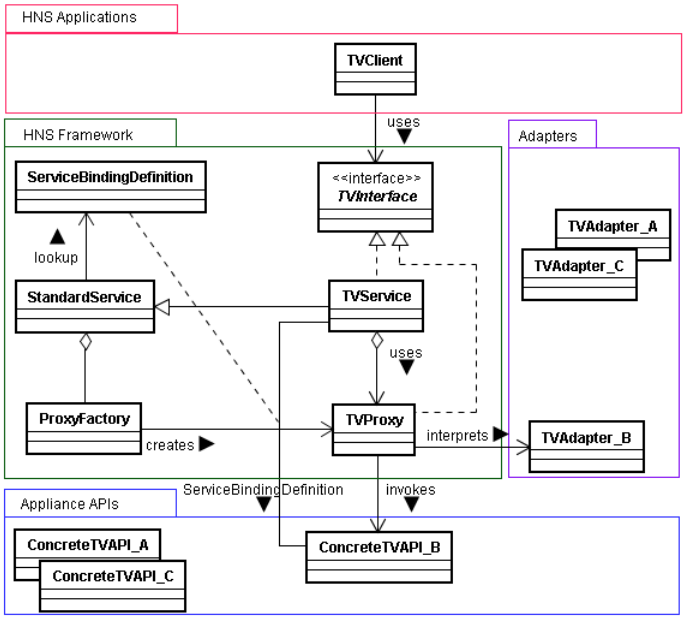

図 9 提案フレームワークのクラス図

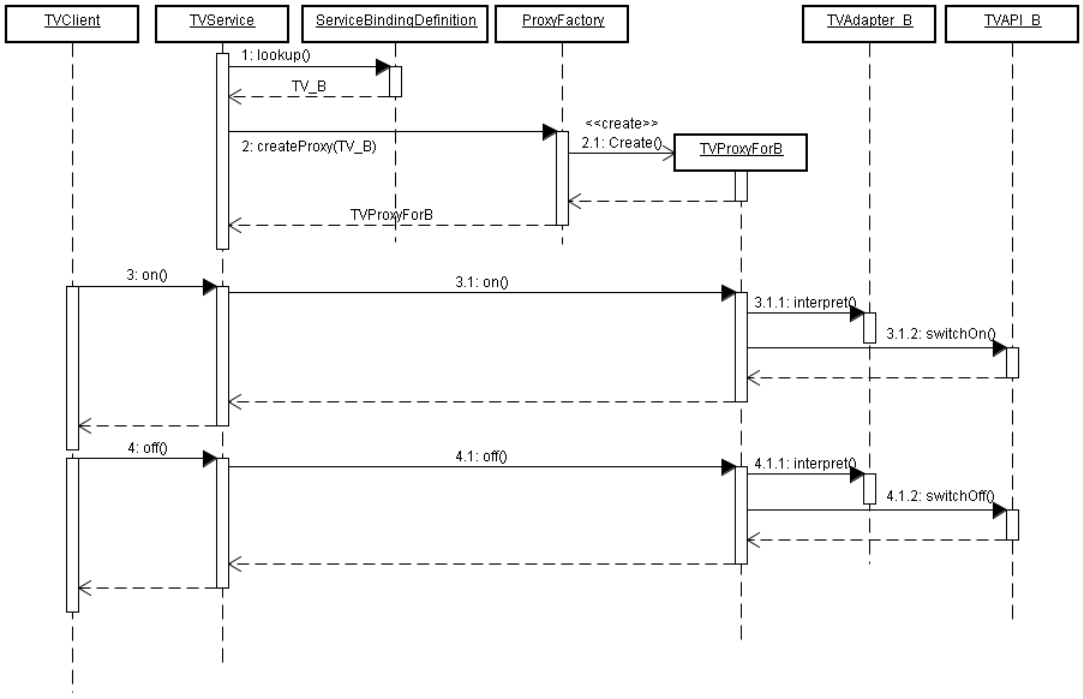

図 10 提案フレームワークのシーケンス図

4. 実装

4.1 概要

提案手法に基づいたHNSフレームワークであるVerbenaを，Javaを用いて実装した．

4.2 標準家電サービスインタフェースの定義

3.3で述べた手法に基づき，各標準家電サービスのインタフェースを定義した．

まず，家電の種類ごとに，現在市場に出回っている各ベンダの製品に搭載されている機能の調査を行った．なお，ネットワーク家電の機種がまだ少ないという現状を踏まえ，この調査は従来の家電を対象として行った．

次に，この調査結果を基に，各標準家電サービスが対象とする標準的な機能の定義を行った．その際，標準家電サービスがあらゆるベンダの家電に対応できることを重視し，全てのベンダの製品が搭載している機能のみを標準的な機能として抽出した．また，各種設定機能など，日常的に使用しない機能は標準的な機能に含めない方針とした．

さらに，標準的な機能を呼び出すためのAPIと，現在の家電の状態を取得するためのAPIを持った，各標準家電サービスのインタフェースを定義した．なお，標準的な機能を呼び出すための各APIは，処理に成功した場合には0を，失敗した場合にはその他の整数値を返す仕様とした．

付録Aに，各ベンダ製品の搭載機能の調査結果，ならびに各標準家電サービスのインタフェース定義を添付する．

4.3 動的サービスバインディング機構の構築

3.4で述べた，各標準家電サービスとネットワーク家電を実行時に結び付ける機構の実装を行った．

図11は，Verbenaにおけるバインディング定義の記述例である．Verbenaは，各標準家電サービスのバインディング定義が，その結び付け先のネットワーク家

電用のアダプタを兼ねる仕様となっている．そのため，標準家電サービスの各 API が呼び出された際に行う家電 API 呼び出し処理を，バインディング定義に直接 JavaScript で記述する．なお，4.2 で標準的な機能を呼び出すための各 API が返す値を定義したが，家電 API によって返り値の型は異なり，さらに返り値を返さない家電 API も存在する．そこで，バインディング定義に返り値の変換や設定を行う処理も併せて記述し，標準家電サービスの各 API が HNS アプリケーションに適切な値を返すようにした．

図 12 は，標準テレビサービスとして公開する `TVService` クラスの定義，また図 13 は各標準家電サービスのクラスが継承する `StandardService` クラスの定義である．`TVService` クラスは実行時にテレビ用プロキシを生成し，各メソッドが呼び出された際に，テレビ用プロキシの同一のメソッドを呼び出す．テレビ用プロキシは，メソッドが呼び出された際にバインディング定義に JavaScript で記述された処理を解釈し，実行する．なお，各ネットワーク家電用プロキシの生成には，ダイナミックプロキシクラス [17] を，またバインディング定義に記述された JavaScript の解釈・実行には，Java で記述された JavaScript の実装である Rhino [10] を用いた．

さらに，各標準家電サービスと HNS アプリケーションとの相互運用性を確保するために，各標準家電サービスのクラスを，Apache AXIS を用いて Web サービス [19] として公開した．

```
// 標準テレビサービスと B 社製テレビを結び付ける.
TVInterface = new function() {
    this.on = function() {  // 標準テレビサービスの on() API が呼び出された場合の処理
        // B 社製テレビの家電 API を呼び出す.
        var ret = new Packages.TV_B().power(true);

        // 返り値を変換する.
        if (ret == true) {
            return new Packages.java.lang.Integer(0);
        } else {
            return new Packages.java.lang.Integer(1);
        }
    };

    ...
}
```

図 11 Verbena のバインディング定義記述例

```
public class TVService extends StandardService implements TVInterface {
    private TVInterface tvProxy;

    public TVService() {
        // テレビ用プロキシを生成する.
        super();
        tvProxy = (TVInterface) factory.createProxy(TVInterface.class);
    }

    public int on() {
        // テレビ用プロキシのメソッドを呼び出す.
        TVStatus status = tvProxy.getStatus();
        if (status.getPower() == false) {
            return tvProxy.on();
        } else {
            return 1;
        }
    }

    ...
}
```

図 12 標準テレビサービスクラス

```
public class StandardService {
    protected ProxyFactory factory;

    protected StandardService {
        try {
            // バインディング定義を読み込む.
            factory = new ProxyFactory("/config.js");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

図 13 StandardService クラス

5. 実験的評価

5.1 概要

提案手法に基づいた HNS フレームワークの実装である Verbena を，実際に各種家電に適用して HNS を構築する実験を行った.

5.1.1 実験環境

本実験は，以下のような環境で行った.

- 家電:
  - プラズマディスプレイ: 日本電気 PX-50XM2
  - DVD プレーヤ: パイオニア HTZ-535DV
  - 照明: 松下電工 HHFZ5310，キシマ KFF-1708，仮想 UPnP 照明
  - カーテン: ナビオ パワートラック
  - 空気清浄機: 日立 EP-V12
  - サーキュレータ: 森田電工 MCF-257NR
- 赤外線学習リモコン: スギヤマエレクトロン クロッサム 2+USB，玄人志向 KURO-RS
- ホームサーバ: PC/AT 互換機
  - CPU: Intel Core Solo Processor U1400 (1.20 GHz)
  - メインメモリ: 1.5 GB
  - HDD: 40 GB (4,200 rpm)
  - OS: Microsoft Windows XP Professional

本実験で使用した家電は，仮想 UPnP 照明を除き，ネットワーク接続機能を持たない従来型の家電である．そこで，PC に接続可能な 2 種類のリモコンと，それぞれに付属の制御用ライブラリを用いてこれらの家電 API を実装し，ネットワーク家電と見なして実験を行った．また，Java 用の UPnP コンポーネントである Cyberlink for Java [8] を用いて，PC 上で動作する仮想 UPnP 照明を実装し，同様に Verbena の適用を試みた．

5.2 バインディング定義の作成

5.1.1 で述べた各家電の API 仕様に合わせて，バインディング定義を作成した．図 14 は松下電工製照明用のバインディング定義，図 15 はキシマ製照明用のバインディング定義，図 16 は仮想 UPnP 照明用のバインディング定義であり，それぞれ標準照明サービスとの結び付けを定義している．図 14 ならびに図 15 では，標準照明サービスと 2 台の照明との結び付けを定義している．

なお，プラズマディスプレイは標準テレビサービスと結び付け，サーキュレータは標準扇風機サービスと結びつけた．

5.3 HNS アプリケーションの実装

各標準家電サービスを用いて，以下の 3 つのサービスを実現する HNS アプリケーションを実装した．なお，これらの HNS アプリケーションは Perl を用いて実装し，Web サービスとして公開されている各標準家電サービスを呼び出すために SOAP::Lite モジュール [9] を使用した．

DVD シアターサービス: テレビの入力切り替えと音量調節を行い，カーテンを閉じ，照明を消灯して，DVD を再生する．

空気清浄サービス: 空気清浄機とともに扇風機を動作させ，室内の空気を効率良く浄化する．

おでかけサービス: カーテンを閉じ，全ての家電の電源を切る．

```
LightInterface = new function() {  // 標準照明サービス
    this.on = function() {  // 照明を点灯する API
        // 家電 API を呼び出す.
        var ret1 = Packages.NationalLightChannel1().on();
        var ret2 = Packages.NationalLightChannel2().on();

        // 返り値を変換する.(両方の API 呼び出しが成功したときに 0 を返す.)
        if ((ret1 == true) && (ret2 == true)) {
            return new Packages.java.lang.Integer(0);
        } else {
            return new Packages.java.lang.Integer(1);
        }
    };

    ...
};
```

図 14 松下電工製照明用のバインディング定義

```
LightInterface = new function() {  // 標準照明サービス
    this.on = function() {  // 照明を点灯する API
        // 家電 API を呼び出す.
        var ret1 = Packages.KishimaLightChannel1().powerOn();
        var ret2 = Packages.KishimaLightChannel2().powerOn();

        // 返り値を変換する.(両方の API 呼び出しが成功したときに 0 を返す.)
        if ((ret1 == true) && (ret2 == true)) {
            return new Packages.java.lang.Integer(0);
        } else {
            return new Packages.java.lang.Integer(1);
        }
    };

    ...
};
```

図 15 キシマ製照明用のバインディング定義

```
LightInterface = new function() {  // 標準照明サービス
    this.on = function() {  // 照明を点灯する API
        // 家電 API を呼び出す.
        new Packages.UPnPLight().power(true);

        // 返り値を設定する.
        return new Packages.java.lang.Integer(0);
    };

    ...
};
```

図 16 仮想 UPnP 照明用のバインディング定義

さらに，実装した HNS アプリケーションを実行した結果，HNS アプリケーションが標準家電サービスを通して各家電を制御できることが確認された．

5.4 家電置き換えへの対応

バインディング定義を変更することで，HNS に収容されている家電の置き換えに対応できることを確認する実験を行った．

本実験では，松下電器製の照明からキシマ製の照明への置き換えを想定し，バインディング定義を図 14 の内容から図 15 の内容に変更した上で，5.3 で実装した各 HNS アプリケーションを実行した．その結果，バインディング定義を変更するだけで，HNS や HNS アプリケーションを再構築することなく，キシマ製の照明への置き換えに対応することができた．また，同様にバインディング定義を図 16 に変更することで，仮想 UPnP 照明への置き換えに対応できることも確認した．

5.5 性能

Verbena は，標準家電サービスとネットワーク家電との結び付けを，標準家電サービスの実行時に動的に行っている．そのため，標準家電サービスとネットワーク家電を静的に結び付ける場合に比べ，オーバヘッドが発生することが予想され

表 3 HNS アプリケーションの応答時間

	おでかけサービス	DVD シアターサービス	空気清浄サービス
標準家電サービス API 数	5	5	2
応答時間: 静的結び付け時 [ms]	9,709	17,369	6,659
応答時間: 動的結び付け時 [ms]	9,916	17,522	6,709
応答時間差 [ms]	207	153	50
相対応答時間差	2.1%	0.9%	0.8%

る．そこで，標準家電サービスとネットワーク家電の結び付けを動的に行うことによるオーバヘッドを明確にするために，Verbena の他に標準家電サービスとネットワーク家電を静的に結び付けた HNS フレームワークを準備し，2 つの環境下で，5.3 で実装した HNS アプリケーションの応答時間を測定した．

その結果を表 3 に示す．最上段は HNS アプリケーションが呼び出す各標準家電サービスの API の数である．また，応答時間，ならびに応答時間差の単位は全てミリ秒である．

表 3 を見ると，全ての HNS アプリケーションにおいて，標準家電サービスとネットワーク家電の結び付けを動的に行った場合の応答時間が，静的に結び付けた場合の応答時間を上回っていることが分かる．しかしながら，それらの応答時間の差，すなわち標準家電サービスとネットワーク家電の結び付けを動的に行う際のオーバヘッドは，最大のおでかけサービス実行時でも 200 ミリ秒強に留まっている．一般的に，HNS アプリケーションに高い実行性能が求められることは少ないため，このオーバヘッドは深刻な問題とはならないと考えられる．

6. 考察

6.1 要求の満足度

提案手法の実装である Verbena が，3.1 で述べた要求 R1，R2，R3 を満たしているか確認する．

Verbena は，ネットワーク家電のベンダに依存しない標準家電サービスと，HNS に収容するネットワーク家電とを実行時に結び付けることによって，HNS アプリケーションが標準家電サービスを通してネットワーク家電を制御できるようにしている．従って，任意のベンダ製のネットワーク家電を HNS に収容することができる．以上の点から．Verbena は要求 R1 を満たしている．

また，Verbena では HNS に収容されているネットワーク家電の置き換えが発生した場合でもバインディング定義を変更するだけで対応でき，その家庭の HNS を再構築する必要は無い．加えて，ネットワーク家電の置き換えが発生しても標準家電サービスの API 仕様は一切変更されないため，既にユーザが使用している HNS アプリケーションを再構築する必要も無い．以上の点から，Verbena は要求 R2 を満たしている．

さらに，3.3 でも述べたとおり，各標準家電サービスが対象とするネットワーク家電の機能，ならびにそれらの機能を呼び出すための API 仕様は，標準家電サービスを準備する HNS サービス提供者の裁量で決定することができる．そのため，ネットワーク家電のベンダのみならず，サードパーティの HNS サービス提供者もユーザに各種 HNS サービスを提供することができる．以上の点から，Verbena は要求 R3 を満たしている．

6.2 提案手法の利点

標準家電サービスのインタフェースは，その家庭に設置されているネットワーク家電のベンダによらず，各家庭に渡って共通している．そのため，提案手法を用いることにより，HNS サービス提供者は家庭ごとに異なるネットワーク家電ベンダの組み合わせを意識することなく，各家庭に共通の HNS アプリケーションを構築し，それぞれの家庭に配布することができる．

なお，Verbena はネットワーク家電の制御を目的として設計・開発されたフレームワークだが，柔軟なバインディング定義を行えるため，ネットワーク家電の制御以外の目的に応用することも可能である．例えば，以下のような応用例が考えられる．

標準ニュースサービス: 各新聞社の Web ページと結び付け，ユーザの好みに合わせた情報源からのニュースを提供する．

標準 SOS サービス: 緊急事態が発生した場合に，警察・消防・親類など，あらかじめ定義された連絡先に通知する．

さらに，これらのサービスと各標準家電サービスを区別することなく，これらを連携させる HNS アプリケーションを容易に構築することができる．

6.3 提案手法の限界

3.3.1 でも述べた通り，提案手法では，できるだけ多くのベンダの製品に対応するために，一部のベンダの製品にしか搭載されていない機能については標準家電サービスの対象から除外している．これらの機能についても HNS アプリケーションから呼び出せるようにする手法として，以下のように，家電 API 名やパラメータを与えることによって，その家電 API を呼び出すメソッドを準備することが考えられる．

```
Object invokeAPI(String name, Object[] params);
```

しかしながら，HNS アプリケーションが標準家電サービスのメソッドを使用せずにこのようなメソッドを使用すると，その HNS アプリケーションは特定のベンダ製のネットワーク家電が設置されている家庭でしか利用できなくなり，HNS アプリケーションのポータビリティの低下を招く．そのため，このようなメソッドの使用については十分に検討すべきである．

6.4 関連研究

2.1 でも述べた OSGi フレームワークは，システム上のオブジェクト (ネットワーク家電を含む) の動的な変化に対応することができる [1][14]．しかしながら，OSGi フレームワークが対応できるのは，システムへのオブジェクトの動的な追加ならびに削除に限られ，機器ごとに異なる API 仕様の変化については考慮されていない．

また，提案手法は，標準家電サービスと家電 API とのインタフェースの差異を吸収するために，Adapter パターン [5] を用いていると捉えることができる．しかしながら，提案手法では実際に用いるアダプタを，開発時ではなく実行時に決定するため，未知のネットワーク家電についても標準家電サービスと結び付けることが可能である．

提案手法における，具体的な機器を抽象的なサービスと捉える考え方は，サービスコンポーネントアーキテクチャ(SCA) [13] の概念と類似している．しかしながら，SCA をマルチベンダ HNS の構築に応用した研究は存在しない．

近年，コンポーネント間の依存性をソースコード内部に含めず，実行時に外部から注入する，Dependency Injection (DI) [4] というデザインパターンが注目されている．DI パターンでは，あらかじめコンポーネントの実装に依存しないインタフェースを準備しておき，そのインタフェースに準じてクライアントを開発する．実際にクライアントが用いるコンポーネントは，外部で定義されたマッピングに基づいて決定される．そのため，このマッピングを変更することによって，インタフェースとコンポーネントの実装との結び付けを自由に変更することが可能である．なお，DI パターンでは，マッピングは主にコンパイル時に参照される．

DI パターンの基本的な考え方は，提案手法と同一である．しかしながら提案手法では，DI パターンにおける，コンポーネントの実装にあたるオブジェクトが，HNS サービス提供者が準備するアダプタとプロキシオブジェクト，さらに家電ベンダが準備する家電 API のように，そのオブジェクトを準備する主体ごとに分かれている点が DI パターンと異なっている．また，インタフェースとコンポーネントの実装との結び付けが実行時にのみ行われる点が，提案手法の特徴である．

7. まとめ

本論文では，大半の HNS アプリケーションが単一ベンダ製のネットワーク家電にしか対応していないという，現在の HNS の問題点を指摘した．そして，この問題を解決するために，提案するマルチベンダ HNS を構築するためのフレームワークが満たすべき 3 つの要求を挙げた．これらの要求を満たすために，ネットワーク家電のベンダに依存しない標準家電サービスを構築し，標準家電サービスとネットワーク家電の結び付けを動的に行う手法を提案した．また，提案手法に基づいた HNS フレームワークである Verbena を実装し，Verbena と複数のベンダの家電を用いてマルチベンダ HNS を構築した．さらに，Verbena が 3 つの要求を満たすとともに，標準家電サービスとネットワーク家電との動的な結び付けを，非常に小さいオーバヘッドで行えることを確認した．

なお，提案手法の実用化を考えた場合，ユーザがより簡単・手軽にネットワーク家電を置き換えられるようにするためには，バインディング定義の変更を支援するアプリケーションや，それに連動して各ネットワーク家電用のアダプタをネットワークを通して提供するような仕組みが必要となる．また，HNS サービス提供者が各ベンダ製のネットワーク家電用のアダプタを効率的に開発できるようにする仕組みも必要になると考えられる．これらの仕組みの提案を，今後の研究対象としたい．

謝辞

本研究は，多くの方々のご指導，ご支援の下，ここまで進めることができました．ここに感謝の意を表します．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には，本研究の主指導教員を担当していただきました．また，2 年間の研究生活に関するご指導は元より，今後の進路に関しても多大なるご指導をいただきました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 杉本 謙二 教授には，副指導教員を担当していただき，普段ご指導いただいている先生方とはまた違った視点から本研究に対するコメントを頂きました．厚く御礼申し上げます．

神戸大学 大学院工学研究科 中村 匡秀 准教授には，副指導教員を担当していただきました．また，HNS や SOA，Web サービスの基礎などの技術的な面から，研究発表や研究論文の執筆に関する事柄に至るまで，きめ細かく暖かいご指導をいただきました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 准教授には，副指導教員を担当していただきました．さらに，自らの研究をアピールすることの大切さをご指導くださいました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助教からは，研究生活全般に渡り様々なご指導や励ましのお言葉をいただきました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 森崎 修司 助教からは，研究室内での研究報告などにおいて，的確かつ有用なアドバイスをいただきました．厚く御礼申し上げます．

神戸大学 大学院工学研究科 井垣 宏 特命助教からは，本研究に関する鋭いご指摘や，実験に用いる機器に関する様々なご助言をいただきました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 西岡 隆司氏からは，Verbena の実装に関して，多大なるご協力とアドバイスを頂きました．氏の協力なくして，Verbena の実装は実現しなかったと思います．本当にありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 大西 洋司氏，同 田中 秀一郎氏，同 前島 弘敬氏には，各ベンダ製家電の搭載機能調査についてご協力いただきました．本当にありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 Web サービス班のみなさまからは，週次のミーティングや日々の研究活動において，貴重なアドバイスや励ましのお言葉をいただきました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座のみなさま，ならびにソフトウェア設計学講座のみなさまからも，様々なご助言やご支援をいただきました．厚く御礼申し上げます．

最後になりましたが，2 年間の研究生活を，経済的，そして精神的に支えてくれた両親ときょうだいに，深く感謝の意を表します．

参考文献

[1] J. Bourcier, A. Chazalet, M. Desertot, C. Escoffier, and C. Marin, "A Dynamic-SOA Home Control Gateway," *Proc of International Conference on Service Computing (SCC 2006)*, pp.18-22, Chicago, USA, Sep. 2006.

[2] Digital Living Network Alliance, http://www.dlna.org/

[3] ECHONET Consortium, http://www.echonet.gr.jp/

[4] M. Fowler, "Inversion of Control Containers and the Dependency Injection pattern," http://www.martinfowler.com/articles/injection.html

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Pattern: Elements of Reusable Object-Oriented Software," Addison-Wesley Professional, 1994.

[6] HomePlug Powerline Alliance, http://www.homeplug.org/

[7] H. Igaki, M. Nakamura, and K. Matsumoto, "A Service-Oriented Framework for Networked Appliances to Achieve Appliance Interoperability and Evolution in Home Network System" (short paper), *Proc. of International Workshop on Principles of Software Evolution (IWPSE 2005)*, pp.61-64, Lisbon, Portugal, Sep. 2005.

[8] S. Konno, "CyberLink for Java,"
http://www.cybergarage.org/net/upnp/java/

[9] P. Kulchenko, and B. Reese, "SOAP::Lite for Perl,"
http://www.soaplite.com/

[10] Mozilla Foundation, "Rhino,"
http://developer.mozilla.org/ja/docs/Rhino

[11] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K. Matsumoto, "Adapting Legacy Home Appliances to Home Network Systems Using Web Services," *Proc. of International Conference on Web Services (ICWS 2006)*, pp.849-858, Chicago, USA, Sep. 2006.

[12] M. Nakamura, H. Igaki, H. Tamada, and K. Matsumoto, "Implementing Integrated Services of Networked Home Appliances Using Service Oriented Architecture," *Proc. of 2nd International Conference on Service Oriented Computing (ICSOC 2004)*, pp.269-278, NY, USA, Nov. 2004.

[13] Open Service Oriented Architecture Collaboration, "Service Component Architecture Home," http://www.osoa.org/display/Main/Service+Component+Architecture+Home

[14] OSGi Alliance, http://www.osgi.org/

[15] M.P. Papazoglou, and D. Georgakopoulos, "Service-Oriented Computing," *Communications of the ACM*, Vol.46, No.10, pp.25-28, Oct. 2003.

[16] Smart Home Systems, Inc., "The X-10 Story," http://www.smarthomeusa.com/info/x10story/

[17] Sun Microsystems, Inc., "ダイナミックプロキシクラス," http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/reflection/proxy.html

[18] UPnP Forum, http://www.upnp.org/

[19] W3C Web Services Activity, http://www.w3.org/2002/ws/

[20] 株式会社東芝, "東芝ネットワーク家電 フェミニティ," http://www3.toshiba.co.jp/feminity/

[21] 経済産業省, "ネット KADEN 2007," http://www.meti.go.jp/policy/netkaden.html

[22] 象印マホービン株式会社, “みまもりほっとライン,”
http://www.mimamori.net/

[23] 田中章弘, 中村匡秀, 井垣宏, 松本健一, “Web サービスを用いた従来家電のホームネットワークへの適応,” 電子情報通信学会技術研究報告, Vol.105, No.628, pp.067-072, Mar. 2006.

[24] 松下電工株式会社, “ライフィニティ,”
http://biz.national.jp/Ebox/kahs/

[25] 三菱電機株式会社, “三菱ルームエアコン みまもりさーばー,”
http://www.mitsubishielectric.co.jp/home/kirigamine/it_b.html

付録

A. 各ベンダ製家電の搭載機能調査結果ならびに標準家電サービスのインタフェース定義

A.1 エアコン

	東芝	日立	松下	三洋	シャープ
電源	✓	✓	✓	✓	✓
温度設定	✓	✓	✓	✓	✓
運転切替	✓	✓	✓	✓	✓
風向調節	✓	✓	✓	✓	✓
風量調節	✓	✓	✓	✓	✓
スイング	✓	✓	–	–	–
タイマ	✓	✓	✓	✓	✓
クリーニング	✓	✓	✓	–	✓
ソフト除湿	–	✓	–	–	–
パワフル	–	–	✓	–	✓

```
// 標準エアコンサービスインタフェース
public interface AirConditionerInterface {
    int on();                                   // 電源入
    int off();                                  // 電源切
    int changeTemperature(int temperature);     // 温度設定
    int changeOperation(int operation);         // 運転切替 (1: 冷房, 2: 除湿, 3: 暖房)
    int changeWindDirection(int windDirection); // 風向調節 (0: 自動, 1: 上, 2: 中, 3: 下)
    int changeWindLevel(int windLevel);         // 風量調節 (0: 自動, 1: 弱, 2: 中, 3: 強)
    int startOnTimer(int hour);                 // オンタイマ
    int cancelOnTimer();                        // オンタイマ解除
    int startOffTimer(int hour);                // オフタイマ
    int cancelOffTimer();                       // オフタイマ解除

    AirConditionerStatus getStatus();           // 状態取得
}

// エアコン状態クラス
public class AirConditionerStatus {
    private boolean power;          // 電源 (true: 入, false: 切)
    private int     temperature;    // 設定温度
    private int     operation;      // 運転モード (1: 冷房, 2: 除湿, 3: 暖房)
    private int     windDirection;  // 風向調節 (0: 自動, 1: 上, 2: 中, 3: 下)
    private int     windLevel;      // 風量調節 (0: 自動, 1: 弱, 2: 中, 3: 強)
    private int     onTimer;        // オンタイマ残り時間 (0: オンタイマ停止中)
    private int     offTimer;       // オフタイマ残り時間 (0: オフタイマ停止中)

    // フィールドの getter/setter
    public boolean getPower() {
```

```
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.2 カーテン

	トーソー	リ・サイト	ナビオ	松装	トヨタ車体
開閉	✓	✓	✓	✓	✓
タイマ	✓	–	–	–	✓
人体感知センサ	✓	–	–	–	–

```
// 標準カーテンサービスインタフェース
public interface CurtainInterface {
    int open();                 // 開
    int close();                // 閉

    CurtainStatus getStatus();  // 状態取得
}

// カーテン状態クラス
public class CurtainStatus {
    private boolean open;  // 開閉 (true: 開, false: 閉)

    // フィールドの getter/setter
    public boolean getOpen() {
        return open;
    }
    public void setOpen(boolean open) {
        this.open = open;
    }
}
```

A.3 ブラインド

	ニチベイ	トーソー	ヨコタ
開閉	✓	✓	✓
停止	✓	✓	✓
グループ操作	✓	✓	–
一斉操作	✓	✓	–
ルーバ角度調節	✓	–	✓

```
// 標準ブラインドサービスインタフェース
public interface BlindInterface {
    int open();               // 開
    int close();              // 閉
    int stop();               // 停止

    BlindStatus getStatus();  // 状態取得
}

// ブラインド状態クラス
public class BlindStatus {
    private boolean open;    // 開閉 (true: 開, false: 閉)
    private boolean moving;  // 動作状態 (true: 動作中, false: 停止中)

    // フィールドの getter/setter
    public boolean getOpen() {
        return open;
    }
    public void setOpen(boolean open) {
        this.open = open;
    }
    ...
}
```

A.4 換気扇

	三菱	東芝	高須産業
電源	✓	✓	✓
タイマ	✓	–	✓
風量調節	–	✓	✓
自動運転	–	–	✓

```
// 標準換気扇サービスインタフェース
public interface VentilatorInterface {
    int on();                      // 電源入
    int off();                     // 電源切

    VentilatorStatus getStatus();  // 状態取得
}

// 換気扇状態クラス
public class VentilatorStatus {
    private boolean power;  // 電源 (true: 入, false: 切)

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
```

```
        this.power = power;
    }
}
```

A.5 空気清浄機

	三洋	日立	ツインバード	セラヴィ	象印
電源	✓	✓	✓	✓	✓
風量調節	✓	✓	✓	✓	✓
急速運転	✓	✓	–	–	–
タイマ	✓	–	✓	–	✓
花粉モード	✓	✓	–	–	✓
タバコモード	–	✓	–	–	–
マイナスイオン	–	–	✓	–	–

```
// 標準空気清浄機サービスインタフェース
public interface AirCleanerInterface {
    int on();                              // 電源入
    int off();                             // 電源切
    int changeWindLevel(int windLevel); // 風量調節 (1: 弱, 2: 強)

    AirCleanerStatus getStatus();       // 状態取得
}

// 空気清浄機状態クラス
public class AirCleanerStatus {
    private boolean power;      // 電源 (true: 入, false: 切)
    private int     windLevel;  // 風量 (1: 弱, 2: 強)

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.6 照明

	東芝	丸善	日立	松下	日本電気	キシマ
電源	✓	✓	✓	✓	✓	✓
明るさ調節	✓	✓	✓	✓	✓	✓
可変色間接光	–	–	✓	–	–	–

```
// 標準照明サービスインタフェース
public interface LightInterface {
    int on();                 // 電源入
    int off();                // 電源切
    int changeBrightness();   // 明るさ調節 (1: 暗, 2: 中, 3: 明)

    LightStatus getStatus();  // 状態取得
}

// 照明状態クラス
public class LightStatus {
    private boolean power;       // 電源 (true: 入, false: 切)
    private int     brightness;  // 明るさ (1: 暗, 2: 中, 3: 明)

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.7 扇風機

	松下	三洋	三菱	東芝	森田	千住	山善	トヨトミ	日立
電源	✓	✓	✓	✓	✓	✓	✓	✓	✓
風量調節	✓	✓	✓	✓	✓	✓	✓	✓	✓
タイマ	✓	✓	✓	✓	✓	✓	✓	✓	✓
首振り	✓	✓	✓	✓	✓	✓	✓	✓	✓
リズム風	✓	✓	✓	✓	✓	✓	✓	✓	✓
マイナスイオン	✓	✓	–	✓	✓	–	–	–	–
チャイルドロック	✓	–	–	–	–	–	–	–	–

```
// 標準扇風機サービスインタフェース
public interface FanInterface {
    int on();                              // 電源入
    int off();                             // 電源切
    int changeWindLevel(int windLevel);    // 風量調節 (1: 弱, 2: 中, 3: 強)
    int startOffTimer(int hour);           // オフタイマ
    int cancelOffTimer();                  // オフタイマ解除
    int startSwing();                      // 首振り
    int stopSwing();                       // 首振り停止
    int startRhythmWind();                 // リズム風
    int stopRhythmWind();                  // リズム風停止

    FanStatus getStatus();                 // 状態取得
}
```

```
// 扇風機状態クラス
public class FanStatus {
    private boolean power;       // 電源 (true: 入, false: 切)
    private int     windLevel;   // 風量 (1: 弱, 2: 中, 3: 強)
    private int     offTimer;    // オフタイマ残り時間 (0: オフタイマ停止中)
    private boolean swing;       // 首振り
    private boolean rhythmWind;  // リズム風

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.8 テレビ

	日立	三菱	ビクター	船井	東芝	シャープ	ソニー	松下
電源	✓	✓	✓	✓	✓	✓	✓	✓
画面表示	✓	✓	✓	✓	✓	✓	✓	✓
音量調節	✓	✓	✓	✓	✓	✓	✓	✓
消音	✓	✓	✓	✓	✓	✓	✓	✓
副音声	✓	✓	✓	✓	✓	✓	✓	✓
チャンネル変更	✓	✓	✓	✓	✓	✓	✓	✓
入力切換	✓	✓	✓	✓	✓	✓	✓	✓
BS 放送視聴	–	✓	–	–	–	✓	–	✓
オフタイマ	✓	✓	✓	✓	✓	✓	✓	✓
オンタイマ	✓	–	–	–	–	✓	–	–
電力量調整	–	✓	–	–	–	–	✓	–
センサ節電	–	✓	–	–	–	–	–	–
字幕	✓	✓	–	–	–	–	–	✓
番組表	–	✓	–	–	–	–	–	✓
番組内容	–	✓	–	–	–	–	–	✓
チャンネル設定	✓	✓	✓	✓	✓	✓	✓	✓
映像設定	✓	✓	✓	✓	✓	✓	✓	✓
音声設定	✓	✓	✓	–	✓	✓	✓	✓
時刻設定	✓	✓	–	–	–	✓	–	–

```
// 標準テレビサービスインタフェース
public interface TVInterface {
    int on();                            // 電源入
    int off();                           // 電源切
    int changeVolume(int volume);        // 音量調節 (0: 最小, 100: 最大)
    int mute();                          // 消音
    int unmute();                        // 消音解除
    int changeVoiceMode(int voiceMode);  // 副音声 (0: 主, 1: 副, 3: 主+副)
```

```
    int changeChannel(int channel);      // チャンネル変更
    int showChannelIndicator();          // チャンネル表示
    int hideChannelIndicator();          // チャンネル表示解除
    int changeInput(int input):          // 入力切替 (0: テレビ, 1: HDD レコーダ, 2: ゲーム機, ...)
    int startOffTimer(int hour);         // オフタイマ
    int cancelOffTimer();                // オフタイマ解除

    TVStatus getStatus();                // 状態取得
}

// テレビ状態クラス
public class TVStatus {
    private boolean power;              // 電源 (true: 入, false: 切)
    private int     volume;             // 音量 (0: 最小, 100: 最大)
    private boolean mute;               // 消音
    private int     voiceMode;          // 副音声 (0: 主, 1: 副, 3: 主+副)
    private int     channel;            // チャンネル
    private boolean channelIndivator;   // チャンネル表示
    private int     input;              // 入力 (0: テレビ, 1: HDD レコーダ, 2: ゲーム機, ...)
    private int     offTimer;           // オフタイマ残り時間 (0: オフタイマ停止中)

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.9 HDD レコーダ

	東芝	シャープ	ソニー	日立	三菱	三洋
電源	✓	✓	✓	✓	✓	✓
再生	✓	✓	✓	✓	✓	✓
停止	✓	✓	✓	✓	✓	✓
送り	✓	✓	✓	✓	✓	✓
戻し	✓	✓	✓	✓	✓	✓
一時停止	✓	✓	✓	✓	✓	✓
録画	✓	✓	✓	✓	✓	✓
DVD トレイ開閉	✓	✓	✓	✓	✓	✓
チャンネル変更	✓	✓	✓	✓	✓	✓
HDD/DVD 切替	✓	✓	✓	✓	✓	✓
2 番組同時録画	–	✓	✓	–	✓	✓

```
// 標準 HDD レコーダサービスインタフェース
public interface HDDRecorderInterface {
    int on();                          // 電源入
```

```
    int off();                          // 電源切
    int play();                         // 再生
    int stop();                         // 停止
    int next();                         // 送り
    int previous();                     // 戻し
    int pause();                        // 一時停止
    int record();                       // 録画
    int openDVDTray();                  // DVD トレイ開
    int closeDVDTray();                 // DVD トレイ閉
    int changeChannel(int channel);     // チャンネル変更
    int changeMedia(int media);         // 記録媒体切換 (0: HDD, 1: DVD)

    HDDRecorderStatus getStatus();      // 状態取得
}

// HDD レコーダ状態クラス
public class HDDRecorderStatus {
    private boolean power;    // 電源 (true: 入, false: 切)
    private int     action;   // 動作 (0: 停止, 1: 再生, 2: 一時停止, 3: 録画)
    private boolean tray;     // DVD トレイ (true: 開, false: 閉)
    private int     channel;  // チャンネル
    private int     media;    // 記録媒体 (0: HDD, 1: DVD)

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.10 オーディオシステム

	ケンウッド	松下	ソニー
電源	✓	✓	✓
再生	✓	✓	✓
停止	✓	✓	✓
送り	✓	✓	✓
戻し	✓	✓	✓
CD	✓	✓	✓
MD	✓	✓	✓
FM/AM	✓	✓	✓
タイマ	✓	✓	✓
ウィークリータイマ	✓	–	–
リピート再生	✓	✓	✓
ランダム再生	✓	✓	✓

```
// 標準オーディオシステムサービスインタフェース
public interface AudioSystemInterface {
    int on();                       // 電源入
    int off();                      // 電源切
    int play();                     // 再生
    int stop();                     // 停止
    int next();                     // 送り
    int previous();                 // 戻し
    int pause();                    // 一時停止
    int record();                   // 録音
    int repeatAllTracks();          // 全曲リピート
    int repeatOneTrack();           // 一曲リピート
    int cancelRepeat();             // リピート解除
    int randomPlay();               // ランダム再生
    int cancelRandomPlay();         // ランダム再生解除
    int eject();                    // 取り出し
    int changeVolume(int volume);   // 音量調節 (0: 最小, 100: 最大)
    int startOnTimer(int hour);     // オンタイマ
    int cancelOnTimer();            // オンタイマ解除
    int startOffTimer(int hour);    // オフタイマ
    int cancelOffTimer();           // オフタイマ解除
    int changeSource(int source);   // 音源切替 (0: CD, 1: MD, 2: FM, 3: AM)

    AudioSystemStatus getStatus();  // 状態取得
}

// オーディオシステム状態クラス
public class AudioSystemStatus {
    private boolean power;     // 電源 (true: 入, false: 切)
    private int     action;    // 動作 (0: 停止, 1: 再生, 2: 一時停止, 3: 録音)
    private int     repeat;    // リピート (0: 解除, 1: 全曲, 2: 一曲)
    private boolean random;    // ランダム再生
    private int     volume;    // 音量 (0: 最小, 100: 最大)
    private int     onTimer;   // オンタイマ残り時間 (0: オンタイマ停止中)
    private int     offTimer;  // オフタイマ残り時間 (0: オフタイマ停止中)
    private int     source;    // 音源 (0: CD, 1: MD, 2: FM, 3: AM)

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
    ...
}
```

A.11 電話

	パイオニア	松下	三洋	シャープ
通話	✓	✓	✓	✓
留守番電話	✓	✓	✓	✓
ナンバーディスプレイ	✓	✓	✓	✓
着信拒否	✓	✓	✓	✓
オンフック	✓	✓	✓	✓
保留	✓	✓	✓	✓
フラッシュ	✓	–	✓	✓
ドアホン	–	✓	–	–

```
// 標準電話サービスインタフェース
public interface PhoneInterface {
    int call(string number);    // 発信
    int receive();              // 受信
    int reject();               // 着信拒否
    int disconnect();           // 通話終了
    int speakerOn();            // オンフック
    int speakerOff();           // オンフック解除
    int hold();                 // 保留
    int unhold();               // 保留解除
    int answeringMachineOn();   // 留守番電話
    int answeringMachineOff();  // 留守番電話解除

    PhoneStatus getStatus();    // 状態取得
}

// 電話状態クラス
public class PhoneStatus {
    private boolean connetcion;         // 通話
    private boolean ringing;            // 呼び出し
    private string  number;             // 相手先電話番号（待ち受け時は空文字列）
    private boolean speaker;            // オンフック
    private boolean holding;            // 保留
    private boolean answeringMachine;   // 留守番電話
    private int     message;            // 伝言件数

    // フィールドの getter/setter
    public boolean getConnection() {
        return connection;
    }
    public void setConnection(boolean connection) {
        this.connection = connection;
    }
    ...
}
```

A.12 テレビ電話

```
// 標準テレビ電話サービスインタフェース
public interface TVPhoneInterface extends PhoneInterface {
    int cameraOn();             // 映像入
    int cameraOff();            // 映像切

    TVPhoneStatus getStatus();  // 状態取得
}

// テレビ電話状態クラス
public class TVPhoneStatus extends PhoneStatus {
    private boolean camera;  // 映像

    // フィールドの getter/setter
    public boolean getCamera() {
        return camera;
    }
    public void setCamera(boolean camera) {
        this.camera = camera;
    }
}
```

A.13 ファクシミリ

	パイオニア	松下	三洋	シャープ
通話	✓	✓	✓	✓
留守番電話	✓	✓	✓	✓
ナンバーディスプレイ	✓	✓	✓	✓
着信拒否	✓	✓	✓	✓
オンフック	✓	✓	✓	✓
保留	✓	✓	✓	✓
フラッシュ	✓	–	✓	✓
ドアホン	–	✓	–	–

```
// 標準ファクシミリサービスインタフェース
public interface FaxInterface extends PhoneInterface {
    int sendFax();                      // ファクシミリ送信
    int receiveFax();                   // ファクシミリ受信
    int copy();                         // コピー
    int printMemory(int memoryNumber);  // メモリ代行受信内容印刷

    FaxStatus getStatus();              // 状態取得
}

// ファクシミリ状態クラス
public class FaxStatus extends PhoneStatus {
    private boolean faxMode;  // ファクシミリモード
    private int     memory;   // メモリ代行受信件数
```

```
    // フィールドの getter/setter
    public boolean getFaxMode() {
        return faxMode;
    }
    public void setFaxMode(boolean faxMode) {
        this.faxMode = faxMode;
    }
    ...
}
```

A.14 ガス栓

	m2m ジャパン
開く	–
閉じる	✓

```
// 標準ガス栓サービスインタフェース
public interface GasCockInterface {
    int shut();                 // 閉じる

    GasCockStatus getStatus();  // 状態取得
}

// ガス栓状態クラス
public class GasCockStatus {
    private boolean open;  // 開閉

    // フィールドの getter/setter
    public boolean getOpen() {
        return open;
    }
    public void setOpen(boolean open) {
        this.open = open;
    }
}
```

A.15 コンセント

	オーム電機	フカダック	野田屋電機
入切	✓	✓	✓

```
// 標準コンセントサービスインタフェース
public interface OutletInterface {
    int on();                   // 入
    int off();                  // 切
```

```
    OutletStatus getStatus();  // 状態取得
}

// コンセント状態クラス
public class OutletStatus {
    private boolean power;  // 入切

    // フィールドの getter/setter
    public boolean getPower() {
        return power;
    }
    public void setPower(boolean power) {
        this.power = power;
    }
}
```

A.16 扉

```
// 標準扉サービスインタフェース
public interface DoorInterface {
    int lock();              // 施錠
    int unlock();            // 開錠

    DoorStatus getStatus();  // 状態取得
}

// 扉状態クラス
public class DoorStatus {
    private boolean open;  // 開閉
    private boolean lock;  // 鍵

    // フィールドの getter/setter
    public boolean getOpen() {
        return open;
    }
    public void setOpen(boolean open) {
        this.open = open;
    }
}
```

A.17 火災報知機

	ノア	ニッタン	松下
煙感知	✓	✓	✓
音声警告	–	✓	–
電池寿命警告	✓	✓	✓

```
// 標準火災報知機サービスインタフェース
public interface FireAlarmInterface {
    FireAlarmStatus getStatus();  // 状態取得
}

// 火災報知機状態クラス
public class FireAlarmStatus {
    private boolean battery;  // 電池 (false: 電池寿命)
    private boolean fire;     // 火災

    // フィールドの getter/setter
    public boolean getBattery() {
        return battery;
    }
    public void setBattery(boolean battery) {
        this.battery = battery;
    }
}
```

A.18 非常ボタン

	MENIX
発報	✓

```
// 標準非常ボタンサービスインタフェース
public interface EmergencyButtonInterface {
    EmergencyButtonStatus getStatus();  // 状態取得
}

// 非常ボタン状態クラス
public class EmergencyButtonStatus {
    private boolean emergency;  // 非常

    // フィールドの getter/setter
    public boolean getEmergency() {
        return emergency;
    }
    public void setEmergency(boolean emergency) {
        this.emergency = emergency;
    }
}
```

備考

テレビ電話，ならびに扉については，適当な機種が見つからなかったため，妥当と考えられる標準家電サービスインタフェース，ならびに家電状態クラスの定義を行った．