

# 修士論文

題目

IoT機器からの通知に基づいた機器監視サービスの開発

学籍番号・氏名

15006・宮坂 虹櫻

指導教員

横山 輝明

提出日

2017年1月28日

神戸情報大学院大学  
情報技術研究科 情報システム専攻

# 目 次

第 1 章 はじめに	1
1.1 研究の背景	1
1.2 IoT サービスの提供者の課題	1
1.3 研究の目的	2
1.4 本論文の構成	2
第 2 章 IoT サービスの維持における問題	3
2.1 IoT サービス	3
2.2 IoT サービスの構造	3
2.3 IoT サービスの開発・運用の事例	4
2.3.1 岡本商店街での実験	5
2.3.2 株式会社ルナネクサスへの聞き取り	9
2.4 IoT サービスの提供者の課題	10
2.5 考察	11
第 3 章 既存の監視手法	13
3.1 サーバからの問い合わせによる監視	13
3.2 監視対象機器からの通知による監視	14
3.3 株式会社 SORACOMA が機能として提供している監視	15
3.4 VPN の利用	16
3.5 既存手法のまとめ	16
第 4 章 IoT 機器からの通知に基づく機器監視サービスの提案	18
4.1 岡本商店街事例で求められたもの	18
4.2 株式会社ルナネクサス事例で求められたこと	18
4.3 既存手法で解決できなかった点	18
4.4 IoT 機器監視サービスの提案	18
4.5 IoT 機器監視サービスの要件	18

4.6 IoT 機器監視サービスの機能 . . . . .	18
<b>第 5 章 機器監視サービスの実装</b>	<b>19</b>
5.1 機器監視サービスの構成 . . . . .	19
5.1.1 エージェントプログラム . . . . .	20
5.1.2 エージェントプログラム用インターフェース . . . . .	20
5.1.3 機器状態データベース . . . . .	20
5.1.4 機器情報データベース . . . . .	20
5.1.5 Web アプリケーションサーバ . . . . .	20
5.1.6 Web アプリケーション . . . . .	22
5.2 機器監視サービスの実装 . . . . .	23
5.2.1 エージェントプログラムの実装 . . . . .	23
5.2.2 エージェントプログラム用インターフェースの実装 . . . . .	26
5.2.3 機器情報データベース . . . . .	26
5.2.4 機器状態データベース . . . . .	27
5.2.5 Web サーバーアプリケーション . . . . .	27
5.2.6 Web アプリケーション . . . . .	29
5.2.7 エージェントプログラムとエージェントプログラム用インターフェース間の通信の実装	29
5.2.8 Web アプリケーションサーバと Web アプリケーション間の通信の実装 . . . . .	30
5.3 ソースコード . . . . .	30
5.4 サービスによる監視のイメージ . . . . .	30
<b>第 6 章 機器監視サービスのユーザーテストと考察</b>	<b>32</b>
6.1 機器監視機能のテスト . . . . .	32
6.1.1 実験のシナリオ . . . . .	32
6.1.2 実験結果 . . . . .	33
6.2 IoT 機器を監視までの手順の比較 . . . . .	33
6.2.1 機器監視サーバーの構築 . . . . .	33
6.2.2 各 IoT 機器の状態を可視化する . . . . .	33
6.3 考察 . . . . .	34
<b>第 7 章 おわりに</b>	<b>35</b>
<b>第 8 章 謝辞</b>	<b>36</b>



# 図 目 次

2.1 IoT サービスの構成図	4
2.2 岡本商店街人流観測 構成図	6
2.3 岡本商店街人流観測 使用した機器	6
2.4 岡本商店街人流観測可視化アプリケーションスクリーンショット	7
2.5 株式会社ルナネクサス サービスイメージ図	9
5.1 システムのブロック図	19
5.2 ログイン画面	22
5.3 機器状態一覧画面	23
5.4 機器状態一覧画面（小さく表示）	23
5.5 機器状態詳細表示	24
5.6 機器追加ダイアログ	24
5.7 機器情報編集ダイアログ	24
5.8 過去の状態表示ページ	25
5.9 機器情報テーブルとユーザテーブルの関連	27
5.10 エージェントプログラムとエージェントプログラム用インターフェースの間のメッセージシーケンス図	30
6.1 IoT サービスの構成図	32

## 内容梗概

近年，IoT が注目を集めている。IoT とは，コンピュータをさまざまなモノに取り付けることで，利便性の向上を図る概念である。近年の半導体技術の進歩により，コンピュータが安価・小型になったこと，インターネットへの通信が様々な場所で安価に行えるようになったことにより，注目が集まっている。

それらのモノが連携して提供するサービスは IoT サービスと呼ばれ，より生活に身近なサービスの登場が期待されている。IoT サービスは，IoT 機器とサーバーがインターネットを介して通信し合うことで，成り立っている。IoT 機器は，モノにコンピュータが取り付けられた機器で，周囲の状況を検知，または，周囲へ働きかける機能を持つ。サーバーは，IoT 機器からの情報を蓄積・分析し，IoT 機器へ指示を送るか，ユーザーへ分析結果を表示する機能を持つ。これら IoT 機器とサーバーが連携することで，IoT サービスは利便性をユーザーへ提供している。

IoT サービスを円滑に提供するには，IoT 機器とサーバーの連携を正常に維持しなければならない。そのため，IoT 機器の動作状態や通信状態の監視が重要となる。数多く，さまざまなネットワークを介して接続される IoT 機器の監視は困難な問題である。IoT 機器が設置される様々なネットワークの構成を把握することは，IoT 機器が多量であることを考えると現実的ではない。また，従来の監視手法はパーソナルコンピュータを対象としたもので，利用しにくい。現状としては IoT サービス開発者が，IoT サービス毎に実装しているため負担が大きいといった問題がある。そのため，設置されるネットワークに関係なく状態が監視できることが求められる。また，IoT 機器の状態を一覧して確認できることや，IoT 機器の過去の動作状態や通信状態を確認できることが必要である。IoT サービスの開発者の負担を減らすためにも，IoT 機器の監視サービスが必要である。

そこで，我々は，IoT 機器からの通知に基づいた機器監視サービスを提案する。IoT 機器が自身の過去の動作状態や通信状態を記録することで，設置されるネットワークに関係なく状態監視をすることを可能にする。また，サービスを機器監視に特化させ独立させることで，IoT サービスに変更を与えること無く，機器を監視することを可能にする。この仕組みを用いることで，IoT 機器が設置されるネットワークに関係無く状態を監視することや，IoT 機器の過去の状態や通信状態を確認することを容易にし，IoT サービス開発者はサービスの開発に専念できる。本研究では，IoT 機器からの通知による機器の設置環境によらない機器の監視を行うことにより，IoT サービスの維持を容易にするシステムの開発に取り組む。

# 第1章 はじめに

## 1.1 研究の背景

近年、IoT が注目を集めている。IoT とは、様々なモノがインターネットにつながり、相互に情報をやり取りすることで、利便性の向上を図る概念である。近年の半導体技術の進歩により、コンピュータが安価・小型になったこと、インターネットへの通信が様々な場所で安価に行えるようになったことにより、注目が集まっている。IoT の具体的な例としては、建設重機の盗難防止、プリンタのトナー発注自動化、体温や脈拍等の収集・可視化等が挙げられる。

IoT サービスとは、IoT による利便性を顧客に提供するサービスの事で、より生活に身近なサービスの登場が期待されている。上記の例では、プリンタのトナー発注自動化が IoT サービスにあたる。IoT サービスは、IoT 機器とサーバがインターネットを介して通信し合うことで成り立っている。IoT 機器は、モノにコンピュータが取り付けられた物で、周囲の状況を検知、または、周囲へ働きかける機能を持つ。サーバーは、IoT 機器からの情報を蓄積・分析し、IoT 機器へ指示を送るか、ユーザーへ分析結果を表示する機能を持つ。これら IoT 機器とサーバーが連携することで、IoT サービスは利便性をユーザーへ提供している。

## 1.2 IoT サービスの提供者の課題

IoT サービスを円滑に提供するには、IoT 機器とサーバーの連携を正常に維持しなければならない。しかし、サーバーの監視は既存手法で解決できるが、IoT 機器の監視には、次のような問題があり、問題に対処する事が提供者の負担となっている。

- IoT 機器が接続されるネットワークの構成は様々であり、接続されるネットワークの変更も出来ないことが多い。
- IoT サービスが提供する機能と、ネットワークや機器の状態の監視は別機能である為、IoT サービスに組み込む事は困難である。
- IoT 機器は安価であるため、大量に使用される。そのため、設定が困難である。

### 1.3 研究の目的

そこで、前述の課題を解決するために、新規に監視サービスを開発し、IoT 機器から通知を送ることにより、IoT 機器の監視における困難を軽減することを目的とする。IoT 機器が自身の過去の動作状態や通信状態を記録することで、設置されるネットワークに関係なく状態監視をすることを可能にする。また、サービスを機器監視に特化させ独立させることで、IoT サービスに変更を与えること無く、機器を監視することを可能にする。この仕組みを用いることで、IoT 機器が設置されるネットワークに関係無く状態を監視することや、IoT 機器監視の設定の簡略化や、IoT サービスに組み込む負担の軽減を行う。本研究では、IoT 機器からの通知による機器の設置環境によらない機器の監視を行うことにより、IoT サービスの維持を容易にするシステムの開発に取り組む。

### 1.4 本論文の構成

本論文では、IoT 機器の監視困難の問題を取り上げ、その問題解決のための監視サービスを開発し、効果を報告する。第 2 章では、IoT サービスの維持に関する背景と、IoT 機器の監視に関する問題を述べる。第 3 章では、第 2 章で述べた問題を分析し、IoT 機器監視サービスの機能要件について述べる。第 4 章では、IoT 機器監視サービスの実装の詳細について述べる。第 5 章では、実験により IoT 機器監視サービスがもたらす効果を検証し、考察を述べる。第 6 章では、本研究に関する評価について述べる。第 7 章では、本研究を通して得られた知見や今後の課題について述べる。

## 第2章 IoT サービスの維持における問題

### 2.1 IoT サービス

IoT とは、Internet of Things の略で「モノのインターネット」とも呼ばれる概念である。IoT では、様々な物がインターネットにつながり、相互に情報をやり取りすることで、多様な自動化を行う。IoT サービスとは、ユーザーに対し IoT による利便性を提供するものである。

IoT サービスは、半導体技術の進歩によりコンピューターが小型且つ安価になったこと、通信ネットワークの整備が進み様々な場所から安価に通信が利用可能になったことで登場した。

例えば、次のような物がある。

- 駐車場の検索・予約・決済サービス [?]
- 太陽光発電の監視

駐車場の検索・予約・決済サービスとは、ドライバーが空いている駐車場を探す手間を省くためのサービスである。周囲の空いている駐車場の検索や、予め駐車場を予約しておくことで、駐車場を探す手間を省いている。このサービスの実現のために、駐車場の駐車スペースにコンピュータを取り付ける。これらコンピュータが、駐車場が空いているか否か・予約が入っているか否か等をサーバーとやり取りする。それにより、サーバーは空いている駐車場の一覧や、利用情報に基づく決済を、ユーザーに提供している。

太陽光発電の監視とは、太陽光発電所の発電量や機器の異常を確認しに行くための手間を省くためのサービスである。このサービスの実現の為に、太陽光発電所の機器にコンピューターを取り付ける。これらコンピュータが発電量や機器の異常の情報をサーバーとやり取りする。それにより、サーバーは、発電量や機器の異常をユーザーに知らせる。

このように、IoT サービスは、IoT 機器とサーバーが連携し、ユーザーに利便性を提供するものである。今後も数多くサービスが登場すると考えられている。

### 2.2 IoT サービスの構造

IoT サービスは、IoT 機器とサーバーが連携し利便性を提供するものである。IoT サービスは、多数の IoT 機器とサーバーがインターネットを介し連携する構造になっている。

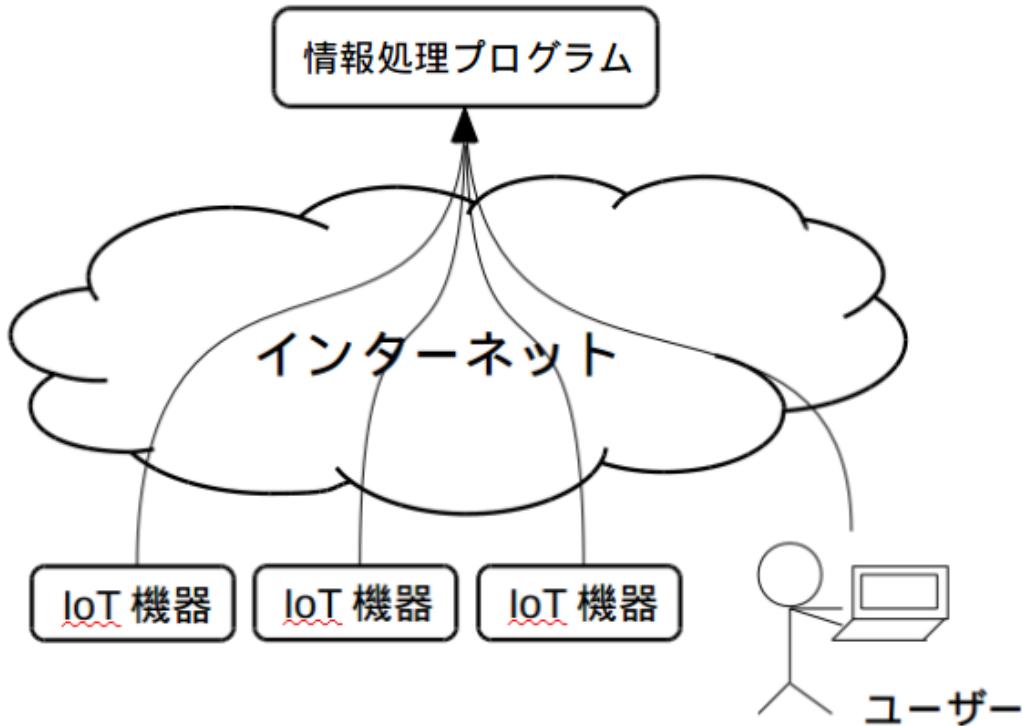


図 2.1: IoT サービスの構成図

IoT 機器は、様々な環境へ設置され、周囲の状況を検知することや、周囲へなんらかの働きを行う為に使用される。駐車場の例では、駐車スペースに車が止まっているか否かを検知している。IoT 機器の上では、サーバと情報を送受するプログラムが常時動作し、周囲の状況をサーバーに送信するか、サーバへ周囲へ何らかの働きかけを行うかどうか問い合わせている。

この機器からの情報を収集し、処理しているのがサーバーである。サーバーは、IoT 機器からの情報を蓄積・分析し、IoT 機器やユーザーに対し何らかの働きかけを行う。駐車場の例では、駐車場の利用情報を蓄積・分析し、ユーザーへ対し可視化を行っている。サーバーにて動作するプログラムが、IoT 機器と通信することで、IoT サービスを構成する。この通信に利用されるのがインターネットである。様々な通信リンクを用いて IoT 機器とサーバー上のプログラムが連携する。

ユーザーは、サーバが蓄積・分析した情報を閲覧したり、サーバが蓄積・分析した情報をもととした通知を受け取る。

このように、IoT サービスの構造は、IoT 機器とサーバー上のプログラムがインターネットを介し通信し、連携することで成り立っている。図 2.1 は、IoT サービスの構造図である

## 2.3 IoT サービスの開発・運用の事例

IoT サービスの開発・運用における課題を分析するために、実験およびヒアリングを行った。

### 2.3.1 岡本商店街での実験

#### 実験概要

2015年12月8日から2016年2月26日まで、NPO法人コミュニティリンクへのインターンシップの一環として、岡本商店街にて人流観測を行った。岡本商店街とは、神戸市東灘区にある阪急岡本駅とJR摂津本山駅の間に位置する商店街のことである。実験は、商店街の方に人流を可視化するIoTサービスを提供し、商店街の活性化に役立てるといった趣旨で行った。観測は、2016年2月7日から2016年3月14日まで行った。

人流観測とは、各地点から各地点迄をある時に移動した人数を観測するものである。通常は、観察員がカウントを用いて数えるが、それでは各地点間を移動した人数はわかるが、その人が以前どの地点に居たのかはわからない。そこで、携帯電話についているWifi機能を利用し、観測を行うこととした。携帯電話のWifi機能は、無線LANの接続に使われるが、接続毎にWifi機能を有効にすることが手間なため、常時ONにしている人も少なくない。携帯電話のWifi機能を有効にしている場合、携帯電話から接続可能な無線LANを探す為、プローブパケットというものが定期的に送出される。プローブパケットには、そのプローブパケットを送出した機器の物理アドレスが含まれており、個々の機器が識別可能である。そのプローブパケットを複数地点で観測し、含まれている物理アドレスと受信時刻を照合することで、携帯電話端末を持った人がどのように移動をしたのかが分かる。岡本商店街では、この原理を利用して、人流観測を行った。

#### 観測・分析・可視化システムの構成

岡本商店街の5店舗に開発した観測機器を設置し、サーバにて蓄積・分析・可視化を行った。構成としては、観測機器内で動作するプログラムが定期的にサーバに観測データを送信する。サーバ上では、分析プログラムと可視化プログラムが動作しており、観測データは分析プログラムへ渡される。分析プログラムは、観測データを可視化プログラムの要求によって分析し、結果を可視化プログラムへ渡す。可視化プログラムは、ユーザからの操作によって分析プログラムを呼び出し、分析結果をWebインターフェースによって可視化する。このシステムのユーザーは、ブラウザからWebインターフェースへアクセスすることで、操作・分析結果の閲覧をすることができる。図2.2は、開発したシステムの構成を示している。

開発した観測機器は、RaspberryPiとBaffalo製のWifiドングル、ampsence・rsyncというソフトウェアを利用し作成した。図2.3は、開発した観測機器である。RaspberryPiとは、小型PCの一つで安価に入手が可能である。また、利用したRaspberryPiにはWifiインターフェースが存在しなかったので、Baffalo製のUSB接続が可能なWifiドングルを接続した。ampsenceとは、プローブパケットを受信しプローブパケットに含まれる物理アドレスと観測日時をファイルへ記録するソフトウェアである。rsyncとは、ディレクトリの中を同期させるプログラムで、遠隔にあるコンピュータの指定したディレクトリの中と、手元のコンピュータの指定したディレクトリを同期することができる。開発した観測機器は、起動時にampsenceを動作させ、定期的に

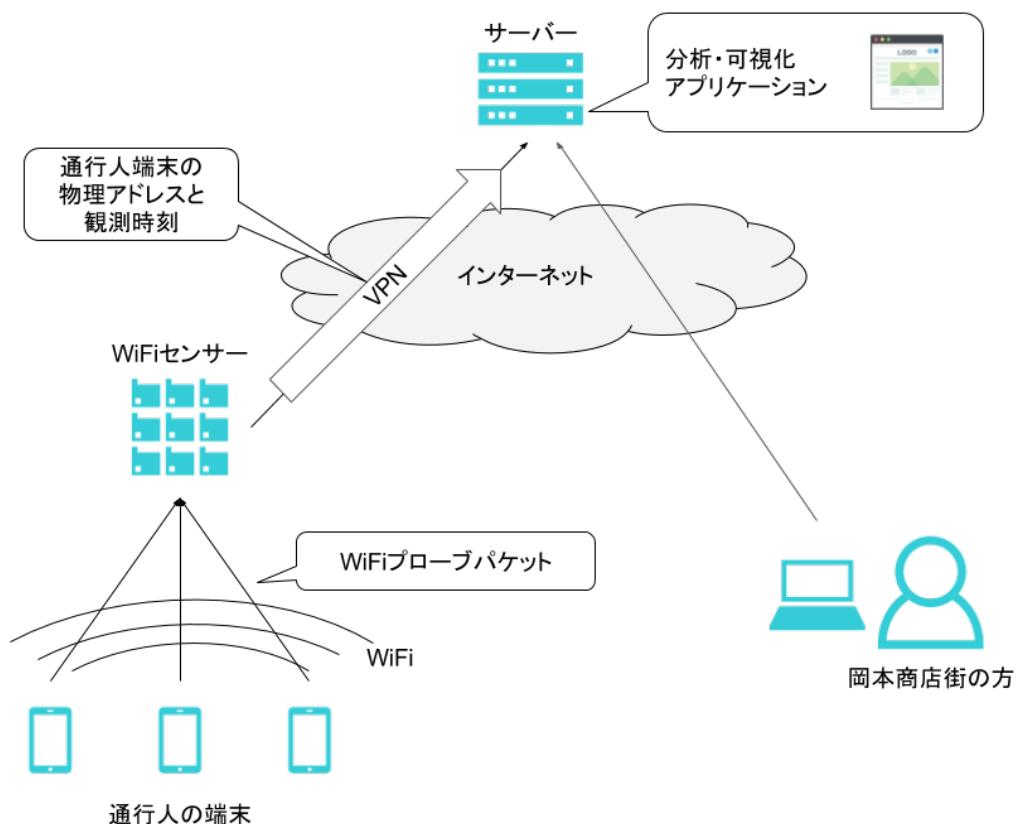


図 2.2: 岡本商店街人流観測 構成図



図 2.3: 岡本商店街人流観測 使用した機器

rsync を実行するよう設定した。これによって、各所に設置された観測機器の情報をサーバー上に集約・蓄積する。

作成した分析プログラムは、可視化プログラムから指定された期間内の観測データを読み込み、地点ごと観測時間ごとの物理アドレスの数の推移・地点ごと観測時間ごとの移動した人数の割合を集計し、可視化プログラムへ渡す。作成した可視化プログラムは、ユーザから指定された期間を分析プログラムへ渡し、分析結果を棒グラフや円グラフといった形で、ユーザーへ表示する。図 2.4 は作成した Web アプリケーションである。

## 岡本商店街動態情報可視化システム

### 2016年 1月 1日 現在の動態情報



図 2.4: 岡本商店街人流観測可視化アプリケーションスクリーンショット

観測機器からサーバまでのネットワークは、観測した物理アドレスが流れるという点、ソフトウェアのアップデートの為にログイン可能にしたいといった要望から、VPN を使用した。VPN には、OpenVPN を使用した。また、観測機器、分析・可視化アプリケーションを開発するにあたって、次のような点を考慮した。

- ネットワーク機器の設定が不要であること

機器が設置されるネットワークは、店舗のネットワークである。そのため、ネットワーク機器の設定を変更することはできない。よって、観測機器と分析・可視化アプリケーションの連携が、ネットワーク機器の設定に左右されないよう設計する必要があった。

- トラブル対応のしやすさ

機器が設置される場所は、岡本商店街内の店舗である。そのため、設置やトラブル対応、回収については、予め店舗に連絡を入れなくてはならない。また、夜間や店舗の休日に対応することは困難である為、遠隔から観測機器にログインする必要があった。

- 設置箇所の問題

店舗によっては、店舗内に無線ネットワークが存在しなかったので、後述する SORACOM Air を用いることとした。

- データ損失に備える

サーバー側の不良によって観測データが損失する事に備え、機器自体にも情報を蓄積しておくことにした。

## 考察

しかし、実験には次のような困難があった。

- ネットワークの設定に手間取った。
- 設定ミスがあった。
- 電源が抜けていたことがあった。
- 一つのトラブルに対し、現地に何度も行かなくてはならなかった
- 長時間に渡って観測データが欠損していることがあった。

これら困難から、このような問題があることが分かった。

- トラブルに気づくのが遅れた問題

トラブルが発生していても、誰も気づかなかつた事があった。そのため、トラブルへの対応が遅れ、観測できなかつた事があった。

- トラブル対応が困難な問題

ネットワークによる問題なのか、電源が抜けているといった問題なのか、切り分けが難しく、トラブルがあつた際に、開発者が対応しなければならないのか運用者で良いのか判断がつきにくかつた。そのため、開発者が行っても、電源が抜けているだけの問題であつたり、運用者が行っても解決できない為、再度現地に行く事になつた。また、トラブルの対応を検討している間に、機器の再起動等により復旧してしまつた事もあつた。

- 観測データの欠損から、満足な分析ができなかつた問題

トラブルと、トラブルへの対応の遅れから、観測データが足りず、満足な分析が行えなかつた。また、観測できていなかつた日時を推測するため、観測データを何度も参照する事になり、分析に手間取つた。

これらから、次のような機能を持つ監視が必要であることが分かった。

- トラブル発生時に、ネットワークの問題なのかそれ以外の問題なのか、ある程度切り分けることができること。
- トラブルのあった日時が記録されている必要があること
- 現在の機器の状態を簡単に閲覧できる必要があること。

### 2.3.2 株式会社ルナネクサスへの聞き取り

株式会社ルナネクサスとは、大阪にある組み込み機器メーカーである。近年の IoT への注目から、IoT サービスを提供している

株式会社ルナネクサスでは、太陽光発電事業を展開している事業主に対し、発電に係る機器の状態や、発電量等を可視化できるサービスを展開している。独自に開発した IoT 機器を発電に係る機器に取り付け、SORACOM Air というインターネット接続サービスを使用して、機器からの情報をサーバーへ蓄積し、ユーザーへ提供している。SORACOM Air とは、後述する IoT 機器向けのインターネット接続サービスのことである。図 2.5 は株式会社ルナネクサスの IoT サービスイメージである。

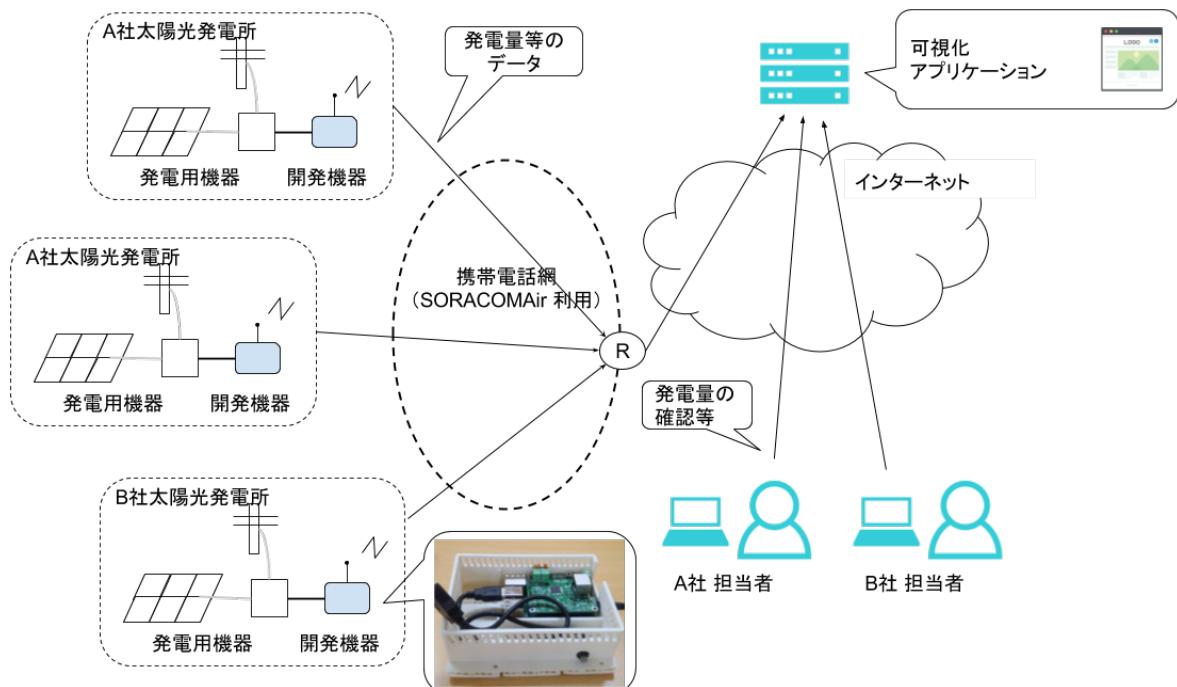


図 2.5: 株式会社ルナネクサス サービスイメージ図

太陽光発電に係る機器は、通常発電所のわきに小さなプレハブを建て、その中に設置する。太陽光発電所は日当たりが良いため、夏場、プレハブの中は太陽の熱と機器が発する熱で機器の標準的な動作温度を超えることがある。そのため、機器が正常に動作しているかどうかを監視する必要がある。

太陽光発電所は僻地にあることが多いので、利用できるネットワークが無いことが多い。このことから、インターネットへの接続に SORACOM Air を選択した。しかし、SORACOMAir 回線は、プライベートアドレスを使用しており、IoT サービスの通信は通過するものの、Ping などによる確認を行うことが出来ない。また、ソフトウェアの更新の為に、ログインするには、現地に行かなくてはならなかった。

この、機器の状態の監視が必要である問題と、遠隔からログインする必要がある問題について、株式会社ルナネクサスでは、ログインの為に VPN を利用したが、監視については新規に作るべきなのか、サービスに組み込むべきなのか迷っているという。

新規に作成するとした場合、どのような事を満たしている必要があるかインタビューを実施した所、以下のような回答をいただくことができた。

- 質問：遠隔からログインする機能があると便利かと思うが、どうか？
- 回答：必要ない。それよりも、単純に動作しているのかどうか知りたい。
- 質問：単純に動作しているのかどうかとは、どういった状態なのか？
- 回答：起動しているのかどうかである。欲を言うならば、CPU の温度も欲しいが、単純に動作しているのか分かれば良い

のことから、IoT 機器の監視について、問題と考えている事が分かった。

## 2.4 IoT サービスの提供者の課題

IoT サービスは、IoT 機器とサーバ上のプログラムがインターネットを介し通信し合うことで成り立っている。このような構造を持つ IoT サービスの提供は、それぞれ異なった役割を持つ開発者と運用者が行っている。開発者はサービスを設計・構築し、運用者はサービスによる供給が止まらないように維持・管理する役割を持っている。

IoT サービスの開発には、次のような問題がある。

- 多分野に渡る技術を知っていなければならない  
IoT 機器の開発には、ハードウェアの知識や、ネットワークの知識が必要不可欠である。また、サービスプログラム（？）の開発のためには、データベースやネットワークの知識が必要となる。このように、多分野に渡る知識を知っていなければ、開発することができない。
- 一つの機能を実装するだけでも困難な作業である  
IoT サービスは、一つの機能を別々の機器上で動く複数のプログラム間のやり取りにて実現するため、簡単な機能であっても、すぐに実現可能なわけではない。

- 設置やトラブル対応が難しい

IoT サービスの開発には、多分野に渡る技術を知っていなければならないため、トラブルが発生した場合、知識のある開発者が現地に行かなければならず、コストが高い。また、設置の時も同様で、予め開発者が現地に行き設置環境を確認しなければならない。

- ネットワークの監視は IoT サービスが提供する機能とは別の機能である

ネットワークの監視や IoT 機器の監視は、IoT サービスが提供する機能とは別の機能である。そのため、IoT サービスをもうひとつ作ると変わらない手間がかかり、開発者の負担となっている。

このように、IoT サービスの開発は、開発者の負担が大きい。

また、IoT サービスの維持・管理においては、次のような課題がある。

- IoT 機器の接続されるネットワークは想定できない

IoT 機器が接続されるネットワークは、IoT 機器の移動や既設ネットワークへの接続の事を考えると、予め想定することが難しい。そのため、サーバに対し、予め設置することは困難である。

- IoT 機器の接続されるネットワークがプライベートアドレスであることがある

IoT 機器の接続されるネットワークは、利便性の都合などからプライベートアドレスを利用し、インターネットとの接続点に NAPT が置かれる事がある。そのため、サーバからの通信が遮断されることがある。

- IoT 機器は安価なため、大量に利用される

IoT 機器は安価であるため、大量に利用されることが多い。そのため、サーバ側へ設定することは負担が大きい。?

このような問題から、IoT サービスの開発・運用は開発者や運用者の負担となっている。

## 2.5 考察

このように、開発者・運用者は、構造の複雑さ・利便性から困っている。ここから、私は、次のような点を問題として取り上げた。

- IoT 機器の監視は必要だが、技術的な問題により、既存の監視手法では監視しづらい

- IoT 機器の監視問題の解決の為に、開発者の負担が大きくなっている

また、これら実験やヒアリングの結果から、監視には次のような物が求められていることが分かった。

- IoT 機器の接続されるネットワークが、プライベートアドレスを使用していても、監視可能であること

IoT 機器が接続されるネットワークを予め予測することは難しい。そのため、IoT 機器にプライベートアドレスが割り振られる可能性がある。そのような状況でも監視することができる必要がある。

- ネットワークが違っていても、一つの画面で確認できること
- IoT サービスの変更が不要であること

サーバの変更は開発者への負担となるため、サーバの変更を避けたい。

- 監視サーバを立てる必要がないこと

新たに監視サーバを立てることは、設定や構築作業が必要となるため、開発者の負担となる。そのため、新規に監視サーバを立てる必要が無いことが求められる。

# 第3章 既存の監視手法

IoT 機器の監視は、次のような要件を満たしている必要があることが分かった。

- IoT 機器の接続されるネットワークが、プライベートアドレスを使用していても、監視可能であること  
IoT 機器が接続されるネットワークを予め予測することは難しい。そのため、IoT 機器にプライベートアドレスが割り振られる可能性がある。そのような状況でも監視することができる必要がある。
- ネットワークが違っていても、一つの画面で確認できること
- IoT サービスの変更が不要であること  
サーバの変更は開発者への負担となるため、サーバの変更を避けたい。
- 監視サーバを立てる必要がないこと  
新たに監視サーバを立てるとは、設定や構築作業が必要となるため、開発者の負担となる。そのため、新規に監視サーバを立てる必要が無いことが求められる。
- 機器に異常があった場合に、運用者に通知を行う機能があること  
常に監視画面を見ているわけには行かないので、アラート機能等があると良い。

そこで、従来からある手法にて対応できるか、考えた。

## 3.1 サーバからの問い合わせによる監視

従来から機器監視に用いられてきた手法として、定期的に機器監視サーバから対象機器に状態を問い合わせる手法がある。機器監視サーバから、監視対象機器上のエージェントプログラムに、現在の状態を問い合わせる事で機器の監視を実現している。

代表的なものとして、次のような物がある。

- Ping
- Ping とは、ICMP メッセージをやり取りするプログラムである。ICMP とは、InternetControlManagementProtocol の事で、IP ネットワークにおいて、IP パケットの不到達等を通知する為の取り決めである。ICMP には、ICMP Echo Request Echo Reply が定義されており、ICMP Echo Request を受け取つ

た機器は、ICMP Echo Reply を返さなくてはならない。監視を行う機器から、Ping を用いて、ICMP Echo Request を監視対象機器の IP アドレスに送信し、Echo Reply の有無から、機器の存在を確認する事ができる。

- SNMP による問い合わせ

SNMP とは、Simple Network Management Protocol の事で、ネットワークの監視・管理を行うための、取り決めである。SNMP には、SNMP マネージャと SNMP エージェントが存在する。SNMP マネージャは、監視を行う機器上で動作し、SNMP エージェントは、監視される機器の上で動作する。SNMP を利用した機器の状態監視の方法として、SNMP マネージャが SNMP エージェントに問い合わせを行い取得する方法がある。

この手法のメリットを以下にまとめる。

- 手軽に機器の状態やネットワークの状態を確認することができる事
- 見やすく表示する事ができるソフトウェアが豊富に存在する事
- コンピューターであれば、ほぼ全ての機器が対応している事

また、デメリットとして、次のような物がある。

- 監視する側のコンピュータに全ての監視対象の IP アドレスを設定する必要があること
- 攻撃の足がかりとして利用される事があるので、間にあるネットワーク機器によってブロックされてしまう事がある

### 3.2 監視対象機器からの通知による監視

機器監視手法として、定期的に監視対象機器から機器監視サーバーへ状態を送信するという手法がある。監視対象機器上のエージェントプログラムが、指定した時間毎に機器監視サーバーへ状態を送信することで、機器の監視を実現している。

代表的なものとして、次のようなものがある。

- SNMP Trap による取得

SNMP では、予め SNMP エージェントに設定をしておくことによって、状態の変化を SNMP マネージャへ知らせることができる機能がある。この機能を利用して、SNMP エージェントの状態を取得できる。

- サーバーのリソース監視ソリューション

サーバのリソース監視ソリューションの中には、独自に開発されたエージェントプログラムを監視対象にインストールすることに寄って、監視を行うものがある。代表的なものとして、以下の 2 つをあげる。

- Teregraf と Influxdb による、機器の状態の通知と蓄積

Teregraf とは、監視対象にインストールされるエージェントプログラムで、監視される機器の上で動作する。Influxdb とは、時系列データを格納することに特化したデータベースで、この中では機器の状態を蓄積することに使用され、監視を行う機器の上で動作する。Teregraf は、機器の状態を定期的に取得し Influxdb に送信する。そして、Influxdb は、それを蓄積する。Influxdb に蓄積されたデータは、Influxdb に対応した可視化アプリケーション（Grafana 等）によって可視化され、監視を行うことができる。

- Logstash によるログの転送による機器の監視

Logstash とは、アプリケーションのログやシステムのログを転送するためのプログラムであり、監視される機器上で動作する。Logstash は、ログファイルの監視をしており、ログファイルに追記があった場合、追記分のデータをデータベースに送信することができる。データベースとしては、Elasticsearch が良く用いられる。データベースに蓄積されたデータは、対応した可視化アプリケーションによって可視化を行うことで、監視をする。

この手法のメリットとして、次のような物が挙げられる。

- 見やすく表示するソフトウェアが多く利用しやすい
- 監視を行う機器に対し、監視対象の機器の IP アドレス等を登録する作業が不要である
- サーバのリソース監視ソリューションを利用した場合、機器ごとアプリケーションごとの監視が可能である

また、デメリットとして、次のような物が挙げられる。

- 機器ごとに、転送先を設定しなければならない
- 複数のソフトウェアを組み合わせるので、設定作業が負担である

### 3.3 株式会社 SORACOMA が機能として提供している監視

ネットワークの提供者がオプションとして提供している監視機能を利用する方法もある。ここでは、SORACOM が提供しているネットワークを利用した場合の監視を取り上げる。

株式会社 SORACOM とは、IoT プラットフォームとして、通信とクラウドへの接続を提供している企業である。通信は MVNO として携帯電話回線網を使用しており、SORACOM 用 SIM を購入し、機器に SIM モジュールを取り付けることで利用できる。株式会社 SORACOM では、各 SIM のアクティベートや通信量設定の変更、クラウドへの接続の設定等を行うための Web インターフェースを提供しており、プラットフォーム

利用者はこの Web インターフェースをインターネットを介して利用し、各種設定を行うことができる。その Web インターフェースには、各 SIM の通信量や状態を把握するためのページが存在しており、ここで各 SIM がリンクアップしているかどうかを監視することもできる。

この機能を用いた場合のメリットをあげる。

- 新たに監視サーバ等を立てる必要が無い。
- インターネットに接続できる環境ならば、どこからでも監視を行うことが可能である。

デメリットとして、下記があげられる。

- SORACOM が提供しているネットワークを利用していないと使用することができない
- リンクアップ、リンクダウンと通信量しか監視できない。
- 他のネットワークに接続された機器を監視することはできない。

### 3.4 VPN の利用

VPN を利用して、既存の監視手法を使用する方法もある。この場合のメリットを以下に挙げる。

- 機器が利用しているネットワークによらない監視を行うことができる

この場合のデメリットを以下に挙げる。

- 機器が利用しているネットワークのアドレスが、VPN のアドレスと重複していないことが求められる。
- 新たに、VPN サーバを立ち上げる必要がある。また、VPN の設定は繁雑であるため、負担となる。
- 監視サーバや機器に対し、IP アドレスを設定することや、各ソフトウェアに設定する事は、改善されない。

### 3.5 既存手法のまとめ

既存の手法として、サーバーからの問い合わせによる監視、監視対象機器からの通知による監視、ネットワークによる監視、VPN の利用といった手法がある。次に、IoT 機器の監視の要件をまとめる。

1. IoT 機器の接続されるネットワークが、プライベートアドレスを使用していても、監視可能であること
2. ネットワークが違っていても、一つの画面で確認できること
3. IoT サービスの変更が不要であること
4. 監視サーバを立てる必要がないこと

次に示すのは、各要件が満たせるかどうかについての表である。

既存手法 \ 要件番号		1	2	3	4
監視サーバからの問い合わせによる監視	VPN と併用しない場合	X			X
	VPN と併用する場合				X
監視される機器からの通知による監視					X
ネットワーク提供者が提供する機能による監視			X		

# 第4章 IoT機器からの通知に基づく機器監視サービスの提案

## 4.1 岡本商店街事例で求められたもの

岡本商店街では、次のような監視の存在が求められた。

- トラブル発生時に、ネットワークの問題なのかそれ以外の問題なのか、ある程度切り分けることができること。
- トラブルのあった日時が記録されている必要があること
- 現在の機器の状態を簡単に閲覧できる必要があること。

## 4.2 株式会社ルナネクサス事例で求められたこと

- 
- 

## 4.3 既存手法で解決できなかった点

## 4.4 IoT機器監視サービスの提案

そこで、私は IoT 機器の監視に特化した IoT 機器監視サービスを提案する。

## 4.5 IoT機器監視サービスの要件

## 4.6 IoT機器監視サービスの機能

# 第5章 機器監視サービスの実装

## 5.1 機器監視サービスの構成

第3章にて述べた要件に基づき、システムを構築した。システムは、エージェントプログラム、機器状態データベース、機器情報データベース、エージェントプログラム用インターフェースプログラム、Web アプリケーションサーバ、Web アプリケーションから成り立っている。エージェントプログラムとエージェントプログラム用インターフェース、Web アプリケーションサーバと Web アプリケーションは、インターネットを介して通信しあう。図 5.10 は、システムのブロック図である。

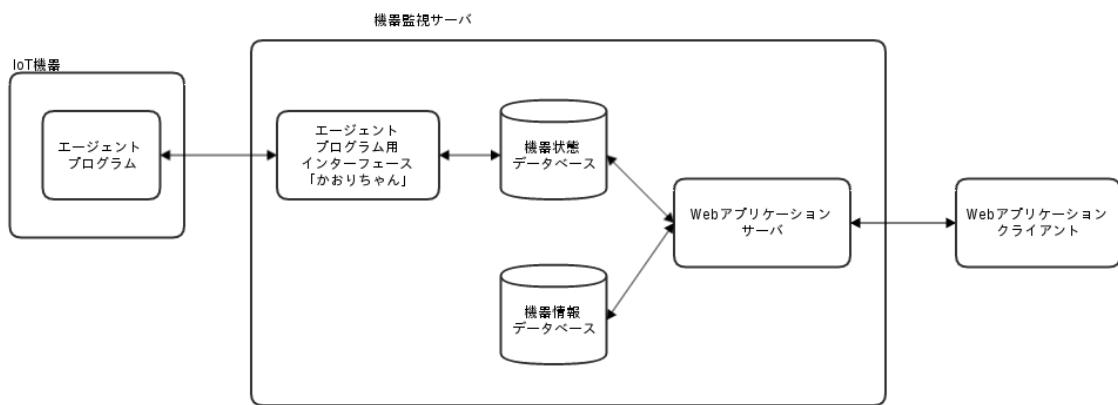


図 5.1: システムのブロック図

エージェントプログラムは、IoT 機器上で動作するプログラムである。定期的に自身の状態を状態蓄積システムへ送信する役割を果たす。定期的かつ自発的に状態を送信することで、ネットワーク環境によらない機器の監視を可能にした。IoT 機器として頻繁に使用される RaspberryPi を想定し作成した。エージェントプログラム用インターフェースは、各 IoT 機器上で動くエージェントプログラムから送られてきた状態を、時刻と共に機器状態データベースへ書き込む役割を果たすプログラムで、機器監視サーバー上で動作する。機器状態データベースとは、機器の状態を時系列に沿って蓄積するデータベースである。機器監視サーバー上で動作する。機器情報データベースとは、機器 ID や、機器名、ユーザー名を記録するデータベースである。機器監視サーバー上で動作する。

Web アプリケーションサーバーは、Web ページや Web アプリケーション自体を配信する。ユーザーが利用するブラウザからの要求に答え、現在の機器の状態や、機器名等を返答する。

Web アプリケーションとは、ブラウザ上で動作するプログラムである。ユーザーからの入力を受け付け、ユーザーへ表示する他、必要な機能を問い合わせる。

これら各要素が連携することで、機器の監視を実現している。

### 5.1.1 エージェントプログラム

エージェントプログラムとは、IoT 機器上にインストールされるプログラムである。エージェントプログラムの役割は、定期的に送信失敗回数をエージェントプログラム用インターフェースへ報告することである。送信失敗回数とは、ネットワークの不具合等により、機器監視サーバーへ送信されなかった報告の数である。自発的に状態を報告するため、IoT 機器にプライベートアドレスが付与されても、状態を検知することができる。また、HTTP を用いるため、間のネットワークにてブロックされることがない。

### 5.1.2 エージェントプログラム用インターフェース

エージェントプログラム用インターフェースとは、サーバー上で動くプログラムである。名前を「かおりちゃん」とした。エージェントプログラム用インターフェースの役割は、エージェントプログラムから送信されたメッセージを受け取り、現在の時刻と正常である旨を機器状態データベースへ書き込む。また、エージェントプログラムから送られた送信失敗回数から、IoT 機器がインターネットから切断された時刻を逆算し、機器状態データベースへ書き込む事も行う。

### 5.1.3 機器状態データベース

機器状態データベースとは、サーバ上で動作するデータベースである。各 IoT 機器の状態を時刻とともに記録する。機器状態監視システムの中心にあるデータベースである。

### 5.1.4 機器情報データベース

機器情報データベースとは、サーバー上で動作するデータベースである。各 IoT 機器の機器 ID、機器名、機器詳細情報、ユーザーのメールアドレスとパスワードを記録する。

### 5.1.5 Web アプリケーションサーバ

Web アプリケーションサーバとは、サーバ上で動作するプログラムである。Web アプリケーションや Web ページの配信、Web アプリケーションからの要求の処理などを行う。必要に応じて、機器状態データベースと

機器情報データベースへアクセスを行う。次に、Web アプリケーションとのインターフェースを挙げ、それについて説明する。

#### ログイン API

Web アプリケーションから、メールアドレスとパスワードを受け取り、機器情報データベースのユーザー テーブルと照合する。照合した結果、ユーザー名とパスワードが合致したユーザーが存在すれば、HTTP クッキーにセッションキーをセットし、機器状態一覧ページへのリダイレクトを返す。合致したユーザーが存在しなかった場合、エラーメッセージを返す。

#### ログアウト API

Web アプリケーションに、HTTP クッキーから該当のセッションキーを削除するよう要求する。

#### 機器情報・機器状態取得 API

ログインチェックを行った後、機器情報データベースと機器状態データベースより、全 IoT 機器の機器情報と機器状態を返す。

#### 機器 ID 生成 API

ログインチェックを行った後、機器情報データベースに存在しない、ランダムな機器 ID を返す。

#### 機器 ID 重複チェック API

ログインチェックを行った後、Web アプリケーションから機器 ID を受け取る。機器情報データベースに該当の機器 ID が存在するかしないかを返信する。

#### 機器作成 API

ログインチェックを行った後、Web アプリケーションから、機器 ID、機器名、機器の詳細と、機器の作成なのか編集なのかの指示を受け取る。機器の作成であった場合、機器 ID に重複が無いことを確認したうえで、機器情報データベースに該当のエントリを作成・機器状態データベースにメジャーメントを作成する。作成されたか、されなかったかを返信する。機器の編集であった場合、機器情報データベースから、受け取った機器 ID を持つものを探しだし、編集する。編集できたか否かを返信する。

## 機器削除 API

ログインチェックを行った後，Web アプリケーションから，機器 ID を受け取る．機器情報データベースから，受け取った機器 ID を持つものを削除する．その後，機器状態データベースから，メジャーメントを削除する．削除されたか否かを返信する．

## 過去の機器状態取得 API

ログインチェックを行った後，機器状態データベースから，全ての IoT 機器の過去の状態をまとめ，返信する．

### 5.1.6 Web アプリケーション

Web アプリケーションとは，ユーザーのブラウザ上で動作するアプリケーションである．ユーザーへグラフィカルインターフェースを提供する．必要に応じて，Web アプリケーションサーバへ必要な情報を要求する．Web アプリケーションは，3 つの Web ページから成り立っている．以下に各ページとそれぞれの役割を述べる．

- ログインページ

正当なユーザーであることを確認するために，メールアドレスとパスワードの入力を求める．メールアドレスとパスワードは，Web アプリケーションサーバへ送信される．図 5.2 は，ログインページのスクリーンショットである．

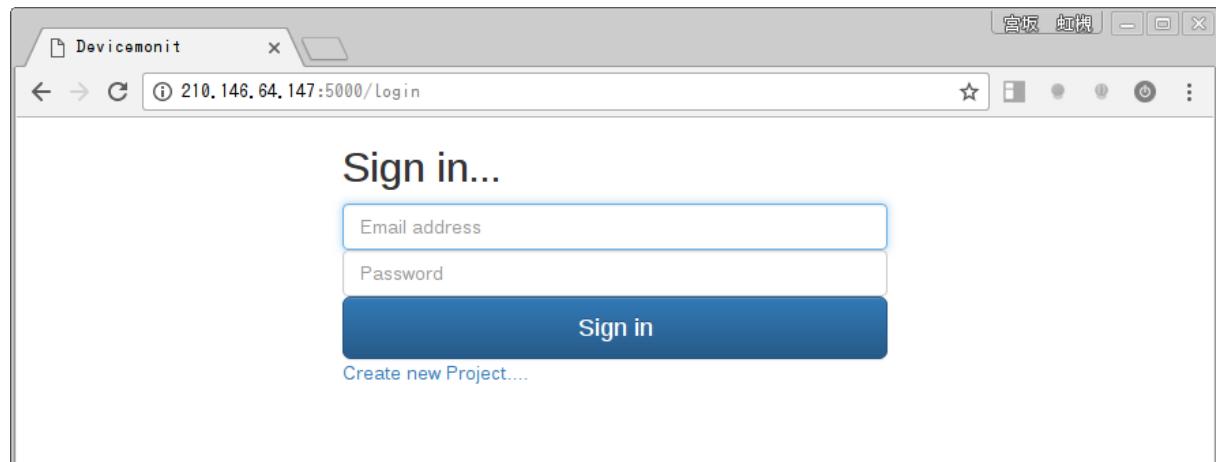


図 5.2: ログイン画面

- 機器状態一覧ページ

機器状態を一覧して表示する他，機器の作成や，機器情報の編集，機器の削除等を行う事ができる．現在の機器の状態を取得するため，定期的に Web アプリケーションサーバと通信する．また，機器の作成や機器情報の編集，削除の為，Web アプリケーションサーバと通信する．

図 5.3・図 5.4・図 5.5 はこのページのスクリーンショットである。それぞれ、一覧表示・縮小一覧表示・詳細な情報の表示をしている。図 5.6・図 5.7 は、それぞれ、機器追加ダイアログ・機器情報編集ダイアログのスクリーンショットである。

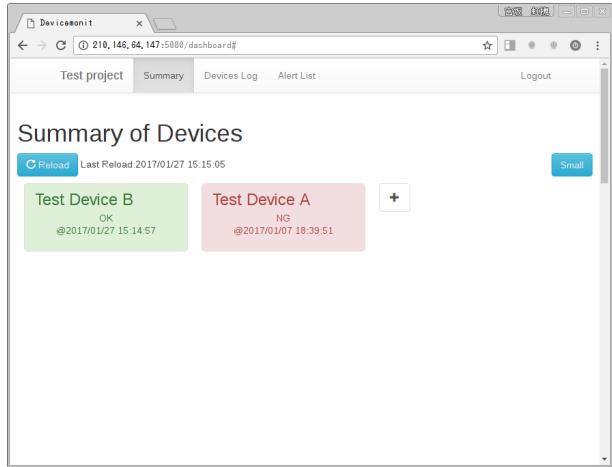


図 5.3: 機器状態一覧画面

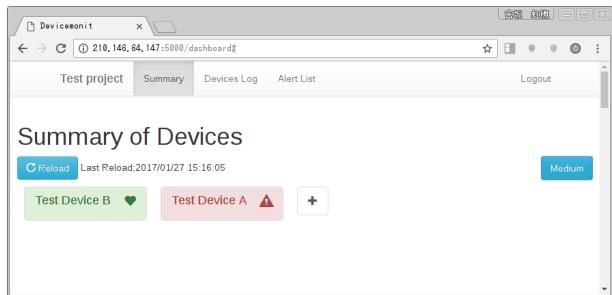


図 5.4: 機器状態一覧画面（小さく表示）

- 過去の機器状態一覧ページ

過去の機器状態を時刻と共に整理し、一覧表示するページである。現在の機器の状態を取得するため、定期的に Web アプリケーションサーバと通信をする。図 5.8 は、スクリーンショットである。

## 5.2 機器監視サービスの実装

### 5.2.1 エージェントプログラムの実装

エージェントプログラムの役割は、送信失敗回数を定期的に IoT 機器監視サーバーに送信することにある。約 1 分おきに、現在の送信失敗回数と過去の送信失敗回数を、エージェントプログラム用インターフェースへ送信する。どのような Linux 環境でも動作することを考え、Shell スクリプトにて実装した。

エージェントプログラムの動作は次のようになる。

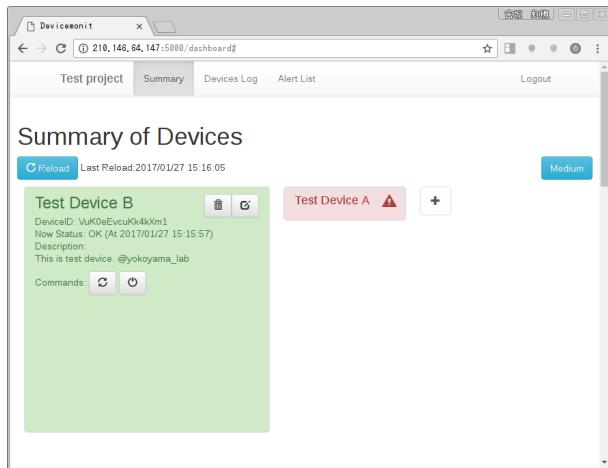


図 5.5: 機器状態詳細表示

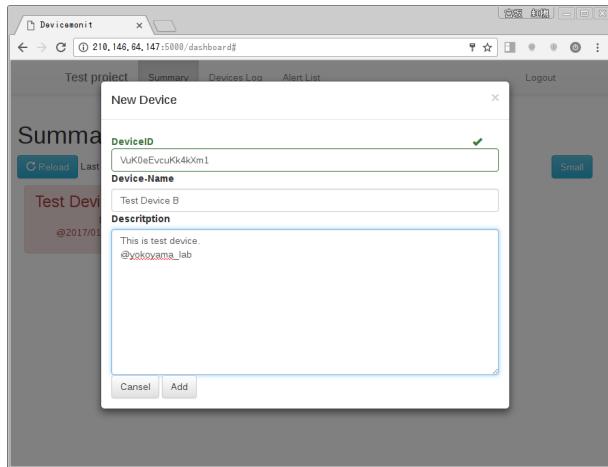


図 5.6: 機器追加ダイアログ

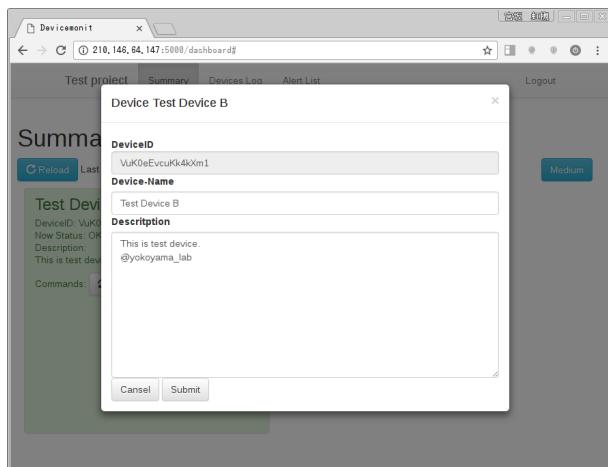


図 5.7: 機器情報編集ダイアログ

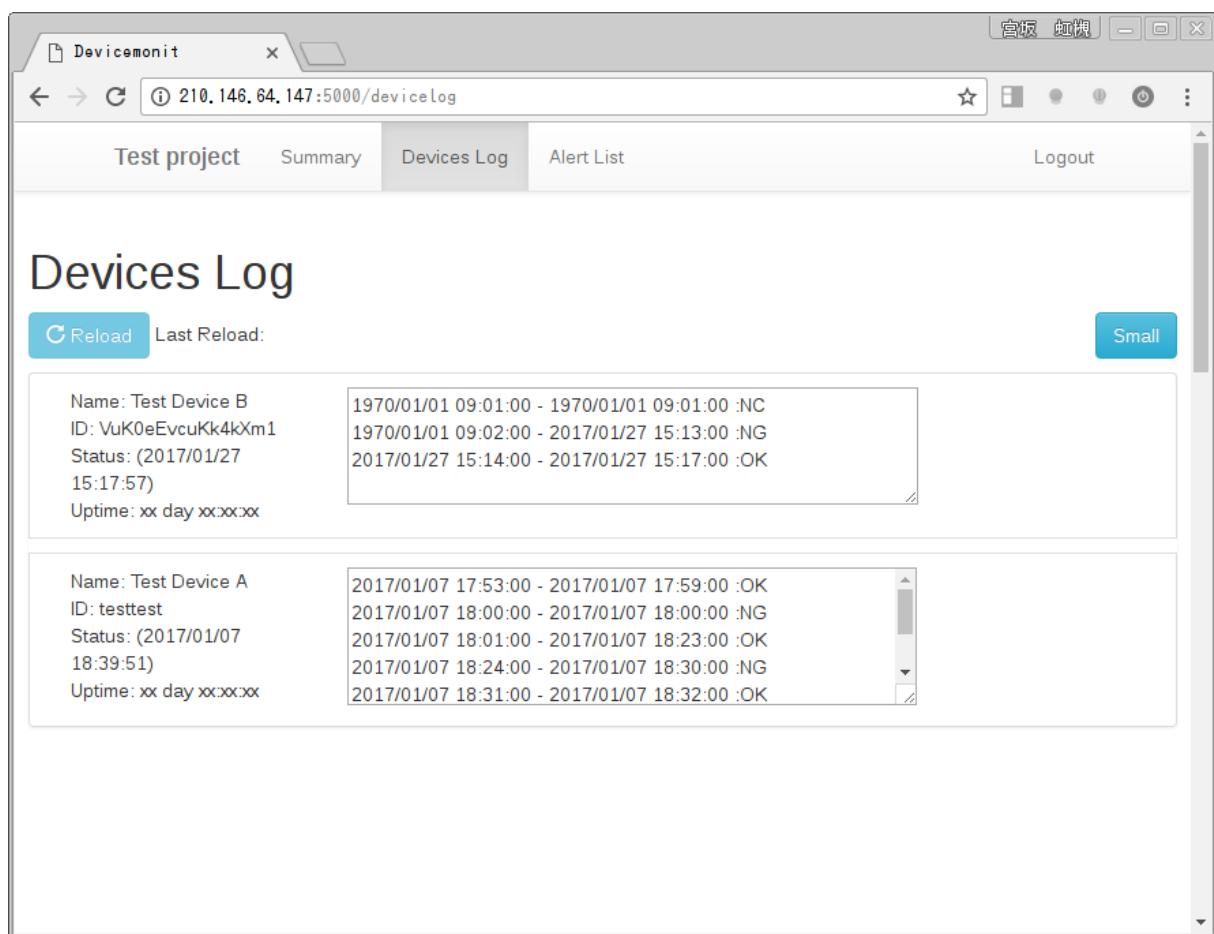


図 5.8: 過去の状態表示ページ

1. エージェントプログラムが起動されると，まず過去に記録された送信失敗回数を読み出す．
2. エージェントプログラム用インターフェースに対し，過去に記録された送信失敗回数と，現在の送信失敗回数を送信する．
3. サーバーから応答があった場合，過去に記録された送信失敗回数と，現在の送信失敗回数をクリアしする．  
サーバーから応答がない場合，現在の送信失敗回数をインクリメントする．
4. 1分間スリープし，再び2から繰り返す．

### 5.2.2 エージェントプログラム用インターフェースの実装

エージェントプログラム用インターフェースの役割は，エージェントプログラムから送信失敗回数を受け取り，時刻と正常である旨を機器状態データベースへ書き込むこと，また，エージェントプログラムから送られた送信失敗回数から，インターネットより切断された時刻を逆算し，機器状態データベースへ書き込む事である．Falcon と呼ばれる API の作成に特化したフレームワークを使用し，Python にて実装した．また，機器状態データベースへの書き込みには，InfluxDBClient というライブラリを使用した．

### 5.2.3 機器情報データベース

機器情報データベースの役割は，機器の名前，機器の詳細説明，機器 ID，ユーザー ID，ログイン用メールアドレスとパスワードを記録し保持する事である．ユーザ ID とは，システムにてユーザーを識別するための識別子である．機器 ID は，システムにて IoT 機器識別するための識別子である．機器情報データベースには，SQLite3 を用いた．機器情報データベースには，次のようなテーブルが用意されている．

#### 機器情報テーブル

機器 ID，ユーザー ID をキーとして，機器の名前，機器の詳細説明を記録し保持するテーブルである．

表 5.1: 機器情報テーブル (物理名: devices)

論理フィールド名	物理フィールド名	型	制約
機器 ID	did	Text	Primary key
ユーザ ID	uid	Integer	Primary key
機器名	name	Text	
詳細説明	description	Text	

## ユーザー テーブル

ユーザー ID をキーとして、ユーザー名とパスワードを記録し保持するテーブルである。

表 5.2: ユーザ テーブル (物理名: User)

論理フィールド名	物理フィールド名	型	制約
ユーザ ID	uid	Integer	Primary key, auto increment
メールアドレス	email	Text	
パスワード	pass	Text	

## 各テーブルの関連

図 5.9 は、各テーブルの関連を表している。ユーザ テーブルと機器情報 テーブルは、ユーザ ID にて関連付けられている。ユーザは 0 以上の機器情報と関連付けられ、機器情報は一つのユーザーに関連付けられている。

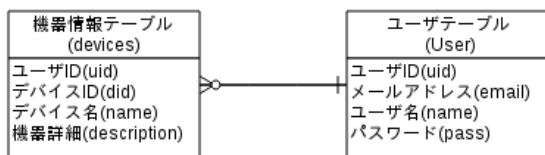


図 5.9: 機器情報 テーブルとユーザ テーブルの関連

## 5.2.4 機器状態データベース

機器状態データベースの役割は、機器の状態を時刻と共に記録・保持することである。機器状態データベースには、Influxdb を用いた。機器 ID をメトリクス名（テーブル名）とし、時刻をキーとして、機器の状態を記録している。

表 5.3:

フィールド名	型	制約
時刻	Timestamp	
状態	Text	

## 5.2.5 Web サーバーアプリケーション

Web サーバーアプリケーションの役割は、与えられた HTTP リクエストを元に、Web ページや、各種情報を返却することにある。Flask と呼ばれる Web アプリケーションフレームワークを用いた。Python を使用し

ている。Web サーバーアプリケーションの設計と Web アプリケーションの設計から、下記の様に実装した。先頭に付いている GET や POST は HTTP メソッド、/login 等は URL を示している。

**GET /login**

ログインページを返す。

**POST /login**

メールアドレスとパスワードを受け取る。クッキーからセッションキーを探し、存在すれば既にログインしているとして、/dashboard へのリダイレクトメッセージを返す。データベースにメールアドレスとパスワードの組が存在するか確認し、存在した場合、セッションキーを返す。存在しなかった場合、ログインエラーページを返す。

**GET /logout**

セッションキーを受け取り、該当のセッションを削除する。

**GET /dashboard**

ログインチェックをし、機器状態一覧ページを返す。

**GET /devicelog**

ログインチェックをし、過去の機器状態一覧ページを返す。

**GET /api/device/all**

ログインチェックをし、現在の状態、最後にメッセージを受け取った日時、機器 ID、機器名、機器詳細のデータを、JSON 形式にまとめたものを返す。

**GET /api/deviceID**

ログインチェックをし、ランダムにデバイス ID を生成し、JSON 形式にまとめたものを返す。

**POST /api/deviceID**

ログインチェックをし、受け取ったデバイス ID が既に存在するか確認し、その結果を JSON 形式にまとめ、返す。

**POST /api/device/{DeviceID}**

ログインチェックをし、デバイス ID と受け取った JSON データから、デバイスの新規作成・編集をし、結果を JSON 形式にまとめ、返す。

**DELETE /api/device/{DeviceID}**

ログインチェックをし、該当のデバイス ID を持つデバイスをデータベースから削除する。

### 5.2.6 Web アプリケーション

Web アプリケーションの役割は、ユーザーインターフェースを提供することである。Bootstrap, JQuery というライブラリを用いて作成した。HTML,CSS,Javascript で書かれている。定期的に Web アプリケーションサーバから状態を取得し、HTML,CSS を用いて表示する。

### 5.2.7 エージェントプログラムとエージェントプログラム用インターフェース間の通信の実装

エージェントプログラムとエージェントプログラム用インターフェースは、インターネットを介して、HTTP というプロトコルを用いて通信する。エージェントプログラムとエージェントプログラム用インターフェース間の通信は次の様な形になる。

1. エージェントプログラムがエージェントプログラム用インターフェースに対し、送信失敗回数を報告する。
2. エージェントプログラム用インターフェースは、時刻と正常である旨を機器状態データベースへ送信する。
3. 機器状態データベースより、正常に書き込んだというメッセージが返ってくる。
4. エージェントプログラム用インターフェースは、必要に応じて送信失敗回数からインターネットから切断された時刻を逆算し、その時刻と切断されていた旨を機器状態データベースへ送信する。
5. 機器状態データベースより、正常に書き込んだというメッセージが帰ってくる。
6. エージェントプログラム用インターフェースは、エージェントプログラムに対し、正常に受け付けたというメッセージを返す。

エージェントプログラムとエージェントプログラム用インターフェースの間の通信は、約 1 分おきに繰り返される。

HTTP 上でやり取りするデータの形式としては、Javascript Object Notation(JSON) という形式を用いる。エージェントプログラムからは、次のような形式でメッセージが送られる。

ソースコード 5.1: エージェントプログラムからエージェントプログラム用インターフェースに送られるメッセージの書式

---

```
1 {"seq":<NOW>, "stat":"OK", "log":{"seq":<PAST>}}
```

---

< NOW > には、現在の送信失敗回数が入る。< PAST > には、過去の送信失敗回数が入る。

また、エージェントプログラム用インターフェースからは、次のような形式で応答がある。

ソースコード 5.2: エージェントプログラム用インターフェースからエージェントプログラムへの応答の形式

---

```
1 {"stat":"OK", "time":<NOWTIME> }
```

---

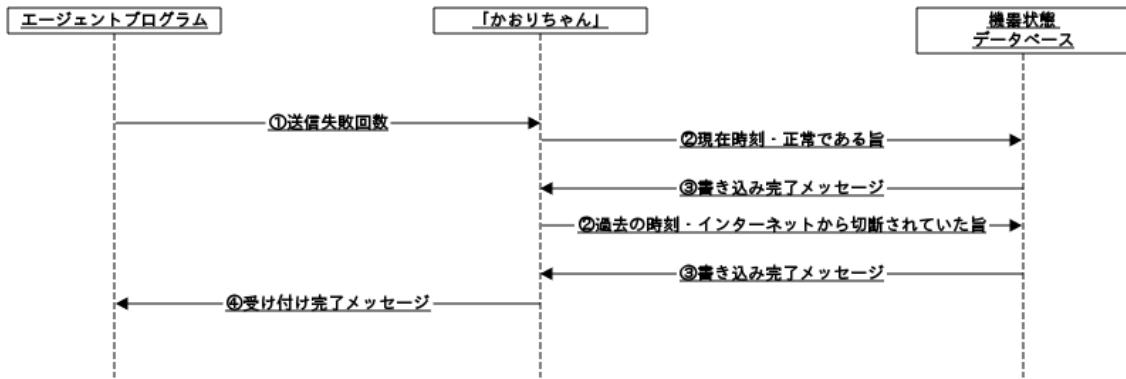


図 5.10: エージェントプログラムとエージェントプログラム用インターフェースの間のメッセージシーケンス図

ここで，< NOWTIME > にはサーバー側の現在時刻が入る．

### 5.2.8 Web アプリケーションサーバと Web アプリケーション間の通信の実装

Web アプリケーションサーバと Web アプリケーションは，インターネットを介し，HTTP というプロトコルを用いて通信する．HTTP 上でやり取りするデータの形式としては，Javascript Object Notation(JSON) という形式を用いる．

## 5.3 ソースコード

下記 URL にて、ソースコードを公開している。<https://github.com/miyasakakoki/devicemonit/tree/feature>

## 5.4 サービスによる監視のイメージ

本サービスで想定するユーザーの動きをまとめる．

- ユーザーが IoT 機器をサービスに登録する場合
  1. ユーザーは最初にサービスにログインし，機器一覧ページへ遷移する．
  2. ユーザーは機器一覧ページにある「+」ボタンを押す．すると，機器追加用ダイアログが表示される．
  3. 機器 ID が既に決まっている場合は，機器 ID 欄を書き換える．  
決まっていない場合は，機器 ID 欄に表示されている機器 ID をメモしておく．
  4. 機器名，機器詳細を入力する．

5. 機器の追加ボタンを押し，機器一覧に追加した機器が表示されていることを確認する．
  6. ユーザーは，IoT 機器へエージェントプログラムをインストールし，自動でエージェントプログラムが起動するよう設定する．
  7. ユーザーは，IoT 機器をインターネットに繋ぎ，サービスの機器状態一覧ページの表示が変わったことを確認する．
- ユーザーが IoT 機器の情報を変更する場合
    1. ユーザーは，サービスにログインし，機器一覧ページへ遷移する．
    2. 該当の機器をクリックし開く．
    3. 該当の機器の編集ボタンを押すと，機器情報編集ダイアログが表示される．
    4. 機器名，機器詳細を編集し，OK ボタンを押す．
  - ユーザーが IoT 機器を削除する場合
    1. ユーザーは，サービスにログインし，機器一覧ページへ遷移する．
    2. 該当の機器をクリックし開く．
    3. 該当の機器の削除ボタンを押すと，確認ダイアログが表示されるので，OK を押す．
    4. 機器一覧ページから，機器が消えたことを確認する．
  - ユーザーが現在の機器の情報，機器の状態を確認する場合
    1. ユーザーはサービスにログインし，機器一覧ページへ遷移する．
    2. 該当の機器をクリックし開くと，機器の情報がと，現在の状態，最後に通信があった日時等が表示される．
    3. また，機器一覧ページにて機器の背景が，緑色である場合は正常で，赤色である場合は異常である．
  - ユーザーが過去の機器状態を確認する場合
    1. ユーザーは，サービスにログインし，機器一覧ページへ遷移する．
    2. ナビバーから過去の機器状態ページへ遷移する．
    3. すると，機器と機器の過去の状態一覧が表示されるので，該当の機器を見つけ出し，確認する．

# 第6章 機器監視サービスのユーザーテストと考察

本システムの機能検証の為に，ユーザーテストを行った．本章では，ユーザーテストの結果を報告し，考察する．

## 6.1 機器監視機能のテスト

必要となる機能が実装されていることを確認するため，検証を行った．

### 6.1.1 実験のシナリオ

ユーザーテストは，1台のIoT機器で成り立っているIoTサービスを想定し行った．IoT機器には，RaspberryPiを使用する．図6.1は使用したRaspberryPi2と，IntelEdisonである．今回は、IntelEdisonは使用していないが，参考の為に上げた．



図 6.1: IoT サービスの構成図

RaspberryPi2は無線LANインターフェースを持たないので，バファロー製の無線LANインターフェースを使用した．

期間は2017年1月28日正午から1時間行い，途中何度かIoT機器(RaspberryPi)の電源を抜き，正常に検知することを確認する．その後，電源が抜けていた期間について，IoT機器の記録を確認する．

### 6.1.2 実験結果

IoT 機器の監視については問題なく動作した事を確認した。機器の電源を抜くと、1 分程度で表示状態が変わり、数分後、機器の電源を再び入れると、2 分程度で表示状態が戻ることを確認した。

## 6.2 IoT 機器を監視までの手順の比較

従来手法による監視の手間と、本サービスによる監視の手間を比較する。ここでは、従来の手法として、Telegraf と Influxdb、Grafana を用いた機器状態の監視システムを比較対象に挙げる。  
順に結果を報告する。

### 6.2.1 機器監視サーバーの構築

従来の手法では、機器監視をするために、次の様な手間があった。

1. 機器監視の為にサーバーを用意する
2. 機器監視サーバへ Influxdb をインストールする
3. 機器監視サーバへ Grafana をインストールする
4. Influxdb に対し、Telegraf との通信に用いるインターフェースの設定を行う
5. Grafana に対し、Influxdb と連携するための設定をする。

本サービスを用いる場合、これら準備は不要となる。

### 6.2.2 各 IoT 機器の状態を可視化する

従来の手法では IoT 機器の状態を可視化するまでに以下の手順を踏んでいた。

1. IoT 機器に Telegraf をインストールする
2. IoT 機器にインストールされた Telegraf の接続先の設定を行う
3. 各 IoT 機器固有の ID を Telegraf に設定する。
4. Grafana にログインする。
5. Grafana へ、現在の状態を表示する為のデータベースクエリを入力する。
6. Grafana へ、過去の状態を表示するためのデータベースクエリを入力する。

7. 各描画パネルに対し，機器名を設定する .

本サービスでは次の手順を踏む .

1. 本サービスにログインする
2. 本サービスの機器追加ボタンを押し，機器 ID をメモ（コピー）する .
3. 本サービスに対し，機器名と機器詳細を入力する .
4. IoT 機器にエージェントプログラムをインストールする .
5. IoT 機器にエージェントプログラムを自動で起動するよう設定する .

比較すると，Grafana ヘデータベースクエリを入力する作業が，なくなっている．代わりに，自動起動の為の設定ファイルを設定する手順が増えている .

### 6.3 考察

ユーザーテストから本システムは，ある程度有効であることが分かったが，以下の点について課題があることが分かった .

- 機器への設定について

設定ファイルの編集等の手間は削減されたが，エージェントプログラムを IoT 機器にインストールするのに手間がかかっていることが分かった．従来手法を用いて機器の監視をする際は，自動起動の設定を行う必要はなかったが，本プログラムでは必要としている．自動起動の為の設定ファイルを自動生成するか，あるいはエージェントプログラム配布の際，簡単なインストールスクリプト等と一緒に配布することで，大きな手間の削減になると感じている .

- ユーザーインターフェースについて

現在，過去の機器状態の記録については，文字記録として表示しているが，グラフ表示等の方が見やすいと感じた．また，期間を指定して閲覧できる機能も必要であることが分かった .

- エージェントプログラムについて

本サービスで提供しているエージェントプログラムは，その他のパッケージに依存しない単純な構成であるため，移植性が高い .

## 第7章 おわりに

IoT とは、様々なモノにコンピュータを取り付け、インターネットを介して相互に情報をやり取りすることで、様々な自動化を図ろうという概念である。IoT サービスとは、IoT による利便性をユーザーに提供するもので、IoT 機器とサーバーのプログラムがインターネットを介して通信し合うことで成り立っている。IoT サービスの円滑な提供のためには、この構造を維持しなければならない。

IoT サービスの構造の維持には、次のような事が障壁となり、提供者の負担となっている。

- ネットワークの監視は IoT サービスが提供する機能とは別の機能である事
- IoT 機器の接続されるネットワークは想定できない事
- IoT 機器の接続されるネットワークがプライベートアドレスである場合が多い事
- IoT 機器は大量に利用される為、サーバへ設定することが負担となること

そこで、本研究では、IoT 機器から通知を送ることで、IoT 機器が接続されるネットワークによらない機器の監視を実現し、IoT サービスとは別個の機器監視サービスとして開発することで、提供者の負担を軽減することを提案した。既存手法では何が足りないのか議論し、IoT サービスの実践やサービス提供者への聞き取りを通して、サービスに必要な要件を抽出し、設計を行った。この機器監視サービスを実現する為、IoT 機器にインストールされるエージェントプログラム、機器監視サーバ上で動作するエージェントプログラム用インターフェース、Web アプリケーションを作成した。

また、要件を満たしているか検証を行い、IoT 機器の動作状態について監視できていることを確認した。今後の課題として、IoT 機器への設定の簡略化や、ユーザインターフェースの向上、様々な IoT 機器への対応があることがわかった。

## 第8章 謝辞

本論文は、著者が神戸情報大学院大学情報技術研究科情報技術専攻在学中に、横山研究室にて行った研究をまとめたものです。本研究に関してご指導ご鞭撻を頂きました横山輝明講師に心より感謝申し上げます。また、本論文をご精読頂き、有用なコメントと励ましをくださった藤原明生准教授に深謝致します。

そして、論文を書く際にアドバイスをくださった、嶋教授と、嶋研究室の渡邊香織さん、田頭潤さん、笠谷拓伸さん、また、研究についてアドバイスを頂いた横山研究室OBの鄒曉明さんと、良き議論相手である京都産業大学大学院M1の石原真太郎さんに感謝致します。

# 参考文献

- [1] トレンド・イノベーション 稲田修一 「ビッグデータ活用でビジネスはどう変わったか～コマツにおけるモノのインターネット事例から考える～」 <https://www.salesforce.com/jp/blog/2013/12/vol3-bigdata.html> (2017年2月10日 閲覧)
- [2] Richo JAPAN Corp. 「出力機器のリモート管理サービス「@Remote」」 <https://www.ricoh.co.jp/remote/> (2017年2月10日 閲覧)
- [3] 日経テクノロジーオンライン 高野 敦 「[ 生体センシング ] "着る "センサーで健康情報を計測」 <http://itpro.nikkeibp.co.jp/article/COLUMN/20140526/559230/> (2017年2月10日 閲覧)
- [4] 株式会社 SORACOM 「SORACOM の概要」 <https://soracom.jp/overview/> (2017年2月7日 閲覧)
- [5] TechCrunch Japan 「【詳報】ソラコムがペールを脱いだ、月額300円からのIoT向けMVNOサービスの狙いとは？」 <http://jp.techcrunch.com/2015/09/30/soracom-launches-mvno-service-for-iot/> (2017年2月7日 閲覧)
- [6] 株式会社沖縄アイオー 金城辰一郎 「ソラコムによるIoTサービス内容とは？非エンジニアがその革新的な魅力と導入事例をわかりやすく徹底解説」 [http://okinawa.io/blog/tech/soracom\\_iot](http://okinawa.io/blog/tech/soracom_iot) (2017年2月7日 閲覧)
- [7] IoTNews.jp 小泉耕二 「【前編】SORACOMが発表した新サービスは、なにがすごいのか？ SORACOM Connected」 <https://iotnews.jp/archives/12037> (2017年2月7日 閲覧)
- [8] THE BRIDGE Takeshi Hirano 「SORACOMの凄さは第三者が「SIM」を自由に発行・運用できるこどーーIoT向けモバイル通信PF、ソラコムが提供開始」 <http://thebridge.jp/2015/09/soracom> (2017年2月7日 閲覧)