REST に基づく異種スマート環境間のセキュアな連携基盤

岩崎陽平[†1]　榎堀優[†2]　藤原茂雄[†3]
田中宏一[†3]　西尾信彦[†4]　河口信夫[†1]

生活環境に様々なセンサや機器を埋め込み，スマート環境を構築する試みが多数なされているが，一般にスマート環境はその環境の中だけで閉じていることが多く，異なる組織により構築されたスマート環境（異種スマート環境）間で，機器やサービス間の連携を行うのは困難であった．本稿では，異種スマート環境間の連携のための要件を示し，それを満たすフレームワークを提案する．本フレームワークでは，シンプルかつ汎用的な枠組みを実現するために，各スマート環境の機器を REST に基づく Web サービスとして公開する．また，ゲストユーザに対してもスケーラブルできめ細かな権限管理を実現するために，ユーザ管理が不要なアクセス権限管理手法として，チケット認証方式を提案する．これは，イベント等で利用される紙のチケットから着想を得たものであり，ユーザが持つデジタル署名されたチケットのデータのみで，サービスの利用権限を判断する．本稿で提案した枠組みを用いれば，異種スマート環境間の連携を，シンプル・汎用的・セキュア・スケーラブルに実現できる．

RESTful and Secure Collaboration Framework for Heteroginious Smart Spaces

Yohei Iwasaki,[†1] Yu Enokibori,[†2] Shigeo Fujiwara,[†3]
Kouichi Tanaka,[†3] Nobuhiko Nishio[†4] and Nobuo Kawaguchi [†1]

It was difficult for a smart space to collaborate with devices and services in heteroginious smart spaces developed by other organizations. In this paper, we propose a collaboration framework for heteroginious smart spaces. To make our framework simple and wide-use, functions of devices and services in the smart space are published as a web service based on REST. Moreover, for scalability for the number of guest users and fine authority management, we propose a ticket authorization scheme, which is an access authorization method without user authentication. In this scheme, the system determine access authority by ticket data with digital signature submitted by a user. If you use our proposed framework, you can make collaboration between heteroginious smart spaces more simple, wide-use, secure and scalable.

1. はじめに

生活環境に様々なセンサや機器を埋め込み，スマート環境を構築する試みが多数なされており，ユビキタスコンピューティングの世界が現実のものとなりつつある．しかし，一般にスマート環境はその環境の中だけで閉じていることが多く，異なる組織により構築されたスマート環境（異種スマート環境）間で，機器やサービス間の連携を行うのは困難であった．

本稿では，異種スマート環境間の連携のための要件

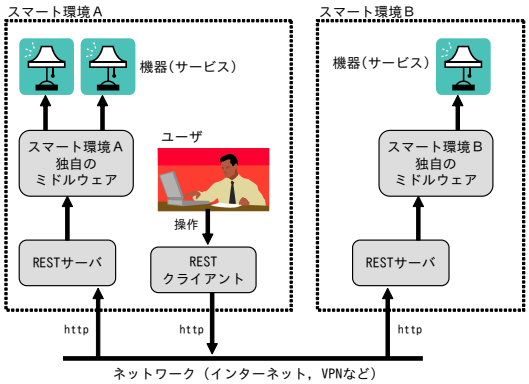

図 1　REST に基づく異種スマート環境間の連携
Fig. 1　REST-based collaboration between heteroginious smart spaces

†1 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
†2 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†3 株式会社内田洋行
Uchida Yoko Co.,Ltd.
†4 立命館大学情報理工学部
Department of Comuter Science, Ritsumeikan University

を示し，それを満たすフレームワークを提案する．異種スマート環境間の連携のためには，シンプルで汎用的なインタフェース，ゲストユーザを考慮した権限管理，スケーラビリティなどを考慮する必要がある．また，各々のスマート環境で独立に開発されたミドルウェアは互いに互換性を持たないため，これらの差異を吸収する必要がある．このため我々は，**図 1** に示すように，各スマート環境の機能を REST に基づく Web サービスとして公開した．各スマート環境に設けられた REST サーバが，スマート環境独自のミドルウェアを抽象化して Web サービスとして公開し，クライアントアプリケーション（REST クライアント）がこれらの Web サービスを利用することによって，異種スマート環境間の連携を行う．

REST（Representational State Transfer）[1)2)] は，HTTP 本来のモデルをそのまま Web サービスのインタフェース（API）として用いるスタイルであり，スケーラブルな分散ソフトウェアをシンプルに構築する手法として近年主流になりつつある．REST に基づきスマート環境を Web サービス化する試みは既に行なわれており，例えば，CASTANET[3)] では，センサのデータや機器の制御機能を，REST に基づく Web サービスとして公開し，コンテキストウェアアプリケーションを構築している．

一般に REST アーキテクチャでは，HTTP のユーザ認証メカニズムを用い，許可したユーザのみにサービスを提供することによって，セキュアなシステムを実現する．例えば，前述の CASTANET では，WSSE 認証[4)] を用いている．しかし，異種スマート環境間での連携を想定した場合，自身の管理外のサービスをゲストユーザとして一時的に利用する状況が多く発生し，利用するユーザ全てをサービス提供側で管理するのは煩雑となってしまう．そもそも，スマート環境を構成するミドルウェアがユーザ管理の機能を持っていない場合もある．

我々は，ユーザ管理が不要な権限管理手法として，チケット認証方式を提案する．これは，イベント等で利用される紙のチケットから着想を得たものであり，ユーザを特定せずにチケットのみでサービスのアクセス権限を判断する．本方式では，サービス利用者に対して「チケット」と呼ばれるデータを発行し，サービス利用者は，サービスへの要求を行う際にこのチケットを添付する．チケットには，アクセス権限の情報が全て含まれ，チケットの情報のみでアクセス権限を判断できる．このため，サービス提供側で別途ユーザの管理をしなくても良く，ユーザの増加に対してスケーラブルな基盤が実現できる．また，サーバ側にゲストユーザの情報を保存しないため，REST のステートレス性の原則ともよく調和する．チケットには，改ざん防止用の署名が付加され，サービス利用者が権限の情報を書き換えることを防止できる．

また，様々な異種スマート環境で利用できるような汎用的なシステムを実現するためには，提供しているサービスの一覧やサービスの種類の情報を取得できることが望ましい．このため，本フレームワークでは WADL(Web Application Descrption Language)[5)] 形式でこれらの情報を提供する

提案手法の実現可能性を示すために，Java 上で動作する Web サーバである Jetty を用いてプロトタイプサービスを実装した．異なるスマート環境で動作するミドルウェアである cogma[6)] および UnitedSpaces[7)] を対象とし，各ミドルウェア上で動作する機器の機能を，それぞれ REST に基づく Web サービスとして公開した．本 Web サービスに対して，チケット認証方式によるアクセス制御を行い，セキュアなサービスを実現した．特に，cogma はユーザ管理の機能を持たないミドルウェアであるが，提案方式によりセキュアな Web サービス化が容易に実現できた．

2. 異種スマート環境間の連携

生活環境に様々なセンサや機器を埋め込んだスマート環境を構築する試みが多数なされており，複数のスマート環境間をまたがって，機器やサービスが相互に連携できることが期待される．例えば，ビデオ会議中の複数のスマート環境間で，会議の流れに合わせてライトの電源を制御したり，出張先のスマート環境のテレビから，自宅のスマート環境の機器の情報を参照したりできれば便利である．

しかし，一般に，スマート環境はその環境の中だけで閉じていることが多く，異なる組織により構築されたスマート環境（異種スマート環境）間で，機器やサービス間の連携を行うのは困難であった．名古屋大学と立命館大学では，それぞれ cogma room[8)] および United Spaces[7)] というスマート環境を構築したが，それぞれのスマート環境で動作するミドルウェアは独立に作成されたものであるため，そのままでは相互に連携を行えず，ミドルウェア間の差異を吸収するソフトウェアが必要となる．2002 年に東京大学の STONE Room[9)] と慶應義塾大学の Smart Space Lab[10)] がインターネット経由で相互連携した例などをはじめ，今後多くのスマート環境が構築され相互に連携を行う状況が想定される．このため，特定のスマート環境間に

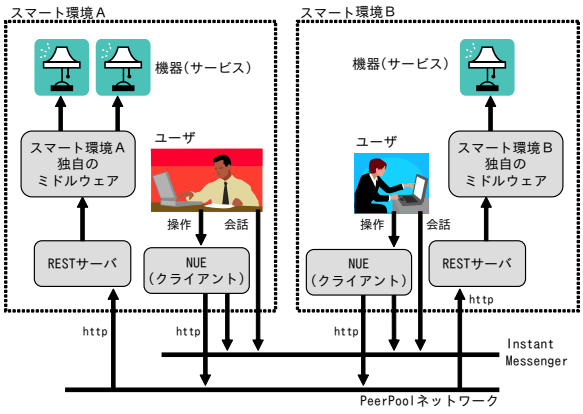

図 2 異種スマート環境間の連携のための統合型フレームワーク
Fig. 2 Integrated framework for collaboration between heteroginious smart spaces

特化した手法ではなく，多くの異種スマート環境間の連携に対して，汎用的かつ容易に適用できる手法が求められる．

このため，我々は，異種スマート環境間の連携のためには，以下のような性質を兼ね備えたソフトウェアフレームワークが必要であると考えた．

(1) **シンプルなインタフェース：**既存のスマート環境をフレームワークに適用したり，アプリケーションを開発・利用したりするためには，シンプルで分かりやすいインタフェース（API）が求められる
(2) **汎用性：**様々な異種プラットフォームで，システムを汎用的に利用できることが望ましい．また，様々な異種サービスが混在するような状況でも，それらを自然に扱えることが望ましい．
(3) **ゲストユーザを考慮した権限管理：**異種スマート環境間を連携させる際には，異なる組織が管理するサービスを利用する場合があり，ゲストユーザにもサービスを安全に公開できるアクセス権限管理手法が必要である
(4) **スケーラビリティ：**スマート環境がオープンになりユーザの数が増加しても，システムがスケールする仕組みが求められる

これらの要件を満たすために，我々は図 1 の仕組みを基本とし，**図 2** に示すような，様々な仕組みを組み合わせた統合的なフレームワークの開発を進めている．本フレームワークでは，シンプルかつ汎用的な枠組みを実現するために，各スマート環境の機器の機能を REST に基づく Web サービスとして公開する．これらの Web サービスにアクセスする際には，ゲストユーザに対してもスケーラブルできめ細かな権限管理を実現する「チケット認証方式」を用いる．ユーザは，インスタントメッセンジャと Web ブラウザをベースとしたクライアントソフトウェアである「NUE」を用いて，会話をしつつこれらの Web サービスを利用する[11]．各々のスマート環境が別々のプライベートネットワークに存在する状況が想定されるため，複数のプライベートネットワーク間で必要な端末のみをネットワーク接続する技術「PeerPool」を用いて，スマート環境間は必要に応じて自動的にネットワーク接続される[12]．

以下本稿では，スマート環境の機器の機能を REST に基づく Web サービスとして公開する手法について 3 節で述べ，この際に用いるチケット認証方式を 4 節で提案する．NUE および PeerPool については別稿[11)12)]にて述べる．

3. REST に基づく異種スマート環境間の連携

様々な異種プラットフォームで，システムを汎用的に利用するためには，既に普及している標準的な基盤の上でシステムが動作することが望ましい．そのため，本フレームワークでは，各スマート環境のサービス (機器の機能など) を，REST に基づく Web サービスとして公開する．

3.1 REST に基づく Web サービス

Web サービスとは，HTTP などの Web 技術の基盤を用いてメッセージ交換を行うことにより，異なるソフトウェアとの連携を行うソフトウェアサービスのことである．REST（Representational State Transfer)[1)2)] は，HTTP[13)] 本来のモデルをそのまま Web サービスのインタフェース（API）として用いるスタイルであり，スケーラブルな分散ソフトウェアをシンプルに構築する手法として近年主流になりつつある．

REST では，全てのリソースに URL をつけ（アドレッシング性），リソースの状態を（html 形式，XML 形式などで）表現可能とする．また，リソースの状態を取得，変更するために，リソースに対して統一されたインタフェース (HTTP の GET, PUT, POST, DELETE メソッドなど) を定義して，状態の表現を転送する．REST の原則に従うことにより，HTTP の仕組み（キャッシュ，プロクシ，Keep-Alive，負荷分散，認証，暗号化等）を有効に活用できる．また，アプリケーションの状態をサーバに持たせないこと（ステートレス性）により，ユーザの増加に対してスケーラビリティを持つ．

HTTP の上にさらにメッセージ交換プロトコルを定義する SOAP[14)] などの仕様と比べて，HTTP をその

まま利用する REST はシンプルであり，アプリケーション開発者に受け入れられやすい．例えば，Amazon の Web サービスでは，SOAP と REST インタフェースの両方を採用したところ，85%の利用が REST 経由だったと言われている[15]．また，HTTP は既に広く普及しており，様々な開発環境で汎用的に利用できる．

3.2 REST サーバ

本フレームワークでは，各々のスマート環境で独立に開発されたミドルウェア間の差異を吸収するため，図 1 に示すように，各スマート環境に設けられた REST サーバが，各スマート環境に存在するサービス（機器の機能など）を，REST に基づく Web サービスとして公開する．クライアントアプリケーション（REST クライアント）がこれらの Web サービスを利用することによって，異種スマート環境間の連携を行う．

各サービスはリソースとして扱われ，URL を持っている．例えば，電源の ON/OFF 機能を持ったあるライトが `https://192.168.207.135/cogma_room/lights/Light1` という URL を持つとする．

リソースの URL に対して，GET メソッドでアクセスすることによりリソースの状態を取得し，PUT メソッドで状態を更新する．例えば，上記の URL に対して GET メソッドでアクセスすると，**図 3** のような XML データによりライトの電源状態を取得できる．また，同様の形式の XML データを PUT メソッドで送信することにより，ライトの電源状態を変更できる．また，生成・削除が可能なリソース（例えば，機器が処理すべきタスク，機器間の接続など）については，親リソースに POST メソッドでアクセスすることにより従属リソースを生成し，DELETE メソッドでリソースを削除する．

リソースの状態は，図 3 のように，名前空間 URI を用いた XML 形式（もしくは MIME タイプ[16] が定義されたメディアデータ）で表す．これは，名前空間 URI を用いることにより，異なる組織により独立に定義された名前が衝突することを防止でき，また XML 形式や MIME タイプは 3.3 節で述べる WADL 形式との相性が良いためである．具体的な XML の形式としては，組織において独自に定めた XML 形式を用いることもできるし，また，機器の状態を表す XML 形式を標準化して複数の組織で共用することにより，クライアントアプリケーションの適用範囲を広げることもできる．

REST サーバは，3.3 節で述べるように WADL 形式によるサービスの一覧を取得する機能を持ち，また

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PowerSwitchState xmlns="http://scoop.cogma.org/app/V1">
on</PowerSwitchState>
```

図 3 REST サーバへ入出力する XML データの例: 電源の ON/OFF 状態が ON の場合を表す
Fig. 3 Example XML data for the REST server, which represents that a power switch is turning on

4 節で述べるように，チケット認証方式を用いたアクセス制限機能を持つ．

3.3 サービス一覧の取得

様々な異種スマート環境で利用できるような汎用的なシステムを実現するためには，提供しているサービスの一覧やサービスの種類の情報を取得できることが望ましい．これにより，例えば，提供されているサービスの種類の情報を解析して，未知のスマート環境に存在する未知のサービスの制御も行えるような，汎用的なクライアントアプリケーションを作成することができる．

しかし，REST に基づく Web サービスの，サービス一覧やサービスの種類を表現するのに適した記述形式は，標準化されたものがまだ存在しない．例えば，SOAP で一般的に用いる WSDL（Web Service Description Language）[17] では，PUT，DELETE メソッドを用いる REST アーキテクチャを十分に表現できず，また Web サイトのエントリ一覧を表現する Atom[18] では，入出力するデータの形式を詳細に記述できない．

本フレームワークでは，WADL（Web Application Descrption Language）[5] を用いてこれらの情報を提供する．WADL は，Marc J. Hadley らによって提唱され，REST に基づく Web サービスのインタフェースを表現する形式として徐々に普及しつつある[2]．WADL は XML をベースとしており，リソースのパス（URL）を，入れ子の `resource` 要素で表現し，また，リソースに対して入出力する XML データの形式を，リンクされた XML Schema[19] データで表現する．

WADL データの例を**図 4** に，XML Schema データの例を**図 5** に示す．本フレームワークでは，個々のサービスの一覧を，入れ子の `resource` 要素（図 4 の Light1, Light2 など）として表現した．通常 WADL では，Web サービスの静的なインタフェースのみ記述し，個々のリソース ID の指定などの動的に変化する部分は URL のパラメータとして表すが，本フレームワークでは，WADL をサービス一覧の取得形式として用いるために，このような（入れ子の `resource` 要素を用いた）表現とした．この例では，スマート環境に存在するライトを表す URL（`/cogma_room/lights/Light1`

または Light2）に対して，電源スイッチの ON/OFF 状態を表す XML データ (図 5 の PowerSwitchState で表される形式) を GET または PUT することにより，ライトの電源の ON/OFF 状態を取得したり変更したりできる．ユーザにサービスの説明を提供するために，WADL データ内の doc 要素，および XML Schema データ内の xsd:documentation 要素で説明を付加している．

なお，個々のサービスが，複数のデータ形式 (XML 形式) を取り扱いたい場合には，サービスを表す resource 要素の下に，さらにデータ形式ごとに入れ子の resource 要素を作成すればよい．単一の機器が複数の機能を持つ場合も同様である．例えば，/cogma_room/PlasmaTV1/speaker/mp3_audio という階層の URL で，cogma room（部屋）に存在する PlasmaTV1（テレビ）のスピーカで演奏する MP3 形式の音声データ，というリソースを表し，PUT メソッドで MP3 データを送信することによって演奏できるサービスが考えられる．

本フレームワークでは，REST サーバに対して /index.wadl というパスを GET メソッドで取得すると，REST サーバ上で管理している全てのサービスの情報を WADL 形式で返す．クライアントアプリケーションが，この WADL データ，およびそれにリンクされた XML Schema データを REST サーバから取得することによって，提供しているサービスの一覧やサービスの種類の情報を動的に取得できる．これにより，クライアントアプリケーションが，特定のサービス（機器）に依存した「電源の ON/OFF」などの概念を事前に知らなくても，WADL データや XML Schema データを解析し，ユーザにサービスの一覧や，「電源 ON」「電源 OFF」などの選択肢を示すことが出来る．[11)]

4. チケット認証方式

一般に REST アーキテクチャでは，HTTP のユーザ認証メカニズム[20)] を用い，許可したユーザのみにサービスを提供することによって，セキュアなシステムを実現する．例えば，1 節で述べた CASTANET では，WSSE 認証[4)] を用いている．しかし，異種スマート環境間での連携を想定した場合，自身の管理外のサービスをゲストユーザとして一時的に利用する状況が多く発生し，利用するユーザ全てをサービス提供側で管理するのは煩雑となってしまう．そもそも，スマート環境を構成するミドルウェアがユーザ管理の機能を持っていない場合もある．

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10"
   xmlns:app="http://scoop.cogma.org/app/V1">
 <grammars>
  <include href="schema/scoop_app.xsd" />
 </grammars>

 <!-- リソース(サービス)の一覧 -->
 <resources base="https://192.168.207.135:443/">
  <resource path="cogma_room">
   <doc title="cogma room">a smart room named 'cogma room'</doc>
   <resource path="lights">
    <resource type="#PowerSwitchResource" path="Light1">
     <doc title="Light1">a cogma light service named 'Light1'</doc>
    </resource>
    <resource type="#PowerSwitchResource" path="Light2">
     <doc title="Light2">a cogma light service named 'Light2'</doc>
    </resource>
   </resource>
  </resource>
 </resources>

 <!-- リソースに対するインタフェース（入出力の形式） -->
 <representation mediaType="application/xml" id="PowerSwitchState"
    element="app:PowerSwitchState" />
 <resource_type id="PowerSwitchResource">
  <method name="GET">
   <doc>get current power switch state</doc>
   <response>
    <representation href="#PowerSwitchState" />
   </response>
  </method>
  <method name="PUT">
   <doc>change power switch state</doc>
   <request>
    <representation href="#PowerSwitchState" />
   </request>
   <response>
    <representation href="#PowerSwitchState" />
   </response>
  </method>
 </resource_type>
</application>
```

図 4 サービス一覧を表す WADL データの例
Fig. 4 Example WADL data representing service lists

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://scoop.cogma.org/app/V1"
   xmlns="http://scoop.cogma.org/app/V1" elementFormDefault="qualified">

 <!-- 電源スイッチのON/OFFを表す列挙型 -->
 <xsd:simpleType name="PowerSwitchStateType">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="on">
    <xsd:annotation>
     <xsd:documentation>turning on</xsd:documentation>
    </xsd:annotation>
   </xsd:enumeration>
   <xsd:enumeration value="off">
    <xsd:annotation>
     <xsd:documentation>turning off</xsd:documentation>
    </xsd:annotation>
   </xsd:enumeration>
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:element name="PowerSwitchState" type="PowerSwitchStateType" />

</xsd:schema>
```

図 5 XML Schema データの例．REST サーバへ入出力する XML データの形式を表す．図 4 の WADL データから，schema/scoop_app.xsd というパスでリンクされている．インスタンスの例は図 3
Fig. 5 Example XML Schema data linked from WADL data in Fig.4

```
<?xml version="1.0" encoding="utf-8"?>
<ticket xmlns="http://scoop.cogma.org/ticket/V1">
  <body>
    <realm>cogma_room.cogma.org</realm>
    <authority>
      <deny path="/cogma_room/lights/Light2"
       methods="PUT" />
      <allow path="/cogma_room/lights/*"
       methods="GET,PUT" />
    </authority>
    <ticket-id>1001</ticket-id>
    <expire-date>2008-05-17T00:00:00+09:00</expire-date>
  </body>
  <body-digest>2935259b7dbde995ad12b98b2d6c9831ba07b8fb
  </body-digest>
</ticket>
```

図 6 改ざん防止用の署名を付加したチケット
Fig. 6 Ticket with degital signature

```
PUT /cogma_room/lights/Light1 HTTP/1.1
Host: 192.168.207.135:443
Content-Type: application/xml
Authorization: ScoopTicket PD94bWwgdmVyc2lvbjOiMS4wIiBlbm
NvZGluZzOiVVRGLTgiPz48dGlja2VOcyB4bWxuczOiaHROcDovL3Njb29
wLmNvZ21hLm9yZy9OaWNrZXQvVjEiPjxOaWNrZXQ+PGJvZHk+PHJlYWxt
PmNvZ21hX3Jvb2OuY29nbWEub3JnPC9yZWFsbT48YXVOaG9yaXR5PjxkZ
W55IHBhdGg9Ii9jb2dtYV9yb29tL2xpZ2h0cy9MaWdodDIiIG1ldGhvZH
M9IlBVVCIvPjxhbGxvdyBwYXRoPSIvY29nbWFfcm9vbS9saWdodHMvKiI
gbWVOaG9kczOiROVULFBVVCIvPjwvYXVOaG9yaXR5PjxOaWNrZXQtaWQ+
MTAwMTwvdGlja2VOLWIkPjxleHBpcmUtZGFOZT4yMDA4LTA1LTE3VDAwO
jAwOjAwKzA5OjAwPC9leHBpcmUtZGFOZT48L2JvZHk+PGJvZHktZGlnZX
NOPjI5MzUyNTliN2RiZGU5OTVhZDEyYjk4YjJkNmM5ODMxYmEwN2I4Zml
8L2JvZHktZGlnZXNOPjwvdGlja2VOPjwvdGlja2VOcz4=

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PowerSwitchState xmlns="http://scoop.cogma.org/app/V1">
on</PowerSwitchState>
```

図 7 チケット認証を用いる HTTP リクエストの例 (抜粋)
Fig. 7 Example HTTP request with ticket authrization (summary)

我々は，ユーザ管理が不要なアクセス権限管理手法として，チケット認証方式を提案する．本方式は，イベント等で利用される紙のチケットから着想を得たものであり，ユーザを特定せずにチケットの情報のみでサービスのアクセス権限を判断する．このため，サービス提供側で別途ユーザの管理をしなくても良く，ユーザの増加に対してスケーラブルな基盤が実現できる．

4.1 チケット認証プロトコル

チケット認証方式を REST アーキテクチャに適用するため，以下のような設計を行った．本方式では，サービス管理者が，サービス利用者（ゲストユーザ）に対して「チケット」と呼ばれるデータを発行し，サービス利用者は，REST サーバへのリクエストを行う際に，このチケットを添付する．

チケットには，REST サーバに対するアクセス権限の情報が全て含まれ，チケットに含まれる情報のみでアクセス権限を判断できる．アクセス権限の情報は XML で記述し，可能なリクエスト内容（どの URL に対して，どの HTTP メソッドが使えるか），チケットの利用期限，認証対象となる REST サーバ（レルム）を特定するレルム名，チケットを特定する ID が含まれる．(これ以外の情報を追加してサーバ側で利用しても良い)

図 6 の `body` 要素が，アクセス権限情報の例である．`realm` 要素はレルム名，`ticket-id` 要素はチケットを特定する ID，`expire-date` 要素はチケットの利用期限を表す．また，`authority` 要素の子要素が，可能なリクエストのルール（アクセス制限）を表し，パス (`path` 属性) と HTTP メソッド (`methods` 要素) が一致した場合に，リクエストを許可 (`allow` 要素) または拒否 (`deny`) する．ルールは上のものほど優先され，どのルールにも一致しなかった場合は拒否される．パスの末尾がワイルドカード*で終わる場合は，指定されたパスを接頭辞に持つ全てのパス（子孫リソース）が対象となる．この図の例だと，`/cogma_room/lights/` 以降の全てのパスに対して，GET，PUT メソッドが原則許可されるが，`/cogma_room/lights/Light2` に対する PUT メソッドのみは拒否される．

チケットには，改ざん防止用の署名を付加し，サービス利用者が権限の情報を書き換えることを防止する．レルム (REST サーバ) ごとに秘密鍵を用意し，アクセス権限情報（`body` 要素）と秘密鍵を合わせたデータに対して，SHA1 ダイジェストを計算したものを，チケットに付加することにより，改ざん防止用の署名を行う．署名されたチケットの例を図 6 に示す．`body-digest` 要素が，計算されたダイジェストである．

サービス管理者は，アプリケーションで利用する署名済みチケットを base64 エンコードした文字列（チケットトークン）を，サービス利用者に渡す．**図 7** の `Authorization: Scoop Ticket` 以降がチケットトークンの例である．複数の異種スマート環境にアクセスするような場合には，複数のレルムに対するチケットをつなげて（複数の `ticket` 要素を子に持った `tickets` 要素をルートとして XML を作成），単一のチケットトークンとすることもできる．

サービス利用者は，Web サーバに対して HTTP リクエストをする際に，このチケットトークンを Authorization ヘッダに添付する．図 7 に例を示す．この例では，図 3 の XML データを PUT リクエストで送信している．なお，チケットの盗聴を防止するために，チケットを添付して REST サーバへアクセスする際には，SSL（Secure Socket Layer）などを用いて通信路を暗号化する．

チケット認証方式は Web サーバ（REST サーバ）に対する拡張として埋め込まれ，チケットに書かれたアクセス権限情報に合わないリクエストは Web サー

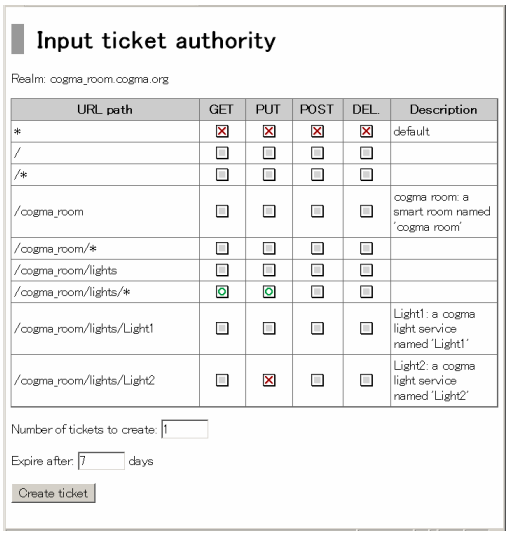

図 8 チケット発行のための Web アプリケーション
Fig. 8 Ticket publishing tool

バ側で自動的に拒否する．また，チケット認証方式では，Web サーバに対して別途ゲストユーザの認証情報を登録する必要が無い．このため，Web サーバ上で動作するアプリケーション開発時には，特にチケット認証方式を意識する必要がない．すなわち，チケット認証方式を用いれば，Web サービスを開発する際に，ゲストユーザに対してスケーラブルかつきめ細かなアクセス権限管理を，容易に実現できる．

4.2 チケット発行ツール

サービス管理者がチケットを発行する際に，毎回アクセス権限の情報を XML で記述するのは煩雑なため，チケットの発行を支援するツール（チケット発行ツール）を，**図 8** のような Web アプリケーションとして開発した．

本ツールでは，REST サーバが提供する WADL データ（3.3 節参照）を元に，存在するリソースのパスの一覧が表示される．サービス管理者は，それぞれのパスの各 HTTP メソッド（GET,PUT,POST,DELETE）を表すセルに対して，アクセス権限を指定できる．セルはデフォルトでは無指定（継承）であり，セルをクリックすることによりアクセスの許可または拒否を指定できる．無指定（継承）の場合は，“親リソース/*”というパスに対する指定が継承される．図 8 は，図 6 のチケットを生成する例である．

チケット発行ツールは，チケット発行の対象となる REST サーバとの間で，チケット署名に用いる秘密鍵を共有している．チケット発行ツール自体はゲストユーザではなく管理者が利用するため，ツールにアクセスする際の認証は一般的な方式（Basic 認証など）を用いれば良い．

本ツールでアクセス権限情報を指定してチケットを発行すると，発行したチケットのデータ（XML 形式，およびチケットトークン形式）が表示されるため，サービス管理者はこのデータをサービス利用者に通知する．また，サービス利用者のクライアントアプリケーションへ，サービス管理者が発行したチケットを自動的にインストールする機構についても今後の課題として検討している．

5. 実　　装

提案したフレームワークの実現可能性を示すために，Java 1.6 と，Web サーバである Jetty 6.1[21] を用いてプロトタイプシステムを実装した．実装したフレームワークには，WADL データの生成やチケット認証等の機構が内蔵されているため，開発者は，スマート環境に特化した部分のみを記述すれば良い．このため，スマート環境の機器の機能を公開するめの REST サーバを容易に構築できる．

異なるスマート環境で動作するミドルウェアである cogma および United Spaces を対象とし，各ミドルウェア上で動作する機器の機能を，それぞれ REST に基づく Web サービスとして公開した．具体的な機器としては，X10 で制御可能な電源スイッチおよび調光ユニット（cogma），ライトを模したソフトウェアエミュレータ（cogma），Universal Plug and Play で電源を制御可能なライト（United Spaces）を対象とした．各機器には URL が割り当てられ，URL に対して GET,PUT リクエストを行なうと，機器の状態（電源の ON/OFF 等）を取得したり，状態を変化させたりすることができる．また，各スマート環境の機器（サービス）の一覧を，WADL 形式で取得でき，ユーザに対して適切なサービスの一覧を推薦するために利用できる．

本 Web サービスに対して，チケット認証方式によるアクセス制御を行い，セキュアなサービスを実現した．特に，cogma はユーザ管理の機能を持たないミドルウェアであるが，提案方式によりセキュアな Web サービス化が容易に実現できた．

6. 評　　価

実装したプロトタイプシステムを用いて，チケット認証方式のオーバーヘッドを評価した．提案したチケット認証方式を用いる場合，および，認証無しの場合，Basic 認証方式，Digest 認証方式を用いる場合[20] の

それぞれついて，HTTP リクエストヘッダの大きさと，1回のリクエストにかかった平均時間を測定した．チケット認証方式については，アクセス制限のルール数 n（`authority` 要素の子要素の数．4.1 節参照）を 1 から 64 まで変化させて比較した．

結果を**図 9** および**図 10** に示す．“`Ticket n=`数値”という形式のラベルは，チケット認証方式でアクセス制限のルール数が n の場合を表す．

6.1 実験方法の詳細

実験用のクライアントソフトウェアの実装には，Jakarta Commons HttpClient 3.1[22] を用いた．REST サーバは CPU Core2 Duo 3GHz, RAM 2GB の PC で，クライアントは CPU Core2 Duo 2.2GHz, RAM 2GB の PC で動作させた．REST サーバとクライアントは 1Gbps の有線 LAN で接続し，暗号化のために SSL 通信を用いた．

1回のリクエストにかかった平均時間は，単一のコネクションで 11000 回のリクエストを連続して行い（Keep-Alive 使用），最初の 1000 回のリクエストを除いて平均値を計算した．従って，SSL 通信路の確立にかかる時間（1 回目のリクエストのみ）は含まれていない．

各リクエストでは，図 7 の例と同様に，REST サーバの `/cogma_room/lights/Light1` というパスに対して，電源 ON を要求する PUT リクエストの送信し，結果の HTTP レスポンスを受信した．

REST サーバ側では，対応するパスで，仮想的なライトのサービス（メモリ上に電源の ON/OFF 状態を保持するのみ）を提供させた．

チケット認証方式（ルール数 n）で用いるチケットには，”`/cogma_room/lights/Light` + 連番（n から 1 までの降順）”というパスに対して GET と PUT メソッドを許可 (allow) する，という内容の，n 個のルールからなるアクセス権限情報を含めた．

Basic，Digest 認証方式においては，ユーザ名 5 文字とパスワード 5 文字の認証情報を用いた．

Digest 認証方式は，1 回目の HTTP リクエストで nonce を受け取り，2 回目の HTTP リクエストで認証情報を送信する方式のため，図 9，図 10 の結果は 1 回目と 2 回目の合計値を示してある．

6.2 結果の考察

図 9 の結果を見ると，チケット認証方式においては，ルール数 n が増加すると HTTP リクエストヘッダの大きさが増加しているのが分かる．可能なリクエストヘッダの大きさの上限は，実装依存だと思われるが[⋆1]，例えば Cookie の大きさの上限値は 4096bytes とされている[23] ため，この程度を上限と見なすのが妥当である．図 9 を見ると，アクセス制限のルール数が 32 程度までならこの制限に収まり，既存の Web 基盤上でチケット認証方式を利用できる．

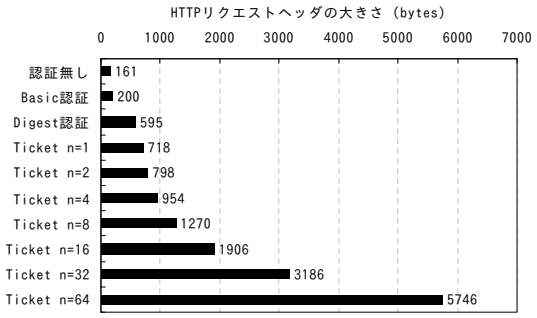

図 9 HTTP リクエストヘッダの大きさ
Fig. 9 Size of HTTP request header (bytes)

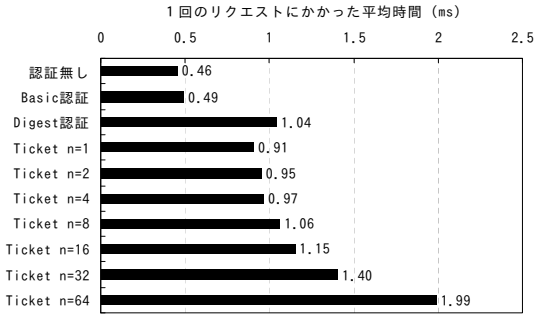

図 10 1 回のリクエストにかかった平均時間
Fig. 10 Average time per request (ms)

チケット認証方式は，Basic 認証方式と比較すると，リクエストヘッダのサイズ，リクエストにかかる時間ともに大きくなってしまう．しかし，チケット認証方式を用いれば，サーバ側でユーザ情報を管理する必要が無く，ゲストユーザに対するきめ細かなアクセス制限を容易かつスケーラブルに実現できる．この利点を考慮すると，ルール数が 32 程度までであれば，このオーバーヘッドは十分に許容できる範囲だといえる．

7. 関 連 研 究

REST に対応したストレージサービスである，Ama-

⋆1 Jetty 6.1 の場合，リクエストヘッダの上限サイズはデフォルトでは 4096bytes に指定されているが，変更可能である．

zon S3 (Amazon Simple Storage Service)[24] では，ストレージの管理者が，ゲストユーザに対して特定のリソースのみへのアクセス権限を提供できる．この際には，主要な HTTP ヘッダ（パス，HTTP メソッド，日時の有効期限など）に対して管理者が秘密鍵で署名（ダイジェストの生成）を行うことにより，署名パラメータ付きの URL を生成し，これをゲストユーザに渡す.[2] これは，4 節で提案したチケット認証方式に類似している．しかし，Amazon S3 の方式では単一のリクエスト内容 (特定のパスおよび HTTP メソッド) に対してのみ署名が行えるのに対し，チケット認証方式では，複数のパスおよび HTTP メソッドに対するアクセス権限をまとめて，単一のチケットとしてゲストユーザに提供できる．特に，複数のデバイスを連携させるようなアプリケーションにおいては，単一のチケットのみで様々なリソースへのアクセスが行えるチケット認証方式は，ユーザやアプリケーション開発者にとってよりシンプルで分かりやすい方式だと考えられる．

Ruby on Rails 2.0[25] の Cookie Session Store では，セッションのデータをサーバ側ではなくクライアントの Cookie[23] に保存し，サーバ側で生成されたダイジェストを付加することにより，このデータの改ざんを防止する．これは，チケット認証方式で用いているチケットの改ざん防止の方式に類似している．チケット認証方式では，REST に基づく Web サービスのアクセス権限管理に特に着目し，これに適したアクセス権限情報のモデル（4.1 節）やツール（4.2 節）を提案している点が異なる．また，チケット認証方式においても，チケットの送信に Authorization ヘッダではなく Cookie を用いることが検討できる．これにより，REST サーバと同一のホスト名でチケットを発行する必要はあるが，Web ブラウザなどの既存のクライアントソフトウェア上で，チケット認証方式を利用することが出来る．

機器やサービスにアクセスするプロトコルを標準化した仕様として，UPnP Standardized DCP[26]，ECHONET[27]，LonMark SNVT[28] などが挙げられる． UPnP Standardized DCP や ECHONET では，機器や機器の機能（サービス）のモデルを規定し，LonMark SNVT では物理量（温度，速さ，風速など）の表現形式を規定している．我々のフレームワークでは，3.2 節で述べたように，REST に基づいてスマート環境をモデル化することにより，部屋，機器，サービス，表現形式などの様々な粒度のリソースに対して，統一されたインタフェースでシンプルにアクセスできる．上記の標準化仕様とは補完関係にあるとも言え，例えば LonMark SNVT の表現形式を XML 形式にマッピングし，我々のフレームワーク上で利用することも検討できる．

CASTANET[3] では，センサのデータや機器の機能を REST に基づく Web サービスとして公開し，スケーラブルなコンテキストアウェアアプリケーションの開発を支援している．しかし，サービスの一覧やサービスの種類・形式などを取得する方法については規定が無く，あらかじめスマート環境の構成を知っている必要がある．我々のフレームワークでは，3.2 節で述べたように，WADL 形式と XML Schema 形式で，サービスの一覧やサービスの種類・形式などのメタ情報を提供する．クライアントアプリケーション側でこれらのメタ情報を解析することにより，異種スマート環境間の連携を行う際に遭遇すると思われる，未知の環境，未知のサービスに対してのアクセスも支援できる．

8. ま と め

本稿では，異種スマート環境間の連携のための要件を示し，それを満たすフレームワークを提案した．本フレームワークでは，シンプルかつ汎用的な枠組みを実現するために，各スマート環境の機器を REST に基づく Web サービスとして公開する．REST サーバでは WADL 形式でサービスの一覧と種類の情報を提供し，これをクライアントアプリケーションから利用できる．また，ユーザ管理が不要なアクセス権限管理手法として，チケット認証方式を提案した．これは，イベント等で利用される紙のチケットから着想を得たものであり，ユーザを特定せずにチケットのみでサービスのアクセス権限を判断する．本稿で提案した枠組みを用いれば，異種スマート環境間の連携を，シンプル・汎用的・セキュア・スケーラブルに実現できる．

今後の課題としては，以下のようなものが挙げられる．

- **アプリケーション：** REST サーバで公開した Web サービスを用いた，より実用的なクライアントアプリケーションを作成し，本フレームワークの応用可能性を示すことが望ましい．
- **非暗号通信路への対応：** 現在，チケット認証方式は秘密情報を平文で送信するため，暗号化通信路 (SSL 等) を用いる必要がある．SSL はコネクションを確立する際の負荷が大きいため，Digest 認証や WSSE 認証に類似した方式を用いて，非暗号化通信路上でもセキュアに利用できるチケッ

ト認証方式を実現することが期待される．

- **他の異種スマート環境への適用：** cogma room や United Spaces 以外に，他の異種スマート環境へも本フレームワークを適用し，その適用可能性を示すことが望ましい．
- **チケットのフォーマットの最適化：** 現在は XML 形式を base64 エンコードしたデータをチケットとして用いているが，よりサイズが小さく，解析に時間がかからないフォーマットを検討することが望ましい．

謝辞 本研究の一部は，総務省「戦略的情報通信研究開発推進制度（SCOPE）」の支援を受けて行われた．

参 考 文 献

1) Fielding, R. T.: *Architectural styles and the design of network-based software architectures*, PhD Thesis, University of California, Irvine (2000).
2) Richardson, L. and Ruby, S.: *Restful Web Services*, Oreilly & Associates Inc (2007).
3) 小澤政博，川原圭博，川西　直，森川博之：REST アーキテクチャスタイルに基づくコンテキストアウェアサービス連携フレームワークの設計，電子情報通信学会総合大会，p.259 (2007).
4) Nadalin, A., et al.: Web Services Security UsernameToken Profile 1.0, `http://www.oasis-open.org/specs/` (2004).
5) Hadley, M. J.: Web Application Description Language (WADL), *Sun Microsystems Laboratories Technical Report SMLI TR-2006-153* (2006).
6) Kawaguchi, N.: Cogma: A Middleware for Cooperative Smart Appliances for Ad hoc Environment, *Proc. of International Conference on Mobile Computing and Ubiquitous Networking (ICMU2004)*, pp. 146–151 (2004). `http://www.cogma.org/`.
7) 西尾信彦：ユビキタス技術実用展開のためのプラットフォーム構築:「インター」ユビキタスネットワーク構築を目指して，電子通信情報学会技術研究報告 (無線通信システム/コミュニケーションクォリティ研究会)， Vol.104, No.21, pp.117–120 (2004).
8) 河口信夫：cogma：ユビキタス情報環境を構築する基盤ソフトウェア，FIT2004 第 3 回情報科学技術フォーラム (2004).
9) 森川博之，南　正輝，青山友紀：「ユビキタスネットワーキングへの道，情報処理，Vol.43, No.6 (2002).
10) 徳田英幸，中澤　仁，岩井将行，由良惇一，村瀬正名：ユビキタス空間を融合するネットワーク技術への課題，情報処理， Vol.43, No.6 (2002).
11) 田中宏一，藤原茂雄，岩崎陽平，榎堀　優，中野悦史，西尾信彦，河口信夫：IM クライアントを用いた異種スマート環境制御のためのサービス推薦機構，マルチメディア，分散，協調とモバイル（DICOMO 2008）シンポジウム論文集 (2008).
12) 榎堀　優，中野悦史，藤原茂雄，田中宏一，岩崎陽平，西尾信彦，河口信夫：サービスアクセスに連動したトンネルネットワークの自動構築，マルチメディア，分散，協調とモバイル（DICOMO 2008）シンポジウム論文集 (2008).
13) Fielding, R.T., Gettys, J., Mogul, J., Nielsen, H.F., Berners-Lee, T., Masinter, L. and Leach, P.: Hypertext Transfer Protocol – HTTP/1.1, *Internet RFC 2616* (1999).
14) Box, D., et al.: Simple Object Access Protocol (SOAP) 1.1, `http://www.w3.org/TR/soap/` (2000).
15) O'Reilly, T.: REST vs. SOAP at Amazon, `http://www.oreillynet.com/pub/wlg/3005` (2003).
16) IANA (Internet Assigned Numbers Authority): MIME Media Types, `http://www.iana.org/assignments/media-types/`.
17) Christensen, E., et al.: Web Services Description Language (WSDL) 1.1, `http://www.w3.org/TR/wsdl` (2001).
18) Nottingham, M., et al.: The Atom Syndication Format, *Internet RFC 4287* (2005).
19) Fallside, D. C., et al.: XML Schema Part 0: Primer Second Edition, `http://www.w3.org/XML/Schema` (2004).
20) Franks, J., et al.: HTTP Authentication: Basic and Digest Access Authentication, *Internet RFC 2617* (1999).
21) Mort Bay Consulting: jetty6 - Jetty WebServer, `http://www.mortbay.org/jetty-6/`.
22) Apache Software Foundation: Jakarta Commons HttpClient, `http://hc.apache.org/`.
23) Kristol, D., et al.: HTTP State Management Mechanism, *Internet RFC 2109* (1999).
24) Amazon.com, Inc.: Amazon Simple Storage Service (Amazon S3), `http://aws.amazon.com/s3`.
25) Hansson, D.H., et al.: Ruby on Rails, `http://www.rubyonrails.org/`.
26) UPnP Forum: UPnP Standards, `http://www.upnp.org/standardizeddcps/`.
27) ECHONET (Energy Conservation and Homecare Network) Consortium: エコーネット規格書, `http://www.echonet.gr.jp/`.
28) LonMark International: LONMARK Application Layer Interoperability Guidelines, `http://www.lonmark.org/`.