高級言語によるネットワーク汎用I/O制御とプロトコル翻訳機構

落合秀也[†] 江崎浩[†]

ネットワーク通信機能を持った汎用 I/O(GPIO) の応用で遠隔制御や遠隔モニタリングが可能になったが，制御においてベンダー固有の通信プロトコルを考慮する必要がある現状では，制御ソフトウェアの開発効率や再利用性を向上させることはできない．本研究は，制御ソフトウェア向けに高級言語で制御インタフェースを提供する一方，ネットワーク接続された組込み機器との固有の通信を翻訳スクリプトとして記述することで，制御ソフトウェアから隠蔽する翻訳機構を提案する．提案する翻訳機構は，(1) 制御ソフトウェアの開発を容易にし，(2) 制御ソフトウェアの再利用性を向上させる．翻訳スクリプトを Web 等で公開させることで (3) 誰でも自由に組込み機器への制御インタフェースを公開できる．本研究では，翻訳機構のプロトタイプ実装を作成し，システム評価を行い，組込み機器のソフトウェアには全く手を加えることなく，これらの性質を満足することを確かめた．

Networked GPIO Control by High-Level Languages and Protocol Translator

Hideya Ochiai[†] and Hiroshi Esaki[†]

Remote control and monitoring enabled by networked GPIO force system developers to aware unique protocols for the devices. This indicates low efficiency in development and low reusablity of application software. We propose an translator that provides high-level language interfaces for application and hides detailed communication protocols of devices. Our proposed translator enables (1) to ease programming, (2) to improve reusability of the software, and (3) anyone to publish the accessibility of their GPIO devices. We confirmed our prototype system satisfies these properties.

1. はじめに

近年の組込みコンピュータの小型化／高効率化により，インターネットを介した汎用 I/O(GPIO) の遠隔制御が可能になっている．独自の機器を GPIO に接続し，遠隔地の環境情報観測や，アクチュエータ機器制御を実現しているアプリケーションも見受けられる[7),15)]．これらのアプリケーションでは，組込み機器ベンダーごとに独自の TCP/IP 上の軽量通信プロトコルを使っている場合も少なくない[2),13),14)]．現在，制御ソフトウェア開発においては，対象とする組込み機器に応じて，様々な環境が要求される．高級言語 (e.g., Java, Perl, C#, XML) で制御ソフトウェアが記述され，機器に依存しない動作環境を用意することができれば，ソフトウェアの開発効率および再利用性を向上させることができると考えられる．

本研究は，高級言語制御インタフェース (単に，制御インタフェースと呼ぶ) と組込み機器が提供する GPIO 制御インタフェース (デバイスインタフェースと呼ぶ) との結合関係を翻訳スクリプトに記述しておき，センサやアクチュエータの制御実行時に，これらを動的に結合させる翻訳機構を提案する．翻訳機構を，制御アプリケーションの基本コンポーネントとして導入することにより，次の 3 つの効果がもたらされる．

(1) 制御ソフトウェアの開発が容易になる
開発者は下位通信プロトコルを意識しなくて済むようになる．

(2) 制御ソフトウェアの再利用性向上
特定の機器だけでなく他の機器にも同じ制御ソフトウェアを使える．

(3) スケーラブルなオープン性
翻訳スクリプトを Web 等で公開することで，誰でも自由にセンサやアクチュエータへのアクセスを提供できる．

これらの効果を組込み機器のソフトウェアを変更せずに提供可能である．

本研究が対象とする組込み機器 (別名，デバイスと呼ぶ) とは，遠隔地に設置され，デジタル I/O やアナログ I/O を，インターネット越しに低速に (100ms 以上

† 東京大学
The University of Tokyo

の精度で) 読書きできるものである．低速なのは，ネットワーク遅延による限界に由来する．GPIO に接続される機器は，標本精度が数秒ほどでも十分なもの，例えば，温度センサや DIP スイッチ，電源の ON/OFF コントローラ，キャラクタディスプレイ，照明の輝度制御装置などを想定する．これら遠隔 GPIO の先に接続される機器をサーバと呼び，汎用コンピュータから観測や制御等のためにリクエストを発行する制御ソフトウェアのことをクライアントと呼ぶことにする．本研究では，サーバへのインターネットからの到達性は確保されているものとする．サーバの利用権限はすべてのクライアントに開かれているものとし，クライアントのアクセスコントロールは将来的な課題である．通信形態は，Remote Procedure Call(RPC) 型のみを考え，イベント通知型は本研究では扱わない．

一般に，異なる表現で定義されたインタフェースを接続する方法としてよく知られたものに Adapter デザインパターン (別名 Wrapper)[5] の応用がある．インターフェース間のメソッド名の対応付けや，データ表現の変換に良く使われるデザインである．制御インタフェースの定義が決まれば，制御インタフェースとデバイスインタフェースとを繋ぐ Adapter を実装すればよい．Adapter により，デバイスインタフェースは隠蔽され，制御ソフトウェアは，制御インタフェースを使ったプログラムとして記述される．

本研究は，Adapter をスクリプト言語で記述しておき，クライアントの通信対象とするサーバに応じて，適切なスクリプトを選択後，クライアントコンピュータ上で実行することで，制御インタフェースとデバイスインタフェースの間を動的に接続する手法を提案する．このスクリプトのことを翻訳スクリプトと呼び，インタプリタを翻訳プロセッサと呼ぶ．各クライアントは，ローカル環境にライブラリ等の方法で，翻訳プロセッサを用意し，サーバへのリクエスト発行時に，対応する翻訳スクリプトを実行させることで，Adapter 機能を再現する．

サーバと通信を行うためには，そのサーバに対応する翻訳スクリプトを取得することができればよい．したがって，例えば Web を介して翻訳スクリプトの提供と取得を行うことにすれば，サーバの公開に分散性とオープン性を与えることができる．

本研究は，すでに翻訳スクリプトが Web 等で提供されている状況を想定し，翻訳機構のシステムが制御ソフトウェア (クライアント) の開発効率を向上させ，そこに再利用性を持たせられることに主眼を置く．翻訳スクリプトを作成する実際的な方法については，第 6 章で考察する．

スクリプト言語を採用したため，処理能力の低下 (e.g., 遅延の増加など) が懸念されるかもしれない．本研究では，プロトタイプシステムを構築し，動作を確認すると共に，スクリプト言語方式が，この点において，遠隔 GPIO のアプリケーションにとって実用的かどうかも検証する．

本論文の構成は以下の通りである．第 2 章で関連技術を述べる．第 3 章で翻訳機構アーキテクチャを述べ，第 4 章で，プロトタイプ実装について言及する．第 5 章で，プロトタイプシステムを使った評価実験を行い，第 6 章で考察を述べ，第 7 章でまとめる．

2. 関 連 技 術

GSN[1] は，Wrapper デザインを応用した仮想化により，複数種類の無線センサネットワークのインタフェースを抽象化している．Janus[4] は，Agent が，無線センサネットワークごとに通信に必要な Adapter を管理しており，クライアントが指定した通信相手への Adapter を動的に選択する．これらの研究においては，Adapter ライブラリの管理運用方法について深く考察されていないようである．我々の提案システムは，誰でも自由にライブラリを翻訳スクリプトの形で公開可能にし，分散的に動作するように構築されている点でこれらの方式と比べ優れている．

高級言語インタフェースをアプリケーションに提供し，遠隔サービスとの通信における，下位通信プロトコルを隠蔽する技術としては，Interface Definition Language(IDL) 関連の技術 (e.g., Microsoft-IDL[6], WSDL[3]) がある．定義された IDL から，RPC のサーバスタブやクライアントスタブを自動的に作成することで，下位通信の詳細を隠蔽する．同一の IDL をグローバルに共有すれば，ソフトウェアの再利用性を向上させることもできる．本研究で提案する翻訳スクリプトも，下位通信プロトコルを隠蔽する点は IDL と同じである．IDL は，通常，対等な言語レベルのソフトウェア間通信を実現することが目的であるのに対し，翻訳スクリプトは，異なる表現レベルの通信を Adapter により結合させることを目的としている．

温度センサなどの機器を操作するための API を，機器の運用者がサービスとして提供し，クライアントがこれを使って，間接的に操作する方式も考えられる．しかしこの手法は，サービスの信頼・安定性，可用性，スループット等についての検討課題を生み出す．本提案システムでは，クライアントとデバイスは IP 到達性があれば通信可能で，中間的なサービスの性質を気

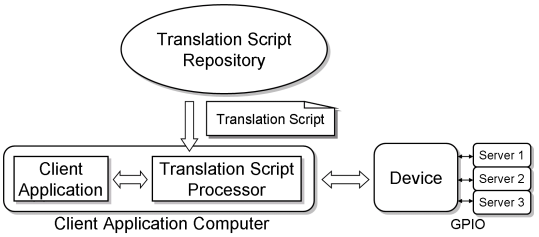


図 1 翻訳機構アーキテクチャ
Fig. 1 Translator architecture

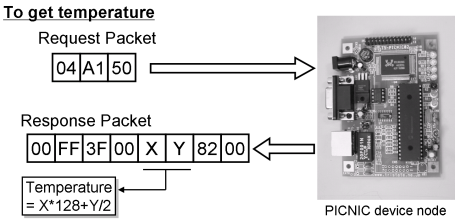


図 2 デバイス (PICNIC) からの温度取得操作
Fig. 2 Temperature acquisition from a device (PICNIC)

にしなくてよい．例えば，ポーリング型のアプリケーションにおいてはすべてのリクエスト／レスポンスが IP レベルのルーティングで配送処理される．したがって，中間サービスを導入する方式に比べて，スループットが問題になるケースは稀である．このように本提案システムを用いると，中間サービスに関する様々な検討課題から解放される利点がある．

ソフトウェアを，動的に他のコンピュータに転送し，その場で実行させる技術は，Code Migration や Mobile Agent として知られている[12]．本研究は，プロトコル翻訳に必要なコードをスクリプト言語として記述し，クライアントコンピュータ上で呼び出せるようにした点で，これらの技術の応用と位置づけられる．

3. 翻訳機構アーキテクチャ

図 1 に，本研究で提案する翻訳機構アーキテクチャを示す．デバイスは，RPC 型でサービスを提供しており，クライアント側のコンピュータには，(1) 制御ソフトウェアのインスタンスであるクライアント・アプリケーション及び (2) 翻訳プロセッサが搭載されている．多数のユーザが参照するレポジトリがあり，ここで翻訳スクリプトが管理／提供されている．

このアーキテクチャ上での基本的な動作は次のようになる．クライアントがサーバを指定し，翻訳プロセッサに RPC 要求を発行すると，翻訳プロセッサはレポジトリからそのサーバに対応するスクリプトを読込み，要求を翻訳後，デバイスに要求を発行する．デバイスからの応答は再び翻訳され，クライアントに返される．

3.1 アプリケーション

サーバ・アプリケーションは，デバイスが提供する GPIO に接続された機器である．具体的には，温湿度センサ，DIP スイッチ，電磁リレー，照明などが GPIO に接続されるサーバ・アプリケーションである．

クライアント・アプリケーションは，翻訳プロセッサから提供されている下記の invoke 命令を通じて，RPC 要求を発行する．

$invoke(s, m, p)$
s: サーバ名
m: 制御インタフェースのメソッド名
p: パラメータ値
戻り値: 応答メッセージ

この命令は，サーバ s に対してメソッド呼出し $m(p)$ を発行することを意味する．デバイスインタフェースの詳細は隠蔽され，クライアントは，常にこの命令によって提供される制御インタフェースを利用する．s, m, p のスキーマは，グローバル権威により管理され，実際のアプリケーション開発は，それに従う．IETF[10] や IEEE[9] などの標準化団体，もしくは IANA[8] などの名前運用権威団体によって，コマンド名やパラメータセットを管理しておき，翻訳スクリプトの作成，および，クライアント開発が，これに従うことで，制御インタフェースに互換性を持たせ，ソフトウェア再利用性が達成されるようになる．

s の例としては，DNS の FQDN や URL が考えられる．$m(p)$ は，具体的には，n を表示することを，$display(n)$; のように定義できるし，XML での命令表記<display number="n" />のように定義することもできる．

3.2 デバイス

デバイスは，TCP/IP での軽量な通信は可能だが，Web サービスほどの高級な通信機能は備えていない．サーバとの通信は，UDP や TCP 上の簡素なプロトコルで提供されている．図 2 に，具体例として，PICNIC[14] から，温度を取得する場合を示す．PICNIC では UDP 上のプロトコルにより，温度センサから値を読み出すためのデバイスインタフェースが提供されている．図のようなリクエストを発行することで得られる応答結果の中から，特定のオクテット列を取り出し，計算式 $X \times 128 + Y/2$ を適用することで，温度 (℃) として取り出すことができる．

サーバ・アプリケーション開発は，複数の組織により独立に行われる状況が一般的である．同じような機能を持つサーバであっても，GPIO への実装方法 (e.g.,

```
m(p){
  ··· parse application parameter (p) ···
  ··· generate request packet (req_packet) ···
  send(sock, req_packet);
  recv(sock,res_packet);
  ··· parse response packet (res_packet) ···
  ··· generate application response (r) ···
  return r;
}
```

図 3 典型的なメソッド (m) の実装形態
Fig. 3 A typical implementation for method(m)

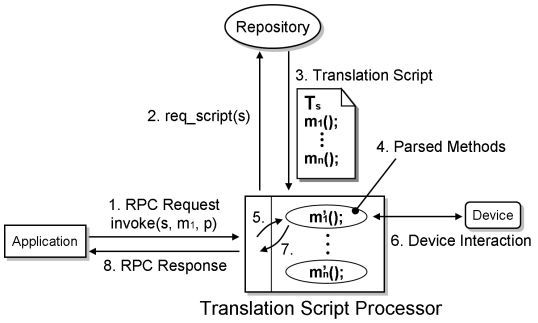


図 4 翻訳プロセッサ周辺の一連の動作
Fig. 4 Behavior of the translation processor

配線方法) や，使用している素子によって，デバイスインタフェースは異なってきてしまう．本研究ではこのような状況を考え，デバイス側のソフトウェアには手を加えずとも，スクリプトを記述することで制御インタフェースに適合できるアーキテクチャとしている．

3.3 翻訳スクリプト

翻訳スクリプトは，サーバごとに定義され，サーバ識別子に対応づけるものとする．クライアントが，$invoke(s, m, p)$ を実行すると，サーバ識別子 (s) を検索キーに翻訳スクリプトレポジトリから読み出される．

本研究では，翻訳スクリプトをメソッド集合として定義する．すなわちサーバ s の翻訳スクリプト T_s は，

$$T_s = \{m_{s1}, \ldots, m_{sn}\}$$

と記述できる．クライアントから発行された $m(p)$ に対応するメソッド m が，翻訳プロセッサにより実行される．例えば，$display(4)$ の場合，$m_{si} =$ "$display$" である m_{si} が 4 をパラメータとして呼び出される．

ここで，典型的な m の実装は，図 3 のようになっており，(1) リクエストデータ変換，(2) 通信，(3) レスポンスデータ逆変換，を通して，デバイスインタフェースを制御インタフェースへ適合させている．

スクリプト言語のデザイン次第で，記述可能な Adapter の範囲が決まってくる．言語が UDP ソケットしかサポートしていなければ，TCP 通信を利用するデバイス向けの翻訳スクリプト (Adapter) を作成することはできない．同じことはデータ加工のための各種演算機能にも言える．記述力および利用可能なリソース数が多いほど，翻訳スクリプトの作成可能な範囲は広くなる．一方，言語の記述力，利用可能なリソースを強化すると，実行先コンピュータのセキュリティ確保の問題もでてくる．これは Mobile Agent に共通する課題[12] であり，本研究の対象外である．本研究では，アーキテクチャとしての提案を行うことに焦点をおき，スクリプト言語のデザインについては，将来的な研究課題としている．

3.4 翻訳プロセッサの動作

図 4 に，翻訳プロセッサの一連動作を示す．

(1) プロセッサは，$invoke(s, m, p)$ を受けると
(2) サーバ名 s でレポジトリに問い合わせ
(3) 翻訳スクリプトを読み込む
ただし，ローカルキャッシュに有効なスクリプトがあれば，これを利用する．
(4) 読み込んだ翻訳スクリプトをパースし，ランタイム環境に展開する
(5) アプリケーションから要求のあったメソッドを判別し，呼び出す
(6) 実行は，翻訳スクリプトに記述された定義に従って行われ，このときにデバイスとの通信を含む
(7) メソッド実行の結果が
(8) invoke メソッドの戻り値として，アプリケーションに返される

3.5 翻訳スクリプトレポジトリ

本研究では，Adapter をスクリプトとして表現するため，実行されるまでは通常のファイルと同じ方法で扱うことができる．そのため，普段はレポジトリに保管しておき，必要時に，ダウンロードして実行することが可能である．実際に利用するレポジトリの名前体系，管理方法，検索性能が，本提案システムの運用形態を決定する．URL で翻訳スクリプトを指定できるようにすれば，サーバ名は URL で識別され，スクリプトの配置も Web サーバの場合とほぼ同じ運用方式で行われる．検索は多くの場合，数秒以内に達成される．

4. プロトタイプ実装

本研究では，提案システムのプロトタイプ実装を作成し，その動作を検証した．動作検証およびアーキテクチャの評価は次章で行い，本章では，どのように検証システムを実装したかを述べる．本研究では，プロトタイプ翻訳プロセッサを，Java v.1.4.2 で実装した．

```
<interface>
  <command name="method1">
    <!-- stub section 1 -->
  </command>
  <command name="method2">
    <!-- stub section 2 -->
  </command>
</interface>
```

図 5 翻訳スクリプトの構造
Fig. 5 Structure of a translation script

Java クラス数は 97 である．以下，翻訳スクリプト言語，翻訳レポジトリ，制御インタフェースの規定，実験環境について述べる．

4.1 翻訳スクリプト言語の規定

本研究では，翻訳スクリプト言語を，XML 上の言語として規定した．理由は，アーキテクチャの検証としては十分な性能を保ちつつ，翻訳プロセッサを短期間で実装することが可能なためなどによる．

図 5 が示すように，XML の上位階層で，command 列 (i.e., メソッド集合) を宣言するよう定義した．それぞれの command の実装部をスタブコードと呼び，この場所に，規定された言語で，翻訳プロセッサの動作を記述する．図 6 の例でその動作を解説する．

図 6 は，図 2 のデバイス (PICNIC) へ UDP 要求を発行し，UDP 応答メッセージから，温度を取り出すプロセスを表現している．IP アドレス 203.178.135.84 宛に UDP10001 のデータグラムソケットを作成し，request の配列を送信する．その後，受信したデータグラムを response 配列として保存する．response 配列の 4 番目, 5 番目のオクテットについて，$response[4] \times 128 + response[5] \times 0.5$ を計算し，temperature の値として格納する．

ここで示したスクリプト言語のコマンドは一部である．本研究のプロトタイプ実装で作成したスクリプト言語のコマンドセットを表 1 に示す．基本的な，四則演算，ビット演算，論理演算などを定義した他，配列，XML 処理を定義した．通信機能には，データグラムソケットを定義してある．これらのコマンドを組み合わせて翻訳処理を記述する．

4.2 翻訳スクリプトレポジトリ

翻訳スクリプトレポジトリは，インターネットに分散的に存在する Web サーバ群を利用する形態を取った．すなわち，アプリケーションサーバ識別子を URL で表現し，指定された URL から翻訳スクリプトを読み込むように作成した．

4.3 制御インタフェースの規定

プロトタイプシステムでは，制御インタフェースを

```
<progn>
  ・・・・
  <setq name="response">
    <dgRecv>
      <dgSend>
        <dgSocket>
          <text>203.178.135.84</text>
          <int>10001</int>
        </dgSocket>
        <getq name="request" />
      </dgSend>
    </dgRecv>
  </setq>
  <setq name="temperature">
    <plus>
      <multiply>
        <byteArrayGet>
          <getq name="response"/>
          <int>4</int>
        </byteArrayGet>
        <int>128</int>
      </multiply>
      <multiply>
        <byteArrayGet>
          <getq name="response"/>
          <int>5</int>
        </byteArrayGet>
        <double>0.5</double>
      </multiply>
    </plus>
  </setq>
  ・・・・
</progn>
```

図 6 メソッド実装の例 (スタブコードの一部)
Fig. 6 An example of method implementation (in the stub code)

入力型と出力型に大別し，それぞれを read/write コマンドで表現した．表 2 に示すように，コマンドの引数で，要求の詳細を識別できるようにした．このインタフェース規定は，実験の間，有効であり，クライアントアプリケーションの発行するコマンドはこれに従う．

4.4 設 定 環 境

実験では，数値入力，ビット出力，パターン出力を行うサーバを設置した状況を想定した．同一観測項目を持つ二台の異なるサーバも設定した．

図 7 に実験環境を示す．ある部屋と，その屋外の環境を想定し，部屋には，(1) 温度センサ，(2)LED 表示器，(3) コントロール SW，(4) 左右の照明，(5) 左右の照度センサが設置されている．(1,3,4,5) は PICNIC に接続し，(2) は仮想デバイスとしてエミュレーションした．屋外に気象センサを設置した状況を設定し，ArmadilloWS が，各種の気象項目を観測している．ArmadilloWS は，Live E!プロジェクト[11] で遠隔地の気象データを収集するために開発されたもので，ハードウェアは Armadillo-210[2] である．それぞれのアプリケーションサーバ識別子は，図に示されている XML ファイルへのディレクトリパスであるが，この図では，プレフィッ

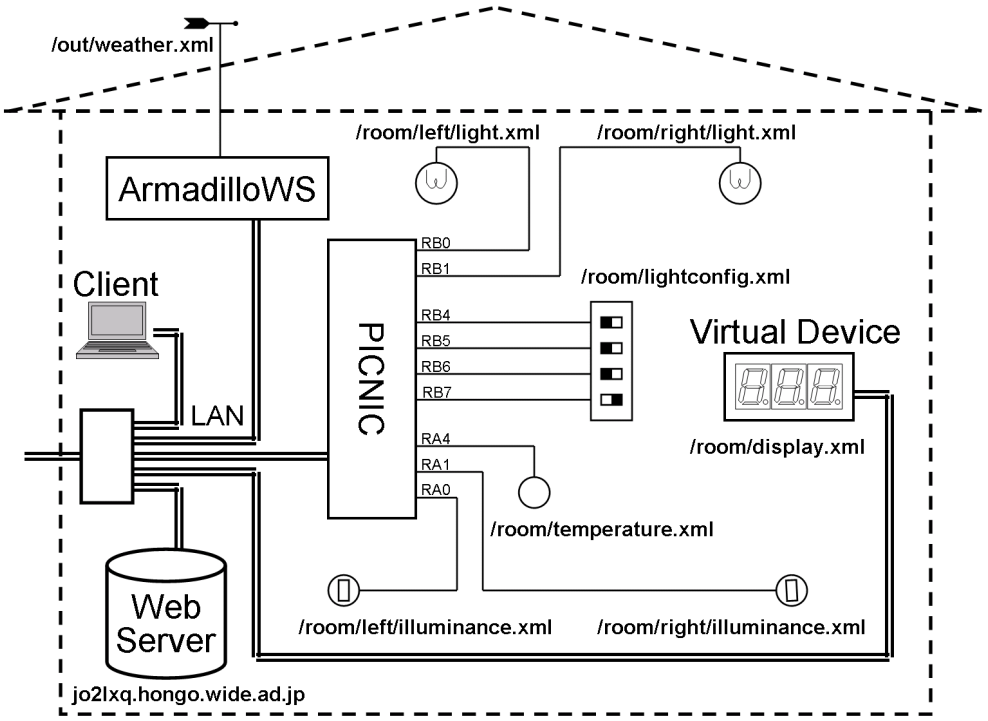


図 7 プロトタイプシステム設定環境
Fig. 7 Experimental organization of the prototype system

表 1 翻訳スクリプト・コマンドセット
Table 1 Translation script command set

Category	Command
Data Type	byte int double text null true false
Operation	plus minus multiply divide mod
Bit Operation	bitand bitor bitxor bitlshift bitrshift biturshift
Logical Operation	and or not xor
Test	eq neq lt gt lteq gteq
Type Cast	castbyte castint castdouble casttext
Branch	if
Loop	while
Procedure	progn
Variable	setq getq
XML DOM	domCreateElement domSetAttribute domSetTextContent domGetAttribute domGetTextContent
Array	byteArray byteArrayAppend byteArrayGet byteArrayLength
Socket	dgSocket dgSend dgRecv dgClose

表 2 プロトタイプシステムで規定した制御インタフェース
Table 2 API defined in the prototype system

コマンド	パラメータの例
<read type=”param”/>	temperature humidity pressure winddir windspeed rainfall illuminance config
<write type=”param0” value=”param1” />	param0=display, param1=-9.9, —, 99.9 param0=switch param1=on,off

クス (http://jo2lxq.hongo.wide.ad.jp/) が省略されている．これらのファイルには，そのアプリケーションサーバへアクセスするための手順がスクリプトで記述されており，Web サーバ (jo2lxq.hongo.wide.ad.jp) から誰でも自由に読出し利用することが可能である．図 7 では，制御クライアントは LAN 内に存在しているが，デバイス (i.e., PICNIC, Armadillo 等) には，グローバル IP アドレスが与えられており，インターネット上から制御可能である．

PICNIC の通信プロトコルは[14] を参照されたい．図 7 での Virtual Device のプロトコルは，以下のようになっている．コマンドは UDP 上のオクテット列 [X Y Z W] で構成されており，X=5A(16 進数) がディスプ

```
String id="http://jo2lxq.hongo.wide.ad.jp/room/temperature.xml";

Document doc=Xml.newDocument();
Element req=doc.createElement("read");
req.setAttribute("type", "temperature");

// invoke(id,"read",{"type":"temperature"});
Element res=RemoteGPIOServer.invoke(id , req);

String temperature=res.getTextContent();
```

図 8 翻訳機構を使用する場合の記述例
Fig. 8 Source code description with translator

レイへの書き込みコマンドを意味し，Y, Z, W は，3 桁の LED ディスプレイの上位桁，中位桁，下位桁にそれぞれ対応している．各桁表示内容は 8 ビットで指定でき，各セグメント (a, ..., g, d.p.) を bit0, ..., bit7 に接続した．[5A 5B 86 7F] であれば，21.8 と表示される．

ArmadilloWS は，UDP 上の 1 オクテット命令を発行することで，対応する気象観測項目を文字列で応答する．具体的には，温度 (10)，湿度 (11)，風向 (12)，風速 (13)，雨量 (14)，気圧 (16) となっている．すなわち，湿度の場合，[11] を発行すると，[37 38 2E 34](=78.4) などと応答を返す．

5. システム評価

5.1 制御ソフトウェアの開発効率

図 8 と図 9 に，PICNIC の温度素子からデータを取得するための，クライアントの記述例 (Java の場合) を示す．図 8 が翻訳機構を使用した場合で，図 9 が，PICNIC プロトコルを直接記述した場合である．翻訳機構を導入することで，記述量が約 30 行から 6 行に削減できている．ソースコード中で利用されている型 (クラス) の数も，11 個から 5 個に削減された．

上記の記述量の比較を，各サーバ・アプリケーションごとに行ったものを図 10 に示す．翻訳機構を使用した場合とプロトコルを直接記述した場合の Java での記述量 (行数) である．すべてのケースにおいて，記述量を 20%以下に削減できている．これは，クライアント側で行われる制御ソフトウェアの開発効率を向上させることを示唆する．

5.2 動 作 検 証

表 2 で規定した XML 制御インタフェースを用いて命令を発行すると，それが正しく翻訳されて，サーバと通信を行うことができることを確認した．以下，値設定と，値取得の場合において，その検証結果を述べる．

図 11 は，部屋の左ライトを ON/OFF する場合 (上段) と数値を表示する場合 (下段) について，発行したコマンドと，変換により生成された UDP データグラムの送信の様子である．指定されたサーバ識別子に対応するスクリプトを動的に読込み，データグラムを正しく生成して，PICNIC および Virtual Device に発行されたことが確かめられた．

図 12 は，PICNIC および ArmadilloWS の温度サーバへ取得要求を発行した後の，これらのサーバからの UDP データグラム応答およびその変換の様子である．この例では，温度変化に追従できていることを示すため，それぞれのサーバへの温度取得要求を，10 分の間隔をおき，2 度発行している．こうして，翻訳プロセッサが応答データグラムからデータを抽出し，高級

```
String name="203.178.135.102";
int port=10001;

String temperature="";
try{
     DatagramSocket s=new DatagramSocket(null);
     InetAddress addr=InetAddress.getByName(name);
     s.connect(addr, port);

     byte[] req=new byte[3];
     req[0]=(byte)0x04;
     req[1]=(byte)0xa1;
     req[2]=(byte)0x50;

     DatagramPacket p=new DatagramPacket(req,req.length,addr,port);
     s.send(p);

     p=new DatagramPacket(new byte[20],20);
     s.receive(p);

     int length=p.getLength();
     if(length==8){
          byte[] rawreceived=p.getData();
          double value=rawreceived[4]*128.0+(rawreceived[5]&0x0ff)*0.5;
          temperature=Double.toString(value);
     }else{
          // Exception
     }
     s.close();
}
catch(java.net.UnknownHostException e){
     // Exception
}
catch(java.net.SocketException e){
     // Exception
}
catch(java.io.IOException e){
     // Exception
}
```

図 9 翻訳機構を使用しない場合の記述例 (PICNIC プロトコルを直接記述した場合)
Fig. 9 Source code description without translator (Description of PICNIC protocol)

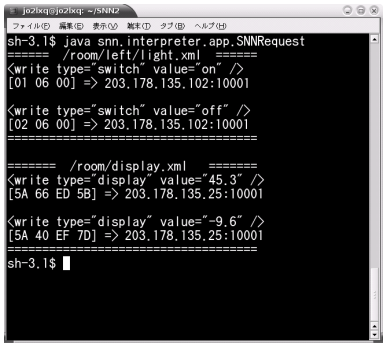


図 11 要求変換の様子
Fig. 11 Translation behavior of request messages

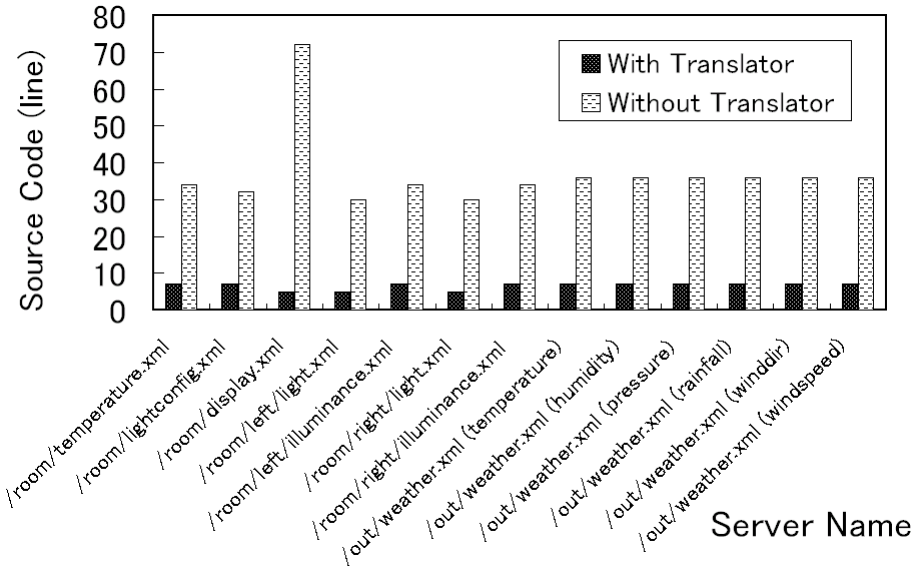


図 10 クライアント側の記述量 (行数) の比較
Fig. 10 A comparison of source code size

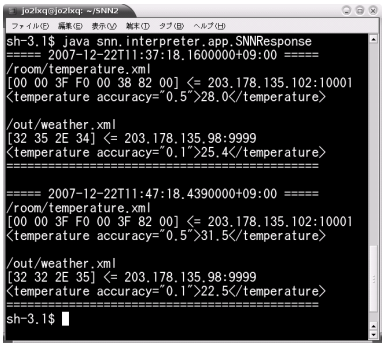


図 12 応答変換の様子
Fig. 12 Translation behavior of response messages

```
updateTD(String src,String dst){
  Document doc=Xml.newDocument();

  // create <read type="temperature" />
  Element t_req=doc.createElement("read");
  t_req.setAttribute("type","temperature");

  // temperature data acquisition from "src"
  // invoke(src,"read",{"type":"temperature"})
  Element t_res=RemoteGPIOServer.invoke(src, t_req);
  String temperature=t_res.getTextContent();

  // create <write type="display" value=temperature />
  Element d_req=doc.createElement("write");
  d_req=setAttribute("type","display");
  d_req=setAttribute("value",temperature);

  // set temperature at the display of "dst "
  // invoke(dst,"write",{"type":"display";"value":temperature})
  RemoteGPIOServer.invoke(dst,d_req);
}
```

図 13 温度表示制御ソフトウェア (updateTD)
Fig. 13 Temperature monitor control software

言語表現を生成できることを確認した．

本動作検証では，クライアントを外部ネットワークに接続した場合での動作も確認した．この実験により，翻訳スクリプトを公開すれば，誰でも自由にアプリケーションサーバを，クライアントに提供可能なことが示された．

5.3 制御ソフトウェアの再利用性

図 13 に，制御ソフトウェア updateTD を示す．updateTD は，サーバ src から温度データを取得し，サーバ dst のディスプレイに値を出力する制御ソフトウェアである．

図 13 のソースコードは実際に通信を行うデバイスの通信プロトコルを意識せずに記述することができている．これは，異なるデバイスを対象にしていてもこの制御ソフトウェア自体は再利用可能であることを示している．

以下，実際に，src に下記の引数を指定して実験を行い，この制御ソフトウェアが，デバイスに依存せず，再利用可能であることを確認した．

(1) src に”/room/temperature.xml”を指定
(PICNIC 温度素子)
(2) src に”/out/weather.xml”を指定
(ArmadilloWS の気象センサ)

両ケースにおいて，dst には”/room/display.xml”を指定した (各サーバ識別子のプレフィックスは省略)．

図 14 に，プログラムの実行の様子を示す．プログ

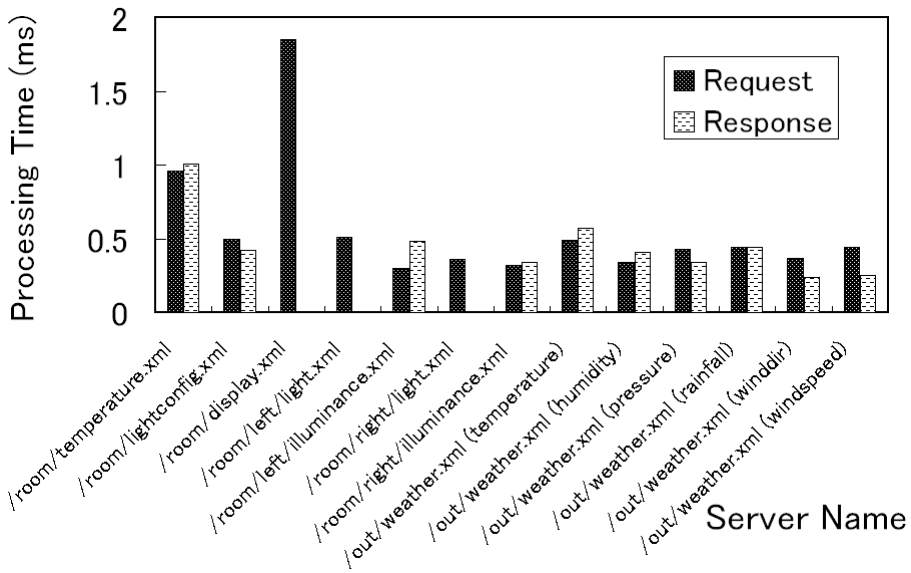


図 15 Request/Response 処理時間
Fig. 15 Processing time for Request/Response

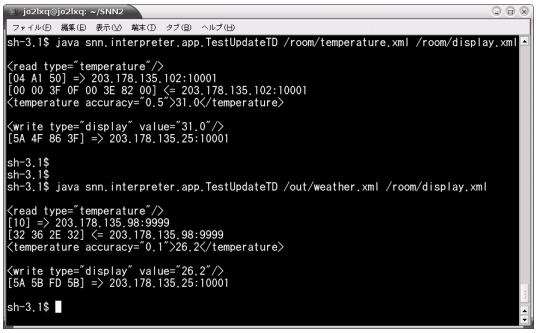


図 14 再利用性の検証実験
Fig. 14 Software reusability test

ラムは 2 度実行されており，前半が (1) のケース，後半が (2) のケースである．前半の結果から読み取れることは，PICNIC プロトコルで温度取得後，ディスプレイに表示コマンドを発行したことである．後半の結果は，Armadillo プロトコルで温度取得後，ディスプレイに表示コマンドを発行したことを示す．この間 updateTD には，変更を加えていない．こうして，この制御ソフトウェアを再利用できることが確認できた．

5.4 翻訳スクリプト処理時間

図 15 は，クライアントが，それぞれのサーバと通信を行う際の，リクエスト変換，レスポンス変換にかかる平均的な処理時間を示している．処理時間の計測にあたっては，Pentium 4, 1GByte メモリを搭載したコンピュータ (Linux 2.6.15) 上で，100 回の平均を取った．この図が示すように，評価実験のシナリオにおいては，翻訳処理にかかる時間は，平均 2ms 以内であった．

6. 考察と課題

翻訳機構アーキテクチャを用いることで，GPIO に接続されたサーバを高級言語インタフェースを使って制御でき，これがクライアント側ソフトウェアの開発効率の向上につながることがわかった．updateTD の記述，および，異なる通信プロトコルを持つ PICNIC と ArmadilloWS からの温度取得実験により，下位通信プロトコルに依存しない形で制御ソフトウェアを作成でき，これが実際に両デバイスに適用できることを実証した．これは，本研究での提案システムが，制御ソフトウェアの再利用性を向上させたことを意味している．

評価実験のシナリオにおいては，翻訳処理にかかる時間は 2ms 以内であった．本研究では，ネットワーク遅延を 100ms 程度と想定しており，2ms の処理時間は無視できる範囲だと考えている．実際には，最初の呼び出しでは，スクリプトをレポジトリから読み込む時間として，数秒がかかると見込まれる．またキャッシュされるスクリプトの数によっても読み出し遅延が懸念されるかもしれない．前者の問題は，事前ダウンロードが可能であれば，積極的に先読みすることで吸収可能だと考えられる．後者の懸念は，java.util.Hashtable 実装を使えば，スクリプト数が 100 万件程度であれば，低遅延で読み出せることが我々の行った簡単な実

験で明らかになっている．もともと遅延が見込まれている環境なため，本研究での実験シナリオの場合もそうだが，数秒程度の遅延は，実際的な運用場面では耐えられる場合も多い．

本研究では，翻訳スクリプトが既に用意されている状況を設定し，その上で，クライアント側の開発効率を考えた．実際に展開する場合には，翻訳スクリプトをどのように用意するかが課題となるが，本研究の段階では，次のような方法で翻訳スクリプトを用意する方法を考えている．(1) 機器のベンダーがテンプレートスクリプトを用意し，(2) 機器の設置者が GUI ツールを使ってテンプレートに加工を施す．末端の開発者が，翻訳スクリプトをテキストエディタなどで直接記述する必要はなく，テンプレートおよび GUI ツールを活用することで，これを比較的簡単に作成できると考えている．この際に，翻訳スクリプトを XML の上に表現することは，自動生成の観点から見て現実的なアプローチである．

今後，本提案システムを実際的な環境に適用していく上での課題としては，制御インタフェースの標準化やクライアントのアクセスコントロールに関するものなどが挙げられる．本研究では，プロトタイプシステムとして制御インタフェースの規定を行い，実験の間，この規定を標準化されたものとして扱っていた．実際的な運用においては，なんらかの標準化団体によって，グローバルユニークに規定されることが必要となってくる．本研究では，アプリケーションサーバに対応するスクリプトを誰でも公開可能で，クライアントはスクリプトを取得できれば制御可能な環境を想定していたが，実際的な環境においては，サーバに対するアクセス権限実現すべき場合がある．これらは実際に展開していく上での課題となると考えられる．

7. おわりに

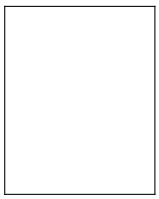
遠隔 GPIO を利用するセンサ／アクチュエータ機器を，高級言語インタフェースで統一的に制御できるようにすることは，制御ソフトウェアの開発効率，再利用性の点で重要である．本研究で提案した翻訳機構アーキテクチャは，翻訳スクリプトを翻訳プロセッサが動的に読込み実行させることで，このような環境を作り出している．提案アーキテクチャでは，翻訳スクリプトはセンサ／アクチュエータ機器と識別子で対応づけられ，レポジトリで管理される．翻訳機構により，制御ソフトウェア開発において，高級言語インタフェースで機器の操作を記述することが可能になった．これにより，(1) デバイスの操作に関する記述を 20%程度に削減できた．(2) 異なる GPIO デバイスで実装される機器にも，同一ソフトウェアが適用可能 (再利用可能) であることが確かめられた．Web 等の分散的なレポジトリ環境を利用することで，(3) スケール性の高い運用を実現できると考えられる．なお，翻訳処理にかかる時間は数 ms であり，ネットワーク遅延と比較し無視できることがわかった．実際的な展開に向けては，翻訳スクリプトの開発支援技術，制御インタフェースの標準化，アクセスコントロールの実現などの課題が残っている．

参 考 文 献

1) Aberer, K., Hauswirth, M. and Salehi, A.: Infrastructure for data processing in large-scale interconnected sensor networks, *IEEE MDM 2007* (2007).
2) Atmark Techno. http://www.atmark-techno.com/
3) Chinnici, R., Moreau, J.-J., Ryman, A. and Weerawarana, S.: Web Services Description Language (WSDL) 2.0, Working draft, World Wide Web Consortium (2007).
4) Dunkels, A., Gold, R., Marti, S.A., Pears, A. and Uddenfeldt, M.: Janus: An Architecture for Flexible Access to Sensor Networks, *ACM DIN'05* (2005).
5) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: オブジェクト指向における再利用のためのデザインパターン，pp.149–161, ソフトバンクパブリッシング (1999).
6) Grimes, R., Stockton, A., Reilly, G. and Templeman, J.: ATL COM プログラミング，chapter1, pp.16–18, Shoeisha, 1 edition (1999).
7) He, D., Bai, Y., Wang, Y. and Wu, H.: A Crop Field Remote Monitoring System Based on Web-server-embedded Technology and CDMA Service, *IEEE/IPSJ SAINT2007* (2007).
8) IANA. http://www.iana.org/
9) IEEE. http://www.ieee.org/
10) IETF. http://www.ietf.org/
11) Nakayama, M., Matsuura, S., Esaki, H. and Sunahara, H.: Live E! Project: Sensing the Earth, *LNCS*, Vol.4311, pp.61–74 (2006).
12) Tanenbaum, A. S. and Steen, M. V.: *Distributed Systems*, chapter Code Migration, pp. 103–112, Peason Education, Inc., second edition (2006).
13) 三岩幸夫：H8/3069F ネットワーク・マイコン・ボードの概要，トランジスタ技術，Vol.39, No.12, pp.142–150 (2002).
14) 落合正弘：PICNIC の製作，トランジスタ技術，Vol.38, No.1, pp.249–261 (2001).
15) 鷺山玲子，土屋　光：PICNIC を利用した He ガ

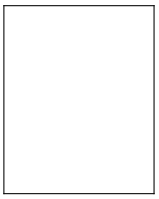
スメータの遠隔自動計測 (2002).

(平成？年？月？日受付)
(平成？年？月？日採録)



落合　秀也（学生会員）

昭和 58 年生．平成 18 年　東京大学・工・電子情報工学科 卒．平成 20 年 東京大学大学院・情報理工学系研究科　修士課程了．同年 同大学大学院・同研究科　博士後期課程，現在に至る．ファシリティマネージメントシステム，広域センサネットワーク，耐遅延ネットワークの研究に従事．



江崎　　浩

昭和 38 年生．昭和 62 年 九州大学･工･電子 修士課程了．同年 (株) 東芝 入社．平成 2 年 米国ニュージャージ州 ベルコア社．平成 6 年 コロンビア大学・客員研究員．平成 10 年 東京大学 大型計算機センター・助教授．平成 13 年 同大学大学院・情報理工学系研究科・助教授．平成 17 年 同大学大学院・同研究科・教授，現在に至る．工学博士 (東京大学)．MPLS-JAPAN 代表，IPv6 普及・高度化推進協議会専務理事，WIDE プロジェクトボードメンバ，JPNIC 副理事長，ISOC 理事．