社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

IoT 機器の環境適応取り込みの提案

山登庸次†

† NTT ネットワークサービスシステム研究所，東京都武蔵野市緑町 3-9-11
E-mail: †yoji.yamato@ntt.com

あらまし　私は，エンジニアが通常 CPU 向けに記述したソフトウェアのコードを，配置される環境に応じて，自動で変換や設定等をして，運用可能とする環境適応ソフトウェアのコンセプトを提案してきた．本稿は，リソースが少ない IoT 機器を環境適応に組み込む事を目指し，ユーザの指定する処理ロジックと IoT データから，IoT サービスを自動で構築する方式を提案する．提案方式は，ユーザの基本サービス選択に基づき，サービスに必要な共通的機能を機能ブロック検知技術で発見し，ユーザ指定の処理ロジックと組み合わせ，IoT PF と IoT GW に処理を設定することでサービス化する．手動でのサービス構築時の作成行数と，提案方式でのサービス構築時の作成行数を比較することで，提案の有効性を示す．

キーワード　環境適応ソフトウェア, IoT サービス，IoT プラットフォーム設定，効率的開発，自動構築

A proposal for adopting environmental adaptation of IoT terminals

Yoji YAMATO†

† Network Service Systems Laboratories, NTT Corporation, 3-9-11, Midori-cho, Musashino-shi, Tokyo
E-mail: †yoji.yamato@ntt.com

Abstract　We proposed the concept of environment-adaptive software, which automatically converts and configures software code written for normal CPUs in order to operate appropriately where it is deployed. This paper proposes a method for automatically building IoT services from user-specified processing logic and IoT data, aiming at incorporating IoT devices with few resources into environmental adaptation. On the basis of the user's selection of basic services, the proposed method discovers common functions required for services using function block detection technology, combines them with user-specified processing logic, and configures processing in the IoT PF and IoT GW to build a service. The effectiveness of the proposal is shown by comparing the numbers of lines written when building a service manually and building a service using the proposed method.

Key words　Environment Adaptive Software, IoT Services, IoT Platform Setting, Effective Creation, Automatic Building

1. はじめに

AI（Artificial Intelligence）の進歩も有り，少コアの CPU（Central Processing Unit）だけでなく，マルチコア CPU, GPU（Graphics Processing Unit），FPGA（Field Programmable Gate Array）等のヘテロジニアスなハードウェアが，多くのアプリケーションに用いられるようになっている．Microsoft 社は FPGA の検索利用等の取り組みをしているし [1]，Amazon 社は，クラウド技術を使い（例えば [2]- [7]），FPGA や GPU インスタンスを提供している [8]．また，ヘテロジニアスなハードウェアとして，アクセラレータだけでなく IoT 機器等の少リソースのデバイスも利用が増えている（ [9]- [19] 等）．

しかし，ヘテロジニアスなハードウェアを効率よく利用するためには，ハードウェア特性を意識したプログラム作成や設定が必要となり，大半のソフトウェア技術者にとっては，壁が高い．マルチコア CPU では OpenMP（Open Multi-Processing）[20]，GPU では CUDA（Compute Unified Device Architecture）[21]，FPGA では OpenCL（Open Computing Language）[22]，IoT 機器ではアセンブリや IoT PF の知識が必要となってくることが多い．

ヘテロジニアスなハードウェアをより活用していくためには，高度な知識を持たない通常のソフトウェア技術者でも，それらを最大限に活用できるようにするプラットフォームが必要と考える．技術者が少コアの CPU と同様のロジックで処理を記述したソフトウェアを，分析して，適用する環境（GPU，FPGA，IoT 機器等）にあわせて，適切に変換，設定を行い，環境に適

応した動作をさせることを，プラットフォームが自動で行うことが今後求められていく．

そこで，私は，一度記述したプログラムコードを，配置先環境の GPU や FPGA 等を利用できるように，変換，リソース量設定，配置決定を自動で行い，動作させる，環境適応ソフトウェアのコンセプトを提案している．合わせて，環境適応ソフトウェアの要素として，コードループ文及び機能ブロックを，GPU，FPGA に自動オフロードする方式等を提案して評価している [23]- [27]．しかし，これまでの私の環境適応検証要素技術は，マルチコア CPU，GPU，FPGA のアクセラレータとして利用するハードウェアへのオフロードだけで，IoT 機器等の少リソースのデバイスは未検討であった．GPU や FPGA の利用は高度な知識が必要なのは確かな事実だが，IoT 機器を用いた IoT サービスの構築も現在多くの知識が必要なのが現状である．

これらの背景を踏まえ，本稿は，IoT 機器を環境適応に組み込む事を目指し，ユーザの指定する処理ロジックから，IoT サービスを自動で構築する方式を提案する．提案方式は，ユーザの基本サービス選択に基づき，サービスに必要な共通的機能とユーザ指定の処理ロジックを組み合わせ，IoT PF と IoT GW に処理を設定することでサービス化する．提案方式で，IoT 機器を環境適応に組み込み，サービス化できることを，平均温度表示のサンプルアプリケーションの自動構築を通じて確認する．手動でのサービス構築時の作成行数と，提案方式でのサービス構築時の作成行数を比較することで，提案の有効性を示す．

2. 既存技術

2.1 市中技術

GPU を一般的計算にも用いる GPGPU（General Purpose GPU）[28] があり，その環境に NVIDIA は CUDA を提供している．FPGA，GPU 等ヘテロジニアスなハードウェアを共通的に扱う仕様として OpenCL がある．OpenCL や CUDA は，C 言語拡張記述として，カーネルと呼ばれる FPGA 等とホストと呼ばれる CPU の間のメモリ情報の移行などを記述するが，ハードウェア知識が必要となる．OpenCL や CUDA を理解していなくても，容易に GPU 等を用いることができるようにするため，ディレクティブで GPU 処理等を行う行を指定して，ディレクティブに基づいてコンパイラが，GPU 等のバイナリファイルを作成する取り組みがある．OpenMP や OpenACC [29] 等の仕様が，それを解釈実行する gcc や PGI [30] 等のコンパイラがある．

ヘテロジニアスなハードウェアを利用はできても，性能改善は容易でないのが現状である．Intel コンパイラ [31] 等は，並列処理可能なループを見つけ，複数のコアに処理を行わせている．しかし，データコピー等により，単に並列でループを処理しても性能が改善しないことも多い．マルチコア CPU でなく，GPU や FPGA の際はメモリも異なるためより複雑で，性能改善には OpenCL や CUDA を駆使した手動チューニングが必要となる．その中で，ループ文の GPU オフロードとして，ループ文の GPU 処理箇所探索を自動化する取り組みとして，著者

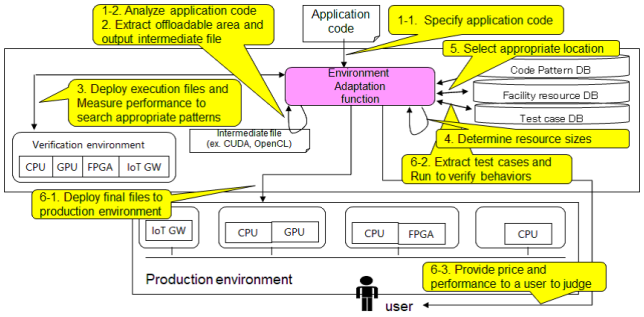


図 1　環境適応ソフトウェアの全体像

は進化計算手法である遺伝的アルゴリズム（GA）[32] を用いたオフロードを提案している．

GPU や FPGA 等のアクセラレータでなく，メモリ等のリソース量が少ない端末が多い IoT サービスも，ヘテロジニアスなハードウェアの利用である．IoT に関する標準としては，M2M プラットフォームの oneM2M [33] が標準化されている．IoT センサ等向けのプロトコルとして，HTTP より軽量な MQTT（Message Queuing Telemetry Transport）[34] が標準化されている．IoT サービスは工場，農業，運輸，ヘルスケア等で多くのサービスが構築されてきたが，IoT センサからセンシングデータを IoT GW で集約して MQTT や HTTP 等のプロトコルでクラウドに送り，データ集計や保持等の共通的処理はクラウド上の IoT PF が行い，必要なデータ加工等の処理を独自プログラムや IoT PF の機能で行って，その結果を企業幹部等のユーザにクラウドのサーバで表示する形が多い．

IoT PF としては，大手クラウド業者の Amazon や Microsoft が，AWS IoT や Azure IoT [35] 等の IoT PF をクラウドの付加的サービスとして提供しており，デファクトスタンダードに近くなっている．日本国内では，SORACOM が提供している IoT PF や，通信キャリアである NTT コミュニケーションズが Enterprise Cloud 上にシステム構築して，IoT PF 機能を提供等している．IoT GW としては，多くのセンサを集約し，標準プロトコルでデータをクラウドに送信することが大きな役割で，Intel 社の IoT GW やアットマークテクノ社の Armadillo IoT GW [36] 等があるが，IoT PF に比べ IoT GW は多くの社が提供しており，製品は極めて多い．GW で IoT データ送信だけでなく，加速度情報を迅速に分析する等，一次計算処理をする場合は，GPU を搭載した NVIDIA 社の Jetson [37] や，Java でプログラム可能な Raspberry Pi 財団の Raspberry Pi 等がある．

以前に著者は，環境適応ソフトウェアの処理として，図 1 の全体像を提案した．環境適応ソフトウェアの処理は，事業者が提供する環境適応機能を中心として，商用環境，検証環境，コードパターン DB，設備リソース DB，テストケース DB 等が連携して行われる．

2.2 本稿の課題の整理

本稿の課題を整理する．ヘテロジニアスなハードウェアの利用は手動での取組みが主流である．著者は環境適応ソフトウェアのコンセプトを以前提案し，ループ文等の GPU や FPGA

への自動オフロード方式を検証しているが，今まではアクセラレータが対象であり，リソースが少ない IoT 機器等の環境適応は検討がされていない．一方，市中の IoT サービス開発に目を向けると，セキュリティ等の共通的処理は Azure や AWS 等の IoT PF に任せ，IoT センサからデータ集約して HTTP や MQTT 等の標準プロトコルを介して IoT PF にデータを送るのは IoT GW が行う役割分担が主流となってきている．そのため，アセンブリ等のセンサ自体の知識は不要だが，IoT PF や IoT GW の高度な知識が必要となってきており，それらの知識がある SIer に IoT サービス構築を依頼することが多い．これらの背景を踏まえ，本稿では，IoT 機器を環境適応に取り込み，高度な知識が無いユーザが SIer に頼まずとも，ユーザが行いたい処理を容易に IoT サービス化できることをターゲットとする．勿論，IoT 技術自体には，開発する際の，標準やフレームワーク，エッジかクラウドのプログラム配置の検討等様々な課題があるが，環境適応という切り口で取組み，GPU や FPGA 等と同様の自動化を特徴に検討する．

3. IoT 機器の環境適応

著者は，環境適応ソフトウェアのコンセプトを具体化するために，これまでに，ループ文の GPU，FPGA 自動オフロード等を提案してきた．これらの要素技術では，深い知識を持たないユーザでも容易にサービス利用できるための自動化に取り組んできており，IoT 機器の環境適応でもその考えは参考にする．

3.1 IoT 環境適応方式の検討

IoT サービスは多くの事例があるが，一から全て開発している例は少なく，データ保持等の共通的処理は IoT PF が良く使われ，多数のセンサデータを集約して標準プロトコルで IoT PF に送信するのは IoT GW やスマホが良く使われる．しかし，IoT PF や IoT GW に関する知識は高度なもので，IoT サービスの構築はそれらの知識を持つ SIer 依頼が必要な事が多い．そこで，ユーザが行いたい独自処理はユーザが通常 CPU 向けと同様の開発で準備するが，IoT PF や IoT GW に関する深い知識が無くても，IoT サービス化できることが必要である．

そこで，ユーザは，基本となるサービスを選び，利用するユーザの IoT データとユーザ独自処理を指定すると，サービスに必要な共通的機能は類似機能ブロック検知技術により探して，ユーザ独自処理と組み合わせてサービス化する方式を提案する．

図 2 を用いて，提案方式を詳細説明する．

事前に，環境適応機能を提供するサービス事業者は，複数の IoT サービスで用いることができる基本サービスと共通的機能を準備する．基本サービスは IoT データを加工して表示や，加工して判断し通知等のアクションをする等が考えられ，複数のサービスを抽象化したものである．共通的機能は，データ蓄積や表示，メール通知等，複数のサービスで利用できる機能であり，IoT PF 等で実処理を行える実プログラムである．

(a) まず，ユーザは基本サービスを選ぶ．選ぶ際は，Web カタログ等から選ぶ形を想定するが，今回は選ぶ方法は対象外とする．基本サービスには，データを集計し，ユーザ指定プログラムで加工し，結果を表示する等のサービスの基本動作が

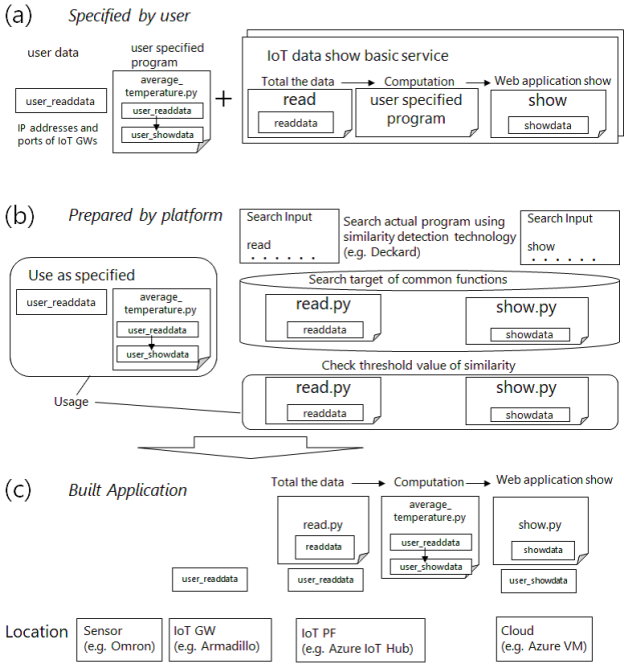


図 2　IoT 環境適応方式．(a) ユーザ指定，(b) プラットフォーム準備，(c) 生成アプリケーション．

記載されている．ここで，ユーザは基本サービスをユーザ向けに使えるようにするため，利用する IoT データとユーザ指定プログラムを指定する．例えば，IoT データは user_readdata で定義される温度データ等であり，ユーザ指定プログラムは user_readdata の温度の平均値を計算し user_showdata に変換する等のユーザが行いたい処理である．IoT データの指定では，取得する IoT GW のアドレスやポート番号等も指定される．

(b) ユーザに指定された情報を元に，サービス事業者が提供するプラットフォームの IoT 環境適応処理機能は，サービスに必要な機能を探す．データを集計する，結果を表示する等の基本サービスに記載された機能は，複数のサービスで利用できる共通的機能であるため，記載された動作に該当する実際のプログラムを，類似機能ブロック検知技術を使って検索する．例えば，Azure IoT PF でデータ集計や Web 表示を行う python プログラム等が見つかる．類似機能ブロック検知技術は，行，字句，抽象構文木，プログラム依存グラフベース等で機能ブロック間類似性を調べる技術で，抽象構文木ベースの C 言語，Java プログラム用に Deckard [38] 等がある．ユーザに指定された，IoT データとユーザ指定プログラムはそのまま利用する．

(c) 発見された共通的機能とユーザ指定プログラムを基本サービスの記載（Main 関数相当）に沿って組み合わせる事でサービス化する．IoT GW から取得された user_readdata を，発見された共通的機能の read.py で集計し，ユーザ指定プログラムの average_temperature.py で平均を計算し，その結果を共通的機能の show.py で表示する．ここで，基本サービスには各処理の動作する場所も指定されており，各処理はその場所に配置される．この基本サービスの場合，クラウドで結果を表示す

るため，値の結果表示の URL は，サービスが検証環境とクラウドで構築された後に，ユーザに通知される．ユーザは動作確認や価格確認は検証環境で行い，利用する判断をした後に，改めて商用環境に IoT サービスが構築される．

このような方式にすることで，ユーザは IoT PF や IoT GW の詳細を知らずとも IoT サービス化ができ，IoT を環境適応の対象にできる．ここで，表示やデータ集約等の共通的機能を，ユーザの独自にしたい場合等は，ユーザはカスタマイズが必要となるため，対象からは外れる．

4. 利用ツール

提案方式を確認するための実装について説明する．サンプルアプリケーションとして，前節の平均温度表示アプリケーションがあり，それで利用されるツール等を説明する．

環境センサは，Omron 2JCIE-BU01 [39] を用いる．Omron 2JCIE-BU01 は，各種センサを評価できる評価ボードで，温度，湿度，照度，気圧，騒音，3 軸加速度，eTVOC，不快指数，熱中症警戒度，振動情報等が出力できる．今回は，その中のデータから温度，湿度の値とその平均を表示に用いる．

IoT GW は，Armadillo-IoT G3L [36] を用いる．Armadillo-IoT G3L は Debian Linux 10 搭載の IoT GW である．通常の Linux 機と同様に IP アドレスが設定できる．今回は USB 接続された Omron 2JCIE-BU01 のセンサ情報を集約し，Internet を介して，クラウド上の IoT PF にデータ送信する．

IoT PF は，Azure IoT [35] を用いる．Azure IoT は数多くの細かいサービスがあるが，大きく分け PaaS の IoT Hub と aPaaS の IoT Central がある．データ処理等で，細かい処理を行うため Azure IoT Hub の REST API を用いて実装を行う．加工したデータの結果表示等は，同じ Azure クラウドの Azure VM で Web 表示される．

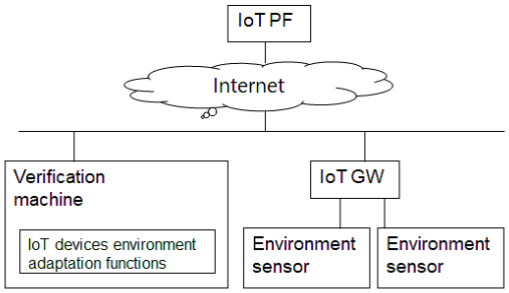
通信する IoT データの形式は JSON（JavaScript Object Notation），データ通信のプロトコルは HTTP を用いる．

共通的機能発見に，類似機能ブロック検知技術を用いるが，Deckard v2.0 [38] を用いる．Deckard は，照合対象となる部分コード（基本サービス中の記載）と，DB に登録されたコード（共通的機能）の機能ブロック類似性を，抽象構文木ベースで判定し，類似度を表示する．利用する際は 95%等の閾値以上の場合に，自動判定するようにする．

ユーザリクエストを受け，IoT サービス化の処理を行う実装は，Python 3 で行う．なお，Python は動的型付け言語であり，センサデータを JSON 文字列で送る際は，ユーザデータの事前定義が無くても，ユーザ指定の処理プログラムで変換を行うことができる．その例のように，ユーザデータの事前定義が不要な場合は，省略してよい．

5. 評　　価

提案方式でサンプルとなるアプリケーションが自動でサービス化出来る事，及び，提案方式でのサービス化で作成行数少なくサービス化出来る事を確認する．



Name	Hardware	CPU	RAM	OS
Verification machine	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX	64GB	Ubuntu 16.04.6 LTS
IoT PF	Azure IoT			
IoT GW	Armadillo-IoT G3L	Arm Cortex-A7 *2	1GB	Debian GNU/Linux 10
Environment sensor	Omron 2JCIE-BU01			

図 3　評 価 環 境

5.1 評 価 条 件

5.1.1 評価対象と評価手法

評価対象は，最も基本的なサービスである，方式提案時のデータ表示アプリケーションとする．IoT データを環境センサから集め，IoT GW が一括して IoT PF に送信し，IoT PF でユーザが指定する処理を行い，結果を Web 表示するので，多くのサービスに応用できるタイプと言える．

基本サービス：IoT データ表示サービス．

センサ：Omron 2JCIE-BU01 から 1 分毎に環境情報の温度，湿度，照度，気圧，騒音，3 軸加速度，eTVOC，不快指数，熱中症警戒度，振動情報を収集．

IoT GW：Armadillo-IoT G3L．

IoT PF：Azure IoT．データ処理は，Azure IoT Hub の REST API を介して行われ，結果等は同じ Azure の VM に渡される．

機能ブロック類似性検知：Deckard v2.0．

ユーザ指定プログラム：温度，湿度の平均値を計算して出力．

温度，湿度の平均値が Web で表示されている事の画面確認は，目視で行う．合わせて，提案方式により，容易に IoT サービス化できることを確認するため，手動で IoT サービス化した際の作成行数と，提案方式で IoT サービス化した際の作成行数を，ツールでカウントして比較する．

5.1.2 評 価 環 境

評価環境とスペックを図 3 に示す．IoT GW に環境センサは USB 接続されている．Verification machine 含め，IoT GW 等は，LAN の同一セグメント上に接続されており，Internet を介してクラウドの IoT PF と通信している．ここで，Verfication machine を用いるユーザは，サービス化したい基本サービスを選択し，ユーザ指定プログラム等を指定する．同じ Verification machine 上に搭載された環境適応機能が，共通的機能発見等をして，IoT PF や IoT GW に設定することで，IoT サービス化する．

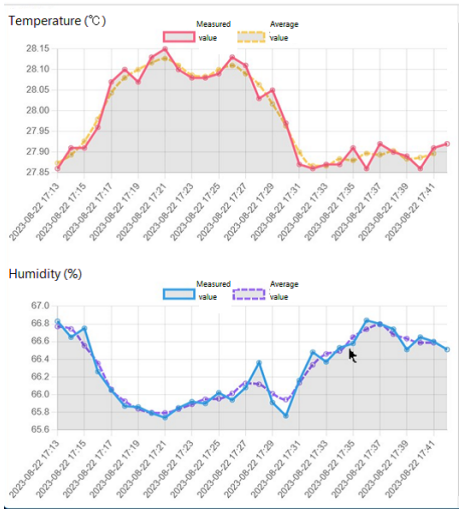


図 4　提案方式で作成の IoT サービス例

表 1　IoT サービスの作成行数比較

Application	Part	Written code (manual)	Written code (proposal)
Show temperature	Common functions	522 Lines	–
	User specified program	8 Lines	8 Lines
	User data collection	259 Lines	259 Lines

5.2　結果と考察

図 4 は，提案方式で作られた IoT サービスの画面である．基本サービスを選び，ユーザは温度，湿度の平均値計算プログラムを指定することで，期待通りの表示がされている．サービス化自体は，ユーザリクエストから秒単位の処理で終了するため，特にユーザの待ち時間は発生しない．勿論，外部サービスとして，Azure を用いているため，Azure をユーザ自身が契約する際は，審査等が必要なため，契約するまでに一定の時間と稼働が必要である．

表 1 は，提案方式で IoT サービス化をした際の稼働を比較するための，手動で平均温度表示アプリケーションを作成した場合の作成行数と，提案方式で作成した場合の作成行数を比較している．アプリケーションは Python 中心であるが，手動作成の場合は Omron センサデータの収集部分が 259 行，温度，湿度の平均値を計算する部分が 8 行であり，共通的機能である IoT PF へのデータ送信部が 20 行，データ集計部が 123 行，Web 可視化部が 379 行である．提案方式での作成の場合は前 2 つだけの 267 行となっており，1/3 に削減していることが分かる．

著者の環境適応ソフトウェアは，これまで，変換，リソース量設定，配置設定，検証，運用中再構成等，一連の流れは検証してきたが，適応対象は，GPU，FPGA，マルチコア CPU 等のアクセラレータだけであり，IoT 機器のようなリソースが少ない端末は検討されていなかった．今回，通常 CPU 向けプログラムしか知識が無いユーザでも容易に IoT サービス化できる，IoT 環境適応方式を提案することで，対象を大きく増やすことができたため，利便性が大きく進展したと言える．

コストを考察する．従来，IoT サービス化には，IoT PF 等に関する豊富な知識が必要で，SIer にサービス構築を依頼することが多かった．事実，著者も，Azure IoT に Armadillo-IoT G3L からデータ送信して処理する習熟に数か月かかっており，容易に開発できることが必要であることを理解している．今回の評価では，サンプルとなるアプリケーションは手動作成の場合は 789 行であり，提案方式の場合は 267 行となっている．これは，提案方式は，ユーザが行いたい処理のユーザ指定プログラムとユーザデータの収集部分だけ用意すれば，Azure でのデータ集約や画面表示は，発見された共通的機能が行ってくれるからである．行数削減は重要な要素だが，IoT PF の習熟が殆ど不要な点が大きいと考えており，数人月単位のコスト削減の効果があると言える．IoT GW に関連しては，データ通信プロトコル等は MQTT や HTTP で標準化されており，データ送信部等は多くの IoT GW で共用できる．用いるセンサのデータ収集部は実装が必要である．

サービス運用開始までのタイムスパンを考察する．共通的機能検索等のプラットフォームの処理時間は殆ど発生していない．IoT データ収集のためのセンサ配布や，基本サービスを Web カタログ等を参照して選ぶ時間の方が長大である．センサ配布は IoT サービスに依存するため難しいが，基本サービス選定方法は今回未検討であるため，次の課題として検討が必要である．また，今までの環境適応は，GPU や FPGA 等のアクセラレータにオフロードして高速化することが主目的であり，アプリケーションの処理内容自体は変化なかったが，IoT 環境適応では，実際の処理内容のユーザ確認は必要となるため，その時間も必要である．ただし，それらの時間を考慮しても，IoT PF の習熟の時間等は削減されるため，サービス開始時間は大きく短縮されると言える．

6.　ま　と　め

本稿では，私が提案している，ソフトウェアを環境に自動適応させ GPU，FPGA，IoT 機器等を適切に利用して，アプリケーションを運用する環境適応ソフトウェアの要素として，リソースが少ない IoT 機器の環境適応方式を提案した．

提案方式では，事前にサービス事業者が，サービスの元となる基本サービスと共通的機能を準備しておく．ユーザのサービス化リクエスト時は，ユーザは基本サービスと入れ込みたいユーザ指定処理ロジックを指定し，その指定に基づき，サービスに必要な共通的機能を類似機能ブロック検知技術 Deckard で検索して，検索した共通的機能をユーザ指定処理ロジックと組合わせ，IoT PF と IoT GW に処理を配置することでサービス化する．提案方式により，IoT 機器を環境適応に組み込み，自動で IoT サービス化できることを，平均温度表示のサンプルアプリケーションの構築を通じて確認した．サンプルアプリケーションを手動で構築した際は作成行数は 789 行であり，提案方式での記述は 267 行であり，1/3 に作成量を減らせる有効性を確認した．

文　　献

[1] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," Proceedings of the 41th Annual International Symposium on Computer Architecture

(ISCA'14), pp.13-24, June 2014.
[2] O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, No.3, 2012.
[3] Y. Yamato, "Cloud Storage Application Area of HDD-SSD Hybrid Storage, Distributed Storage and HDD Storage," IEEJ Transactions on Electrical and Electronic Engineering, Vol.11, Issue.5, pp.674-675, DOI:10.1002/tee.22287, Sep. 2016.
[4] Y. Yamato, "Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," Journal of Information Processing, Vol.25, No.1, pp.56-58, Jan. 2017.
[5] Y. Yamato, "Server Selection, Configuration and Reconfiguration Technology for IaaS Cloud with Multiple Server Types," Journal of Network and Systems Management, Springer, DOI: 10.1007/s10922-017-9418-z, Aug. 2017.
[6] Y. Yamato, et al., "Fast Restoration Method of Virtual Resources on OpenStack," IEEE Consumer Communications and Networking Conference (CCNC 2015), pp.607-608, Jan. 2015.
[7] Y. Yamato, "Automatic Verification for Plural Virtual Machines Patches," The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, 2015.
[8] AWS EC2 web site, https://aws.amazon.com/ec2/instance-types/
[9] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," Technical report of General Electric (GE), Nov. 2012.
[10] M. Takemoto, et al., "Service Elements and Service Templates for Adaptive Service Composition in a Ubiquitous Computing Environment," The 9th Asia-Pacific Conference on Communications (APCC 2003), Vol.1, pp. 335-338, 2003.
[11] Y. Yamato, et al., "Study and Evaluation of Context-Aware Service Composition and Change-over Using BPEL Engine and Semantic Web Techniques," IEEE Consumer Communications and Networking Conference (CCNC 2008), pp.863-867, 2008.
[12] Y. Yamato, et al., "Study of Service Processing Agent for Context-Aware Service Coordination," IEEE International Conference on Service Computing (SCC 2008), pp.275-282, July 2008.
[13] H. Sunaga, et al., "Service Delivery Platform Architecture for the Next-Generation Network," ICIN 2008, Oct. 2008.
[14] H. Noguchi, et al., "Autonomous Device Identification Architecture for Internet of Things," 2018 IEEE 4th World Forum on Internet of Things (WF-IoT 2018), pp.407-411, Feb. 2018.
[15] Y. Yamato, et al., "Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles," International Journal of Information and Electronics Engineering, Vol.6, No.5, pp.289-293, 2016.
[16] Y. Yamato, et al., "Method of Service Template Generation on a Service Coordination Framework," 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov. 2004.
[17] H. Noguchi, et al., "Device Identification Based on Communication Analysis for the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2019.2910848, Apr. 2019.
[18] H. Noguchi, et al., "Distributed Search Architecture for Object Tracking in the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, Oct. 2018.
[19] M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.
[20] T. Sterling, et al., "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN: 9780124202153, 2018.
[21] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, ISBN: 0131387685, 2010.
[22] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.
[23] Y. Yamato, "Study and Evaluation of Improved Automatic GPU Offloading Method," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1941010, June 2021.
[24] Y. Yamato, "Study and Evaluation of Automatic GPU Offloading Method from Various Language Applications," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1971666, Sep. 2021.
[25] Y. Yamato, "Improvement Proposal of Automatic GPU Offloading Technology," The 8th International Conference on Information and Education Technology (ICIET 2020), pp.242-246, Mar. 2020.
[26] Y. Yamato, "Proposal of Automatic Offloading for Function Blocks of Applications," The 8th IIAE International Conference on Industrial Application Engineering 2020 (ICIAE 2020), pp.4-11, Mar. 2020.
[27] Y. Yamato, "Automatic Offloading Method of Loop Statements of Software to FPGA," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1916020, Apr. 2021.
[28] J. Fung and M. Steve, "Computer vision signal processing on graphics processing units," 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, pp.93-96, 2004.
[29] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.
[30] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.
[31] E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP (EWOMP'02), Sep. 2002.
[32] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.
[33] oneM2M web site, https://onem2m.org/technical/published-specifications/release-4
[34] MQTT web site, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html
[35] Azure IoT Hub web site, https://learn.microsoft.com/ja-jp/azure/iot-hub/iot-concepts-and-iot-hub
[36] Armadillo web site, https://armadillo.atmark-techno.com/about/iot-gw
[37] Jetson web site, https://www.nvidia.com/ja-jp/autonomous-machines/embedded-systems/jetson-nano/
[38] Deckard web site, https://github.com/skyhover/Deckard
[39] Omron sensor web site, https://github.com/nobrin/omron-2jcie-bu01/blob/master/README_ja.md