# Webプロトコルを用いたホームコンピューティングミドルウェアの統合

上野乃毅[†1] 中島達夫[†2]
佐藤一郎[†3] 副島康太[†4]

実用的なアプリケーションを構築する場面で生じる要求に応えるために，ミドルウェアは個々のアプリケーションに特有の特徴を備えるようカスタマイズできることが望ましい．Webプロトコルを介して様々なアプライアンスを統合するミドルウェアを開発した．本稿では，開発したミドルウェアがホームアプライアンスの統合に十分な機能を持つことを示したうえで，将来のユビキタスコンピューティング環境の基盤としての展開を見据えたアーキテクチャの設計を解説する．

# Web Based Middleware for Integrating Home Appliances

Daiki Ueno,[†1] Tatsuo Nakajima,[†2] Ichiro Satoh[†3]
and Kouta Soejima[†4]

We have developed a universal middleware that allows various appliances to be incorporated together. It is desirable for such a middleware to be customized for several applications to preserve their advantages. In this paper, we will show that our middleware is enough to integrate applications in home computing environments and give some experiences with the design of the architecture on which we are aiming at constructing ubiquitous computing environments.

## 1. はじめに

チップの小型化・高集積化にともない，TVやVCRをはじめとする様々なアプライアンスにコンピュータが内蔵されるようになった．これらのアプライアンスの中にはネットワークコントローラを搭載したものが少なくない．

ネットワーク機能を利用してアプライアンスを相互に連携させることができれば，ホームコンピューティング環境はより豊かなものになると期待される．個々のアプライアンスの機能性を補うだけではなく，実用的なアプリケーションを構築できるだろう．

多くのアプライアンスを接続するための基盤として，ATM（Asynchronous Transfer Mode）ネットワーク，IEEE1394[1)]，Bluetooth[2)]，VIA[3)]といった多くのネットワークシステムが開発されている．同様に，研究分野ではPEN[4)]や，Networked Surfaces[5)]，CLAN[6)]といったものが提案されている．

ネットワーク機能を提供するアプライアンスのうち，現在市販されているものの多くはインターネットプロトコルに対応している．このことから，将来のアプライアンスの接続の形態は，インターネットを直接に利用するものが主流になると予想される．

多くのアプライアンスを接続するコンピューティング環境は，ユビキタスコンピューティング環境と呼ばれ，1990年代初頭から研究されてきた．将来のアプライアンスのほとんどがインターネットプロトコルに対応するという予想が正しければ，ユビキタスコンピューティング環境についての従来の考え方は拡張されるべきである．また同時に，インターネット規模のユビキタスコンピューティング環境を実現するうえでの様々な問題に取り組む必要がある．

アプライアンス間の通信の基盤を提供するミドルウェアには，サン・マイクロシステムズのJini[7)]や，

†1 早稲田大学大学院理工学研究科
Graduate School of Science & Engineering, Waseda University
†2 早稲田大学理工学部
School of Science & Engineering, Waseda University
†3 国立情報学研究所
The National Institute of Informatics
†4 富士通LSIソリューション株式会社
Fujitsu LSI Solution Limited

マイクロソフトの UPnP[8] のほかに，日本の家電メーカの提唱する HAVi[9] などがある．

これらのミドルウェアの主要な機能として，プラグアンドプレイ機能と位置透過性があげられる．プラグアンドプレイ機能により，アプライアンスをネットワークに接続した時点からネットワークを介した利用が可能となる．位置透過性により，アプライアンスに割り当てられた IP アドレスやポート番号などの物理的な位置情報を参照することなくアプライアンスを特定することができる．

これらのミドルウェアのプロトコルには互換性がなく，それぞれ異なるミドルウェアに限定的に対応したアプライアンスどうしを相互に接続することはできない．ミドルウェア間の非互換性が多数のアプライアンスを連携させるうえでの大きな障壁となっている．

本稿では，Web プロトコルを用いて既存のミドルウェアネットワークを連絡し，多種多様なアプライアンスを接続する新たなミドルウェアを提案する．

論文の構成は以下のようになっている．2 章では研究の目的を述べる．4 章では，アーキテクチャの主要な概念であるオーバーレイネットワークの概要を紹介し，システムに要求される機能の分析から設計を述べる．5 章では実装の詳細を述べる．6 章では，我々の実装したミドルウェアを，可用性と拡張可能性の両面から評価・議論する．

## 2. 研究の目的

システムの観点から，インターネット規模のユビキタスコンピューティング環境を実現するために，3 段階の目標を定めている．

第 1 段階では，すでに普及したホームコンピューティングミドルウェアの間で相互接続性を確保する．これらのミドルウェアは高い抽象度を持つ API（アプリケーションプログラミングインタフェース）を提供しているが，それゆえに複数のミドルウェアに対応したアプライアンスの実装は困難となっている．ミドルウェアの共通部分を明らかにし，ホームコンピューティング環境の実現に必要な機能の洗い出しがこの段階の目標である．

第 2 段階では，第 1 段階の知見を基に，ホームコンピューティング環境での使用を想定して，軽量で標準化されたミドルウェアを開発する．開発したミドルウェアの主要なコンポーネントを小型のハードウェアに組み込むことで，大規模なユビキタスコンピューティング環境を構築するための基盤として広く普及・展開を図る．

第 3 段階では，ホームコンピューティング環境に新たな機能を系統的に組み込む手段を提供する．コンテキストアウェアネスやデバイスの自動設定などの先進的な機能をミドルウェアの実装に変更を加えることなく追加できるようにフレームワークとして実現する．

## 3. 関 連 研 究

### 3.1 異種ミドルウェアの相互運用

Jini と UPnP を相互接続する研究には，双方のネットワークに仮想的なサービス生成する試み[10] がある．しかしながら，新たなプロトコルへの対応は考慮されていない．

VNA（Virtual Networked Appliances）[11] では，サービスの制御に用いる低位のプロトコル（RTP，RTSP，SMTP，HTTP など）を取り換え可能にする柔軟な機構を用意している．しかしながら，ミドルウェアの主要な機能には，このほかにアプライアンスの参加・離脱管理やイベントの処理などがあり，それぞれ連携する別種のアプリケーションプロトコルで実現されていることが多い．たとえば，UPnP では，イベントの管理に GENA（General Event Notification Architecture）[12] を，サービスの制御には SOAP（Simple Object Access Protocol）[13] を利用している．このため，単純に低位のプロトコルを変更可能にするだけでは，ミドルウェアの相互運用性は達成できない．

### 3.2 アプライアンス制御のための標準プロトコル

標準化されたインターネットプロトコルを利用してアプライアンスを遠隔操作する研究には，中間プロトコルに SIP（Session Initiation Protocol）[14] を採用した手法[15] がある．SIP はマルチメディアセッションを扱うアプリケーションレベルのプロトコルである．

HTTP に比較すると広く普及しているとはいいにくいが，主な優位点としては，イベントやストリームメディアの扱いがプロトコルとして用意されていることがあげられる．また，移動透過性を保証するための手段も提供されている．

### 3.3 サービスの統合

また，複数のアプライアンスで提供されるサービスを統合して，1 つの論理的なアプライアンスとして利用可能にする研究には，VNA や uBlocks[16] などがある．しかしながら，それぞれ単一のミドルウェアプロトコルのみを扱っており，既存の複数のミドルウェア間の相互運用性は考慮されていない．

### 3.4 オーバーレイネットワーク

INS（Intentional Naming System）[17] は，インター

ネット上のリソースを特性情報から発見可能にする研究である．名前解決要求はリゾルバのネットワークを経由するため，一種のオーバーレイネットワークと見なすことができる．

本システムでは，名前解決はすべてゲートウェイコンポーネント（5 章）が責任を負い，ゲートウェイコンポーネントは固定的に配置されるが，INS ではリゾルバのネットワークは動的に構成され，名前解決要求は適切にルーティングされる．

## 4. 設　　計

### 4.1 相互接続の難しさ

異なるミドルウェアに対応したアプライアンスを相互に接続するためには，プロトコルの変換が必要となる．しかしながら，この変換は簡単ではない．主な原因は 2 つある．

1 つは，プラグアンドプレイ機能を備えたミドルウェアのプロトコルは複数のプロトコルの集合であることが多く，互いのプロトコルによる通信の状況を把握しなければならないことである．このため，プロトコル単位の変換だけでは十分ではなく，ネットワークの状態を管理しなければならない．

もう 1 つは，既存のミドルウェアにおけるアプライアンスの抽象は，それぞれ大きく異なることである．たとえば，Jini にはアプライアンスとサービスを区別する抽象は存在しないが，UPnP や HAVi には存在する．UPnP ではアプライアンスがサービスを内包する単純なモデルに従うが，HAVi では提供されるサービスを検索のキーとして，ネットワーク上のアプライアンスを特定することができる．

このようなことから，既存のミドルウェアの相互接続を支援する新たなミドルウェアが必要となる．

### 4.2 アーキテクチャ

アーキテクチャには，アプリケーションレベル仮想オーバーレイネットワークを採用した．オーバレイネットワークは，上位レベルのネットワークをサポートするために，ベースネットワーク上に構築するネットワークであると定義できる．アプリケーションレベル仮想オーバーレイネットワークは，オーバレイネットワークの概念を拡張したものであり，アプリケーションごとにカスタマイズ可能な抽象的なプロパティをソフトウェアのレベルで追加できるようにする．この考えはアクティブネットワーク[18] に似ている．アクティブネットワークでは，ネットワークから供給されるプロパティや，すでに存在しているプロパティを変更することなく，アプリケーションに特有のコードを追加するために既存のネットワークをカスタマイズできる．

同様に，アプリケーションレベル仮想オーバーレイネットワークは，既存のネットワークシステムやすでに接続されているアプリケーションに影響を与えることなく，それぞれのアプリケーションのために容易にカスタマイズできる．このため，我々のアプローチは，直接的にインターネットプロトコルを拡張するアプローチよりも現実的なものといえる．

アプリケーションレベル仮想オーバーレイネットワークは，インターネット上の多くの問題を解決するためにしばしば適用されており，新しい概念ではない．しかしながら，従来の研究では，ネットワーク化されたアプライアンスの統合のためにアプリケーションレベル仮想オーバーレイネットワークを応用した例は存在しない．我々は，アプリケーションレベル仮想オーバーレイネットワークの概念を最大限活用することを目指した．

### 4.3 中間プロトコル

インターネット規模の運用形態を視野に入れると，中間プロトコルには相互運用可能な標準プロトコルを採用すべきである．本システムでは，オーバーレイネットワークの中間プロトコルとして HTTP を採用した．HTTP を採用した理由を以下に述べる．

**広く普及している**

HTTP はすでに十分に普及しており，専用のクライアントを用意しなくても，身近な Web ブラウザを介して容易に扱うことができる．近年，SIP 対応の電話機などが市販されているが，すでに広く普及した HTTP と比較して手軽とはいいにくい．

また，Jini や UPnP や HAVi といったミドルウェアには対応していないが，HTTP を通じて制御できる，廉価なアプライアンスも多く市販されている．これらのアプライアンスとの混在により新たなアプリケーションを考えることが可能であろう．

**軽量である**

5 章で述べるように，本システムのプロトコルでは HTTP のメッセージボディに含まれる情報をいっさい利用しない．このため，十分に軽量に実装できるばかりではなく，携帯電話に搭載された低機能な Web ブラウザからも操作が可能である．また，この特長を活かし，遠隔地のアプライアンスと連携した Macromedia Flash のアプリケーションを構築することも可能である．

**ユーザインタフェースをカスタマイズできる**

近年，Web アプリケーションにおけるセッション管理は標準的な技術となりつつあり，これを用いること

**表 1** 提供する機能
Table 1 Offered functions.

| 基本サービス | 補助サービス |
|---|---|
| アプライアンスの名前付け | スタブサービスの活性化 |
| 参加・離脱管理 | |
| プロトコル変換・転送 | 特性情報管理 |

で，利用者の好みを反映したユーザインタフェースを容易に実装できる．

**十分なセキュリティ機能が組み込まれている**

認証や TLS による通信路の暗号化[19] などのセキュリティ機能がプロトコルに組み込まれていることも利点である．

### 4.4 機能の選定

ホームコンピューティング環境での利用を想定すると，ミドルウェアは軽量であることが望ましい．このため，既存のミドルウェアで標準的に提供される機能のうち，必要最低限の機能を注意深く選別する必要があった．提供する機能の一覧を**表 1** に示す．

大規模なユビキタスコンピューティング環境では，多くのアプライアンスの中から好ましいものを特定するために労力が払われる．このような場面で，アプライアンスの物理的な位置情報を利用者に直接指定させるのは現実的ではない．

開発したミドルウェアでは，特にアプライアンスの名前づけの手法を工夫することにした．レイトバインディングに基づくアプライアンスの名前づけの仕組みを提供することで，特性情報の指定だけでアプライアンスを特定できるようになった．

また，プラグアンドプレイを実現するためには必須の機能として，アプライアンスの参加・離脱を管理する仕組みを用意した．最後に，アプライアンスで提供されるサービスを呼び出すために中間プロトコルとミドルウェアプロトコルを相互に変換する枠組みを用意した．

これらの基本的なサービスに加え，既存のミドルウェアネットワークに対応したアプライアンスやクライアントとの相互接続性を確保するために，補助的な 2 つの機能を取り入れる必要が生じた．1 つは，スタブの透過的な活性化機構である．これは，クライアント側のミドルウェアネットワーク上に，クライアントからの制御リクエストを中間プロトコルに変換・転送するスタブを必要に応じて起動する仕組みである．もう 1 つは，アプライアンスで提供されるサービスに関連する様々な情報を管理する仕組みである．これにより，別のミドルウェアネットワークから，アプライアンスで提供されるサービスの特性を知ることが可能となる．

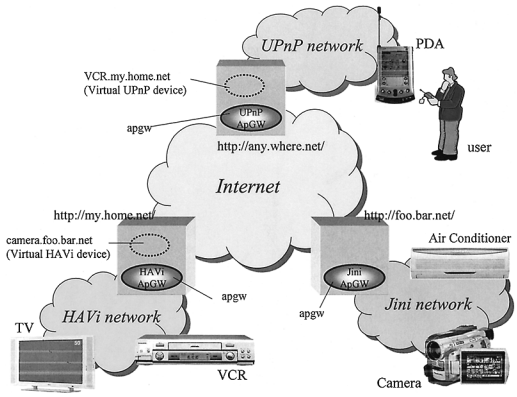


**図 1** 運用例
Fig. 1 Typical scenario.

### 4.5 運 用 形 態

我々のミドルウェアの実際の運用例を**図 1** に示す．この例では，Jini，UPnP，HAVi の各ミドルウェアに対応したアプライアンスを互いに操作可能にするため，各ミドルウェアネットワーク上にゲートウェイを配置している．UPnP のクライアントプロトコルに対応した PDA を持った利用者は，UPnP ネットワークに設置されたゲートウェイ，および Jini や HAVi ネットワーク上に配置されたゲートウェイを経由してアプライアンスを操作する．

## 5. 実 装

我々のアーキテクチャは，主に以下の各コンポーネントによって構成される．

- ゲートウェイコンポーネント
  - Web サーバ
  - プロトコル変換
  - ミドルウェアネットワークの構成管理
- ブートストラップサービスコンポーネント
  - スタブの透過的な活性化
  - アプライアンスの参加・離脱の管理
- エミュレーションコンポーネント
  - 各ミドルウェアに対応したアプライアンス（Jini，UPnP，HAVi）
  - Web ベースのアプライアンスエミュレータ（X10，クロッサム 2+）

ゲートウェイコンポーネントは Jini や UPnP といったミドルウェアネットワーク上に配置され，主に中間プロトコルである HTTP からミドルウェアのプロトコ

ルへの橋渡しの役割を果たす．ゲートウェイコンポーネントはWebサーバを内包し，次節で解説する拡張されたURLを解釈する．URLの一部として与えられた制御リクエストは適切に変換され，アプライアンスに転送される．また，公開するアプライアンスの限定や，検索の振舞いなど，Web上からミドルウェアネットワークの構成の管理を可能にする．

ブートストラップサービスコンポーネントは，従来のミドルウェアに対応したクライアントとの互換性を保つ役割を担う．ミドルウェアネットワーク上に代理となる仮想的なサービスを適宜生成することで，ネイティブのミドルウェアプロトコルによるアプライアンスの遠隔制御を可能にする．また，アプライアンスの高度な検索を実現するために，参加・離脱に追従する機構も用意した．

エミュレーションコンポーネントは，評価のためのアプライアンスの実装を提供する．各ミドルウェアに対応したアプライアンスだけではなく，直接Webから制御できるアプライアンスも実装した．

以降では，我々のミドルウェアを構成する各コンポーネントにおいて，特に重要なモジュールを順を追って解説する．初めに，ゲートウェイからアプライアンスを特定するための手法を解説し，名前づけの根拠となるアプライアンスの機能性の記述に言及する．

### 5.1 特性情報によるアプライアンスの特定

物理的な位置を指定することなく，ミドルウェアネットワークに接続されているアプライアンスを特定するために，新たな名前づけの仕組みを用意する必要があった．ヒントとなる特性情報を与えることにより，利用者にとって好ましいアプライアンスを特定できる仕組みが望まれる．我々のシステムではURL[20]の一部にアプライアンスの特性情報を埋め込む記法を採用した．

例として，会議室にあるライトの機能をもったアプライアンスを特定するには，次のように指定する．

```
http://somewhere.example.com?
  ?location=conferenceroom&?function=light
```

検索式には“?”を接頭し，特性名とパターンを“=”を挟んで与える．単一の検索式を“&”で区切って連ねることで，複数の検索式を指定することができ，段階的な絞り込みが可能となる．

このような単純な検索に加え，特定の時点におけるアプライアンスの状態を知るために，新たな記法を導入した．会議室にあるライトが点灯しているか否かを調べるには，“!”に続いて特性名を指定する．

```
http://somewhere.example.com?
  ?location=conferenceroom&?function=light&
  !power
```

上記のURLを含むHTTPリクエストがゲートウェイコンポーネントで処理されると，HTTPのリダイレクト機能により以下のURLに転送される．これにより，利用者は会議室のライトが点灯していないことを知ることができる．

```
http://somewhere.example.com?
  ?location=conferenceroom&?function=light&
  ?power=OFF
```

リダイレクト先のURLは新たな検索式であることから，次回の検索にそのまま利用することができる．我々のアーキテクチャでは，同様にURLによる指定だけでアプライアンスの制御も行えるよう，この仕組みをさらに拡張している．

送出する制御リクエストには，“!”を接頭し，コマンド名と引数を“=”を挟んで続ける．先程特定したライトを点灯させるためには，検索結果のURLに“&”を後続し，次のように記述する．

```
http://somewhere.example.com?
  ?location=conferenceroom&?function=light&
  ?power=OFF&!power=ON
```

上記のURLを含むHTTPリクエストがゲートウェイコンポーネントで処理され，適切にプロトコルの変換が行われてライトが点灯すると，HTTPのリダイレクト機能により以下のURLに転送される．このようにして，利用者はHTTPを通じて会議室のライトを操作することができる．

```
http://somewhere.example.com?
  ?location=conferenceroom&?function=light&
  ?power=ON
```

アプライアンスの状態検査と同様に，制御リクエストの実行結果は，次回の検索式の一部として継続的に利用することができる．HTTPのリダイレクト機能を利用することにより，制御リクエストの送出を自然なセッション管理の一部と見なすことも可能となる．また，例示した記法に加え，複数の制御リクエストを

```
path           =   search-part
                   ["&" command-part]
search-part    =   search *("&" search)
search         =   "%3F" pair ;%3F is ?
command-part   =   command *("&" command)
command        =   "!" name ["=" value]
name           =   <HEX escaped string>
value          =   <HEX escaped string>
```

**図 2** 名前づけと制御リクエストの構文
Fig. 2 A syntax for naming of appliances.

“&” で区切って与えることで，一括して送信することも可能である．この特徴を活用することで，利用者と親和性の高いユーザインタフェースを容易に実現することができる．ABNF [21] で表現した記法の構文を**図 2** に掲載する．

### 5.2 アプライアンスの機能性記述

制御リクエストを送出する時点で，利用者はアプライアンスの提供するサービスを把握している必要がある．しかしながら，すべてのアプライアンスの詳細な知識をあらかじめ利用者に期待するのは現実的ではない．このため，ミドルウェアネットワークの外部から任意のアプライアンスの機能性を参照できる必要がある．

現在の実装では，ゲートウェイに対して空の制御リクエストを送出することで，HTTP のレスポンス中にアプライアンスの持つ機能の一覧が送出される．

アプライアンスの機能性の記述には UPnP で使用されるデバイス記述用スキーマと，サービス記述用スキーマ（SCPD）を採用した．これらは**図 3** に示すような XML 形式のファイルである．

UPnP での利用上，記述は整形式であればよく，要素の曖昧性が許容されている [22]．これを利用することで，機器の所有者や物理的なロケーション情報などの様々な特性情報を埋め込むことが可能である．先にあげた記述の例では，`device` 要素の直下に，“location” 特性および “function” 特性を指示する 2 つの `property` 要素がある．

### 5.3 検索アルゴリズム

Jini，UPnP，HAVi には組み込みの検索機能が備わっているが，検索に利用可能な語彙が共通ではないため，直接には利用できない．

また，UPnP のサービス発見プロトコルである SSDP の検索機能は，Jini や HAVi のルックアップサービスと比べて低機能であり，サービスの型による検索は可能だが，アプライアンスの特性情報による検索はできない．

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
...
  <device>
...
    <property name="location" value="conferenceroom"/>
    <property name="type" value="ceillight"/>
    <serviceList>
      <service>
        <property name="function" value="light"/>
...
        <SCPDURL>/xalSCPD.xml</SCPDURL>
      </service>
    </serviceList>
  </device>
</root>
```

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <actionList>
    <action>
      <name>SetDimLevel</name>
      <argumentList>
        <argument>
          <name>Dim</name>
          <relatedStateVariable>Dim</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>Dim</name>
    </stateVariable>
  </serviceStateTable>
</scpd>
```

**図 3** アプライアンスの機能性記述の例
Fig. 3 An example for an appliance’s service description.

そこで，アプライアンスの参加・離脱に追従し，公開された機能性記述をディレクトリサービスに保管する仕組みが必要となる．

アプライアンスの検索は複数回に分けて行われる．最初の段階では XPath 式による単純な文字列比較で，ディレクトリサービスに保管されたデバイス記述ファイルを検索する．

ディレクトリ中に該当するものがない場合には，事前に与えられた検索式間の依存情報を利用して検索式を展開し，再度 XPath による検索を行う．

例として，検索式 `?location=conferenceroom` に該当するものがないとする．ここで会議室が 4 階の 3 号室にあるという依存情報があれば，最初の検索式を `?floor=4&?room=3` と展開し，再度検索を行う．

依存情報の管理には階層構造化モデル [23] を用いている．アルゴリズムを**図 4** に示す．

### 5.4 スタブの活性化

我々のミドルウェアと，Jini や UPnP といったミドルウェアプロトコルとの併用を考えた場合，クライアントからの制御リクエストを中間プロトコルに変換・

転送するスタブが必要となる．Jini や UPnP などの従来のミドルウェアの多くはプラグアンドプレイ機能をサポートしており，オーバーレイネットワークに接続されたアプライアンスの構成が，静的に与えられて変化することがないと仮定するのは柔軟性に欠ける．アプライアンスの動的な参加・離脱に従うために，スタブの透過的な活性化の仕組みが必要となる．スタブの活性化のモデルを明確化し，統一しておくことで，将来現れるであろう新しいミドルウェアを我々のオーバーレイネットワークに適合させることが容易となる．

我々のシステムにおけるスタブの活性化の流れを**図 5** に示す．この図では Jini および UPnP に対応したクライアントが，相異なるミドルウェアネットワーク上にあるアプライアンスに制御リクエストを送出するまでの各段階を表している．

スタブはミドルウェアネットワーク上のサービスとして振舞い，クライアントのネットワークに配置される．最初の段階では，制御対象のミドルウェアネットワークで利用可能なアプライアンスの一覧を取得する．次の段階では，目的のアプライアンスに対応するスタブを手元のミドルウェアネットワーク上に活性化する．

(1) $M = D + E$ ($D$: 隣接行列，$E$: 単位行列)
(2) 可達行列：$M' = M^k = M^{k-1} (k \geq 2)$
(3) 可達集合：$R(S_i) = \{S_j | M'(i,j) = 1\}$
(4) 先行集合：$A(S_i) = \{S_j | M'(j,i) = 1\}$
(5) 階層構造の決定：

```
for (k = 0; R(S) ≠ φ ; k++) {
  for (i = 0; i < n; i++)
    if ( R(Si) ∩ A(Si) = R(Si) )
    要素 Si は k レベル;
  R(S), A(S) から k レベルの要素を除去;
}
```

**図 4** 階層構造化モデルによる依存情報の管理
Fig. 4 Managing dependency information using a hierarchical structural model.

ここまでの処理はスタブ活性化サービスと呼ばれる特別なサービスを介して行われる．クライアントは自分の対応するミドルウェアのプロトコルだけを知っていればよく，HTTP や相手側ミドルウェアのプロトコルの詳細を知る必要はない．スタブが活性化された後は，クライアントは従来と同様の手法でスタブを介してアプライアンスを制御することができる．スタブはミドルウェアプロトコルを HTTP に変換し，アプリケーションレベル仮想オーバーレイネットワークを介して転送することで，アプライアンスの制御を肩代わりする．一連の処理で，クライアントはミドルウェアプロトコルのみを知っていればよい．

### 5.5 実装の詳細

ゲートウェイコンポーネントは Java サーブレットとして実装し，Linux 上の Tomcat 4.1.16 上で動作する．UPnP に対応したアプライアンスの実装には Intel UPnP SDK for Linux [24] を，SOAP クライアントには Apache Axis [25] を使用している．UPnP デバイスの参加・離脱の告知の管理には，専用の SSDP ライブラリを Java で実装した．

Jini に対応したクライアントおよびアプライアンスの実装にはサン・マイクロシステムズのリファレンス実装を使用した．

HAVi の実装には富士通 LSI ソリューションの FLS HAVi 1.0 を使用した．附属の dvController プログラムにより，IEEE1394 で接続した DV カメラの制御が可能である．

評価用の家電機器としては，X10 [26] のライトモジュールと，ハル・コーポレーションのプログラミング可能な赤外線リモコンのクロッサム 2+をそれぞれ制御可能とした．

プラグアンドプレイに対応したミドルウェアは単一のプロトコルに収まらないものが多い．このため，ソ

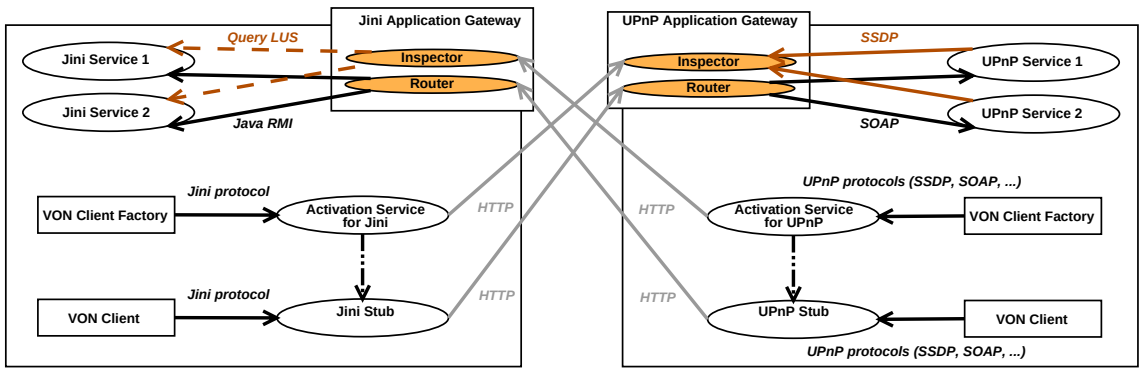


**図 5** スタブの活性化の流れ
Fig. 5 A flow of a stub's activation.

フトウェアの開発の場では，従来と異なる手法が有効であることが多い．たとえば，クライアントおよびアプライアンスの参加・サービスの利用・後処理といった，一連の流れを意識しつつ，テスト・デバッグを並行して進めることで，作業の効率を高めることができる．

アーキテクチャの動作検証のためのテストベッドを容易に構築するために，各ミドルウェアに対応したエミレーションコンポーネントを少ない手間で開発するための開発ツール群の開発も同時に行った．

## 6. 評価と議論

前章で述べたスタブの活性化の仕組みにより，ゲートウェイを介したミドルウェアネットワークは回数の制限なく多段化することができる．この特徴を利用してオーバヘッドを評価した．評価に用いた機材は以下のとおりである．

- クライアント PC（PentiumIII 1.2 GHz，1 GB）
- サーバ PC（Crusoe TM5800，867 MHz）
- 100 Mbps Ethernet

最初に UPnP ネットワーク上に評価用のエミュレーションサービスを起動しておく．同一のネットワークからネイティブのクライアントを用いてスタブを生成し，さらに生成されたスタブに対してスタブを生成し，サービスを呼び出すまでの往復時間を計測した．10 回の呼び出しにおいて総計で 4.235 秒であり，その内訳はスタブの活性化に 4.082 秒，URL による制御リクエストの送出から SOAP サービスの実装が呼び出されるまでに 0.053 秒であった．

スタブの活性化にほとんどの時間が費されていることを考えると，近隣のすでに活性化されたアプライアンスを自動的に選択したり，事前にスタブを活性化しておくことで，さらに起動時間を短縮できると期待できる．

接続されるアプライアンスの数が増加した将来のホームコンピューティング環境では，複数のアプライアンスを統合することで，複雑に多段化したサービス統合の形態を考えることができる．このような運用形態では，活性化が必要となるアプライアンスはある程度機械的に予測できるため，サービスの呼び出しについては十分な性能を発揮できると期待できる．

特性情報によるアプライアンスの特定に関しては，コンテクスト情報をいっさい利用しない現在の手法には限界があると考えている．現在の名前づけの問題点は，物理的に近い距離にあるアプライアンスを操作するのにも，多数の検索式を組み合わせねばならないことである．将来的にはクライアント側のコンテキスト情報を検索に利用する予定であり，現在実装を進めている．また，検索式の構文に関しても，現在の URL を拡張した記法は十分なものではない．任意の論理式を表現できるよう拡張する必要がある．

アプライアンスの機能性記述に関しても，特性情報を文字列として埋め込む方式は拡張性に欠ける．RDF などの拡張可能なメタデータ記述方式を採用することで，表現力を高めることが望まれる．

SOAP や WSDL [27] といった，Web ベースの標準プロトコルとの併用も検討中である．これにより，普及の兆しを見せている UPnP 対応の情報家電機器や，現在市販が始まっている Web ベースの情報家電を直接操作する際のオーバヘッドを減らすことができる．また，他の Web サービスとの連携も容易になると期待できる．

## 7. 今後の課題

本稿で提案したミドルウェアを広く普及させるためには，いくつかの課題について研究する必要がある．

### 7.1 小型のハードウェアへの組み込み

第 1 の課題は，軽量な実装という特長を生かし，実際に小型のハードウェアにゲートウェイコンポーネントを組み込むことである．これにより，次代のユビキタスコンピューティング環境の基本構成要素としての展開が期待できる．

### 7.2 ストリームメディアの取扱い

第 2 の課題は，複数のミドルウェアを接続することで実現される実用的なアプリケーションを構築することである．それぞれのアプライアンスの機能性に応じたストリームメディアの変換も望まれる．

### 7.3 イベントの取扱い

第 3 の課題は，アプライアンスからのイベントの能動的な通知である．ゲートウェイコンポーネントに対して HTTP でポーリングすることで実現可能であるが，狭い時間間隔でポーリングするのは効率的ではない．履歴から中間値を求めることにより，イベントの生じるタイミングを学習するなどの手法が有効であると考えられる．

### 7.4 オーバーレイネットワークの構成支援

第 4 の課題は，アプリケーションごとにカスタマイズされたアプリケーションレベル仮想ネットワークを系統的に構築する手法の模索である．アプリケーションごとに異なる構成のオーバーレイネットワークを構築することは現実的ではない．オーバーレイネットワークの構成を支援するツールキットの充実も重要であると考えられる．

### 7.5 高度なコンテキスト情報の活用

身近なホームコンピューティング環境への展開を考えると，より高度なコンテキスト情報の活用も望まれる．たとえば，利用者の好みによりアプライアンスを特定するためには，時間や温度など，様々な曖昧な情報からコンテキスト情報を獲得し，効率的に管理する仕組みが必要となる．

## 8. ま と め

本稿では，多種多様なアプライアンスを統合するためのミドルウェアについて解説した．開発したミドルウェアには次にあげる特長がある．第1に，異なるミドルウェアネットワークに属するアプライアンスを相互に接続できる．第2に，軽量であり，ネットワーク上の基盤サービスに依存しない．第3に，中間プロトコルにHTTPを採用していることから，Webから制御可能な安価なアプライアンスをも接続できる．また，他のWebサービスとの連携も容易である．

アーキテクチャにはアプリケーションレベル仮想オーバーレイネットワークの手法を採用した．各ミドルウェアネットワーク上にゲートウェイを配置することで，利用者の状況に応じて，アプライアンスを適切に選択する機構を提供することができた．

これにより，新たなミドルウェアへの対応が容易となり，ホームコンピューティング環境で要求される様々な機能を，既存のミドルウェアの実装に変更を加えることなく追加する手段を提供することができた．

## 参 考 文 献


1) IEEE Std 1394-1995: Standard for a High Performance Serial Bus.
2) Miller, B.A. and Bisdikian, C.: *Bluetooth Revealed*, Prentice Hall (2000).
3) von Eicken, T. and Vogels, W.: Evolution of the Virtual Interface Architecture, *IEEE Computer*, Vol.31, No.11 (1998).
4) Jones, A. and Hopper, A.: The Prototype Embedded Network (PEN), Technical Report, AT&T Laboratories, Cambridge (2000).
5) Scott, J., Hoffman, F., Addlesee, M., Mapp, G. and Hopper, A.: Networked Surfaces: A New Concept in Mobile Networking, *International Workshop on Mobile Computing, Systems and Applications* (2000).
6) Pope, S., Roberts, D., Riddoch, D., Mansley, K., Clarke, D., Mills, T. and Hopper, A.: CLAN Scalable High Performance User Level Networking, *IEEE Gigabit Networking Workshop GBN* (2001).
7) Sun Microsystems: The Community Resource for Jini™ Technology. http://www.jini.org
8) Microsoft: The Universal Plug and Play Forum. http://www.upnp.org
9) The HAVi Organization: HAVi: Home Audio Video Interoperability. http://www.havi.org
10) Allard, J., Chinta, V. and Gundala, S.: Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability, *2003 International Symposium on Applications and the Internet* (*SAINT 2003*) (2003).
11) Nakazawa, J., Tobe, Y. and Tokuda, H.: On Dynamic Service Integration in VNA Architecture, *IEICE Trans. Inf. & Syst.*, Vol.E00-A, No.1, (2001).
12) Cohen, J., Aggarwal, S. and Goland, Y.Y.: General event notification architecture base: Client to arbiter (Sep.2000). http://www.upnp.org/download/draft-cohen-gena-client-01.txt
13) Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
14) Handley, M., Schulzrinne, H., Schooler, E. and Rosenberg, J.: SIP: Session Initiation Protocol, RFC 2543 (Mar. 1999).
15) Moyer, S., Marples, D. and Tsang, S.: A Protocol for Wide-Area Secure Networked Appliance Communication, *IEEE Communications Magazine* (Oct. 2001).
16) 岩井将行，中澤 仁，徳田英幸：複数個インタフェースからの一貫性のある分散アプリケーション構築に関する研究，日本ソフトウェア科学会ソフトウェアシステム研究会SPAサマーワークショップ SPA-SUMMER 2002 論文集，8 (2002).
17) Adjie-Winoto, W., Schwartz, E., Balakrishnan, H. and Lilley, J.: The design and implementation of an intentional naming system, *17th ACM SOSP* (1999).
18) Tennenhouse, D., et al.: A Survey of Active Network Research, *IEEE Communication Magazine*, Vol.35, No.1 (1997).
19) Rescorla, E.: HTTP Over TLS, RFC 2818 (May. 2000).
20) Berners-Lee, T.: Universal Resource Identifiers in WWW, RFC 1630 (June 1994).
21) Crocker, D. and Overell, P.: Augmented BNF for Syntax Specifications: ABNF, RFC 2234 (Nov. 1997).
22) Goland, Y.Y. and Schlimmer, J.C.: draft-goland-fxpp-01.txt: Flexible XML Processing Profile (FXPP) (Jun. 2000). http://www.upnp.org/download/draft-goland-fxpp-01.txt.
23) Warfield, J.N.: Binary Matrices in System Modeling, *IEEE Trans. on Systems Man & Cy-*

*bernetics vSMC-3 n5* (1973).
24) Intel Corporation: Open Source UPnP SDK for Linux. http://upnp.sourceforge.net
25) The Apache Software Foundation: Apache Axis. http://xml.apache.org/axis/index.html
26) X-10 Technology and Resource Forum. http://www.x10.org/
27) Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

(平成 14 年 12 月 21 日受付)
(平成 15 年 4 月 16 日採録)

**上野　乃毅**

2001 年早稲田大学大学院理工学研究科修士課程（情報科学専攻）修了．現在，同大学大学院博士課程在学中．

**中島　達夫**（正会員）

早稲田大学コンピュータネットワーク工学科教授．専門はオペレーティングシステム，分散システム，ネットワークシステム，ユビキタスコンピューティング．日本ソフトウェア科学会，ACM，IEEE，USENIX 各会員．

**佐藤　一郎**（正会員）

1991 年慶應義塾大学理工学部電気工学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻後期博士課程修了，博士（工学）．1996 年お茶の水女子大学理学部情報科学科助手，1998 年同学科助教授．2001 年より国立情報学研究所助教授．このほか，1994〜1995 年 Rank Xerox Grenoble 研究所客員研究員．1999〜2001 年科学技術振興事業団さきがけ研究 21（「情報と知」領域）研究員．1996 年度情報処理学会論文賞，1999 年度同学会山下奨励賞，1998 年日本ソフトウェア科学会高橋奨励賞ほか受賞．分散システムの理論，プログラミング言語，ミドルウェア等の研究に従事．日本ソフトウェア科学会，電子情報通信学会，IEEE，ACM 各会員．

**副島　康太**（正会員）

1992 年東京都立科学技術大学電子システム工学科卒業．同年，日本鋼管株式会社電子デバイス本部綾瀬研究所所属．2000 年，富士通 LSI ソリューション株式会社ソリューション開発部所属．2000 年より 2001 年まで早稲田大学にて情報家電向けネットワークプロトコルの研究に従事．現在も同テーマに関する研究および開発を行っている．