Vol.2016-ARC-219 No.9
Vol.2016-SLDM-175 No.9
Vol.2016-EMB-40 No.9
2016/3/24

分散組込みシステム向き Webベース開発環境 Blue-SkyのROS拡張

アモーンタマウット プラウィーン[1,a)]　早川 栄一[2,b)]

概要: 我々が開発している分散組込みシステム向き Web ベース開発環境 Blue-Sky から、ロボットフレームワーク ROS 対応の各種デバイスを利用するためにに拡張した．Blue-Sky と ROS とでは通信モデルが異なることから，Blue-Sky を拡張し，ROS の publish/subscribe 通信をハンドリング可能にした．また，roscore が持つ通信に関する情報を取得することで，Blue-Sky と ROS のノードが混在した環境において，両ノードの状態の監視や通信状態のトレースを行う事が可能になった．

キーワード: CPS, IoT, Blue-Sky, Web, ROS, 分散組込みシステム

ROS Extension of Blue-Sky Web based Development Environment for Distributed Embedded Systems

Praween AMONTAMAVUT[1,a)]　Eiichi HAYAKAWA[2,b)]

Abstract: We extended Blue-Sky that we are developed, to support ROS that is a robotic framework, featuring the web based development environment for distributed embedded systems. The ROS is a framework based on publish/subscribe messaging model. Whereas, Blue-Sky has a different messaging model with ROS. In order to solve the problem, we extended Blue-Sky for enabling publish/subscribe messaging model of ROS , then we collect the messaging data from roscore in order to reduce the confusion among Blue-Sky and ROS nodes with tracing and monitoring the both nodes accessing status.

Keywords: CPS, IoT, Blue-Sky, Web, ROS, Networked Embedded System

1. はじめに

近年，CPS[1] や IoT[2] といった，ネットワークに接続された複数の組込みデバイスを用いてセンサデータの取得や加工，物理世界の操作などを行うシステムが数多く登場している．これに伴い，組込みシステムおよびネットワークシステムの両方に精通したエンジニアの育成 [3][4] が必要とされている．特に，従来のシンプルなセンサやアクチュエータだけではなく，複数のセンサが搭載されたセンサボードや，ロボットといったデバイスを遠隔から操作することも行われている．

このような状況に対して，拓殖大学工学部早川研究室では，分散組込みシステムの教育や開発を対象としたフレームワークである Blue-Sky[5] の開発を行っている．Blue-Sky は，IoT を主な対象として，Linux が動作する組込み機器のマシン管理および監視システムと，通信状態やマシンの監視を行うゲートウェイ，および Web ブラウザベースの開発環境を提供することで，インストールのコストを下げつつ，多くのマシンの操作および管理を可能にするフレームワークである．これを用いて，演習室などでの IoT の教

[1] 拓殖大学大学院工学研究科電子情報工学専攻
Takushoku University, Graduate School of Engineering, Electronics and Information Sciences
[2] 拓殖大学工学部情報工学科
Takushoku University, Faculty of Engineering, Department of Computer Science
[a)] praween@hykwlab.org
[b)] hayakawa@cs.takushoku-u.ac.jp

育環境 [5] およびシステム開発を行うことを想定している．

しかし，Blue-Sky では，組込み機器に搭載するデバイスは，GPIO や SPI といった低レベルなインタフェースへアクセスする API だけを提供している．これは，当初のターゲットデバイスが Raspberry Pi などの CPU ボードであり，シンプルなデバイスだけを対象としていたためである．しかし，最近の IoT システムでは，センサボードやロボットなど高機能かつ複雑なデバイスプログラミングを必要とするデバイスを扱うことも増えてきている．このことから，Blue-Sky においても，このようなデバイスに対応する必要が出てきた．

このような問題に対して，我々はロボットアプリケーション開発のフレームワークである ROS[6][7] に着目した．ROS は，ハードウェア抽象化機構，デバイスドライバ，ライブラリ群，可視化ツール，pub/sub の基づいたメッセージ通信，およびパッケージ管理を提供し，複数コンポーネントにおけるアプリケーション開発をサポートしているオープンソースプロジェクトである．現在，多くのデバイスベンダが ROS 対応のライブラリを公開していることから，デバイスに関する複雑なプログラミングを行うことなく，センサやアクチュエータ，さらにはロボットなどを利用することができる．

そこで，本研究の目的は，IoT の学習者を対象として，Blue-Sky から ROS 対応の各種デバイスを扱えるように，システムを拡張することである．ROS と Blue-Sky では通信のモデルが異なることから，これらの違いを吸収する機構を設計実装した．また，ROS および Blue-Sky のノードが混在する環境において，Blue-Sky が提供する監視および可視化システムを利用できるようにすることで，Blue-Sky の開発環境内でシステム全体での動作を理解しやすくすることが可能になった．

2. ROS

本章では ROS について述べる．

ROS は，ロボットソフトウェアの開発を目的としたフレームワークである．ROS は，コンピュテーションを行うプロセスである各ノードを XML-RPC[8] ベースにおける publish/subscribe (pub/sub) 通信することによって動作する．

本報告で対象とする ROS のコンポーネントは次の三つである．

1) **プロジェクトの作成ツール群** 作成ツール群は CMake マクロのビルドパッケージである catkin[9] のコマンド群によってワークスペースや複数階層で作成可能なパッケージとして構成される．ROS パッケージはこのマクロ群で宣言されたパッケージライブラリを指定することによって catkin によるビルドが可能である．これらのツール群やライブラリはシステム開発を行う OS 上にインストールする必要がある．
2) **ノードの可視化ツール** 可視化ツールとしては rqt_graph[10] を提供している．ROS のノード [7] は実行されているプロセスのことを定義している．さらに，ROS はモジュール化されていて，ノードはソフトウェアのモジュールや実行自体そのものとすることも可能である．これで，rqt_graph は各 ROS のコンピューテションというノードの状態や通信間の繋がりを可視するプラグインである．
3) **pub/sub の通信を処理する publisher** ROS は，roscore[11] と呼ぶ基本的なプログラムやノードの集合である．ノード間で通信する場合には，roscore は ROS のパッケージで作成されたプログラムやノードのデータを publish/subscribe(以下，pub/sub) のメッセージング方式で収集し，“/rosout”[12] のトピックでログを publish する機能を持つ．ちなみに，pub/sub 通信モデルは，非同期モデルでありメッセージの送信者である publisher は受信者である subscriber の情報を持たずにメッセージを送る．subscriber は，publisher の情報を持たずに，受信対象となるクラスだけを指定し，そのクラスのメッセージだけを受け取る．このため，送信者と受信者とは疎結合であり，スケーラブルなシステムを作ることができる．一つの ROS の基に作成されたすべてのプログラム群やノードの状態は，roscore から取得することが可能である．

3. 問題分析

我々の開発したフレームワークである Blue-Sky を ROS に適用する場合の問題点は次のとおりである．

1) **Blue-Sky と ROS とでの提供する機能の違い:**
ROS は，各 OS に属する言語のライブラリやそのビルドシステムまで統合されたシステムである．一方，Blue-Sky は，サーバが提供する Blue-Sky API を用いてデバイスの機能を呼び出すことで，アプリケーションを構築し，実行する．ROS のロボットアプリケーションから Blue-Sky の機能をアクセス可能にするには，低オーバヘッドでこの二つの仕組みを繋ぐ必要がある．
2) **Blue-Sky と ROS との通信:**
ROS のノードの状態を取得する場合，roscore との通信が必要となる．roscore との通信は，pub/sub 通信モデルによるが，Blue-Sky は HTTP による通信を基本としていることから，この二つの通信モデルを変換する必要がある．Blue-Sky の開発観葉は，ブラウザ上で JSON ベースで通信を行うために，ROS がベースにしている XML-RPC のデータ形式をそのまま扱うことができない．また，Blue-Sky の開発環境 [5] は，Javascript を利用したブラウザアプリケーションなの

Vol.2016-ARC-219 No.9
Vol.2016-SLDM-175 No.9
Vol.2016-EMB-40 No.9
2016/3/24

で，roscore の XML データ形式を利用することが難しい．

3) **Blue-Sky 上での ROS の可視化のための拡張:**
ROS は rqt_graph と呼ぶ可視化ツールを備えているが，単一のツールであり，システムのインストールや設定が必要であり，複数台数を管理しなければいけない教室環境においては管理コストが上がる．また，複数のツールをまたいで利用することは，利用者の利便性を阻害することになる．

4. 設計方針

前章の問題分析に対して，我々が開発した Blue-Sky の環境を ROS に適用可能に拡張することで対応する．拡張する部分は次の2 点である．

1) **Blue-Sky の開発ライブラリの ROS 拡張:**
Blue-Sky で組んだ IoT アプリケーションから ROS のノードが提供するデバイスに低オーバヘッドでアクセスできるようにする．これにより，Blue-Sky から ROS が提供する高水準な API を備えたデバイスを利用することが可能になる．

2) **Blue-Sky の可視ツールの ROS 可視ツール拡張:**
Blue-Sky の Web ベース開発環境でブラウザ上に Blue-Sky のノードと ROS のノードの両方を可視化可能にする．これにより，開発者や学習者は各ノードの状態や通信を監視することができる．IoT では，多くの機器の状態の監視や，通信オーバヘッドによるプログラムでの対応などが必要になることから，そのような課題を容易に行えるようにする．

5. 設計

5.1 全体構成

図 1 にシステムの全体構成を示す．本報告では，利用者は Blue-Sky の開発環境およびノードを利用し，さらにそこから ROS が動作するノードを利用することが可能になる．Blue-Sky のサーバは，ROS と通信をするために，Multi-purpose Handler と呼ぶプロトコル変換の機構を実装した．この機構を用いて，roscore のトピックをハンドルすることで，ROS ノードのデータを取得することが可能になる．

そして，ROS ノードの状態は本システムの skycoder[5] で可視化する．可視化データは JSON 形式にするため，Blue-Sky サーバの Multi-purpose Handlers で，XML-RPC を JSON-RPC[13] に変換する機能を追加することで行う．

5.2 ROS-Blue-Sky プロキシ

Blue-Sky から，ROS の各ノードを呼び出すために，roscore が動作するシステム上で Blue-Sky との通信を行うプロキシを起動する．プロキシは次の役割を持つ．

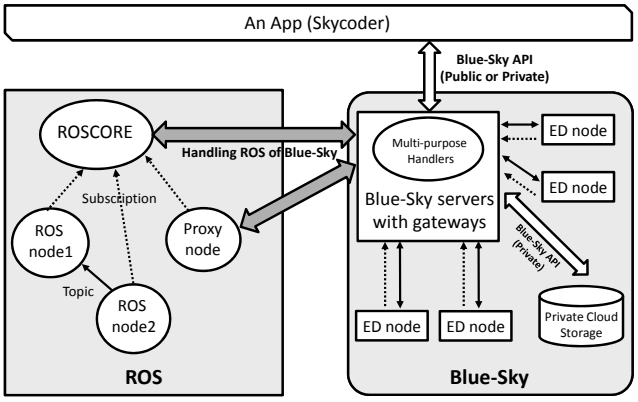

図 1 Blue-Sky の ROS 拡張モデル
Fig. 1 ROS Extensional Model of Blue-Sky

- Blue-Sky からのトピックアクセス要求を ROS の各ノードへの通信に置き換える
- ROS から生成されたデータを Blue-Sky gateway に中継する
- roscore に対して，各ノードの操作を依頼する
- ノード名の管理
- 通信オーバヘッドの測定

プロキシは，ROS のノードの一つとして扱われている．このような設計にすることで，他のノードやライブラリを変更することなく，その機能を利用することができる．また，プロキシは，pub/sub 通信の時間情報を取得し，それを Blue-Sky へ送信することでノードの通信オーバヘッドを測定する．

```
# Flashing LED and getting the light
# sensor data for 10 times.
loop 10
lsn 172.16.4.103 gpio set 22 1    # ED の利用node
sleep 300
lsn 172.16.4.222 get a 3 0    # ED の利用node
sleep 600
lsn ros://node0/onLed   # ノードの利用ROS
sleep 600
lsn ros://node0/offLed   # ノードの利用ROS
sleep 600
end
```

図 2 コマンドラインによる ROS ノードへのアクセス例
Fig. 2 Example to access ROS node in command line interface.

```
/**
 * Flashing LED and getting the light sensor data
 * for 10 times with API oriented function.
 */
for(var i = 0; i < 10; i++) {
  l_sensornetwork("172.16.4.103", "gpio",
        "set", "22", "1");
  Sleep(300);
  l_sensornetwork("172.16.4.222", "get",
        "a", "3", "0");
  Sleep(600);
  l_sensornetwork("ros://node0/onLed");
  Sleep(600);
  l_sensornetwork("ros://node0/offLed");
  Sleep(600);
}
```

図 3 Javascript による ROS ノードへのアクセス例
Fig. 3 Example to access ROS node in Javascript

図 2 は BlueSky から ROS ノードを利用する例である．

skycoder のコマンドラインから利用する場合，ros://ノード名/トピック名で，ros のノードとトピックを指定する．例では，node0 において onLed と offLed というトピックがすでに定義されている場合は，Blue-Sky の ED ノードと同じアクセスメソッドによってアクセスすることができる．skycoder で Javascript を用いたプログラムの例を図 3 に示す．これも，Blue-Sky のノードと同じメソッドによってアクセスすることが可能である．

5.3 roscore からの情報取得機能

roscore から通信に関する情報取得を行うために，Blue-Sky サーバ上に Multi-purpose Handlers と呼ぶハンドラを提供する．本機能は roscore の XML-RPC のメソッドを Blue-Sky サーバから呼び出すために用いる．

```
<?xml version='1.0'?><methodCall>
<methodName>getSystemState</methodName>
<params><param>
<value><string>/rosnode</string></value>
</param></params></methodCall>
```

図 4 Multi-purpose Handlers に追加した roscore に対するハンドリング RPC の呼び出しメゾッド
Fig. 4 Calling Method of RPC Handling Roscore Added to Multi-purpose Handlers.

図 4 に ROS で用いる XML-RPC の呼出しメソッド例を示す．このデータを本サーバのハンドリングで roscore に渡し，ROS のノード XML データを取得する．そして，図 5 のように JSON のログに変換し，Blue-Sky 上のストレージへ格納したり，skycoder に提供する．

```
XML-RPC response data:
<?xml version='1.0'?>
<methodResponse><params><param><value>
<array><data><value><int>1</int></value>
<value><string>current system state</string>
</value>...</param></params></methodResponse>

converted to JSON-RPC
{''ETLog'':{''ros'':{''jsonrpc'':{''jsonrpc'':
''2.0'',''result'':{{''params'':{''param'':
{''value'':{''array'':{''data'':{''value'':[{''int'':1},
{''string'':''current system state''}...}}}}}
```

図 5 Serializing the Response Data to JSON-RPC
Fig. 5 応答データを JSON-RPC に変換する．

5.4 ROS ノード可視ツールの設計

ROS ノードについても，Blue-Sky の ED ノードと同じように可視化可能なように，skycoder 上の可視化ツールを拡張した．ROS のノードをブラウザ上で可視化するには，本システムの Multi-purpose Handlers によって変換された JSON 形式の ROS ノードデータを取得し，本システムで扱っている可視化ライブラリである visjs[14] の dataset[15] にマッピングした後に Network[16] 可視化モジュールで表示する．マッピングオーバヘッドを減らすために，ROS ノードが起動されないか roscore が起動していない場合は可視化の処理をしないように設計する．

これによって， ROS ノードに対しても，従来の可視化ツールのように開発環境上にインストールする必要なく利用することが可能になった．

6. 評価

本節は実現および評価について述べる．

表 1 ROS 実験環境
Table 1 Environment for ROS Implement Experiment.

System Name	Specification	Unit
ROS	・ indigo	1
GUEST OS	・ Ubuntu 14.04.3 LTS	1
HYPERVISOR	・ VMware Player v. “7.1.2”	1
NATIVE OS	・ Ubuntu 12.04.5 LTS ・ Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz.	1

表 1 に本システムの実装実験における ROS 実験環境を示す．ROS は indigo を仮想マシンにインストールし，次の二つのワークスペースを作成する．

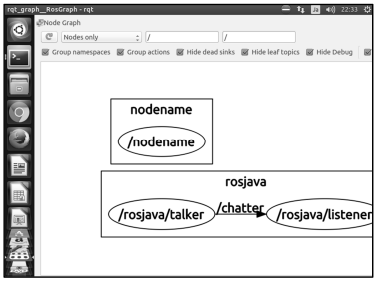

図 7 rqt_graph of ROS
Fig. 7 ROS の rqt_graph

- **ワークスペース 1:** nodename のパッケージを作成し nodename というノード名を初期化する std_msg の標準出力にメッセージを送信するアプリケーション
- **ワークスペース 2:** rosjava のパッケージを作成し，デフォルトに提供されている Talker と Listener のアプリケーション

roslunch や roscore を起動させて，各ワークスペース

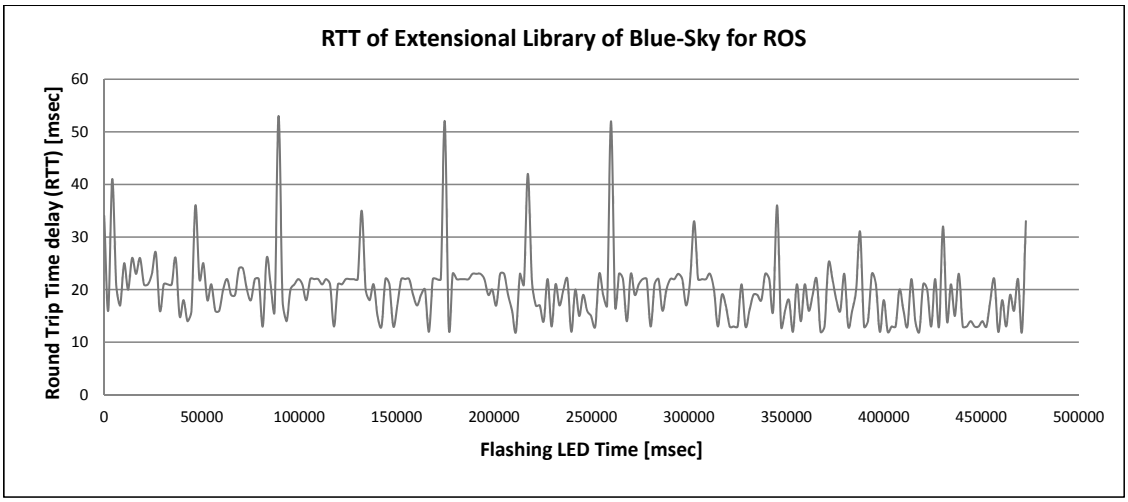

図 6　Blue-Sky の ROS 拡張ライブラリアクセスの往復遅延時間
Fig. 6　RTT of Flashing LED Execution of Extention Library of Blue-Sky for ROS

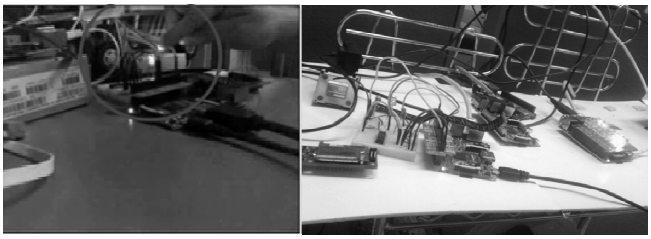

図 8　Physical Embedded Device Nodes
Fig. 8　物理的な組込み機器のノード

のアプリケーションを起動する．ここで，図 7 のように rqt_graph で ROS ノードを表示する．

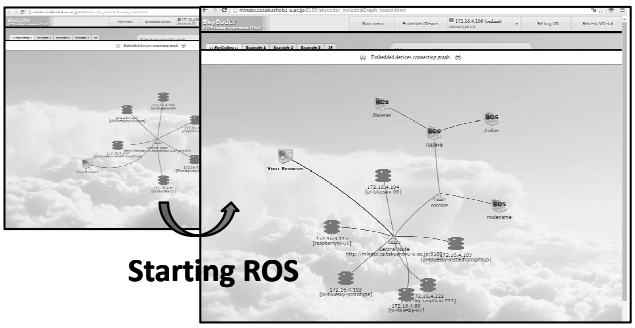

図 9　Blue-Sky の skycoder で ROS ノードの可視
Fig. 9　Visualizing the ROS Nodes with Skycode of Blue-Sky

そして，表 2 に示したように Blue-Sky のシステムを起動し，ワークスペース 2 のアプリケーションの初期化を行った後，本拡張ライブラリを用いてアプリケーションを作成する．

1) Blue-Sky の ROS 拡張ライブラリを ROS のパッケージにおける “lib” に入れる．
2) 表 2 に示したように 100Base-TX で接続した Raspberry Pi 上で動作する Blue-Sky の組込み機器ノード上の LED を 2 秒ずつに点滅するアプリケーションを作成する．

表 2　Blue-Sky 実験環境
Table 2　Environment of Blue-Sky.

System Name		Specification	Unit
Skycoder	Browser	・ Google Chrome v. “48.0.2564.103 m”	1
	OS	・ Windows 7	
		・ Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz	
Blue-Sky servers	JAVA	・ java version “1.7.0_55” OpenJDK Runtime Environment(IcedTea 2.4.7)	2
	OS	・ Ubuntu 13.10	
		・ AMD Athlon(tm) II X2 260 Processor	
Raspberry Pi	JAVA	・ java version “1.8.0”	4
		・ Java(TM) SE Runtime Environment	
	OS	・ Debian GNU/Linux 7.8 (wheezy)	
		・ Arm (BCM2708)	
Intel Edison	JAVA	・ java version “1.8.0_25”	1
		・ Java(TM) SE Runtime	
	OS	・ Linux 3.10.17-poky-edison+	
		・ Genuine Intel(R) CPU 4000 @ 500MHz	
KZM-A9	Platform	・ Android v. “2.2”	1
		・ Linux Kernel v. “2.6.29”	

これによって，図 8 のように物理的な組込み機器の LED が 2 秒ずつに点滅することで動作を確認した．さらに，本システムの skycoder 上に図 9 はワークスペース 1 とワークスペース 2 で作成した ROS のノードを表示することができた．

8 分間実行した場合の RTT は，図 6 のようになった．本拡張ライブラリの RTT は平均 19.897 ミリ秒であり，下限値は 12〜53 ミリ秒と小さい値である．このことから，拡張ライブラリとしては十分な性能を持っているものと思われる．拡張ライブラリとして有効であると考える．

さらに，ローカルなマシンで rqt_graph を利用不可能な ROS の開発環境においても，本可視ツールを利用することによって，インストールせずに ROS ノードを起動するだけで，ROS ノードと本システムのノードの同時に容易にトレースや監視を行うことが可能になった．これは遠隔地に設置されたデバイスの監視や，教室のように多くのデバイスを扱う環境では有効であると考えられる．その一方，各ノードの通信状態の表示は不十分であり，この部分には改善が必要であることが明らかになった．

7. 関連研究

ROS[6][7] の rqt_graph[10] は OS 上の GUI で ROS ノードを可視することで，そこで開発した ROS アプリケーションにおける roscore[11] のトピックを解析し，ノードとして表示するものである．

遠隔地に設置されたデバイスなどのようにディスプレイを持たない環境では，rqt_graph で可視することが難しい．遠隔地のロボットやセンサ類を操作することで，IoT の効果を学ばせたい場合にはこのような機能では不十分である．リモートデスクトップ [17] を用いることも可能であるが，台数が増えた場合には管理が難しいことや，データ転送に関する負荷や遅延が高い．

さらに，rqt_graph はネットワークに接続するノードを想定していないため，ネットワーク間に実施されるノード間の通信に関する往復遅延時間の測定を含めて，トレースや監視が不可能である．これに対して，本システムでは，通信時間を含めた測定や，ノードの監視が可能であり，組込みシステムとネットワークの両面を理解する必要がある IoT の学習環境としては有効であると考えられる．

8. おわりに

本原稿は分散組込みシステム向き Web ベース開発環境 Blue-Sky の ROS 拡張の方式と実装について述べた．Blue-Sky に対して，ROS 上のデバイスを扱うための拡張を行い，ROS の状態の可視化機能を追加した．これによって，Blue-Sky が備える Web ブラウザベースの開発／実行環境上で，ROS と Blue-Sky のノードの操作や，状態監視，通信のトレースを透過的に行うことが可能になった．今後の課題は，実利用における評価と，可視化の見やすさの改善およびプログラミングシステムの改善である．

参考文献

[1] Lee, E.: Cyber Physical Systems: Design Challenges, *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pp. 363–369 (online), DOI: 10.1109/ISORC.2008.25 (2008).

[2] Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M.: Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions, *Future Gener. Comput. Syst.*, Vol. 29, No. 7, pp. 1645–1660 (online), DOI: 10.1016/j.future.2013.01.010 (2013).

[3] 岩野和夫，高島洋典: サイバーフィジカルシステムと IoT(モノのインタネット)，情報処理，Vol. 57, No. 11, pp. 826–834（オンライン），DOI: 10.1241/johokanri.57.826 (2015).

[4] Rajkumar, R. R., Lee, I., Sha, L. and Stankovic, J.: Cyber-physical Systems: The Next Computing Revolution, *Proceedings of the 47th Design Automation Conference*, DAC '10, New York, NY, USA, ACM, pp. 731–736 (online), DOI: 10.1145/1837274.1837461 (2010).

[5] アモーンタマウットプラウィーン，早川栄一: 分散組込みシステム向き Web ベース開発環境，組込みシステムシンポジウム 2015 論文集，Vol. 2015, pp. 34–39 (2015).

[6] contributors, R.: About ROS, ROS (online), available from ⟨http://www.ros.org/about-ros⟩ (accessed 2016-02-01).

[7] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y.: ROS: an open-source Robot Operating System (2009).

[8] Merrick, P., Allen, S. and Lapp, J.: XML remote procedure call (XML-RPC) (2006).

[9] Troy Straszheim, Morten Kjaergaard, B. G. and Thomas, D.: catkin, ROS (online), available from ⟨http://docs.ros.org/api/catkin/html/⟩ (accessed 2015-12-08).

[10] Thomas, D. et al.: rqt_graph, ROS (online), available from ⟨http://wiki.ros.org/rqt_graph⟩ (accessed 2015-12-17).

[11] Thomas, D.: roscore, ROS (online), available from ⟨http://wiki.ros.org/roscore⟩ (accessed 2015-12-17).

[12] Saito, I.: rosout, ROS (online), available from ⟨http://wiki.ros.org/rosout⟩ (accessed 2015-12-27).

[13] Morley, M. et al.: JSON-RPC 2.0 Specification, JSON-RPC google group (online), available from ⟨http://www.jsonrpc.org/specification⟩ (accessed 2016-02-11).

[14] Almende, B. et al.: vis.js, visjs (online), available from ⟨http://visjs.org⟩ (accessed 2015-01-11).

[15] Almende, B. et al.: DataSet, visjs (online), available from ⟨http://visjs.org/docs/data/dataset.html⟩ (accessed 2015-01-11).

[16] Almende, B. et al.: network, visjs (online), available from ⟨http://visjs.org/docs/network/⟩ (accessed 2015-01-11).

[17] Bhogal, K., Peterson, R. and Seacat, L.: Bandwidth usage and latency reduction of remote desktop software based on preferred rendering of a user selected area (2011).