オープン環境による IoT プロトタイプ構築の取り組みと適用事例

清尾　克彦[1]，大江　信宏[1]

１．はじめに

IoT（Internet of Things）は，センサを含む機器同士または機器とクラウド上のサーバをネットワーク経由で接続し,機器からのデータを分析して様々なサービスを提供する「モノのインターネット」と言われ，DX（Digital Transformation）や 5G の世界を実現する技術の１つとして注目されている．1991 年に Mark Weiser がユビキタスコンピューティングを提唱し，2000 年代から M2M（Machine-to-Machine，マシン間通信）という言葉が使われるようになり，2014 年頃から現在の IoT という言葉に置き換わってきた．このような背景のもとに，2010 年に M2M 研究会（のちに NPO 法人 M2M・IoT 研究会に衣替え，以下 M2M・IoT 研究会と称する）[1)]を立ち上げて，M2M の普及とそれを担う人材育成の取り組みを開始した．また，電気学会第 2 次 M2M 技術調査専門委員会の一員として，M2M/IoT システム全般についての調査研究を行い，「M2M/IoT システム入門」[2)]の編纂に参加してきた．サイバー大学においても 2014 年度春学期からゼミナールで講義ベースの「M2M を学ぶ」を立ち上げ，2014 年度秋学期より演習ベースの「M2M をはじめる」で M2M プロトタイプ構築演習（のちに IoT プロトタイプ構築演習に変更）を開始した．前者は 2017 年度春学期から専門基礎科目「IoT 入門」に衣替えして現在まで継続し，AI テクノロジープログラムの 1 つに組み込まれている．後者については，ゼミナール「IoT をはじめる」として継続して取り組んでいる．M2M 時代の取り組みについては，「M2M（Machine to Machine）技術の動向と応用事例」（サイバー大学紀要第 5 号/2013）等[3, 4)]で報告済みなのでこちらを参照ください．

IoT システムは多くの技術の組み合わせで実現される複雑系システムであり，実現するためには広範な技術力と労力が必要となる．また，IoT システムはアイデア次第でその応用範囲は多岐に渡ると考えられる．今まで ICT に疎遠であった応用分野（例えば農林水産業，介護事業等）の専門家との連携も必要である．アイデアが本当に効果を発揮するかを確認するためにアイデアの創出からプロトタイプ構築による評価を短期間に繰り返して，アイデアのブラッシュアップを図り，実用化を目指すことが求められている．

[1] サイバー大学 IT 総合学部・教授

このような課題を解決して，DX 時代に期待される IoT システムの実現を図るためには，オープン化及び標準化の成果を活用し，多くの技術から構成されるプロトタイプを ICT 分野の初心者でも容易に構築できるプラットフォームの整備が必要となる．

既に「オープン環境による M2M-IoT システム構築の動向と取組事例」（サイバー大学 e ラーニング研究第 4 号/2015）[5]で報告したようにプラットフォームの整備に取り組んできているが，本稿では，2014 年度秋学期以降継続して取り組んできた IoT プロトタイプ構築演習（ゼミナール「IoT をはじめる」）での成果として，「オープン環境による IoT プロトタイプ構築の取り組み」と「発展系への挑戦を含めた適用事例」について報告する．

第 2 章で IoT プロトタイプ構築演習に求められる要件をまとめ，第 3 章で IoT プロトタイプ構築演習のシステム構成を説明し，第 4 章でオープン環境を活用した IoT プロトタイプ構築の全体像を説明する．第 5 章で IoT デバイスでの事例を，第 6 章で IoT ゲートウェイでの事例を，第 7 章で IoT サービスを実現する IoT サーバでの事例を紹介する．第 8 章でゼミナール「IoT をはじめる」のカリキュラムの概要と 7 年半に渡り取り組んできた内容について受講生のアンケート結果を踏まえて分析を行う．第 9 章で今後の取り組みなどまとめを行う．

本稿で記載している内容は，7 年半の取り組みの中で実装した時点の状態を示しており，最新の状態とは異なっていることを了承ください．

２．IoT プロトタイプ構築演習に求められる要件

2.1. IoT 人材育成の必要性

今まで ICT に疎遠であったあらゆる分野で IoT の応用の可能性が広がり，応用分野のユーザである専門家が IoT システムを理解して実用化することが期待されており，誰でもが IoT を活用できるような IoT 人材育成が必要とされている[6,7]．

DX や 5G の時代を迎え，AI やクラウドとともに IoT が重要なキーテクノロジーとなっており，IoT 人材の育成は急務と言える[6]．

IoT システムは後述するように，エッジ側のセンサやアクチュエータを扱う IoT デバイスや，データの中継・変換を行う IoT ゲートウェイにおける組込みシステム技術，IoT サービスでの広範なクラウドやサーバの構築技術，エリアネットワークやアクセスネットワーク（IP ネットワーク）などのネットワーク技術などの幅広い技術の融合体であり，全体を俯瞰し，応用につなげていく技術力が必要とされる．

また，IoT システムにより，既存の事業の生産性向上と，新規事業としての新しい価値の創出が期待されている．このように実現してみないとなかなか価値を評価しにくい場合にはデザイン思考[8]のアプローチで，短期間に効率よく安価にプロトタイピングを実現しユーザを含めた評価を繰り返すことにより，真に期待されるものを実現することが求められている．

2.2. IoT 人材の育成目標

IoT システムは幅広い技術の融合体であることから，データの処理の流れに沿って，ステップバイステップで必要となる技術を積み上げていく必要がある．本稿で総括する演習の取り組み（以下,「本演習」と略記する）では，第 2.4 章に示す四部構成で各ステップを着実に理解することで，最後には全体を俯瞰して新たな応用を考えられるレベルを狙っている．特に理解を深めるには座学ではなく手を動かす演習が必要である．

また，IoT の応用分野の広がりにより，今まで ICT に疎遠だった分野の専門家が参画することが想定される．このように IoT の応用に挑戦する人材の ICT スキルは未経験者レベルから初・中・上級者レベルまで幅広く存在することが予想され，受講生のスキルレベルに対応した教育方法を提供していく必要がある．

さらに，デザイン思考のアプローチで真に役に立つ IoT 応用システムが提案できるように，オープン環境で安価で使いやすい IoT プロトタイプ構築環境を提供することが必要である．

本演習では，これらの IoT 人材の育成目標を踏まえて，四部構成によりステップバイステップで学習するスタイルを踏襲し，いろいろなスキルレベルの人材が混在してもそれぞれのスキルレベルに応じた満足度が得られるオープン環境による IoT プロトタイプ構築演習のプラットフォームを提供していく．

2.3. 対象とする IoT システム

IoT プロトタイプ構築演習の対象とする IoT システムの全体像を図 2.1 に示す．大きく 4 つの機能から構成されている．

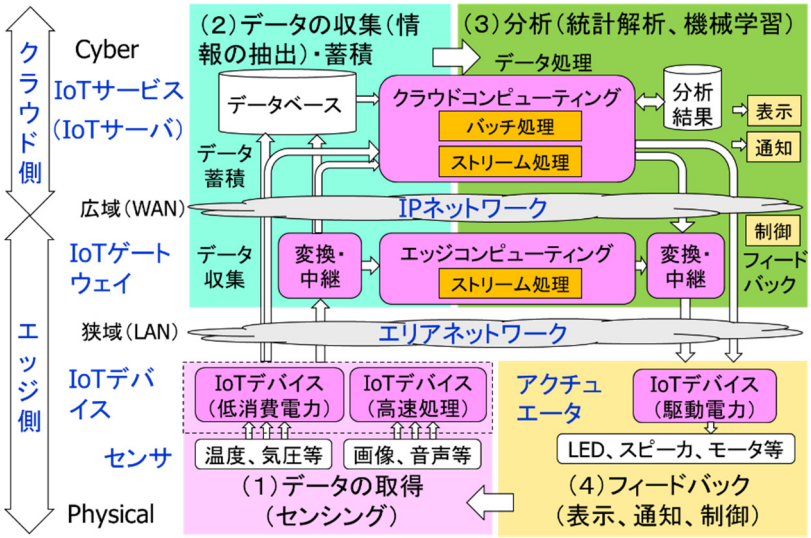

図 2.1　IoT システムの全体像

（Ｉ）データの取得（センシング）：エッジ側に位置する IoT デバイスでセンサ等のデータを取得する．センサとして，温度や照度のように少量のデータを低頻度で扱う低消費電力志向のものと，信号波形や画像を扱う高速処理志向のものが存在するが，本演習では前者の低消費電力志向のセンサを扱う．

（２）データの収集・蓄積：IoT デバイスで取得したセンサデータを収集しクラウド側に位置するデータベースに蓄積する．大量に収集されたデータはビッグデータと呼ばれる．

（３）分析（統計解析，機械学習）：蓄積されたデータをバッチ処理で，または，時々刻々収集されるデータをストリーム（リアルタイム）処理で分析を行い，分析の結果に応じていろいろなサービスを提供する．分析は従来からの統計解析に加え，最近では人工知能（AI）の機械学習が使われている．サービスとして，データを表示したり，メール等でイベントを通知したり，IoT デバイスのアクチュエータ等を駆動するための制御データをフィードバックしたりする．（２）と（３）の機能は IoT サービスと呼ばれ，IoT サーバ上で実現される．

（４）フィードバック（表示，通知，制御）：IoT サービスでの分析結果からフィードバックされる表示・通知・制御データに基づいて，IoT デバイス上のアクチュエータである LED を点灯したり，スピーカで音で知らせたり，モータを駆動したりする．（１）から（４）を繰り返すことにより，IoT システムの効率化・最適化・自律化が実現される．

IoT デバイスと IoT サーバの間で直接送受信するのではなく，一旦エッジ側に位置する IoT ゲートウェイで中継・変換処理を行うことを想定している．2 つ目のデータ収集・蓄積

機能と 3 つ目の分析機能をクラウド上で行う場合をクラウドコンピューティングと呼び，IoT ゲートウェイ上で行う場合をエッジコンピューティングと呼ぶ．セキュリティとリアルタイム性能を重視する場合にエッジコンピューティングが使われる．

IoT デバイスと IoT ゲートウェイの間のネットワークはエリアネットワークと呼ばれ，無線局免許不要で比較的狭い地域で複数の IoT デバイスからの情報を集約することができる．IoT デバイスへの電源の供給が難しい環境では低消費電力のネットワークが求められる．IoT ゲートウェイと IoT サーバの間のネットワークはアクセス/コアネットワークと呼ばれ，免許を取得した通信業者が運営するネットワークで，本演習は IP ネットワークでの接続を対象にしている．また，IoT システムでは，セキュリティが重要になってきているが，本演習では本格的な取り扱いはしていない．

2.4. IoT プロトタイプ構築の実現方法

（1）IoT システムは図 2.1 のようにいろいろな技術を組み合わせて実現される複雑系システムであるので，データの処理の流れに沿って，以下の四部構成で 1 ステップずつ着実に動作を確認しながら全体像を実体験していくことが必要である．

・第一部：IoT デバイスを動かす
・第二部：IoT ゲートウェイにつなぐ
・第三部：IoT サーバ（クラウド）につなぐ
・第四部：IoT アプリケーションを創る

（2）個人で容易に安価で実現できるオープンな環境が求められる．IoT デバイスや IoT ゲートウェイのコンピュータシステムとして安価なオープンハードウェアを，ソフトウェアツールとして無料のオープンソフトウェアを，IoT サーバとして無料で利用できるサービスを，ネットワークとして標準化され無料で使えるものを想定している．なお，オープン環境では突然の変更（サービスの制限や中止など）が起こるので，リスクマネジメント（代替策の準備）が必要である．

（3）開発と本格的な運用を分離し，まずは誰でもが容易に取り組める PC（Windows）ベースの開発環境を提供することで，効率の良い開発を行い，開発完了後，最適な運用形態への容易な移植を可能にする．具体的には，PC（Windows）上で，IoT ゲートウェイを開発し，その後で Raspberry Pi（Linux）に移植して 24 時間運用を行うなどのケースが該当する．

（4）受講生のスキルレベルがばらばらであることから，それぞれのスキルを持つ人の異なる達成目標を満足させられるように，コースを基本系，発展系，応用系で構成することで対応する．基本系は個別の設定以外は変更することなく動くものを提供する．発展系は基本系に埋め込まれた機能を自分の力で変更して動くようにする．応用系は基本系や発展系をベースに自分の力で機能を追加していく．

（5）サイバー大学のような非同期の遠隔授業/演習では，質問やトラブル時の対応として，公開 Q&A のディベートルーム，Zoom によるオンラインでの遠隔相談，最悪は宅急便送付による現物確認など，質問やトラブルの内容に応じて使い分ける形をとっている．

３．IoT プロトタイプ構築演習のシステム構成

3.1. IoT プロトタイプの実装構成

4 つの機能から構成される IoT プロトタイプを構築するために，図 3.1 に示すような実装構成を想定している．

・第一部：IoT デバイスを動かす：アナログ系センサやデジタル系センサからのセンサデータを取得したり，制御データに基づいて ON/OFF 系や PWM 系のアクチュエータを駆動したりする．

・第二部：IoT ゲートウェイにつなぐ：エリアネットワークを介して，IoT デバイスからのセンサデータを受信してリアルタイムでデータを処理（表示）したり，制御データを IoT デバイスにフィードバックしたりする．

・第三部：IoT サーバ（クラウド）につなぐ：アクセスネットワーク（IP ネットワーク）を介して，IoT デバイスや IoT ゲートウェイとの接続を行い，受信したセンサデータをリアルタイムで表示したりする．

・第四部：IoT アプリケーションを創る：IoT サーバ上で受信・収集したセンサデータを IoT データベース（SQL 系，NoSQL 系）に蓄積したり，ダッシュボードに表示したり，分析結果（トリガ条件の結果）に基づいてイベントをメールや SNS で通知をしたりする．このような応用はいろいろな既存の Web サービスと連携して実現を図ることができる．また，分析結果に基づいて制御データを IoT ゲートウェイ経由で IoT デバイスにフィードバックすることで IoT デバイス上のアクチュエータを駆動したりする．

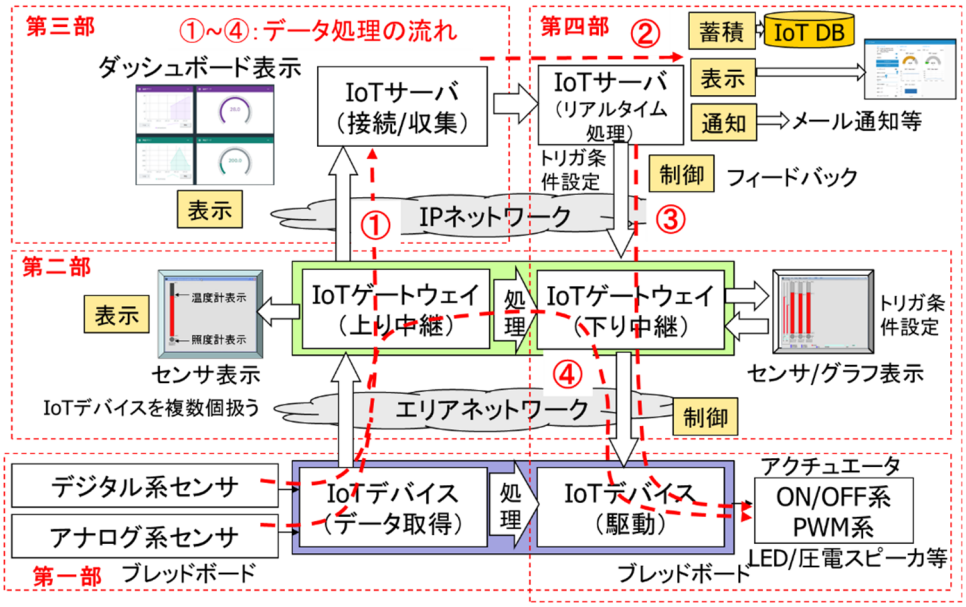

図 3.1　IoT プロトタイプの実装構成

データの処理の流れを①～④の矢印で示す．①では IoT デバイスのセンサデータの取得から，IoT ゲートウェイを経由して，IoT サーバに接続・収集するまでの上り方向のデータの流れを示す．②では IoT サーバで収集されてきたデータをリアルタイム処理（クラウドコンピューティング）で，IoT データベースに蓄積したり，ダッシュボードに表示したり，分析結果（トリガ条件成立等）に基づいて e メールや SNS を使って通知を行う．③では②と同じような分析処理の結果に基づいて，制御データを IoT ゲートウェイ経由で IoT デバイスに送り，アクチュエータを駆動する．④では IoT ゲートウェイにおいてエッジコンピューティングとして③と同じような機能を実現する．

3.2. IoT プロトタイプ構築演習のコース設定

本演習に参加する受講生のスキルレベルは，モノづくりが初めての方（初心者）から，少しやったことがある程度の初級者，似たような機器を開発したことがある中・上級者まで混在している．初心者でも上級者でも，それぞれ達成感が得られ満足度が高くなるようにコースを設定している．

IoT プロトタイプ構築演習のコース設定を図 3.2 に示す．基本系は全員が取り組む内容で第二部の左側のシリアル接続と真ん中の ZigBee 接続を選択できるようにしている．ZigBee 接続に挑戦してうまく行かない場合は，シリアル接続で最後の第四部までたどり着くことができるように構成している．最悪，エリアネットワークが不調でも，右側の仮想 IoT デバイス機能で最後まで取り組めるようにしている．基本系は受講生個別の設定が必要となる部分以外はそのまま使えば動くものを提供しているので，プログラミングの力が弱くても IoT の動作を体験できる．

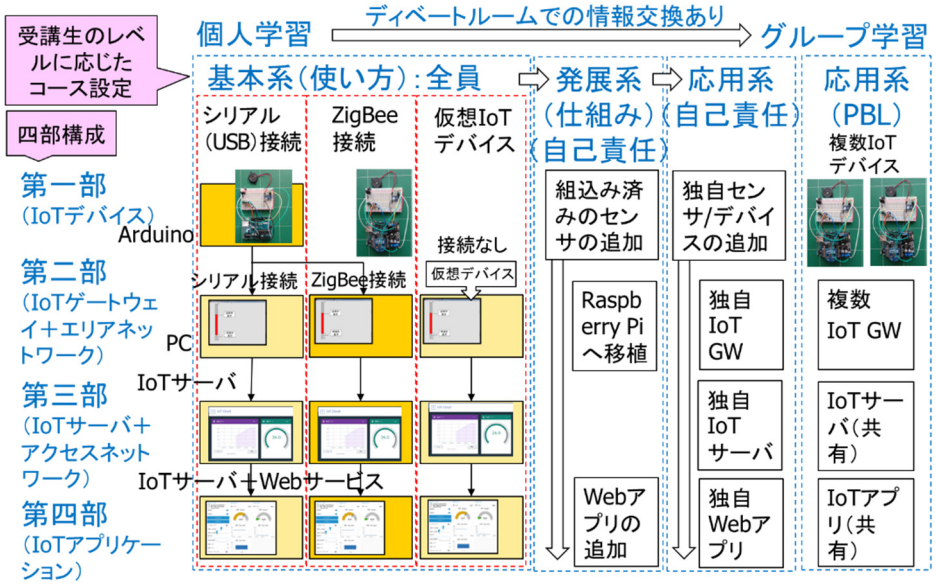

図 3.2　IoT プロトタイプ構築演習の取り組みレベル

発展系や応用系は，基本系を完成できた受講生が，基本的に自分の力で課題を解決しながら自分のスキルレベルに応じて挑戦していくコースである．発展系では，基本系にあらかじめ組み込まれた機能を自分の力で動くように拡張することで実現できる．また，ホストマシンを PC（Windows）から，例えば Raspberry Pi（Linux）に置き換えて，プログラムを移植（変更はわずか）する場合も含まれる．また，IoT サーバでの異なる Web サービスの利用なども含まれるため，ある程度のプログラミング能力を必要とする．

応用系（個人学習）では，基本系に準拠した形で，独自の機能を自分の力で組み込んでいく．ある程度レベルの高いプログラミング能力が必要である．応用系（グループ学習）では，グループ演習（PBL：Project Based Learning）を想定したもので，複数の IoT デバイスや IoT ゲートウェイと，IoT サーバを組み合わせて，グループとして 1 つの応用システムを構築する．グループ演習はサイバー大学のゼミナールではまだ取り入れていないが，外部の講習会（組込み適塾）[9]では適用している．

3.3. IoT アプリケーションのイメージ（我が家の気象台）

IoT プロトタイプ構築演習で実現する IoT アプリケーションのイメージ（我が家の気象台）を図 3.3 に示す．自宅に複数台（基本系では 1 台）の IoT デバイスを設置する．第一部対応の IoT デバイスには気象に関するセンサ（デジタルセンサ：基本系は温度センサのみ，発展系として湿度センサと気圧センサを内蔵）と照度センサ（アナログセンサ）を搭載する．アクチュエータとして LED2 個（基本系）と圧電スピーカ（発展系）を搭載する．

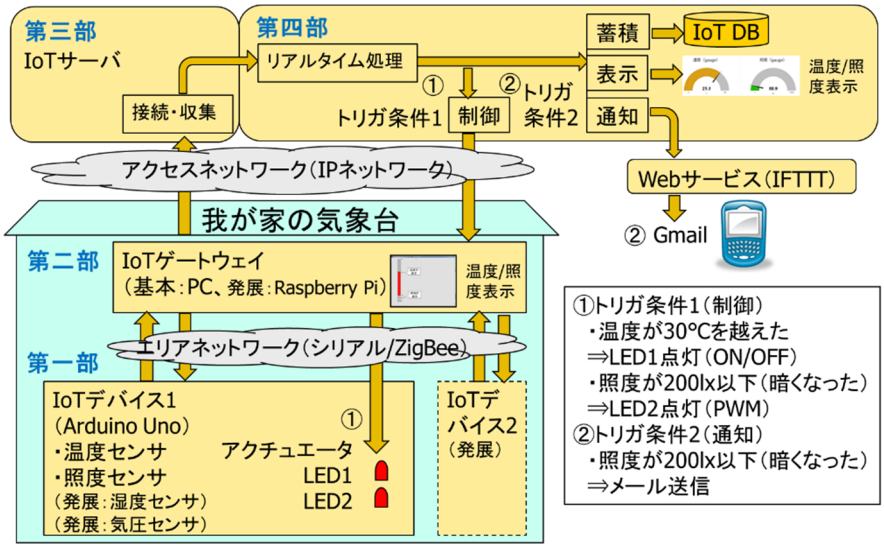

図 3.3　IoT プロトタイプ構築演習（基本系）のイメージ（我が家の気象台）

第二部対応の IoT ゲートウェイには PC（基本系：Windows）を使用する．24 時間運用する場合は，Raspberry Pi（発展系：Linux）に移植する．

第三部対応の IoT サーバでは，クレジットカードの登録なしに無料で使える範囲内で

IBM Cloud を利用していたが，2021 年 9 月以降クレジットカード登録が必要になったことから，自分の PC 上に IoT サーバを構築する形態（OwnCloud と呼ぶ）に変更している．

第四部対応の IoT アプリケーションとして気象台機能を提供する．気象データ（温度，照度データ）を IoT データベースへ蓄積し，ダッシュボードに気象データをビジブルに表示する．トリガ条件 1 の「温度が 30℃を越えたら LED1 を点灯（ON）」と「照度が 200lx 以下で LED2 を点灯（PWM=255：最大輝度：点灯に相当）のイベントに対応した制御データを IoT ゲートウェイ経由で IoT デバイスにフィードバックする．トリガ条件 2 の「照度が 200lx 以下でメール通知」のイベントに対応して，Web サービスの IFTTT に HTTP プロトコルの POST メソッドを送信（Webhook）し，アクションとして Gmail にメールを送信する．

４．オープン環境による IoT プロトタイプ構築の全体像

4.1. 全体像

IoT プロトタイプ構築の全体像（基本系＋発展/応用系）を図 4.1 に示す．

センサは非常に多くの種類があるが，本演習では基本系としてアナログの照度センサとデジタルの温度センサを扱う．発展系として基本系に内蔵されているデジタルの湿度センサと気圧センサを扱う．アクチュエータもいろいろな種類があるが，基本系では ON/OFF 制御と PWM 制御ができる LED と圧電スピーカを使用する．発展系/応用系としていろいろなセンサやアクチュエータが使われてきたが，本稿では煩雑になるので省略している．

IoT デバイスは安価なものが数多く存在するが，基本系として初心者でも扱いやすい Arduino（C/C++系）を使用する．発展系として Arduino 互換である ESPr Developer や M5Stack, 低消費電力の XBee ダイレクトや, JavaScript 系の obniz, すぐ使える SensorTag（BLE 接続等）などが存在する．詳しくは第 5 章を参照ください．

IoT ゲートウェイでは，基本系として開発が容易な Windows PC 版の Processing（Java 系）を使用する．発展系として，ほぼ同じ機能を Python や Node-RED（JavaScript 系）で実現することも可能である．また，発展系の一環として 24 時間稼働の本番を想定した Raspberry Pi への移植を容易に行うことができる．詳しくは第 6 章を参照ください．

IoT サーバでは，基本系として自分の PC 上に構築した OwnCloud（IaaS 系，Node.js＋Node-RED 環境）を使用する．IoT サーバ環境の変化により，当初の Xively/M2X（BaaS 系，無料版）から，クレジットカード登録不要の IBM Cloud（PaaS 系，無料版）を経て，

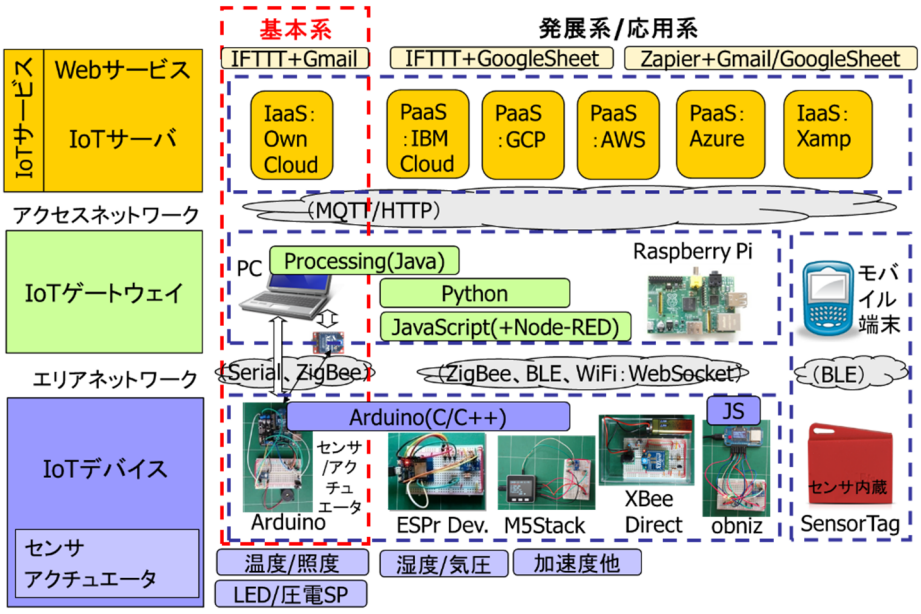

図 4.1　IoT プロトタイプ（基本系＋発展/応用系）の全体像

現在の OwnCloud に落ち着いた経緯がある．発展系として PaaS 系の IBM Cloud，AWS IoT，Google Cloud Platform（GCP），Azure IoT などが想定される．また，Web サービスとして，基本系では IFTTT が，発展系では Zapier などが考えられる．詳しくは第 7 章を参照ください．

エリアネットワークとしては，基本系では 1 対 1 有線のシリアル接続（USB）とメッシュ型無線の ZigBee を取り扱う．発展/応用系では BLE や Wi-Fi 系（IP ネットワークプロトコル）等も取り扱う．

アクセスネットワークとしては，IP ネットワークの HTTP プロトコルや MQTT プロトコルを使用する．

4.2. IoT デバイス/IoT ゲートウェイのハードウェアプラットフォーム

図 4.2 に IoT デバイス/IoT ゲートウェイのプラットフォームの全体像を示す．

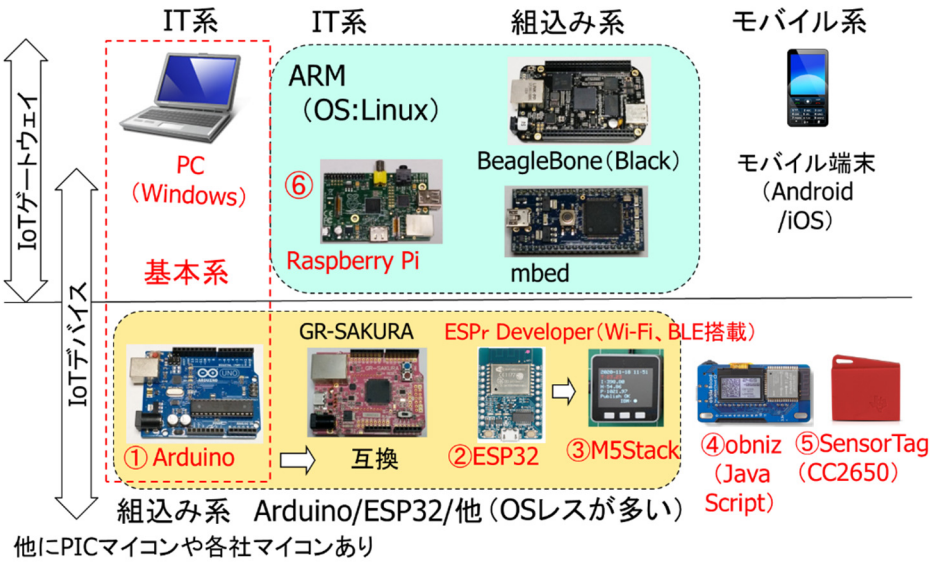

図 4.2　IoT デバイス/IoT ゲートウェイのプラットホーム

IoT デバイスでは，基本系としてイタリア発祥のオープンハードウェアの Arduino を利用する．互換機が多く存在し，ルネサスエレクトロニクスの GR-SAKURA や中国系の ESP32 マイコンを使った ESPr Developer や M5Stack 等がある．また，国内系の JavaScript 系の obniz や豊富なセンサ群を搭載した TI 社の SensorTag がある．

IoT ゲートウェイでは，基本系では身近な Windows 版の PC を想定している．本番の IoT システムでは 24 時間稼働が行われるので PC ではなく，専用の Linux 系コンピュータボードとして，イギリス発祥の Raspberry Pi や，ARM 社系の mbed，TI 社系の BeagleBone などが使われる．PC で開発される基本系のプログラムは，Java 系の Processing なので容易に Linux 系のマシンに移植が可能である．また，モバイル端末とし

て Android 系や Apple 系のスマートフォンやタブレットも使われる．

（1）Arduino[10] ボード：2005～，イタリア

フィジカルコンピューティングの元祖の 1 つで，Atmel 社の AVR マイコンを使用している．電子回路はシールドとして，プログラムはスケッチとして公開されている．シンプルであるが性能や機能に制約がある．オープンハードウェアとして基板回路図が公開されており，多くの互換機（ルネサスエレクトロニクス社の GR-SAKURA，ESPr Developer，M5Stack 等）が存在する．Arduino の概要を図 4.3 に示す．

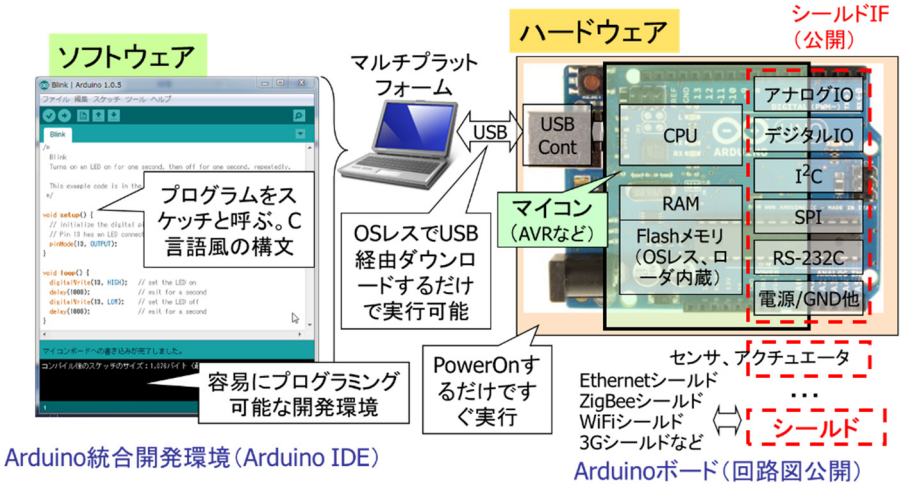

図 4.3　Arduino の概要

（2）ESPr Developer[11]：2016～，中国

中国 Espressif Systems 社の無線通信モジュール（Wi-Fi，BLE）を内蔵する低コスト，低消費電力で Arduino に比べて高性能な SoC のマイクロコントローラ（ESP32：Tensilica の Xtensa LX6 マイクロプロセッサを採用）と USB シリアル変換 IC を搭載したボードである．Arduino 環境での開発が可能である．

（3）M5Stack[12]：2017～，中国

液晶ディスプレイと Wi-Fi+Bluetooth を搭載したマイコンモジュール（ESP32）で，5.5cm の正方形の小さな筐体に，MCU，液晶，Wi-Fi，加速度センサ，スピーカ，バッテリが搭載されている．Arduino 環境での開発が可能である．さらに別売りの GPS や LoRa，各種センサやキーボードなどを積み重ねる（=stack）ように接続して拡張が可能である．

（4）obniz[13]：2017 発表，2018 販売～，日本

IoT デバイスでのソフトウェア開発は不要で，内蔵された obnizOS により汎用的な入出力端子に接続された電子部品を Web 経由で直接制御することができる．Web 上で JavaScript 等を使って容易に IoT アプリケーションを開発することができる．同様なコンセプトで，WWW（World Wide Web）で利用される技術の標準化を推進する国際的な非営利団体である W3C が標準化を進めている WoT（Web of Things）を活用した CHIRIMEN

オープンハードウェア[14]の取り組みも行われている.

（５）SensorTag[15]：米国 TI 社

手のひらサイズのパッケージに温湿度,照度,加速度など多種類のセンサと 3 種類(BLE, ZigBee，6LoWPAN）の無線規格に対応するデバイスを内蔵している．コイン・セル・バッテリを使用したバッテリ動作が可能で,低消費電力を実現している.容易に IoT ゲートウェイや IoT サーバに接続可能である.

（６）Raspberry Pi[16]：2012～，英国

ARM コアを搭載した Broadcom 社のマイクロコンピュータを使用している．IT 教育用に考案されたもので，コストパフォーマンスに優れた Linux ベースの小型 PC である．IT 教育用のため，他に比べて，I/O インタフェースが少ない．OS は組み込まれておらず，推奨の Raspbian を SD カードにダウンロードする必要がある.Raspberry Pi の概要を図 4.4 に示す.

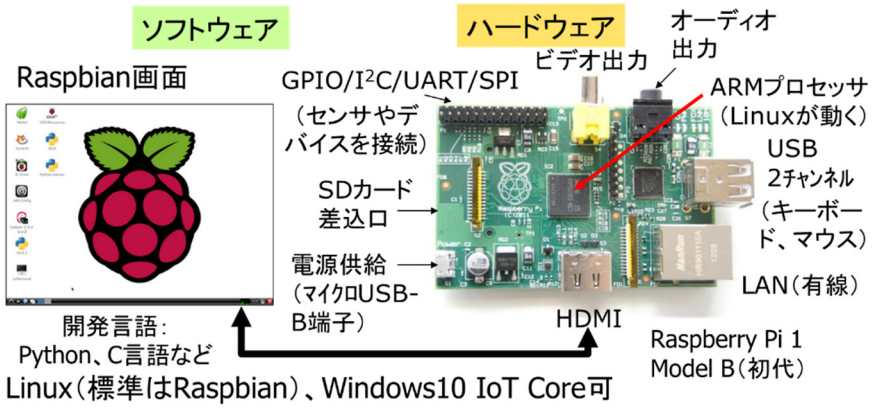

図 4.4　Raspberry Pi の概要

4.3. IoT デバイス/IoT ゲートウェイのソフトウェアプラットフォーム

図 4.5 に IoT デバイス/IoT ゲートウェイのソフトウェアプラットフォームの一覧を示す．基本系として，IoT デバイスでは Arduino IDE を，IoT ゲートウェイでは Java 系の Processing を使う．Processing（Java 系），発展系で使う Python，JavaScript（Node-RED）は，いずれも PC（Windows）で開発を行い，Raspberry Pi（Linux）へ移植することが容易である.

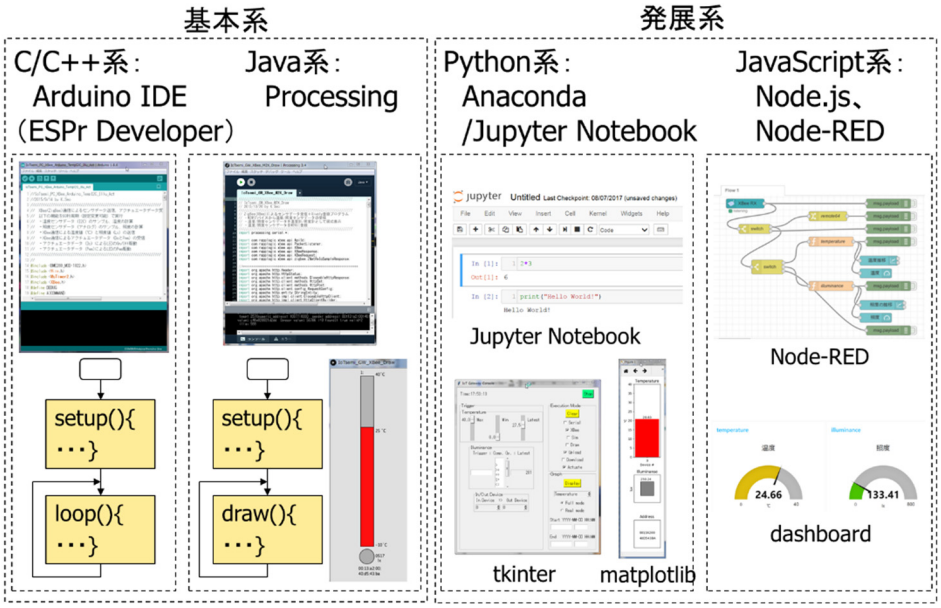

図 4.5　IoT デバイス/IoT ゲートウェイのソフトウェア開発環境（プラットフォーム）

（1）Arduino IDE[10]（統合開発環境）：C/C++系

未経験者でも簡単に利用できるようにシンプルな構成（OS レス）で，PC（Windows, Mac，Linux）上で C 言語風の「Arduino 言語」によってプログラムを制作・コンパイル・デバッグし，それを Arduino ボードに転送するだけですぐに実行できる．ライブラリが充実しており教育用としてよく使われている．プログラムのエントリポイントは図 4.5 に示すように setup()と loop()という 2 つの関数になっている．

（2）Processing[17]：Java 系

Processing はオブジェクト指向の Java をベースとしたオープンソースのプログラミング言語及び統合開発環境である．Java を初心者に使いやすくした言語で，よく使うグラフィック，サウンド，シリアル通信，ネットワークなどに関して単純化することにより，簡単に扱うことができる．特に画像やアニメーションを用いた視聴的な表現を得意としている．Windows，Mac，Linux などの Java 仮想マシンを搭載可能なコンピュータで利用できる．開発環境は Arduino IDE に似ており，エントリポイントは図 4.5 に示すように setup()と draw()という 2 つの関数になっている．

（3）Python[18]：

インタープリタ型オブジェクト指向の高水準汎用プログラミング言語で，文法が簡単で，誰でも書きやすいという特徴がある．プログラムの文書化が重視されており，言語の基本機能の一部となっている．分析アルゴリズムが共通化しやすい AI では機械学習や深層学習のライブラリが数多く提供されている．

（4）JavaScript[19]/Node-RED[20]：

当初は Web のクライアントサイドのオブジェクト指向スクリプト言語であったが，

JavaScript の実行環境として Node.js[21]のフレームワークを使うことでサーバサイド言語としても使われるようになってきた．Web や IoT のアプリケーションを作成する分野で活用されている．

Node-RED は IBM によって開発され，現在はオープンソースのビジュアルプログラミング用のフローベース開発ツールである．あまりコード（JavaScript）を書かなくてもプログラムを作成できる LowCode ツールの 1 つとして注目されている．

（５）PC（Windows 版）から Raspberry Pi（Linux 版）への移植

Java 系の Processing や Python，JavaScript（Node-RED）によるプログラムを PC（Windows 版）で開発を行い，その後本番の運用では 24 時間稼働を目指して Raspberry Pi（Linux 版）にほとんど変更なしに移植が可能である．Processing の場合は RXTX ライブラリと Port 設定部分の変更以外は修正しなくて済む．

4.4. IoT サーバ

IoT サーバのプラットフォームと IoT データベース，Web サービスについて紹介する．

4.4.1 IoT サーバのプラットフォーム

図 4.6 にクラウド環境における 3 通りの IoT サーバの実現方法を示す．また，図 4.7 に IoT サーバの 3 通りの構築例を示す．以下では無料で利用できる環境を紹介する．

（１）BaaS（Backend as a Service）版：IoT に特化した形で提供される PaaS の発展系に相当する．基本的な設定を行うことで容易に IoT サーバを実現できる．初期の頃は Xively（のちに Google が買収）[22]が有名であった．Xively の無料枠が終了したため，2016 年からほぼ機能が似ている AT&T 社の M2X[23]に移行したが，こちらも現在は無料枠のサービスを停止している．

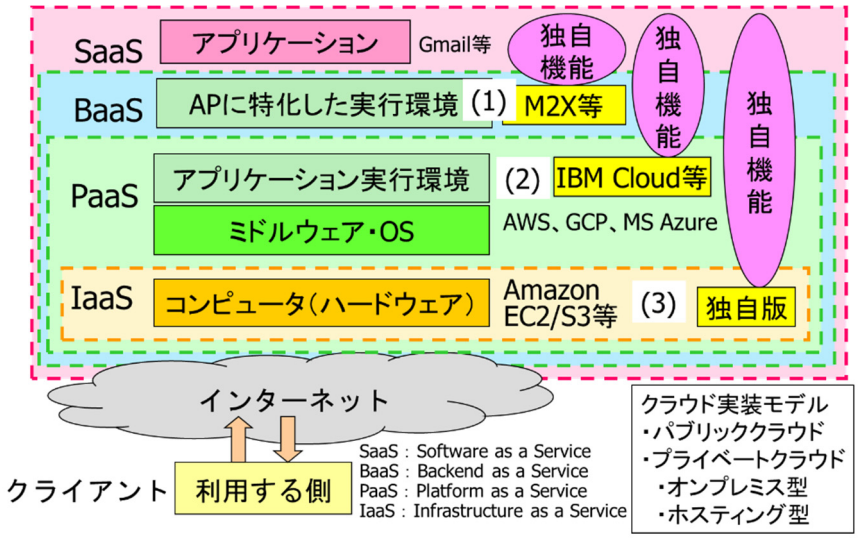

図 4.6　IoT サーバの実現方法

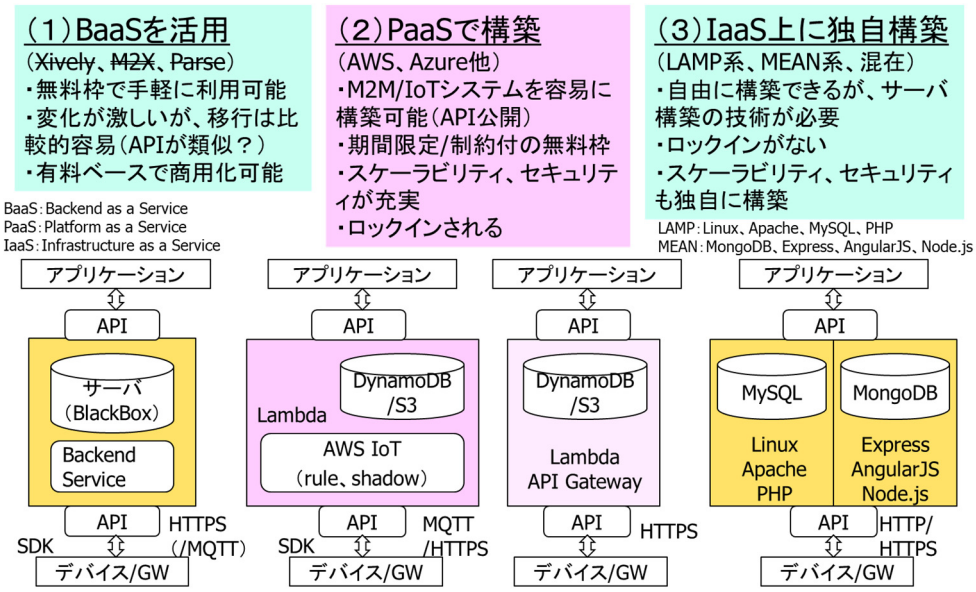

図 4.7　IoT サーバの構築例

（２）PaaS（Platform as a Service）版：アプリケーションを容易に構築できるように多くの API や機能モジュールが公開されており，それらを組み合わせることで比較的容易に IoT サーバを構築できる．本稿では 4 つの大手クラウド企業（AWS IoT, IBM Cloud, GCP, Azure IoT）を利用した構築例を示す．スケーラビリティやセキュリティが充実している．一般にセキュリティの関係でクレジットカードの登録が必要である．詳細については第 7.3 章で紹介する．

（３）IaaS（Infrastructure as a Service）版：クラウド上で提供されるハードウェアと OS 環境に独自の IoT サーバを構築する場合で，自由度があるがサーバ構築の技術を必要とする．スケーラビリティやセキュリティも独自に構築する必要がある．従来の LAMP（Linux, Apache, MySQL, PHP）系と IoT 向きの MEAN（MongoDB, Express, AngularJS, Node.js）系が存在する．なお，後者の一部の機能（Node.js＋Node-RED）を組み合わせて容易に簡易な IoT サーバを構築できるので基本系として採用している．自分の PC 上に IoT サーバを構築することから OwnCloud と呼んでいる．

4.4.2 IoT データベース（IoT DB）

IoT で扱うデータは，多種多様なデバイスやセンサから発生する多様なデータであり，いろいろな発生頻度で膨大な量のデータが収集される．このような膨大な量（Volume）・発生頻度（Velocity）・多様性（Variety）のあるビッグデータを扱うには，従来のデータの一貫性を厳密に保証するリレーショナルデータベース（RDB）では十分に対応できない．

そのため，IoT データベースとして，性能を落とさずに規模拡大に容易に対応（スケールアウトに対応）でき，非定形のデータを扱える大規模分散データベースに対応した

NoSQL のデータベースがよく使われる．図 4.8 に NoSQL の 4 つのモデルを示す．本演習では，AWS IoT でキーバリュー型の DynamoDB を，IBM Cloud でドキュメント指向型の Cloudant を取り扱っている．なお，GCP で扱う BigQuery は，NoSQL ではないが，大規模データを対象に SQL 形式の分散クエリを実現することができる．

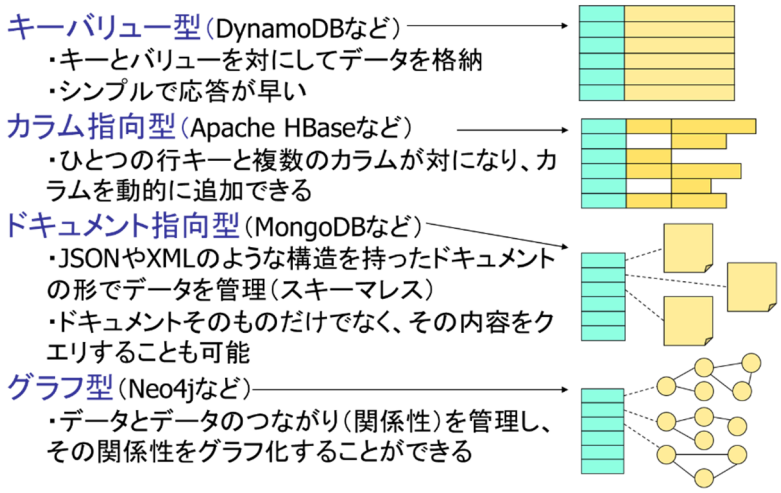

図 4.8　主な 4 つの NoSQL モデル

4.4.3 Web サービス（タスク自動化サービス）

Web サービスには多様なものがあるが，以下では IoT でよく使われるタスク自動化サービスについて紹介する．図 4.9 に一般的なタスク自動化サービスの概要を示す．IFTTT が最初に無料サービスをはじめ，その後 Zapier などのサービスが提供されている．

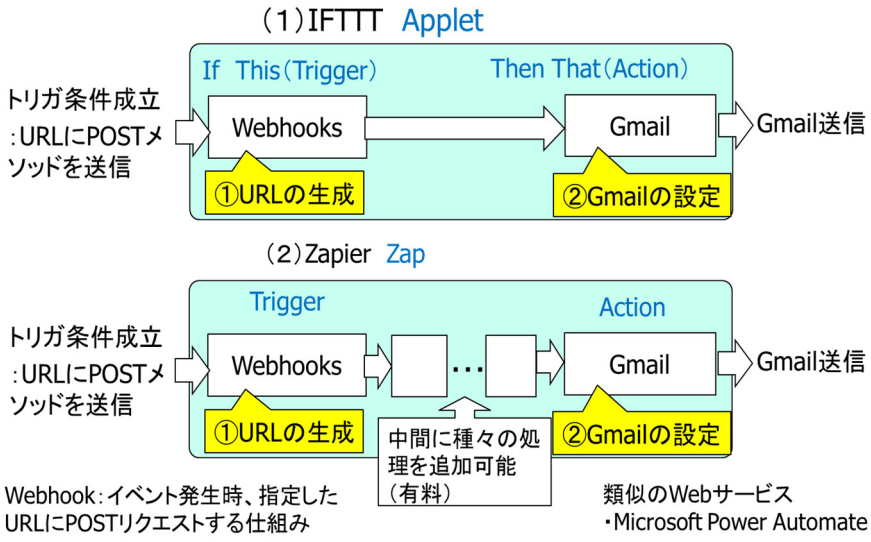

図 4.9　主な Web サービス（タスク自動化サービス）の概要

（１）IFTTT（イフト）[24]：

「if this then that」というシンプルなコンセプトに基づく Applet と呼ばれるプロフィールを使って，いろいろな Web サービス（Facebook，Evernote，Weather，Dropbox など）同士を連携することができる．本演習では，Applet の「this」の部分は「Webhooks」がトリガになり，「that」の部分は「Gmail 送信」といったアクションを扱っている．シンプルな操作性と安価なことから個人ユーザを中心に使われている．本演習では基本系として無料枠内で使っている．

（２）Zapier[25]：

Zap が IFTTT の Applet に対応する．本演習では，Zap の Trigger 部分の「Webhooks」がトリガになり，Action 部分の「Gmail 送信」がアクションとなる．トリガとアクションの間に追加のアクションを記述できるなど IFTTT に比べて高機能であり，連携アプリも多く複雑な連携や様々なアプリを使って作業する企業向きである．無料枠もあるが，この範囲では IFTTT より制約が多い．

4.5. IoT ネットワーク

図 4.10 に IoT ネットワークの全体像を示す．①は有線で USB（UART）による 1 対 1 接続を，②～④は 2.4GHz 帯のライセンス不要の無線ネットワークを，⑤は IoT デバイスと IoT サーバを直結する移動体通信型を示す．本演習では⑤については対象外としている．

本演習で取り扱う 2.4GHz 帯のライセンス不要の 3 種類の IoT 無線ネットワーク（Wi-Fi，BLE，ZigBee）について，図 4.11 にネットワーク通信階層を，図 4.12 に無線ネットワーク構成と通信手順について示す．最近，IoT ネットワークとして，LPWA（Low Power Wide Area）ネットワークが注目されているが，本演習では取り扱っていない．

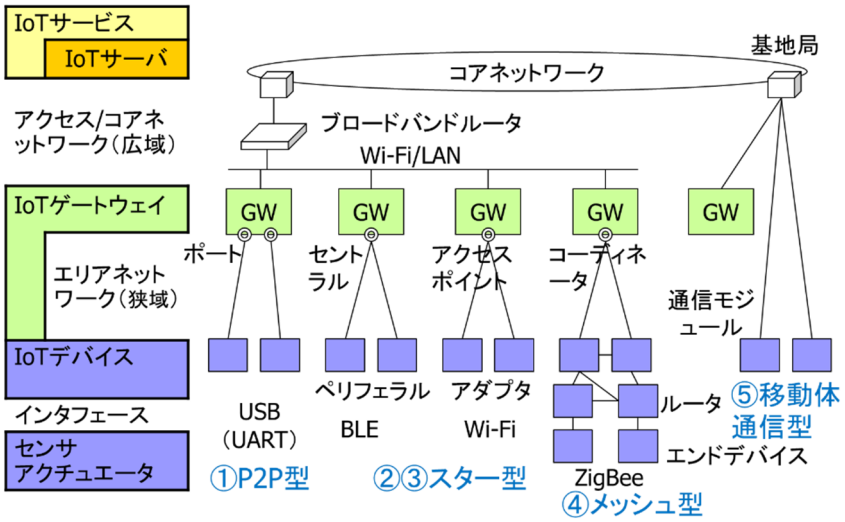

図 4.10　IoT ネットワークの全体像

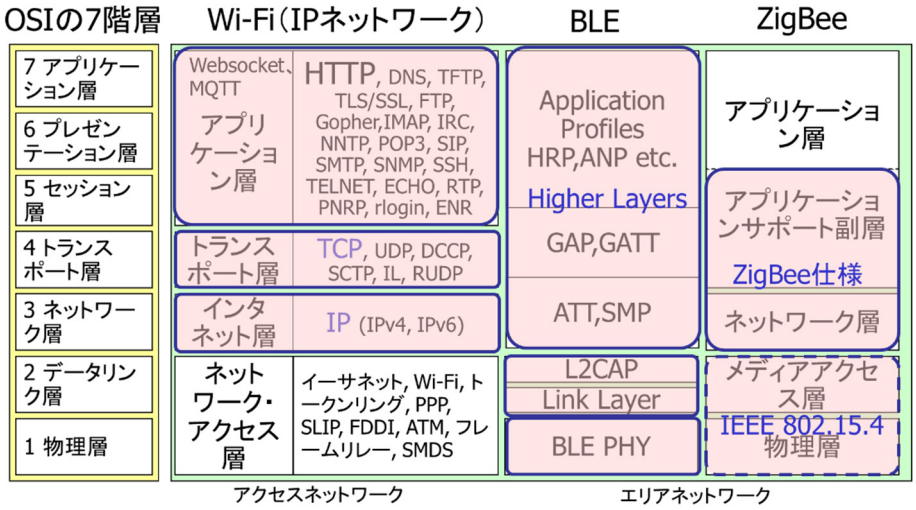

図 4.11　主な無線ネットワークのプロトコル

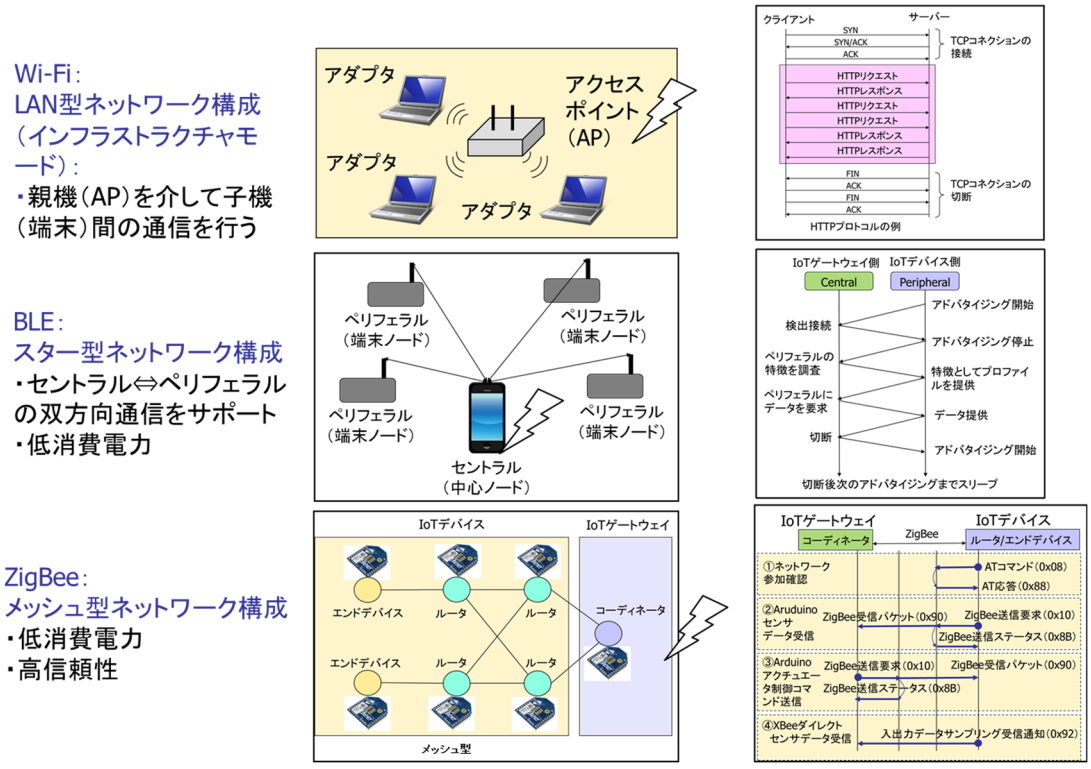

図 4.12　主な無線ネットワークのネットワーク構成と通信手順

エリアネットワークでは基本系として有線の 1 対 1 接続の USB（UART）と無線のメッシュ型の ZigBee を扱う．発展系で Wi-Fi（WebSocket 等）や BLE を利用する．

アクセスネットワークでは，Wi-Fi を使い IP 関連プロトコルを使用する．基本系では Web 操作でよく使われる HTTP や IoT 向けの MQTT プロトコルを使用する．

4.5.1 エリアネットワーク

（１）Wi-Fi：LAN 型のネットワーク構成で，アクセスポイントを経由してアダプタ間で IP 関連プロトコルによる送受信を行う．高速で多量のデータ通信を行うことができるが消費電力は大きい．

（２）BLE（Bluetooth Low Energy）：低消費電力向けの標準としてモバイル端末に装備されているスター型ネットワークで，セントラルに接続されたペリフェラルノード間で双方向通信を行う．１対１通信であれば容易に接続できるが，マルチ接続の場合はそれなりのプログラミング（通信手順）が必要となる．

（３）ZigBee[26]：メッシュ型の低消費電力・高信頼性の双方向ネットワークを提供する．通信手順に従ってプログラミングすることで，メッシュ型のネットワークを容易に構成できる．本演習ではエリアネットワークとして ZigBee を基本系として使用する．ZigBee チップとして，Digi International Inc.製の XBee モジュール[27]を採用し，メッシュ型ネットワークの設定に XCTU ツール[28]を使用する．メッシュ型ネットワークは，ネットワークの元締め（IoT ゲートウェイ側）になるコーディネータノード，IoT デバイスとして末端になる低消費電力のエンドデバイスノード，これらの間で中継点となるルータノードから構成される．複数のルータノードを経由すること（マルチホップ）で遠くまで通信できる．また，一部のルータノードが通信不能になっても他のルータノードを経由して通信できる利点がある．XCTU ツールで各 XBee モジュールを該当するノードに設定する．XCTU ツールによる XBee モジュール設定後の接続例を図 4.13 に示す．

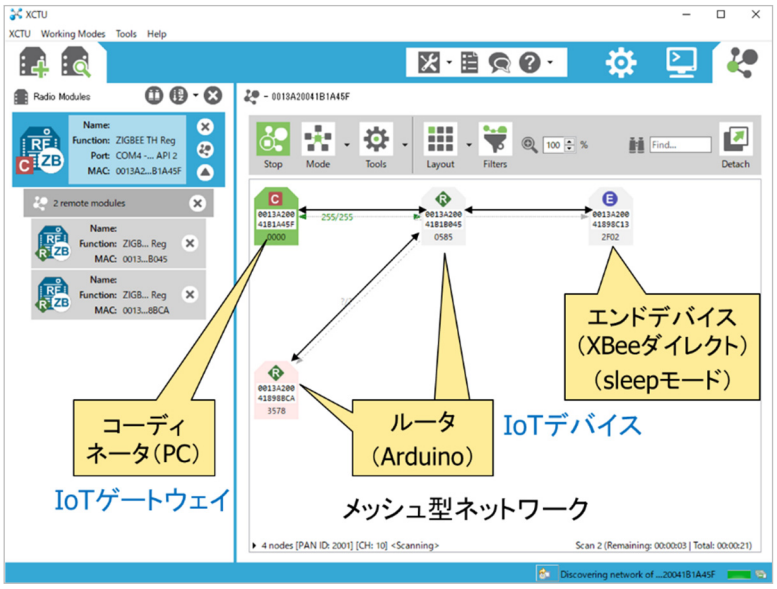

図 4.13　XCTU によるメッシュ型ネットワークの接続例

4.5.2 アクセスネットワーク（IP ネットワーク）

図 4.14 にアクセスネットワークで使う 3 種類の IP 関連通信プロトコルを示す．なお，

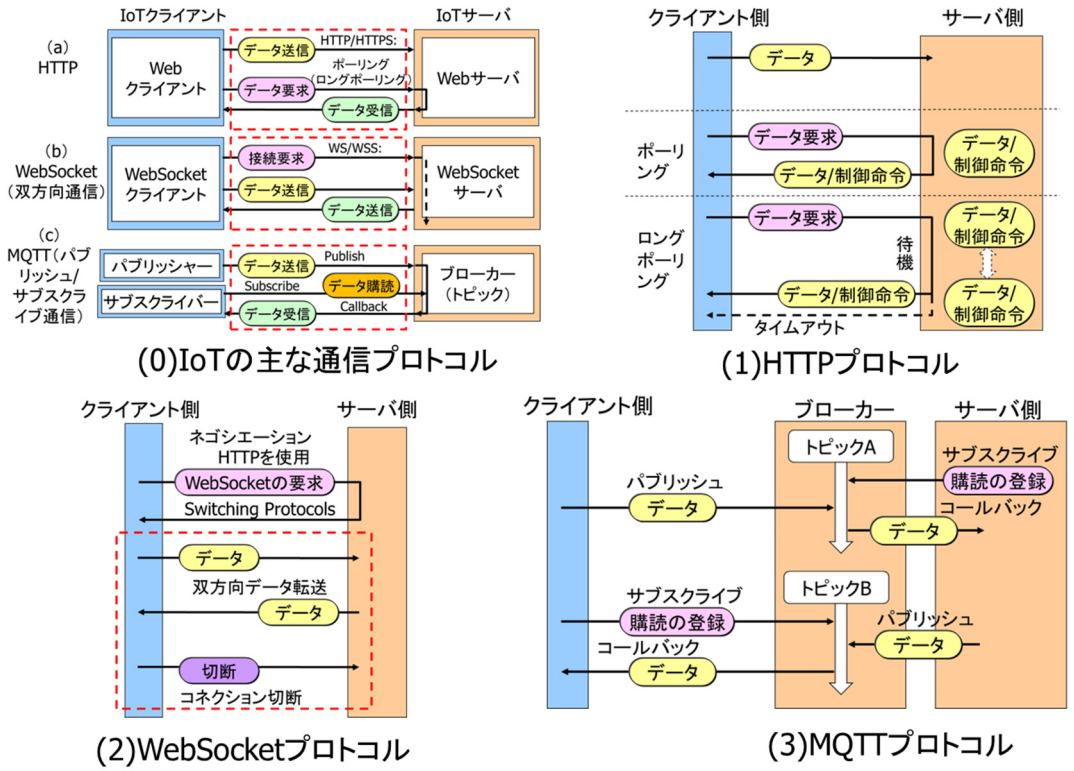

図 4.14　IoT の主な通信プロトコル（IP ネットワーク）

IoT 向けに低消費電力化を狙いとした CoAP や 6LoWPAN 等のプロトコルについては本演習では取り扱っていない.

（1）HTTP プロトコル : クライアント/サーバ型のプロトコルでクライアント側から起動してデータの送受信を行う．プロトコルの制御部分の処理が重く，微小な IoT データに対してはオーバーヘッドが多いと言われている．Web アクセス時に使う標準のプロトコルで，本演習でも基本系で取り扱う.

（2）WebSocket[29] プロトコル : HTTP プロトコルを使ってネットワークを確立した後は，クライアント側とサーバ側から双方向でデータ転送できるのが特徴である．本稿では，IoT デバイスと IoT ゲートウェイ間での実装例を第 5.4 章で紹介する.

（3）MQTT[30] プロトコル：HTTP プロトコルに比べて軽量な IoT 向けのパブリッシュ/サブスクライブ型のネットワークである．もともとは IBM 主導で開発され，現在は標準化されている．MQTT ブローカーにトピックを設定し，送信側（パブリッシャー）は指定したトピック向けに送信データをパブリッシュ（発行）する．受信側（サブスクライバー）はあらかじめ指定したトピックに対してサブスクライブ（登録）しておき，そのトピックにパブリッシュされたデータをコールバック関数で受信する．2020 年度以降の基本系では MQTT プロトコルを取り扱う.

５．IoT デバイスでの事例

第一部の「IoT デバイスを動かす」における基本系と発展系の事例について説明する．

5.1. IoT デバイスの全体像

IoT デバイスの全体像を図 5.1 に示す．IoT デバイスは，マイクロコンピュータボードとセンサやアクチュエータを搭載したブレッドボードから構成され，5 つの機能を搭載する．

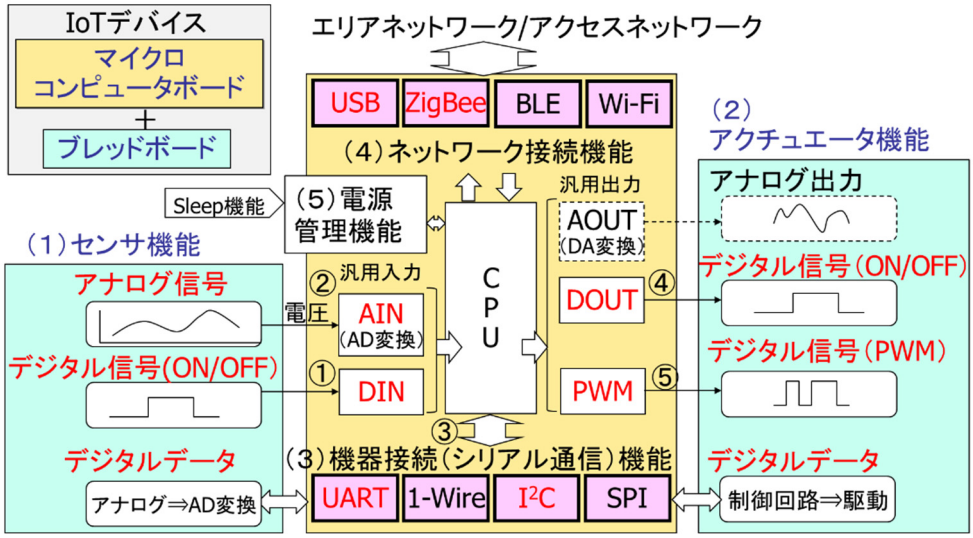

図 5.1　IoT デバイスの全体像

（１）センサ機能：デジタル信号入力（ON/OFF），アナログ信号入力，デジタルデータ入力（シリアル通信経由）の 3 種類を取り扱う．図 5.2 に示す①タクトスイッチ，②照度センサ，③温度・湿度・気圧センサがそれぞれに対応している．特にアナログセンサについては，アナログ信号の扱い方，アナログ/デジタル変換，測定値のバラツキについての理解が重要と考えている．

（２）アクチュエータ機能：デジタル信号出力（ON/OFF），デジタル信号出力（PWM），デジタルデータ出力（シリアル通信経由）の 3 種類を取り扱う．図 5.3 に光を出力する LED，音を出力する圧電スピーカ，回転運動するモータ（応用系）の例を示す．

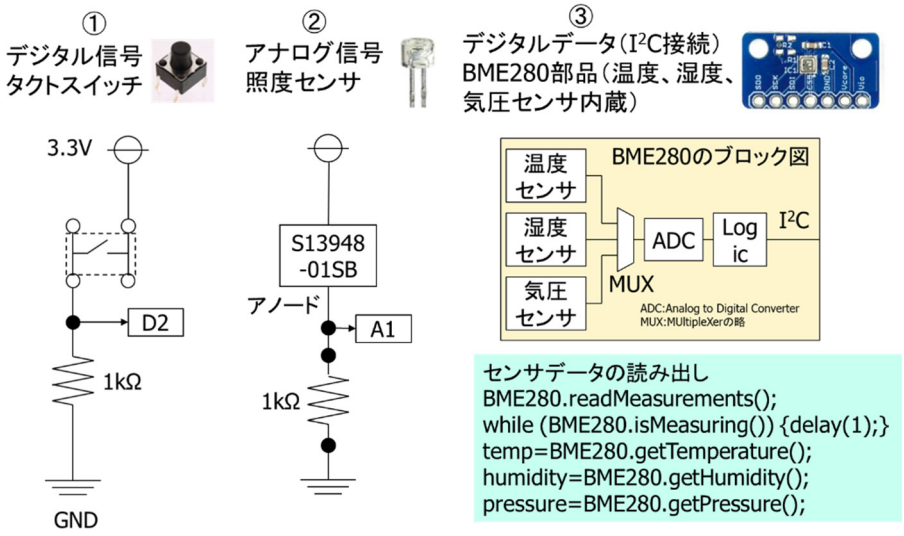

図 5.2　センサ（信号入力処理）機能

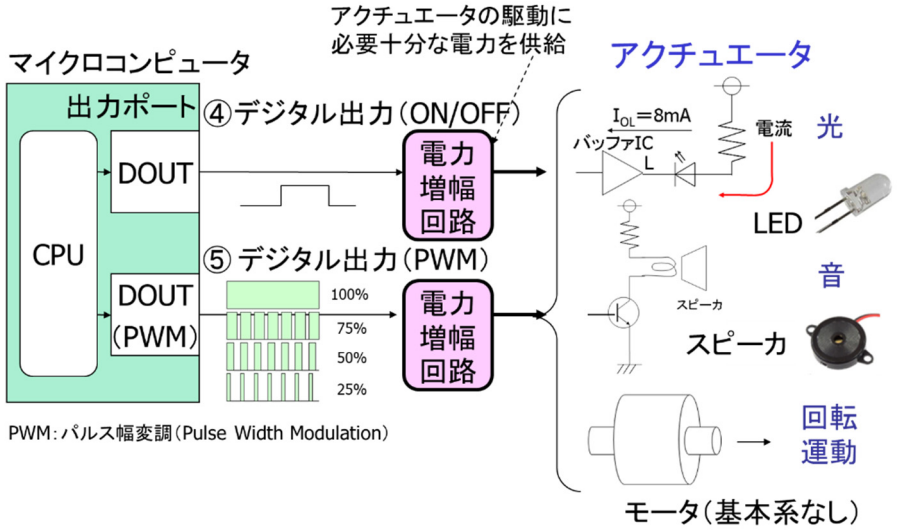

図 5.3　アクチュエータ（信号出力処理）機能

（3）機器接続（シリアル通信）機能：1 対 1 接続の UART，1 線式の 1-Wire，2 線式の I²C，3 線式の SPI が存在する．図 5.4 にそれぞれのシリアル通信方式を示す．基本系では UART と I²C を使用する．

（4）ネットワーク接続機能：1 対 1 接続の USB（UART），低消費電力志向でメッシュ型の ZigBee，低消費電力志向の BLE，高速処理が可能な Wi-Fi 等が存在し，基本系では USB（UART）と ZigBee を使用する．

（5）電源管理機能：低消費電力で運用する場合には必要最小限の期間だけ電源を供給する sleep 機能が使われる．第 5.3 章の XBee ダイレクトの例がこれに該当する．

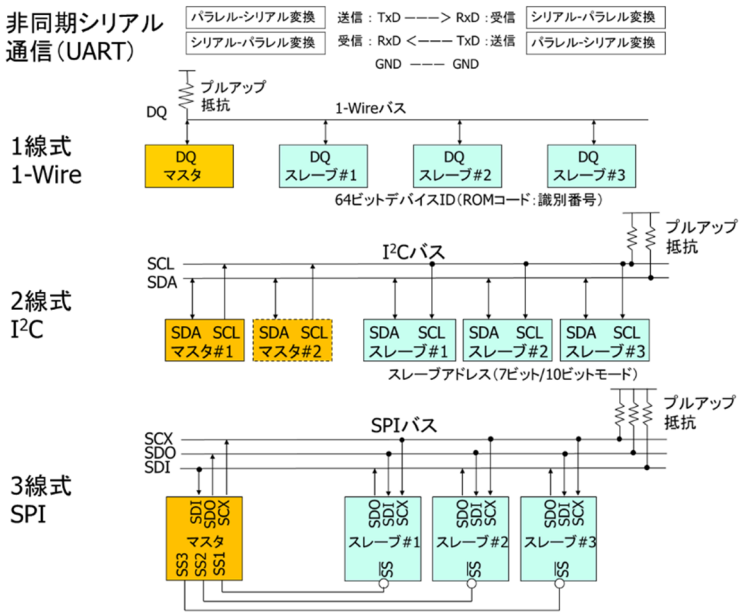

図 5.4　機器接続（シリアル通信）処理機能

5.2. IoT デバイスの基本系（Arduino Uno 版＋ZigBee 接続版）

IoT デバイスの基本系で使う部品一覧を図 5.5 に示す．13 種類の部品で総額約 1 万 6 千円になり，各自インターネット通販などを通じて調達する．

これらの部品を組み立てた完成品を図 5.6 の（a）に示す．上側のブレッドボード上にセンサやアクチュエータ等の部品を搭載し，部品端子間をジャンパーワイヤで接続する．下側の Arduino Uno ボード上に XBee モジュール（ZigBee 接続部品）付き XBee シールドを搭載し，上側のブレッドボートとの間にジャンパーコードで配線することにより IoT デバイスの基本系（ZigBee 接続版）が完成する．

図 5.5　IoT デバイス（基本系）の部品一覧

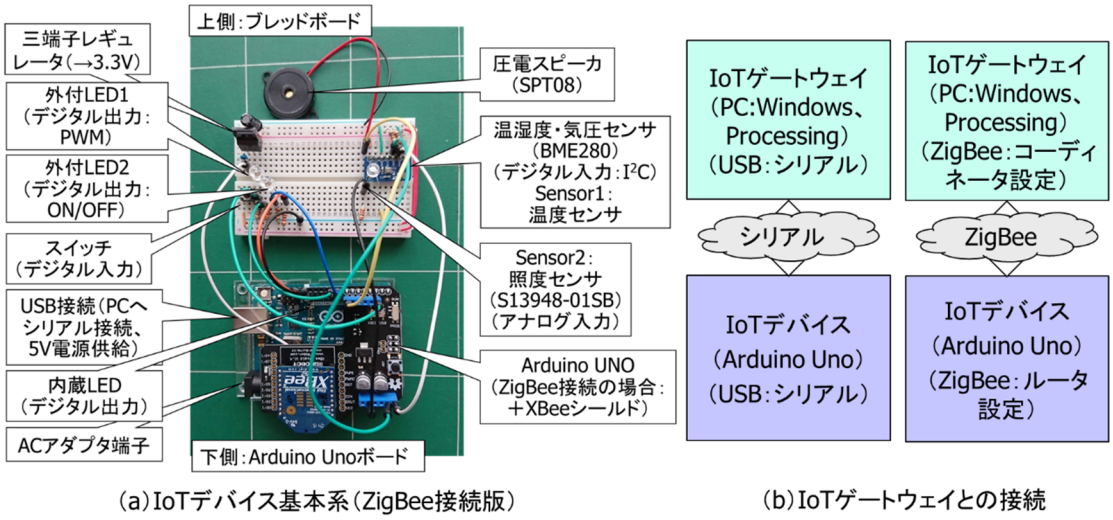

図 5.6　IoT デバイス基本系（Arduino Uno 版）の概要

IoT デバイス基本系の回路図（Fritzing[31] で作成）を図 5.7 に示す．回路図を理解して部品間の接続を確認しながら組み立てることで接続誤りを減らすことができる．

IoT デバイス基本系のプログラムフローを図 5.8 に示す．最初にライブラリ設定や変数定義などを行い，setup()関数の中で 1 回だけ実行する初期設定やエリアネットワークへの参加と確認，タイマーの設定を行う．loop()関数の中でタイマー設定した周期で，①センサデータの取得と測定値の計算，IoT ゲートウェイへの送信処理，②IoT ゲートウェイから

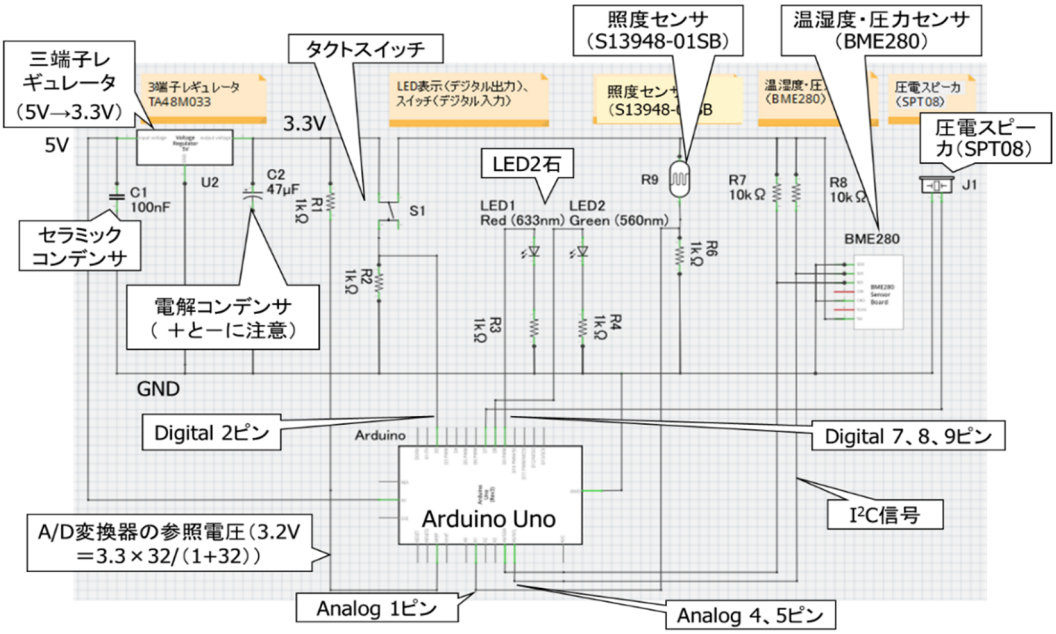

図 5.7　基本系（Arduino Uno 版）の回路図（Fritzing）

の受信処理と受信した制御データからアクチュエータ（内蔵 LED，外付 LED）の制御，③送受信処理完了の LED5 回点滅を実行する．図 5.8 の右側の上段にシリアル通信時のシリアルモニタの表示例を，下段に ZigBee 通信時の XCTU コンソール画面の表示例を示す．

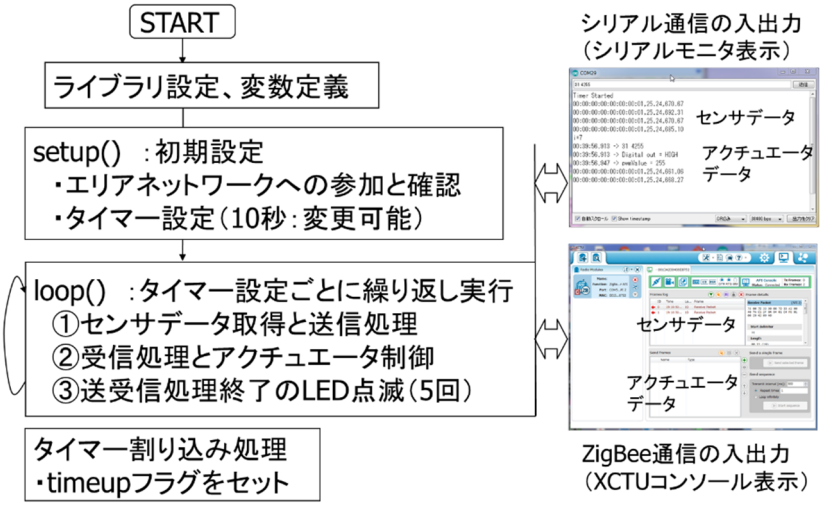

図 5.8　Arduino 基本系のプログラムフロー図

5.3. IoT デバイスの発展系：XBee ダイレクト（低消費電力版）の例

IoT デバイスへの電源供給が難しい場合は，電池や太陽電池などで駆動するので，低消費電力で実現する必要がある．

図 5.9 に示す XBee ダイレクトは，ZigBee（XBee）モジュール内に搭載されている低消費電力マイコンを使って，アナログ系温度センサと照度センサのデータを読み取り，内蔵の AD 変換器でデジタルデータに変換して，ZigBee ネットワーク経由で IoT ゲートウェイに送信する．ZigBee ネットワークノードとしてエンドデバイスに設定することで sleep モード（電源停止）の実行が可能になる．本例では 15 秒間 sleep した後に立ち上がり，データ送受信後 5 秒間経って sleep に入ることを繰り返す．

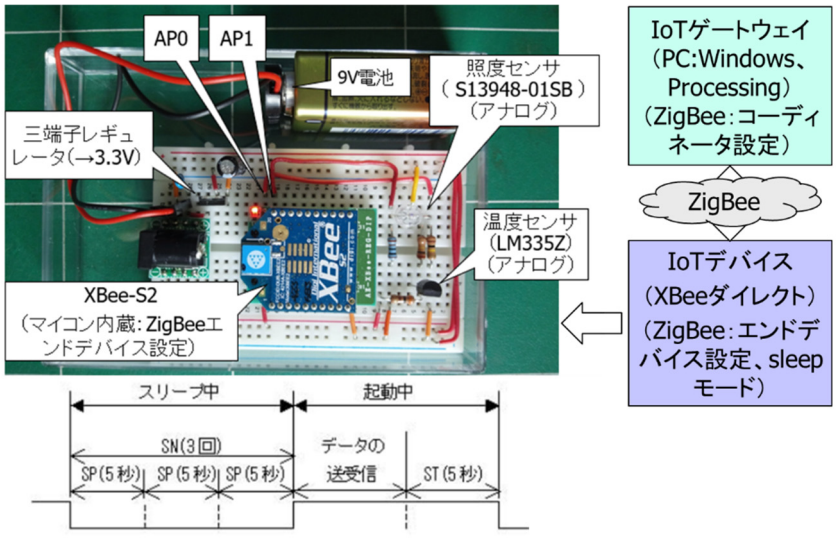

図 5.9　XBee ダイレクト（Sleep モード）の概要

5.4. IoT デバイスの発展系：ESPr Developer（高機能 Arduino 互換機）の例

Wi-Fi 機能搭載の高機能 Arduino 互換機である ESPr Developer を使った IoT デバイスの発展系を図 5.10 に示す．エリアネットワークとして Wi-Fi を使い，WebSocket プロトコルで IoT ゲートウェイと送受信する場合と，HTTP プロトコルにより IoT サーバと直接送受信する例を示している．IoT デバイス上のセンサ構成は基本系と同じである．

図 5.11 に ESPr Developer のプログラム構成を示す．Arduino 互換機なので，プログラム構成はネットワークが Wi-Fi 経由の WebSocket プロトコルであることを除いて基本系と同じである．本例では，下り方向の IoT ゲートウェイからの制御データによるアクチュエータの駆動は実装していない．

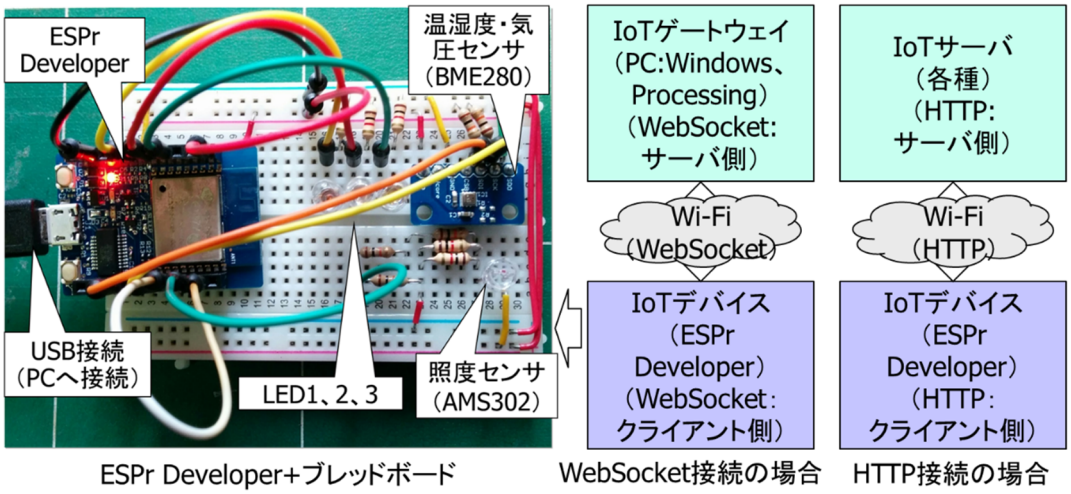

図 5.10　ESPr Developer の概要

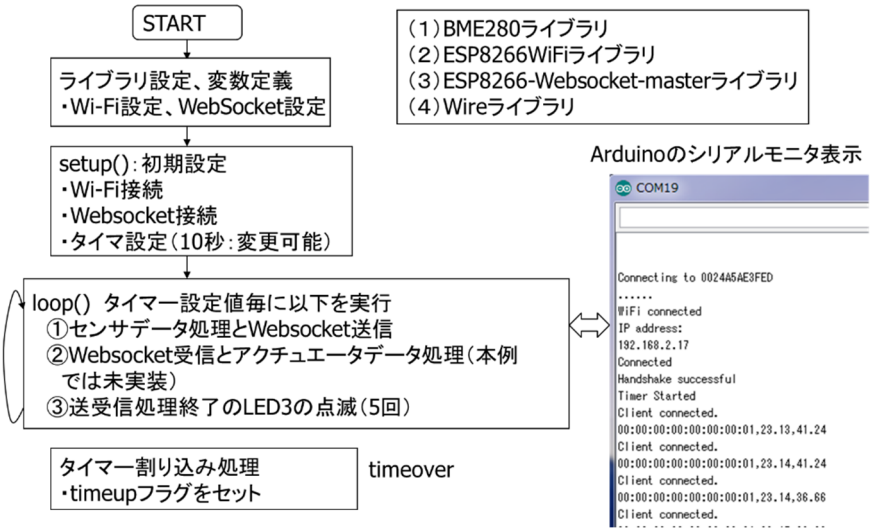

図 5.11　ESPr Developer のプログラム構成

5.5. IoT デバイスの発展系：M5Stack（高機能 Arduino 互換機）の例

Wi-Fi 機能搭載の高機能 Arduino 互換機である M5Stack Gray を使った IoT デバイスの発展系を図 5.12 に示す. エリアネットワークとして Wi-Fi を使い, MQTT プロトコルで, IoT サーバの IBM Cloud と直接送受信しているのが特徴である. IoT デバイス上のセンサ構成は基本系と同じである.

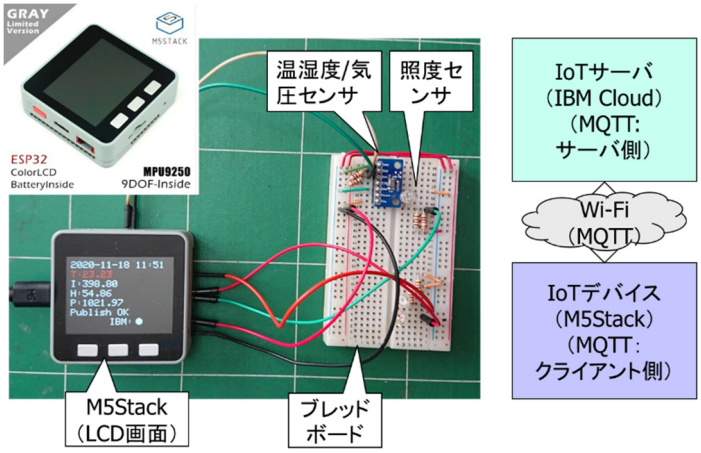

図 5.12　M5Stack の概要

図 5.13 に M5Stack のプログラム構成を示す. Arduino 互換機なので, プログラム構成はネットワークが Wi-Fi 経由の MQTT プロトコルであることを除いて基本系と同じである. 本例では, IoT サーバへの上り方向はポーリングによる 1 分周期で Publish し, IoT サーバからの下り方向では Subscribe 登録に対する callback によるイベント処理を行う（なお下り方向の制御データによるアクチュエータの駆動は実装していない）.

また, M5Stack には LCD 画面が付いているので時刻やセンサデータの表示や送信制御の指示入力が可能である.

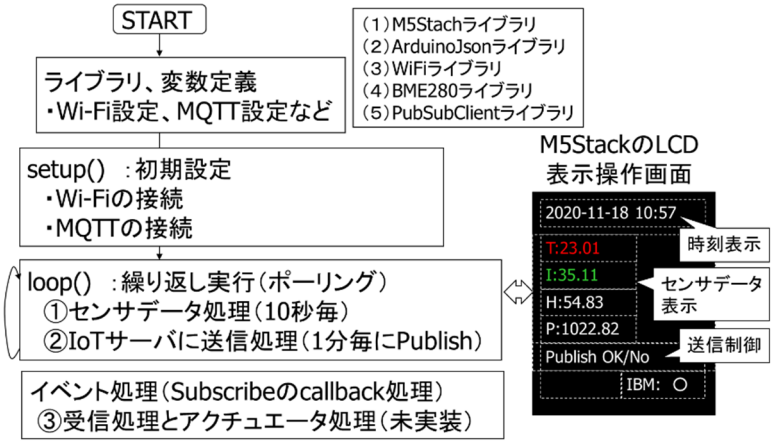

図 5.13　M5Stack のプログラム構成

5.6. IoT デバイスの発展系：obniz（JavaScript）の例

Wi-Fi 機能搭載の obniz を使った IoT デバイスの発展系を図 5.14 に示す.

IoT デバイス上のセンサ構成は基本系と同じである.

IoT デバイスには obnizOS が搭載されており，IoT ゲートウェイから Wi-Fi 経由（WebSocket プロトコル）でハードウェアを制御する API により，センサからのデータを収集したり，アクチュエータを駆動したりすることができる.

図 5.15 に obniz の IoT ゲートウェイプログラム（JavaScript）の構成を示す．ライブラリや表示パネルの設定，IBM Cloud 接続の設定等を行い，IoT デバイスである obniz に接続する．obniz に接続されているセンサやアクチュエータの設定を行い，IBM Cloud と MQTT プロトコルによる接続を行う．続いて，60 秒周期で，センサデータの読み込み，センサデータのパネル表示，Google Sheet へのデータ送信（Google Sheet ロギングモードの時），IBM Cloud への送信（IBM Cloud ロギングモードの時）を繰り返し行う．なお，本例では，IBM Cloud からの制御データによるアクチュエータの駆動は実装していない.

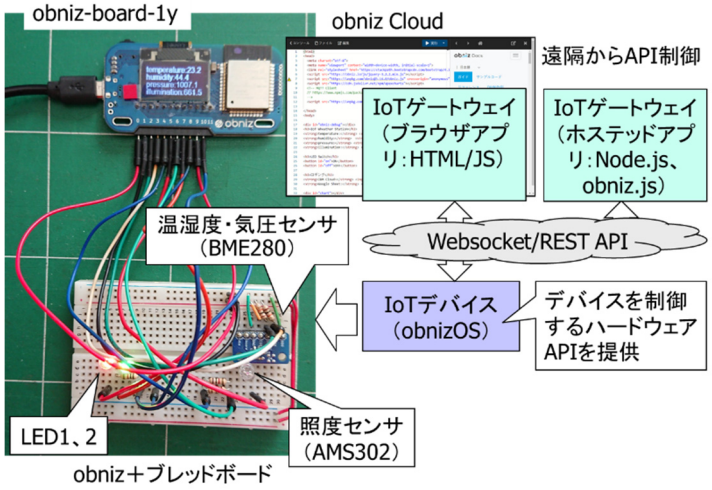

図 5.14　obniz の概要

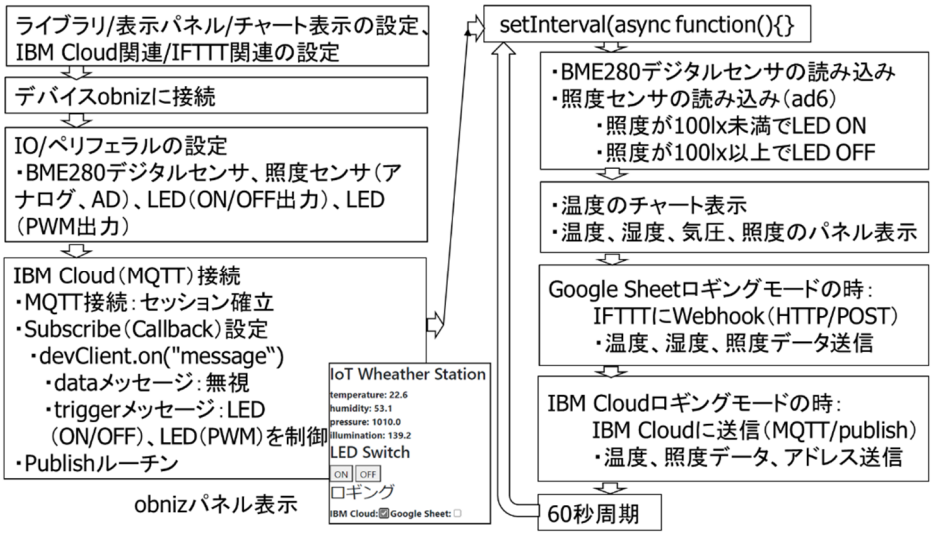

図 5.15　obniz プログラム（IoT ゲートウェイ，JS）の構成

5.7. IoT デバイスの発展系：SensorTag の例

図 5.16 に SensorTag を使った IoT デバイスの概要を示す．本例では，BLE 経由で IoT ゲートウェイに相当する Android 系のスマートフォンに接続している．コイン電池駆動の SensorTag には 10 種類のセンサがあらかじめ搭載されており，電源ボタンをオンにするだけで動作を開始する．

スマートフォンでは，Android アプリとして提供される「Simplelink SensorTag アプリ」（無料）をダウンロードすることにより，アプリの画面から操作・設定することで，10 種類のセンサを表示したり，IoT サーバにセンサデータをアップロードすることができる．なお，本例では，IoT サーバへのアップロードは実装していない．

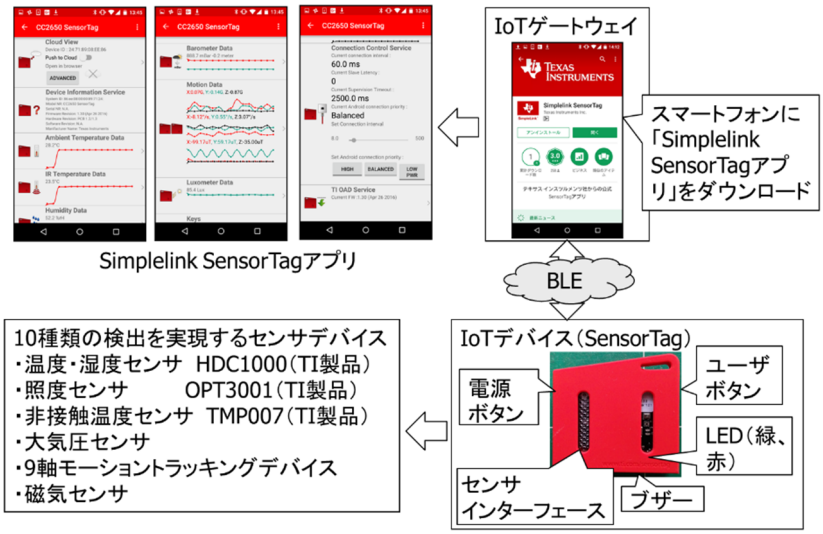

図 5.16　SensorTag の概要

5.8. IoT デバイスのまとめ

IoT デバイスの基本系としては今後も最も普及している Arduino 系を利用していく．性能不足の場合は，Arduino 互換機の ESPr Developer を想定している．

エリアネットワークの基本系としては，IoT ネットワークの特徴の 1 つであるメッシュ型の理解を重視しているので，今後も ZigBee を利用していく．

なお，IoT ゲートウェイを省略して直接 IoT サーバと接続する場合には，Wi-Fi 接続を活用したり，Arduino の代わりに Raspberry Pi を使う方法も考えられる．

６．IoT ゲートウェイでの事例

第二部の「IoT ゲートウェイにつなぐ」における基本系と発展系の事例について説明する．

6.1. IoT ゲートウェイの全体像

IoT ゲートウェイの全体像を図 6.1 に示す．IoT ゲートウェイは，エリアネットワークを介して IoT デバイスと，アクセス/コアネットワークを介して IoT サーバと接続されている．

IoT ゲートウェイは図 6.1 に示すように 5 つの機能から構成されている．

（１）エリアネットワーク接続機能：エリアネットワーク（USB（UART），ZigBee，BLE，Wi-Fi）を介して IoT デバイスとの間で送受信する．

（２）アクセス(IP)ネットワーク接続機能:アクセス(IP)ネットワーク(HTTP, WebSocket, MQTT）を介して IoT サーバとの送受信を行う．

（３）プロトコル/メッセージ変換（上り）機能：エリアネットワーク経由で IoT デバイスからのセンサデータを受信・集約し，アクセス/コアネットワークのプロトコルに変換して IoT サーバに送信する．

（４）プロトコル/メッセージ変換（下り）機能：アクセス/コアネットワーク経由で IoT サーバからの制御データを受信し，その制御データに従って，エリアネットワークのプロトコルに変換して，IoT デバイスのアクチュエータを制御する．

（５）エッジコンピューティング機能：クラウド側にセキュリティの関係でデータを出したくない場合や，迅速にリアルタイムで処理したい場合は，IoT サーバで処理するのではなく，この IoT ゲートウェイで処理を行う．本演習では，データ表示機能（基本系）やトリガ条件によるフィードバック（IoT デバイスのアクチュエータ駆動）機能（発展系）が該当する．

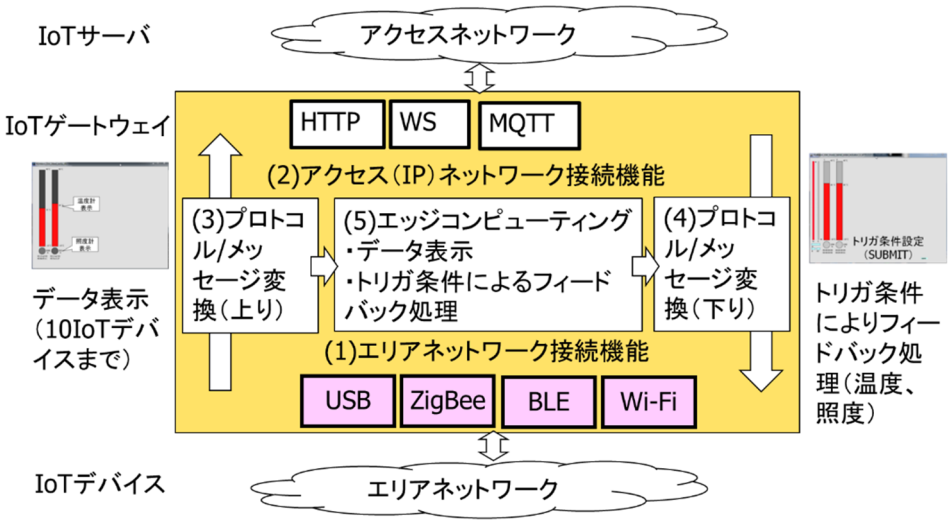

図 6.1　IoT ゲートウェイの全体像

6.2. IoT ゲートウェイのプログラム構成

図 6.2 に IoT ゲートウェイの典型的なプログラム構成例を示す．

①において IoT デバイスからセンサデータを受信(周期的または非同期の受信イベント)し，一旦 Device テーブルに格納する．Device テーブルは可変であるが本演習では上限を 10 個としている．②において Device テーブルに格納されたデータをビジブルに表示する．③において Device テーブルに格納されたセンサデータを IoT サーバに周期的に送信する．基本系では最初のデータのみに制約しているが，発展系では格納されたすべてのデータを扱う．④において IoT サーバからフィードバックされる制御データを受信（周期的または非同期の受信イベント)し，データコマンドの場合は Device テーブルに追加登録してデータ表示に反映する．また，トリガコマンドの場合は⑤のように IoT デバイスに制御データを送信（周期的または非同期のイベントとして受信）し，IoT デバイスのアクチュエータを制御する．⑥は IoT ゲートウェイでのトリガ条件処理で，温度データと照度データに対してトリガ条件を設定し，条件が成立した時に制御データを IoT デバイスにフィードバックする．⑦は IoT サーバに蓄積された IoT データベースの内容を HTTP プロトコルで検索してグラフ表示する．①～⑤までが基本系の機能で，⑥と⑦は発展系としている．

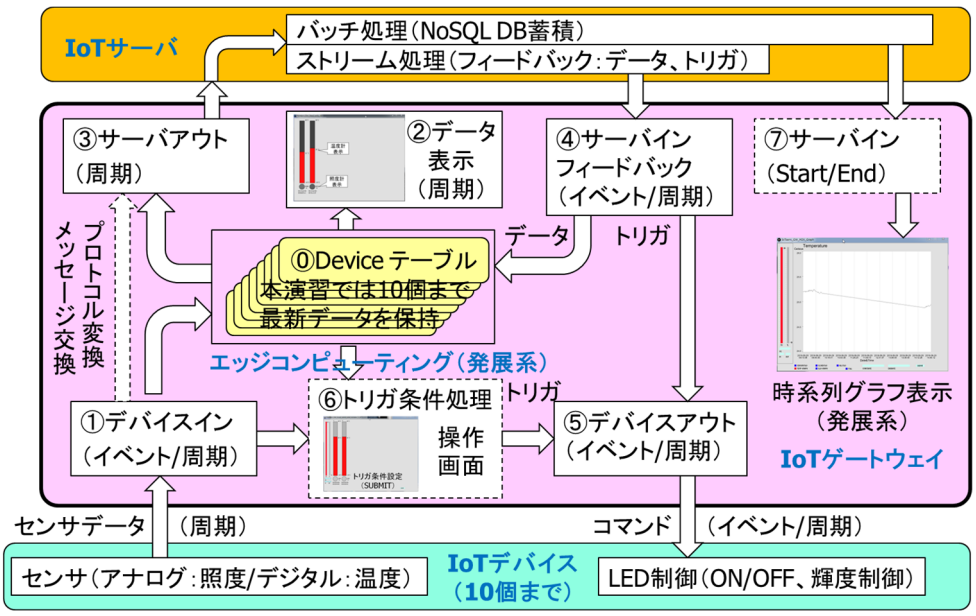

図 6.2　IoT ゲートウェイのプログラム構成

6.3. IoT ゲートウェイの基本系：Processing（Java 系）版

図 6.3 に IoT ゲートウェイの基本系として Processing（Java 系）プログラムの実装例を示す. プログラムの流れとして, 最初にライブラリ定義や変数定義等の初期設定を行い, setup()関数で 1 度だけ実行する ZigBee や MQTT 等の通信関連や GUI 関連の定義を行う. 続いて draw()関数でセンサデータの表示（10 秒周期）や IoT サーバへの送信（MQTT プロトコルの Publish で 60 秒周期）を周期的に行う. IoT デバイスとの ZigBee による送受信は非同期のイベント処理で行う. IoT サーバからの制御データの受信も MQTT プロトコルの Subscribe 登録に対応した callback のイベント処理で行う. IoT デバイスから受信したセンサデータと IoT サーバからフィードバックされたセンサデータは一旦 Device テーブル（最大 10 個まで）に格納される.

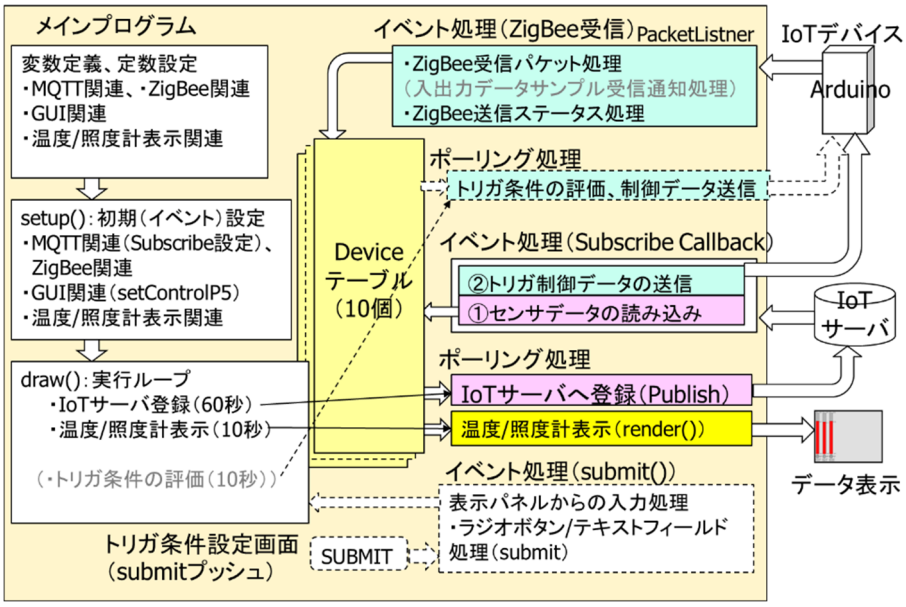

図 6.3　Processing プログラムの構成

図 6.4 に Processing プログラムにおける UI（ユーザインタフェース）画面の表示例を示す. 左側（基本系）は Device テーブルに格納されたセンサデータの値をビジブルに表示（この例では 1 つの IoT デバイス）している. 右側（発展系）は IoT サーバからフィードバックされたセンサデータを含めて表示している. また, UI 画面からトリガ条件を設定して submit することで, トリガ条件の結果に応じて, IoT デバイスのアクチュエータを制御することができる.

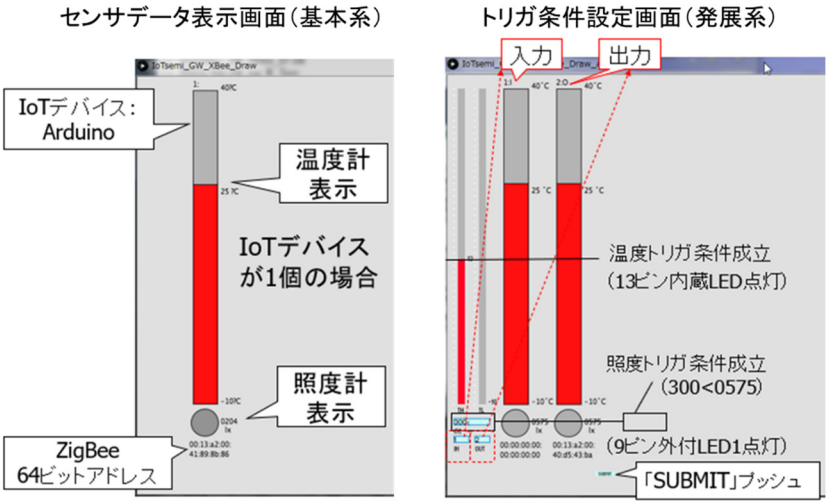

図 6.4　Processing プログラム表示画面例

トリガ条件の例（発展系）を表 6.1 に示す．上段に温度のトリガ条件を，下段に照度のトリガ条件を示す．設定値と受信データとの比較結果により IoT デバイスへの制御データをフィードバックする．

表 6.1　トリガ条件の設定と判定方法

トリガ条件の種類		受信データとの比較	IoTデバイスへの出力データ
温度のトリガ条件⇒デジタル出力（13ピン）	温度MAX比較値フィールドにMax温度を設定	Max温度<=（温度）	デジタル出力 ・成立→1（LED点灯） ・不成立→0（LED消灯）
	温度MIN比較値フィールドにMin温度を設定	Min温度>=（温度）	デジタル出力 ・成立→1（LED点灯） ・不成立→0（LED消灯）
照度のトリガ条件⇒アナログ（PWM）出力（9ピン）	照度比較値フィールドに照度値を設定 照度比較演算子フィールドにトリガ条件式を設定	LTE：設定照度<=（照度） LT：設定照度<（照度） GTE：設定照度>=（照度） GT：設定照度>（照度） EQ：設定照度==（照度）	アナログ（PWM）出力 ・成立→255（LED点灯） ・不成立→0（LED消灯）
		AS：=	アナログ（PWM）出力：（照度）を0-1800から0-255にマッピングした値
		null：null	アナログ（PWM）出力：設定値を出力 255以上の場合は255に、0以下の場合は0に設定

6.4. IoT ゲートウェイの発展系：Python 版

図 6.5 に IoT ゲートウェイの発展系として Python プログラムの実装例の全体像を示す．

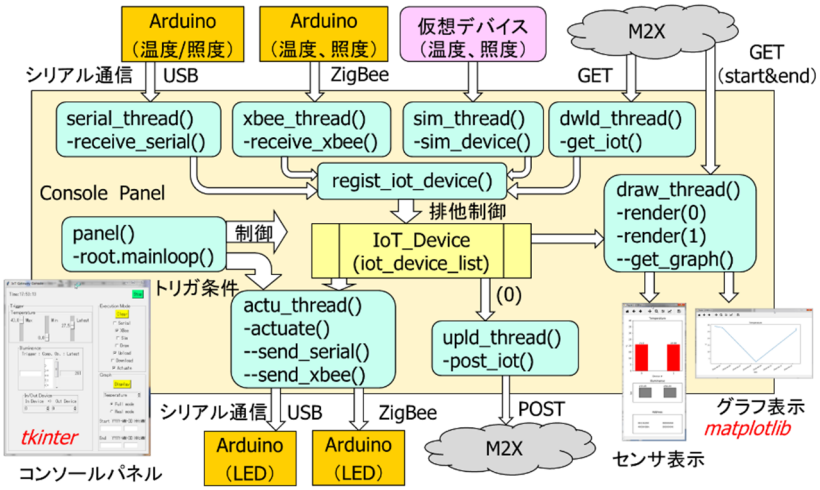

図 6.5　Python プログラムの全体像

機能としては，基本系と同じである．それぞれの機能をスレッドに分けて実行する形態をとっている．IoT デバイスからのシリアル通信接続による入力，ZigBee 接続（複数）による入力，仮想デバイス（複数）からの入力，及び IoT サーバ（本例では M2X）からのフィードバックデータは，一旦 Device テーブル（本例では IoT_Device が該当）に格納されてから目的別に処理される．コンソールパネルは tkinter ライブラリで，表示パネルは matplotlib ライブラリで実装している．コンソールパネルでの表示指定により Device テーブルに格納されたデータがビジブルに表示される．コンソールパネルでの IoT サーバへの送信指定により Device テーブルの先頭のデータを IoT サーバ（本例では M2X）に送信する．また，コンソールパネルからトリガ条件を設定することにより，IoT デバイスのアクチュエータを制御することもできる．

図 6.6 に Python プログラムにおける UI(ユーザインタフェース)画面の表示例を示す．図の左側は tkinter で実装したコンソールパネルで各フレームに対応する機能の処理を指定する．図の右側は matplotlib で実装したセンサデータの表示画面で本例では 2 個の IoT デバイスからのデータを周期的に表示している．

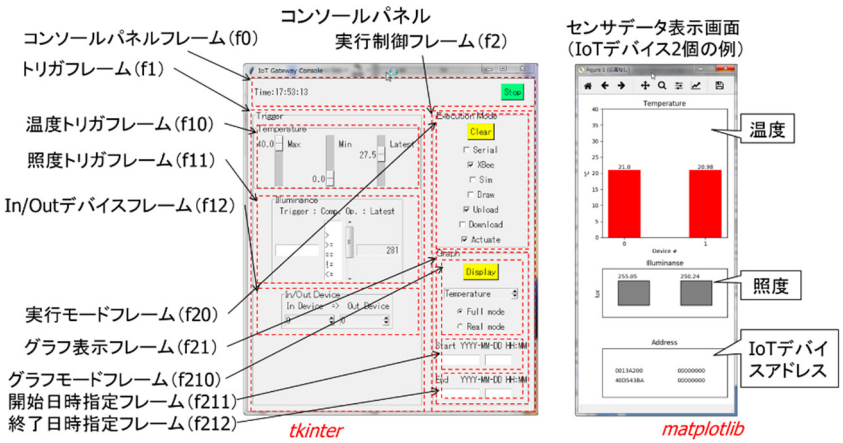

図 6.6　Python プログラムの画面

6.5. IoT ゲートウェイの発展系：Node.js＋Node-RED 版

図 6.7 に IoT ゲートウェイの発展系として Node-RED（JavaScript）プログラムの実装例の全体像を示す.

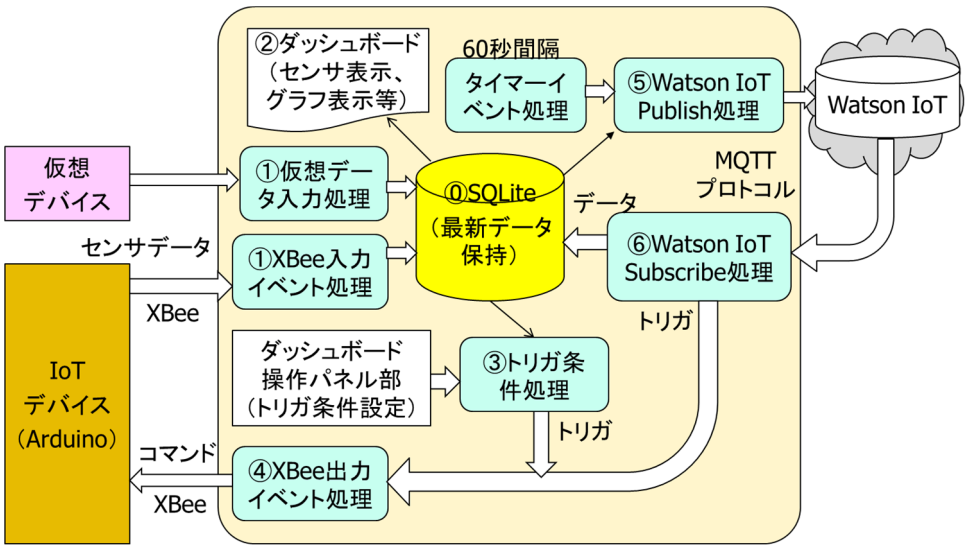

図 6.7　Node-RED プログラムの全体像

機能としては，基本系と同じである．それぞれの機能はイベントの起動により実行される．IoT デバイスからの ZigBee（XBee）接続（複数）による入力，仮想デバイス（複数）からの入力，及び IoT サーバ（本例では IBM Cloud（Watson IoT））からのフィードバックデータは，一旦 Device テーブル（本例では SQLite で実装）に格納されてから目的別に処理される．コンソールパネルとデータ表示パネルは Dashboard ライブラリを使って実装している．コンソールパネルでの表示指定による Device テーブルに格納されたデータがビジブルに表示される．コンソールパネルでの IoT サーバへの送信指定により Device テーブルの先頭のデータを IoT サーバ（本例では IBM Cloud）に送信する．また，コンソールパネルからトリガ条件を設定することにより，IoT デバイスのアクチュエータを制御することができる.

図 6.8 に Node-RED プログラムにおける UI（ユーザインタフェース）であるダッシュボード画面の表示例を示す．操作パネル部で対応する機能の処理を指定する．温度/照度表示部に Device テーブルに格納されたセンサデータをビジブルに表示している．また，トリガ条件設定部でトリガ条件に関する設定を行うことにより，IoT デバイスのアクチュエータを制御することができる.

図 6.8　Node-RED プログラムのダッシュボード画面

6.6. PC（Windows 版）から Raspberry Pi（Linux 版）への移植

既に紹介した基本系の Processing（Java 系）版，発展系の Python 版と Node.js＋Node-RED 版については，開発・評価を PC（Windows）で行い，本格的な運用時に Raspberry Pi（Linux）に移植することが容易である．

基本系の Processing（Java 系）の場合について，以下に手順を示す．

①Raspberry Pi 上で Processing の Linux-ARM 版をインストールする

②Processing を起動して環境設定を行う

③シリアル通信用ライブラリ（RXTX ライブラリ）をインストールする

④Windows 版の Processing プログラム（スケッチ，必要なライブラリを含む）を Raspberry Pi にコピーする

⑤Linux-ARM 版との相違点を 1 か所修正（Port 名の変更）する

Raspberry Pi に移植した実装例（複数の IoT デバイスに接続）を図 6.9 に，実行例を PC（Windows）から VNC（Virtual Network Computing）機能を使って表示させた例を図 6.10 に示す．

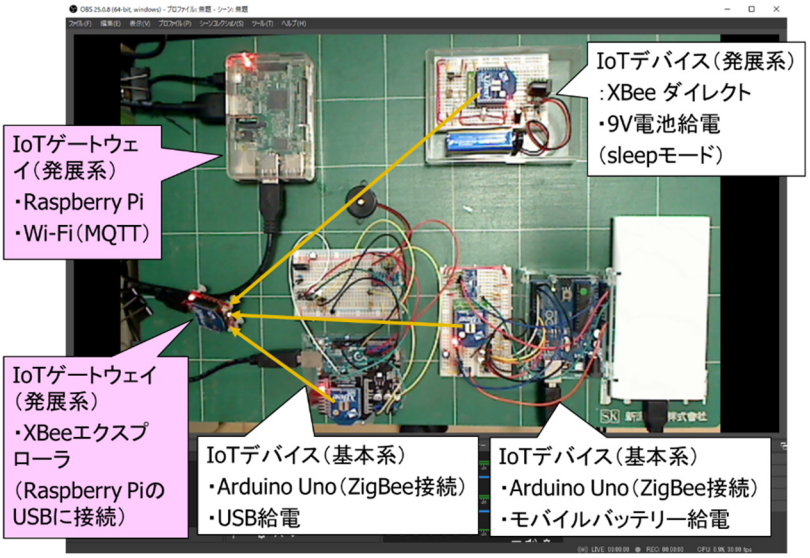

図 6.9　Raspberry Pi への移植（実装例）

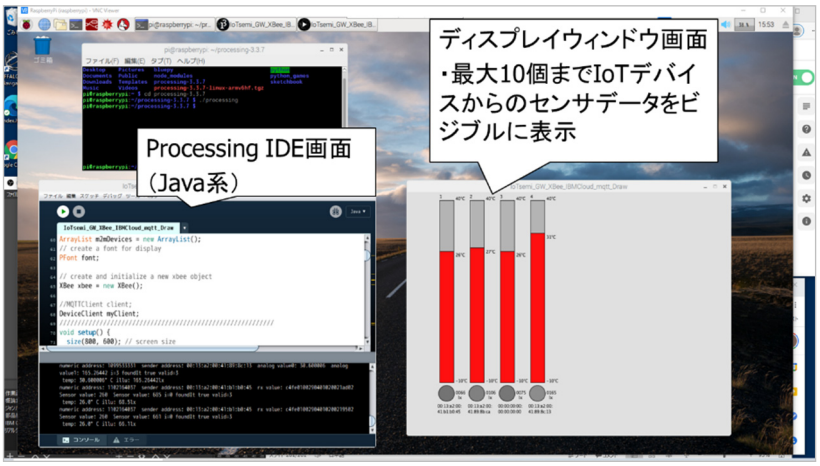

図 6.10　Raspberry Pi での実行例（VNC 機能で Windows 画面上に遠隔表示）

6.7. IoT ゲートウェイのまとめ

基本系として PC（Windows）の Processing（Java 系）版を 2014 年度秋学期の初期から使用しているが，発展系として紹介したように Python 版や，Node.js＋Node-RED 版が存在する．提供されるライブラリに依存するところがあるが，Processing（Java 系）版に比べて，Python 版や Node.js＋Node-RED 版の方が使い勝手が良い感触である．分析に AI を使うケースが今後出てくると想定されるので，Python 版を基本系にすることを検討していく．ちなみに，Eclipse 財団のアンケート調査によれば，IoT ゲートウェイで使われているプログラミング言語の順位は 2018 年では Java→Python→C→C++であったが，2021 年では Python→C++→C→Java に変化してきている[32, 33]．

7．IoT サービスを実現する IoT サーバでの事例

第三部の「IoT サーバ（クラウド）につなぐ」と第四部の「IoT アプリケーションを創る」における基本系と発展系の事例について説明する．

7.1. IoT サーバの全体像

本演習で扱う典型的な IoT サーバでの基本機能の全体像を図 7.1 に示す．大きく，①接続/収集インタフェース，②リアルタイム（ストリーム）処理，③データ蓄積，④データ利用・分析（AI）の 4 つの部分に分けられる．なお，本演習ではデータ分析（AI）については扱っていない．

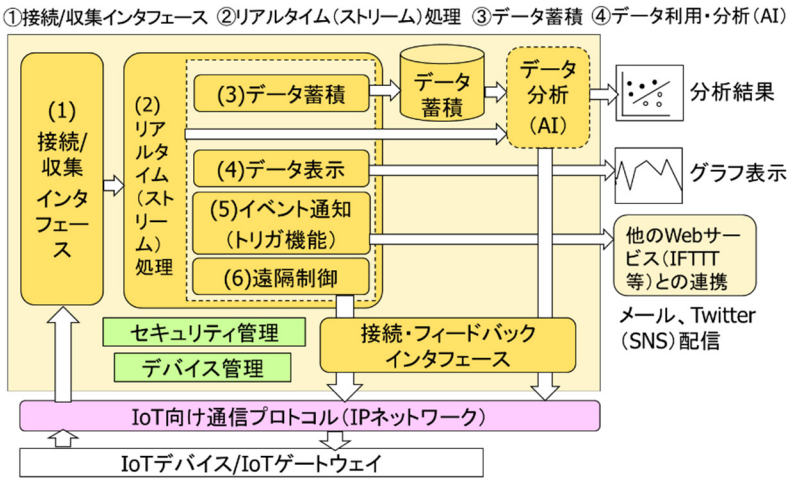

図 7.1　IoT サーバの基本機能構成

それぞれを構成する機能部分について簡単に説明する．

（1）接続/収集インタフェース：IoT サーバの送受信インタフェース部分で IP ネットワークプロトコルによる通信接続を行う．HTTP の場合は Web サーバが，MQTT の場合は MQTT ブローカーが該当する．セキュリティや認証チェックについても行われる．

（2）リアルタイム（ストリーム）処理：接続インタフェースで受信されたデータがそれぞれ設定された機能（データ蓄積機能，データ表示機能，イベント通知機能，遠隔制御機能）ごとにリアルタイムで処理される．

（3）データ蓄積機能：IoT データベース（NoSQL が多い）に蓄積され，後でデータ分析が行われる．

（4）データ表示機能：リアルタイムで収集されたデータをビジブルに表示する．

（5）イベント通知（トリガ）機能：設定されたトリガ条件の結果に応じて，他の Web サービス（IFTTT 等）と連携して，イベント通知（メール送信，SNS 送信）等を行う．

（6）遠隔制御機能：設定されたトリガ条件の結果やデータ分析結果に基づいて，制御データを IoT ゲートウェイ経由で IoT デバイスにフィードバックしてアクチュエータ（LED 等）を駆動する．

7.2. IoT サーバ：BaaS の例

最近では，PaaS 上に特定のアプリケーションサービスの運用に必要な汎用的なサーバ機能を提供する BaaS（Backend as a Service）が注目されている．IoT 分野でもある制約条件の中で無料で使用できるようになってきている．

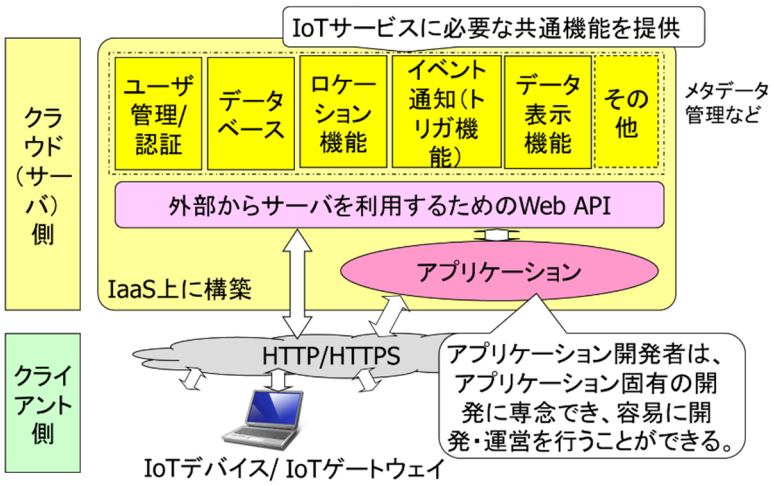

図 7.2　BaaS のシステム構成例

BaaS の一般的な構成を図 7.2 に示す．BaaS では IoT サービスとして共通的に必要となるユーザ管理/認証，データベース，ロケーション機能，イベント通知（トリガ機能），データ表示機能などの機能を Web API を使って容易に実現することができる．IoT アプリケーション開発者は，ほとんどプログラムコードを記述しなくても容易に開発・運営を行うことができ，アプリケーション固有の開発に専念することができる（最近注目されている NoCode に近い）．

M2M/IoT 向けに特化した BaaS として，当初注目されていた Xively[22]（元 Pachube，2011 年福島原発事故の放射線量の見える化で注目された）の無料サービスを利用した．その後無料サービスが中止されたため，ほぼ同じ機能を持つ AT&T 社の M2X[23] に代替したが，M2X も 2020 年に無料サービスを停止してしまった．

本演習の取り組み時点ではこの BaaS 版を基本系として使ってきたので，M2X（AT&T 社）のシステムの全体像を図 7.3 に示す．

なお，現状では同等機能を持つ BaaS として手ごろなサービスが無いことから，BaaS については取り組みをやめている．

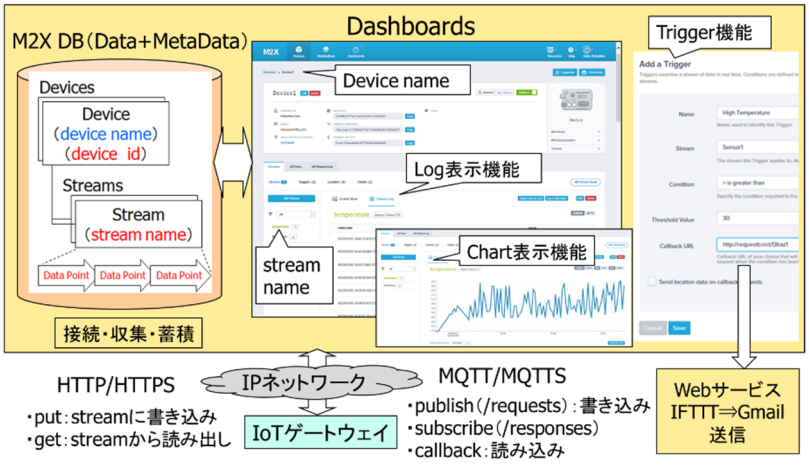

図 7.3　M2X（AT&T 社）システムの全体像

7.3. IoT サーバ：PaaS の例

大手のクラウド企業が，アプリケーション開発に必要な機能を PaaS としてサポートしており，IoT 向けのプラットフォームが提供されている．PaaS として以下の特徴を持っており，比較的容易に堅牢でスケーラビリティのある IoT システムを構築できる．

（1）クラウドネイティブ：クラウドで提供されたサービスを組み合わせて，効果的・効率的にシステムを構築できる．

（2）フルマネージサービス：システムの提供側によってすべて管理されたサービスで，利用者による可用性や全体のキャパシティの調整などの管理は不要となる．データベースやコンピュートエンジンなどのサービスが該当する．

（3）マイクロサービス：システムを複数のサービスの集合体として構成し，サービス相互を RESTful API のようなシンプルで軽量な手段で連携する手法で，小規模なサービス群を疎結合することにより，変化に強いシステムを構築することができる．

本演習では 4 つの大手クラウド企業（Amazon 社の AWS IoT[34]，IBM 社の IBM Cloud/Watson IoT Platform[35]，Google 社の Google Cloud Platform(GCP)[36]，Microsoft 社の Azure IoT[37]：発展系に取り組んだ順）から提供される PaaS を対象にする．4 大クラウド企業の主な特徴をシェア順に表 7.1 に示す[38]．

表 7.1　4 大クラウド企業の主な特徴

4大クラウド企業（シェア順）	主な特徴
①AWS（2006～）	・取り扱っているサービスが多数 ・クラウドでは最古参
②Microsoft Azure（2010～）	・マイクロソフト製品との連携 ・国内拠点多数
③GCP（2008～）	・価格性能比が良い ・データ分析や大量のトラフィックを支えるインフラに強み
④IBM Cloud（Bluemix：2014～）	・基幹システムとの連携 ・Watsonとの連携、機械学習、AI分野に強み

4 大クラウド企業（PaaS）での課金体系（無料枠）について図 7.4 に示す．本演習の IoT プロトタイプ構築の例では，無料クレジットや無料枠を利用することでほぼ無料で実現可能である．なお，頻繁に使用すると無料枠を超えて課金の恐れがあるので十分な注意が必要である．

AWS		無料クレジット	12か月	以降
無料クレジット	（+α枠）	$200（30D）	課金	課金
有料サービス	無料枠		（12M）	課金
常時無料（多数+）	無料枠			

Azure		無料トライアル	12か月	以降
無料クレジット	（+α枠）	個別（短期）	停止	停止
有料サービス	無料枠		（12M）	停止
常時無料（40+）	無料枠			

GCP		無料クレジット	以降
無料クレジット	（+α枠）	$300（90D）	停止
常時無料（20+）	無料枠		

IBM Cloud		無料クレジット	以降
無料クレジット	（+α枠）	$200（20D）	停止
常時無料（40+）	無料枠		

図 7.4　4 大クラウド企業の課金体系（無料枠）

また，セキュリティ強化と本人確認のために無料版の利用でもクレジットカード登録が必要になってきている(2021 年 9 月までは IBM でクレジットカード登録なしで使えた)．クレジットカードを使えない受講生がいるため，本演習では IoT サーバの基本系として後述の自 PC を使った IaaS 版の OwnCloud を利用している．

7.3.1. 4 大クラウド企業（PaaS）の全体像

4 大クラウド企業から提供される IoT プラットフォームのシステム例を図 7.5 に示す．

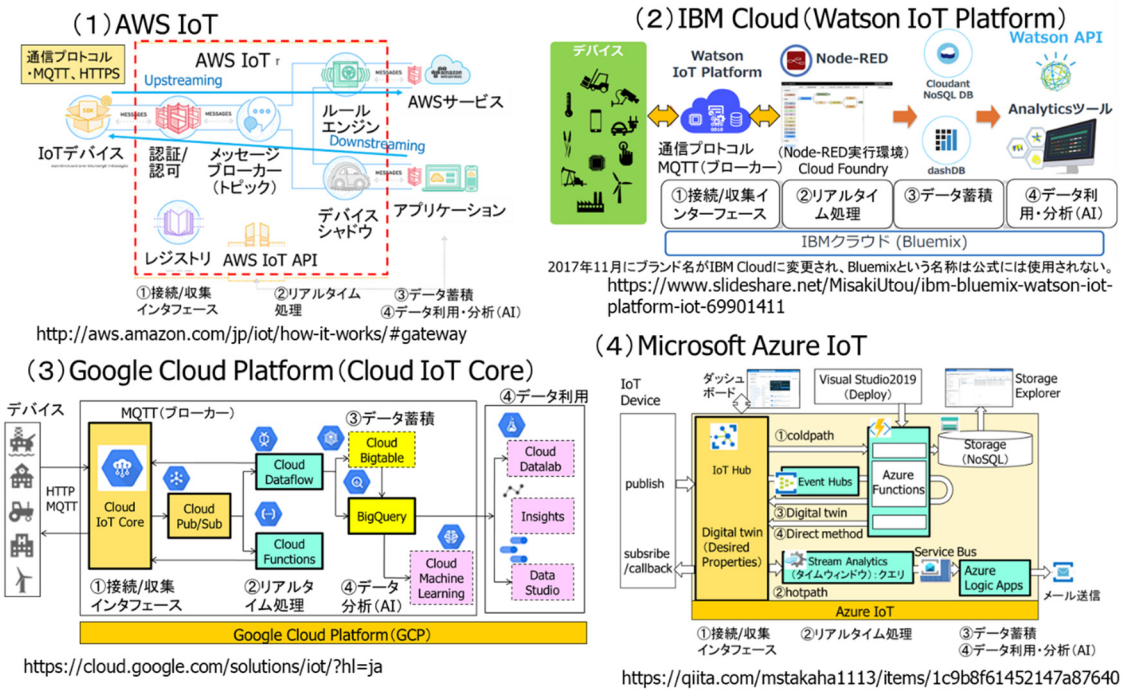

図 7.5　4 大クラウド企業による IoT プラットフォームのシステム例

それぞれ，①接続/収集インタフェース，②リアルタイム処理，③データ蓄積，④データ利用・分析（AI）の 4 つの部分から構成されている．

個々の企業の IoT プラットフォームの紹介の前に，共通的な項目について説明する．

（1）認証：最初のアクセス時に接続/収集インタフェース部でユーザのアクセス認証が行われる．詳細はそれぞれ異なるが，あらかじめユーザ登録した HP から認証コードを取得しそれを使ってアクセスすることになる．

（2）送受信プロトコル：基本は IoT 向きの MQTT プロトコルを扱う．接続/収集インタフェース部で MQTT ブローカーによりトピック経由での送受信が行われる．また，HTTP プロトコルでアクセスする仕組みも提供されている．

（3）デジタルツイン：実 IoT デバイスの状態のコピーを IoT サーバ上に保持することで，実 IoT デバイスの状態（正常に接続されている場合と電源 OFF などで切り離されている

場合）に関わらず，IoT サーバ上で同じサービスを継続できるようにした仕組みが提供されている．実 IoT デバイスが電源 ON で接続したときに，IoT サーバ上にコピーされた情報との違いがあれば，その内容を反映するような操作が自動的に行われる．AWS IoT のデバイスシャドウ機能と Microsoft Azure IoT のデジタルツイン機能が該当する．

（４）FaaS（Function as a Service）：リアルタイム処理において，サーバレスでイベントに応じてプログラムコード（ファンクション）を実行する機能である．AWS IoT の Amazon Lambda，GCP の Cloud Functions，Microsoft Azure IoT の Azure Functions が該当する．

7.3.2. PaaS の例：AWS IoT[34)]

Amazon から提供される AWS IoT による IoT プロトタイプ構築例を図 7.6 に示す．

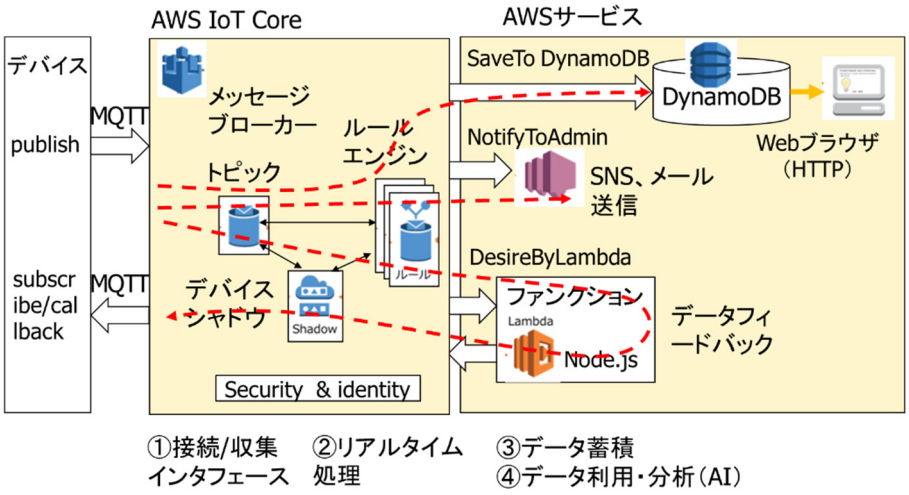

図 7.6　AWS IoT の実装例（全体像）

接続は MQTT プロトコルの場合を示している．AWS IoT Core 部分におけるメッセージブローカーはデバイス（AWS では Thing と呼ぶ）と AWS IoT の間のメッセージ送受信を行うサービスである．トピックは Publisher からのメッセージの送信先であり，Subscriber のメッセージの受信元になる．ルールエンジンは各トピックに配信されたメッセージをフィルタリングやデータの属性を編集した後，他の AWS サービスに連携する機能である．デバイスシャドウはデバイスから通知された現在の設定情報（reported）とアプリケーションから受け取ったデバイスに設定されるべき設定情報（desired）の両方を保持し，両者の情報を比較して差分がある場合にデバイスに対して差分情報をプッシュする機能である．この機能によりデバイスの状態（電源 OFF で切り離し状態）を気にせずにアプリケーションを構築することが可能になる．

AWS サービス部分は，ルールエンジンから起動されるサービス群より構成されている．

本演習では，データ蓄積機能として NoSQL の DynamoDB を使う．イベント通知機能として，メール送信や SNS 送信を行うことができる．また，ファンクション機能（Lambda）によりイベント方式でプログラムコード（この例では JavaScript）を実行することでリアルタイムなアプリケーション（この例では受信データのフィードバック）を実現することができる．

AWS IoT の実装例（ダッシュボード）を図 7.7 に示す．この画面で，デバイス（Things）やセキュリティの設定を行う．また，各ルールついてフィルタリングの設定や連携するサービスの設定も行う．右側にデバイス（Things）の設定内容の詳細と，デバイスシャドウの状態が表示されている．なお，表示内容については 2017 年当時の内容で現在とは相違している．

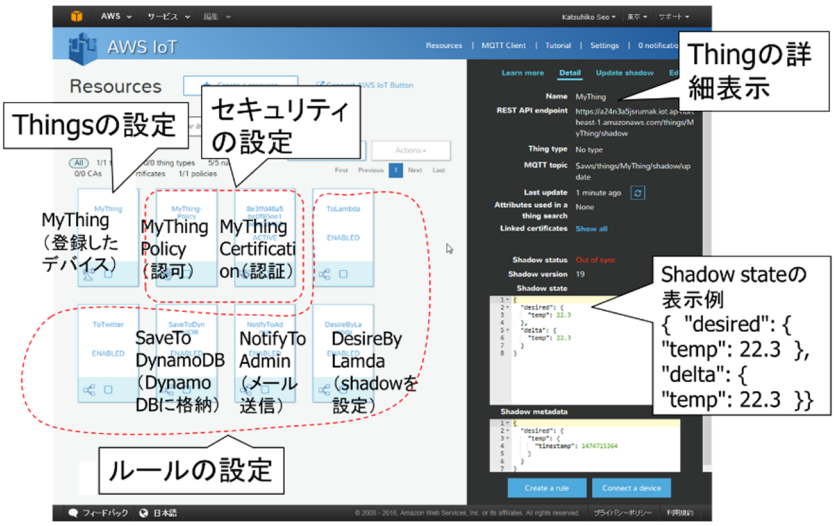

図 7.7　AWS IoT の実装例（ダッシュボード）

7.3.3. PaaS の例：IBM Cloud[35)]

IBM から提供される IBM Cloud による IoT プロトタイプ構築例を図 7.8 に示す．

接続は MQTT プロトコルの場合を示している．接続/収集インタフェース部分（Watson IoT Platform）における MQTT ブローカーはデバイスと IBM Cloud との間でのメッセージの送受信を可能にしている．トピックは Publisher からのメッセージの送信先であり，Subscriber のメッセージ受信元になる．本例のリアルタイム処理部において，IBM の IaaS 基盤である Cloud Foundry 上に Node-RED-Cloudant（NoSQL DB）環境を build して，その上でフロー型のプログラムコードである Node-RED によりアプリケーションを構築することができる．

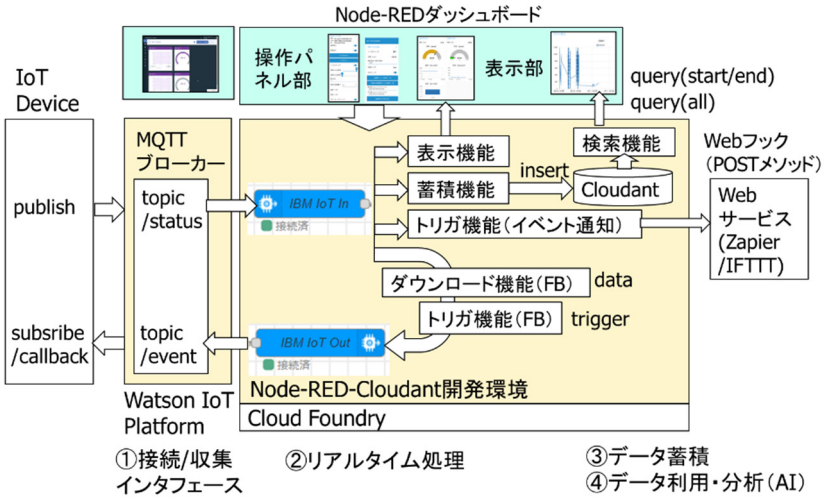

図 7.8　IBM Cloud の実装例（全体像）

Node-RED のダッシュボードの操作パネル部分からあらかじめ組み込まれた操作を行うことで該当する機能を実行できる．表示機能ではリアルタイムでトピック（/status）を経由して収集されるデータをダッシュボードの表示部に表示する．蓄積機能では NoSQL の Cloudant に蓄積する．検索機能では Cloudant に蓄積された内容を検索して表示する．トリガ機能（イベント通知）では，トリガ条件の結果に基づいて，外部の Web サービスと連携してメール送信や SNS 送信を行う．ダウンロード（フィードバック）機能では収集したデータをトピック（/event）経由でフィードバックする．トリガ（フィードバック）機能では，トリガ条件の結果に基づいて制御データを同じようにフィードバックする．

IBM Cloud の実装例（画面例）を図 7.9 に示す．Watson IoT Platform 画面でデバイスを定義し，IBM Watson IoT カード画面で表示形式を定義するだけで，トピック経由で収集されたデータがビジブルに表示される．また，Node-RED のダッシュボード機能により操作パネル部や温度表示部/照度表示部を簡単に設定することができる．

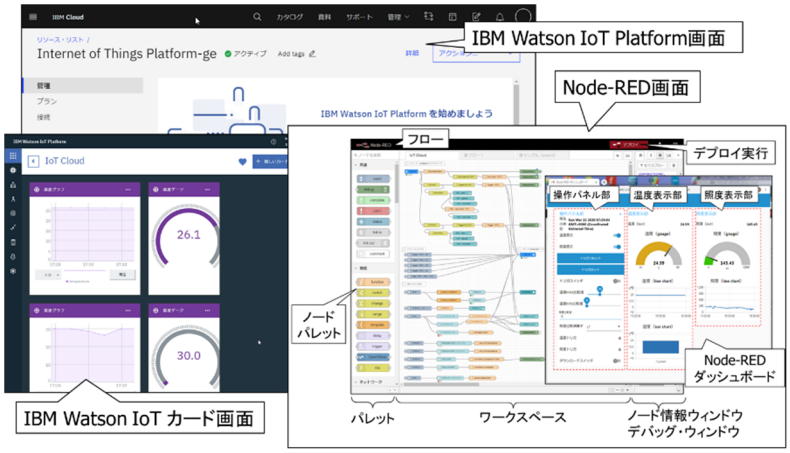

図 7.9　IBM Cloud の実装例（画面例）

7.3.4. PaaS の例：Google Cloud Platform（GCP）[36)]

Google Cloud Platform（GCP）による IoT プロトタイプ構築例を図 7.10 に示す.

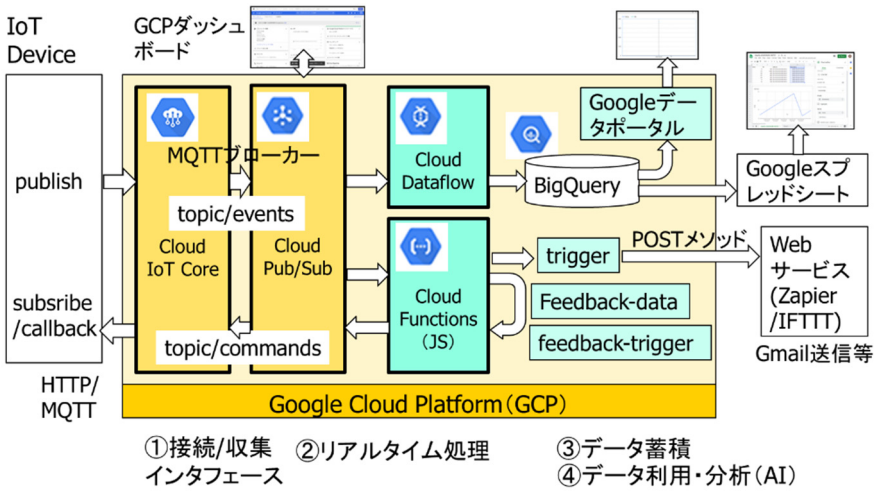

図 7.10　Google Cloud Platform の実装例（全体像）

接続は MQTT プロトコルの場合を示している．接続/収集インタフェース部分における Cloud IoT Core 部はセキュリティのもとにデバイスを設定・管理するマネージャ機能を持ち，デバイスからのすべてのデータを Cloud Pub/Sub にパスする．Cloud Pub/Sub 部ではトピック経由とすることでフィルタリングを行いながら送受信を行う．トピック（/events）は Publisher からのメッセージの送信先であり，トピック（/commands）は Subscriber のメッセージ受信元になる．

リアルタイム処理における Cloud Dataflow 機能は様々なデータ処理のパターンの実行に対応したマネージドサービスで，バッチ及びストリーミングのデータ処理を実行する．本例では大規模データベースの BigQuery にデータを蓄積する．BigQuery に蓄積されたデータは Google データポータル機能を使って表示したり，Google スプレッドシートに出力することもできる．また，Cloud Functions 機能はイベントドリブンのサーバレスコンピューティングプラットフォームでサーバにプロビジョニングせずにプログラムコード（この例では JavaScript）を実行できる．Trigger 機能では，設定したトリガ条件の結果に基づいて外部の Web サービスと連携してメール送信や SNS 送信を行う．Feedback-data 機能では，受信したデータをトピック（/commands）経由でフィードバックする．Feedback-trigger 機能では，トリガ条件の結果に基づいて制御データを同じようにフィードバックする．

Google Cloud Platform の実装例（画面例）を図 7.11 に示す．GCP ダッシュボード画面より必要な操作画面に移動する．IoT Core 画面でデバイスを定義し，Pub/Sub 画面で MQTT ブローカーのトピックの設定等を行う．図の右側に BigQuery の設定画面と表示画面を示す．

図 7.11　Google Cloud Platform の実装例（画面例）

7.3.5. PaaS の例：Azure IoT[37)]

Azure IoT による IoT プロトタイプ構築例を図 7.12 に示す.

接続は MQTT プロトコルの場合を示している．接続/収集インタフェース部分における IoT Hub 部は，デバイスと Azure サービス間でセキュリティで保護された信頼性のある双方向通信（Pub/Sub 機能）を実現している.

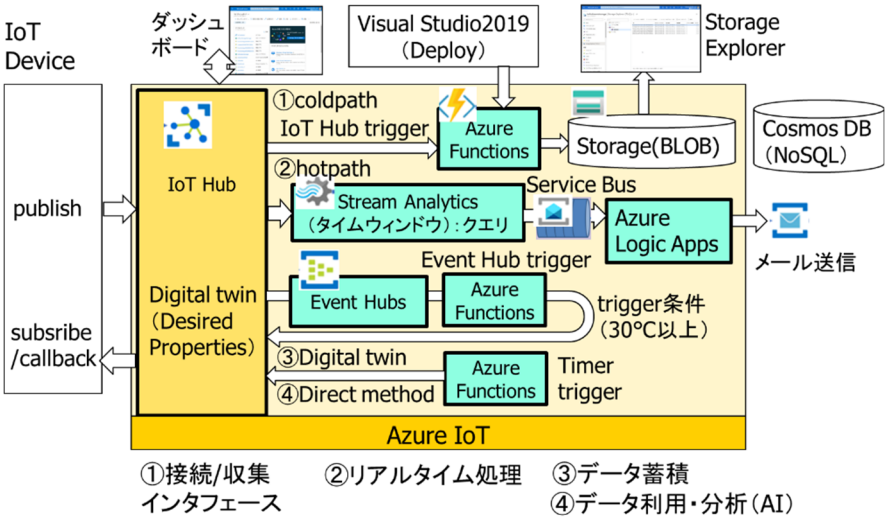

図 7.12　Azure IoT の実装例（全体像，未適用）

リアルタイム処理部における Azure Functions 機能はイベントドリブンのサーバレスコンピューティングプラットフォームで，AWS の Lambda に相当する．①の coldpath の例では，IoT Hub でのデータ受信トリガにより起動された Azure Functions が受信データを IoT DB（Storage（BLOB）または NoSQL の Cosmos DB）に蓄積する．②の hotpath の例では，Stream Analytics（リアルタイムのイベント処理エンジン）機能で，SQL に似たクエリ言語でデータ分析を行い，その結果に基づいて Azure Logic Apps を使用してイベント通知をメールで送信する．③の Digital twin の例では，スケーラブルなデータ受信サービスである Event Hubs のトリガにより起動された Azure Functions がトリガ条件の結果に基づいて Digital twin 機能を使って制御データをデバイスにフィードバックする．Azure の Digital twin 機能は，デバイスの状態をデジタルで Azure 側に保持する機能で，通信が確立している状態では同じ情報を保持し合う．AWS のデバイスシャドウ機能に相当する．④の Direct method の例では，Timer でトリガされた Azure Functions が周期的に制御データを Direct method 通信機能を使ってデバイスに直接フィードバックする．

Azure IoT の実装例（画面例）を図 7.13 に示す．Azure ダッシュボード画面より必要な操作画面に移動する．IoT Hub 画面でデバイスを定義し，MQTT ブローカー等の設定を行う．図の右上側に Azure Functions 画面（JSON ビュー）を示す．右下側に Storage Explorer 画面を示す．

本例の Azure IoT については試行評価レベルで，IoT サーバの基本系に相当する機能は実装できていない．残された課題の 1 つである．

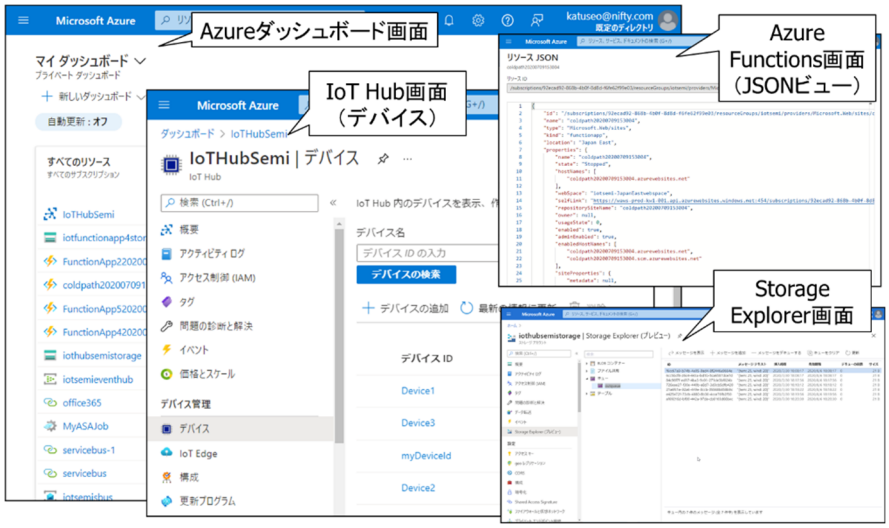

図 7.13　Azure IoT の実装例（画面例，部分適用）

7.4. IoT サーバ：IaaS の例

7.4.1. IaaS の全体像

クラウド上の IaaS 環境またはオンプレミス環境において，独自に IoT サーバ環境を構築するケースで，自由に構築できるがサーバ構築技術を必要とする．また，セキュリティやスケーラビリティについてすべて独自に構築が必要で，本格的な運用に適用するのはかなり難しい．しかし，IoT サーバの仕組みを理解するためのプロトタイプレベルとしては実現可能である．

図 7.14 に独自 IoT サーバの実現例を示す．図 7.14 の(a)は従来のサーバ方式である LAMP 版（Linux＋Apache＋MySQL＋PHP）で，(b)は IoT 向きの MEAN 版（MongoDB＋Express＋AngularJS＋Node）による実装である．(c)は 3 種類の独自 IoT サーバの実装形態を示している．

以下では，LAMP 版と，IoT サーバの基本系として採用した MEAN 版のサブセット（Node.js＋Node-RED/JavaScript，OwnCloud と呼ぶ）の実装例について説明する．

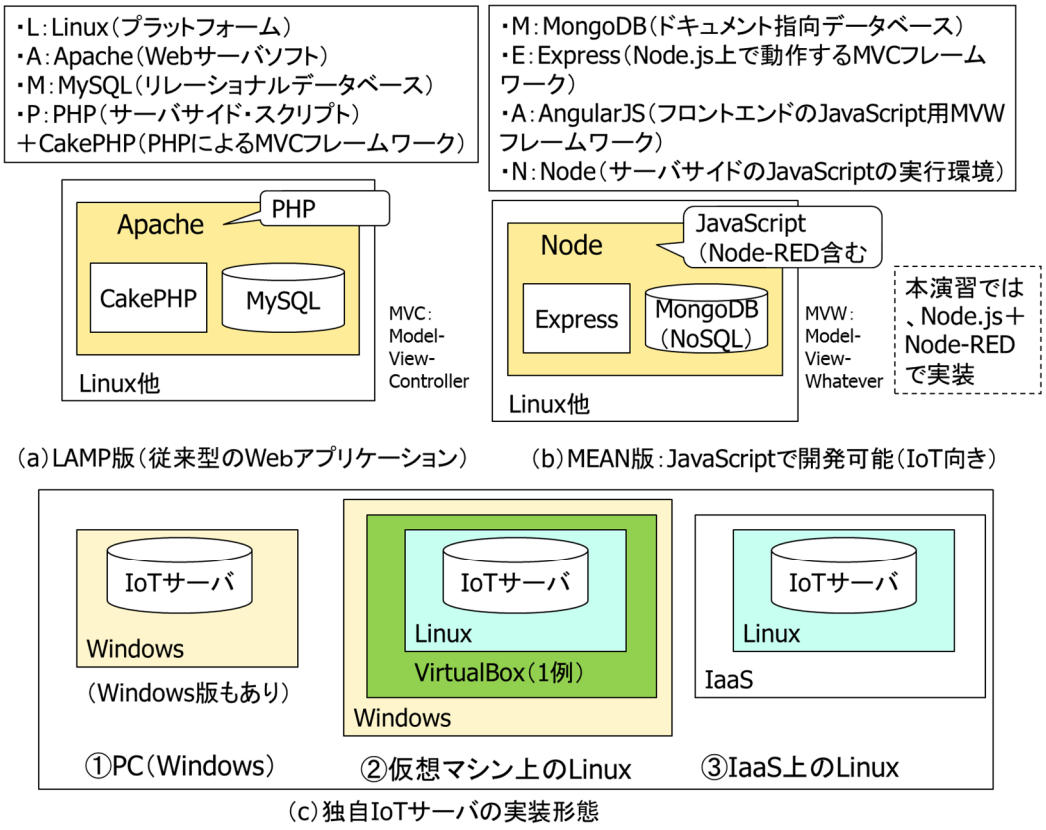

図 7.14　独自 IoT サーバの実現方法（LAMP 版，MEAN 版）

7.4.2. IaaS の発展系：LAMP 版

独自 IoT サーバ（LAMP 版）の実装例（全体像）を図 7.15 に示す.

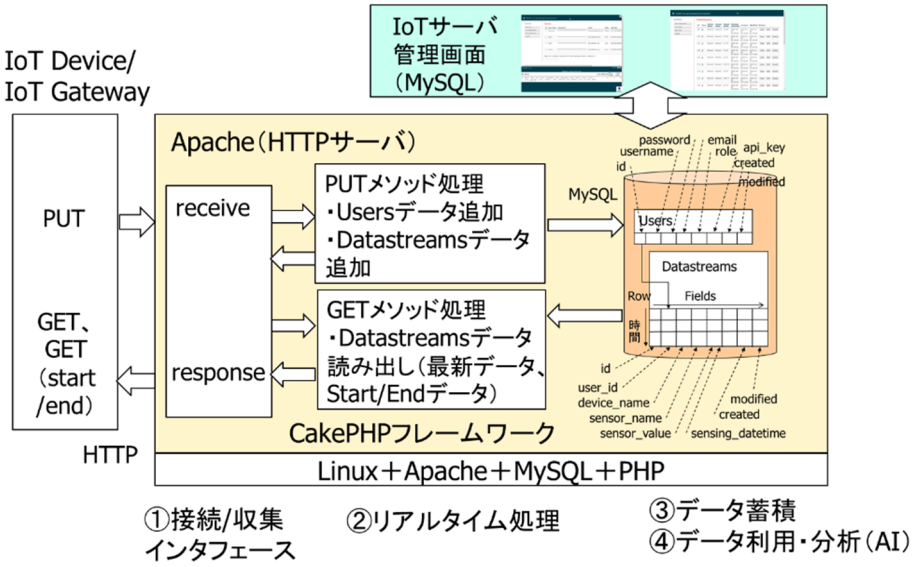

図 7.15　独自 IoT サーバ（LAMP 版）の実装例（全体像）

この LAMP 版では，OS に Linux を，Web サーバに Apache を，IoT DB としてリレーショナルデータベースの MySQL を，プログラミング言語として PHP を使う．サーバ構築のフレームワークとして MVC（Model-View-Controller）フレームワークを構築できる CakePHP を使用している.

接続/収集インタフェース部では Apache による HTTP サーバ機能により HTTP プロトコルの PUT メソッドと GET メソッドに対応した処理を行う．リアルタイム処理部で，PUT メソッドで送信されてきたセンサデータを取り込み，MySQL に Users データの場合はユーザ登録を，Datastreams データの場合はセンサデータを蓄積する．GET メソッドの場合は MySQL から Datastreams を読み出し（最新データ検索，日時による start/end 検索）を行い，エッジ側にフィードバックする.

独自 IoT サーバ（LAMP 版）の実装例（管理画面例：MySQL）を図 7.16 に示す．Users テーブルは MySQL に登録されたユーザ登録の一覧を示す．Datastreams テーブルは MySQL に蓄積されたセンサデータの一覧を示す.

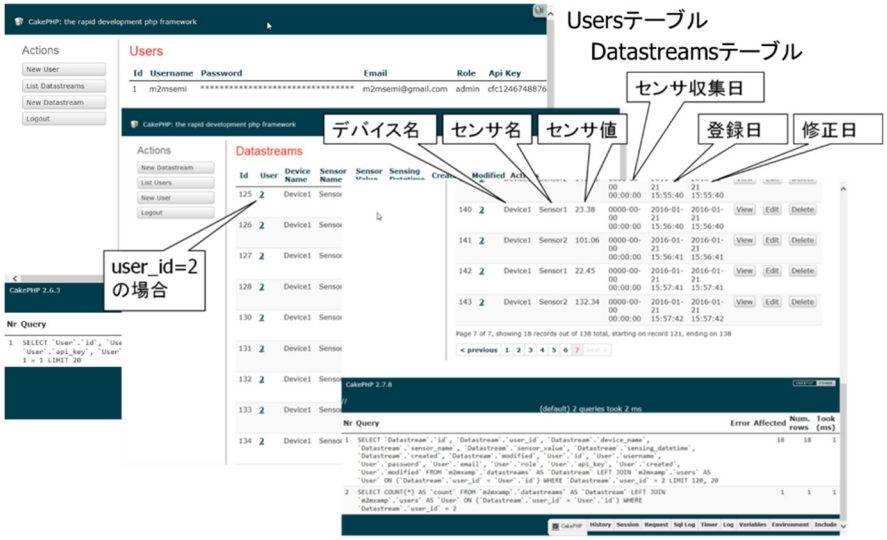

図 7.16　独自 IoT サーバ（LAMP 版）の実装例（管理画面例：MySQL）

7.4.3. IaaS の基本系：Node.js＋Node-RED 版

独自 IoT サーバ（Node.js 版，OwnCloud）の実装例（全体像）を図 7.17 に示す．7.3.3 の PaaS（IBM Cloud）の例とほぼ同じ機能となっている．

この Node.js 版では，OS に Windows を，接続/収集インタフェース部として Web サーバに Node.js を，MQTT ブローカーに Node-RED で動作する aedes ブローカーを使用する．トピック（/status）は Publisher からのメッセージの送信先であり，トピック（/event）は Subscriber のメッセージ受信元になる．

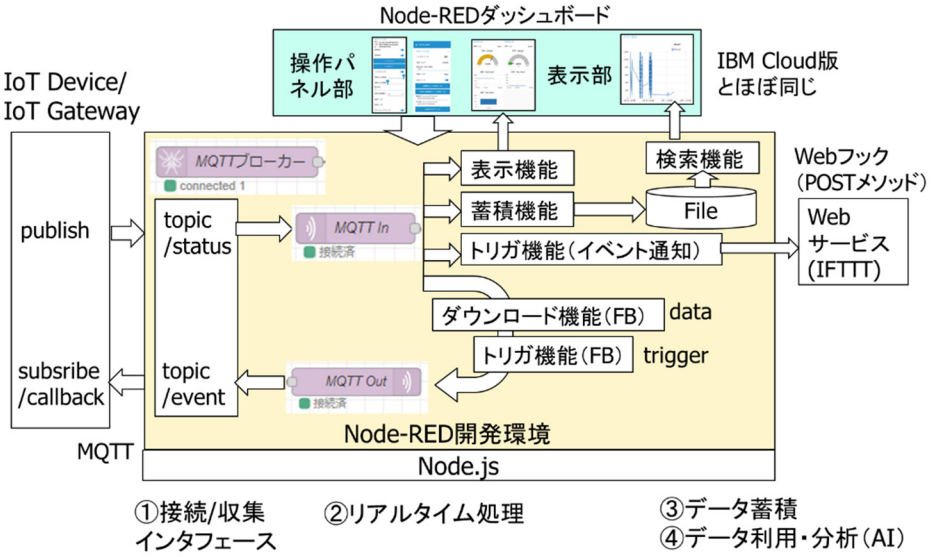

図 7.17　独自 IoT サーバ（Node.js 版）の実装例（全体像）

リアルタイム処理部ではフロー型プログラミング言語である Node-RED を使用する．Node-RED のダッシュボードの操作パネル部分からあらかじめ設定した操作を行うことで該当する機能を実行できる．表示機能ではリアルタイムでトピック（/status）を経由して収集されるデータをダッシュボードの表示部に表示する．蓄積機能では Node-RED に標準装備された通常の File に蓄積する．検索機能では File に蓄積された内容を検索して表示する．トリガ機能（イベント通知）では，トリガ条件の結果に基づいて，外部の Web サービスと連携してメール送信や SNS 送信を行う．ダウンロード（フィードバック）機能では収集したデータをトピック（/event）経由でフィードバックする．トリガ（フィードバック）機能では，トリガ条件の結果に基づいて制御データを同じようにフィードバックする．

独自 IoT サーバ（Node.js 版）の実装例（画面例）を図 7.18 に示す．左側にフロー型の Node-RED プログラム作成例を示す．右側上段に表示機能，ダウンロード（フィードバック）機能とトリガ（フィードバック）機能を扱う Node-RED プログラム（OwnCloud フロー）のダッシュボード画面の表示例を示す．右側下段にトリガ機能（イベント通知）と検索機能を扱う Node-RED プログラム（OwnCloudExt フロー）のダッシュボード画面の表示例を示す．

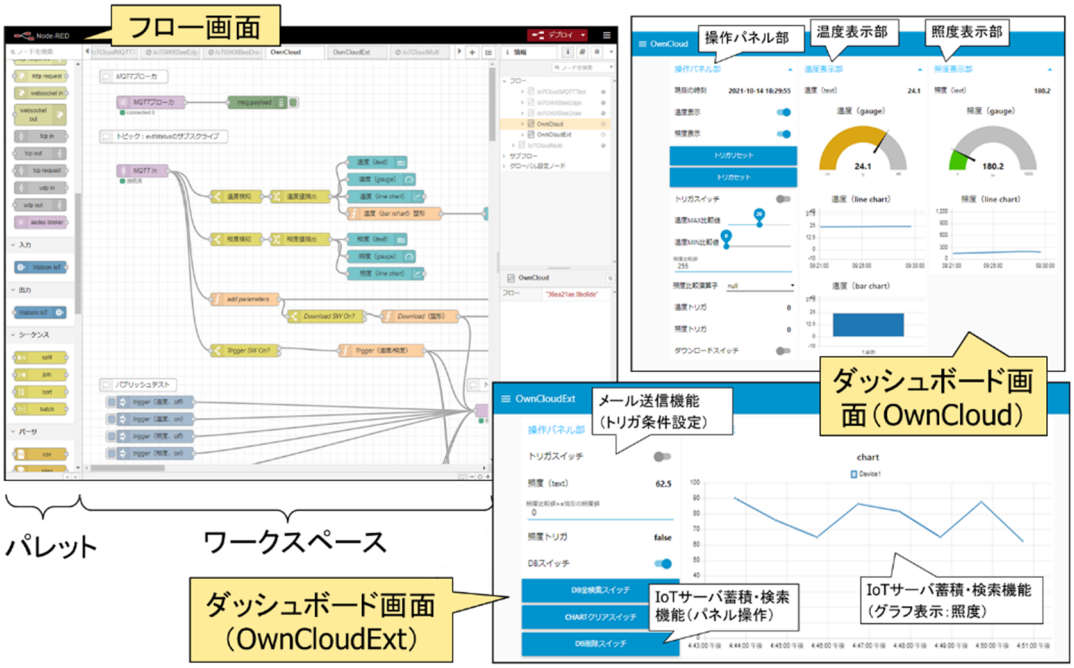

図 7.18　独自 IoT サーバ（Node. js 版）の実装例（画面例）

7.5. IoT サーバのまとめ

IoT サーバの基本系として，最終的にクレジットカード登録の必要のない自 PC 上の IaaS 版（Node.js＋Node-RED）を採用した．自 PC 上だとクラウドのイメージがつかみにくいところもあり，商用の優れた PaaS 環境を利用したいところである．クレジットカード登録による PaaS を基本系として使えるようにすることが課題である．

4 大クラウド企業（PaaS）のプロトタイプ構築例の中で，AWS IoT，Azure IoT と GCP はマイクロサービスの特徴を生かした構築例になっている．それぞれの IoT アーキテクチャを理解して，使用する機能の設定を行い，独自処理部分をファンクション（FaaS）機能でコーディングしデプロイすることでシステムを実現している．

逆に，IBM Cloud の場合は，接続/収集インタフェース以降は Cloud Foundry 上の Node-RED プログラムですべての処理を行う構築例になっている．前者はそれぞれのクラウド企業のアーキテクチャを理解して作りこむ必要があり，相互の移植は面倒に感じられた．逆に後者の場合では IBM Cloud と独自 PC 上のサーバ環境でほぼ同じ Node-RED プログラム（フロー）で実現することができ，移植しやすいと感じられた．いずれにしても実際のコード作成は少なく，NoCode/LowCode で構築できる環境が整ってきている．

８．ゼミナール「IoT をはじめる」での適用例

2014 年度秋学期から 2021 年度秋学期までの 7 年半に渡りゼミナール「M2M をはじめる」(2017 年度秋学期より「IoT をはじめる」に変更) で取り組んできた IoT（当初は M2M）プロトタイプ構築演習の成果について第 2 章から第 7 章にまとめた．

この章では，ゼミナールのカリキュラムの概要，7 年半に渡る IoT プロトタイプ構築演習の変遷と受講生のアンケート結果を含めた運営状況について報告する．

8.1. ゼミナールの概要

第 2 章における IoT プロトタイプ構築演習に求められる要件に基づいてゼミナールのカリキュラムを作成した．

IoT システムの構築には多くの技術を組み合わせる必要があることから，四部構成でステップバイステップで着実に技術を身に付けていく方式をとっている．また，受講生には文系や理系の様々なスキルを持った人材がそれぞれのレベルで満足度が高くなるように，基本系をベースに，スキルに応じて発展系や応用系に挑戦できるようにコースを設定した．IoT プロトタイプを安価に構築できるようにオープン系のハードウェアやソフトウェア環境を利用するとともに，オープン系特有の突然の変更に対応できる柔軟性も考慮している．

図 8.1 にゼミナールのカリキュラム構成と運営方法を示す．サイバー大学の場合は，非同期式の遠隔形式で授業が行われる．ゼミナールは 15 回で構成され各回は 1 週間間隔で実施される．演習内容の講義部分については VOD（ビデオオンデマンド）で提供し，演習に必要な操作方法や詳細については講義ノートとして提供している．また，演習で使うプログラムやデータは共通ディベートルームからダウンロードできるようにしている．

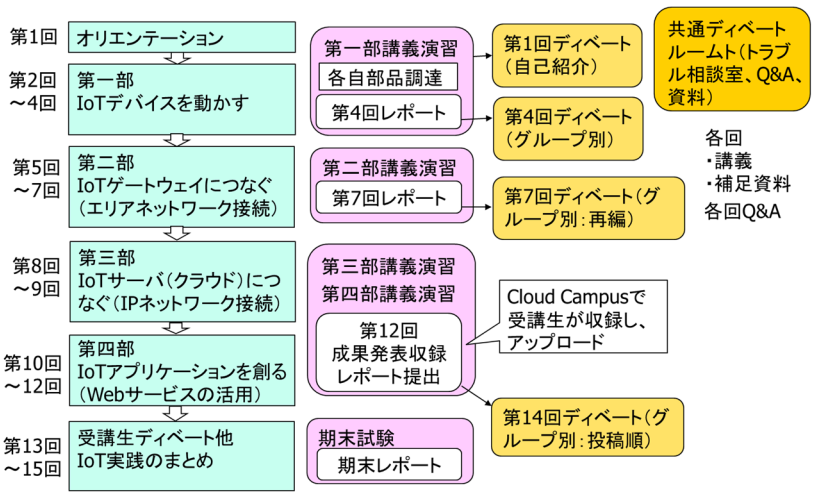

図 8.1　ゼミナールのカリキュラム構成

15 回の内訳は，四部構成になっており，第一部（第 2 回～4 回）の「IoT デバイスを動かす」では，基本系の Arduino Uno とセンサやアクチュエータを搭載したブレッドボードを接続して IoT デバイスを組み立て，動作を確認する．第二部（第 5 回～7 回）の「IoT ゲートウェイにつなぐ」では，IoT デバイスを PC（Windows）による IoT ゲートウェイにシリアル通信で接続した後，メッシュ型の ZigBee 通信の設定を XCTU で行い接続を確認する. 引き続き PC 上の Processing プログラムにより IoT デバイスからのデータを受信してビジブルに表示されるかを確認する．また，IoT デバイスに制御データを送信し IoT デバイスのアクチュエータ（LED）が駆動されることを確認する．第三部（第 8 回～9 回）の「IoT サーバ（クラウド）につなぐ」では，IoT サーバ（2021 年度秋学期は自 PC 上の OwnCloud）の設定を行い IoT ゲートウェイから IP ネットワークの MQTT プロトコルを使って送信できることを確認する．第四部（第 10 回～12 回）の「IoT アプリケーションを創る」では, IoT サーバ上にフロー型の Node-RED プログラムを構築し, Node-RED ダッシュボード上の操作パネル部から操作することにより以下の動作を確認する．センサデータをデータ表示部に表示したり，トリガ条件によって制御データを IoT デバイスにフィードバックして LED を駆動したり，Web サービスの IFTTT を使って Gmail でイベント通知したり，IoT データベース（2021 年度秋学期は File）に蓄積したりする．

各部の終了回でレポートを提出するとともに，グループ内で主に受講生間でディベートルームを使ってレポート内容について質疑応答を行う．第 12 回でゼミナールでの成果を「私の IoT プロトタイプ構築法」と題して発表する. 基本系は誰もが同じ結果になるので，得られた知見やトラブルの対処法を中心に成果をまとめ，挑戦した発展系について紹介する形をとっている．発表はサイバー大学提供のビデオ収録ツールを備えた Cloud Campus を使ってビデオ収録したものを公開し，第 14 回でその収録内容に対してグループ内で主に受講生間でディベートを行う．

本演習のようなモノづくり演習を遠隔非同期で行っているので，トラブル時の対応が重要になる．基本的なトラブルの対応は，共通ディベートルームの掲示板で行っている．受講生がトラブル内容を掲示板にアップすると原則講師陣が回答するが，受講生同士の情報交換も多くそれで解決することも多い．掲示板ベースで解決できない場合は，受講生と講師陣の間で遠隔会議システムを使って画面共有しながら解決を図っている．初期の頃は Skype を利用していたがコロナ禍以降（2020 年度以降）は Zoom を活用している．モノづくりの場合はそれでも解決しない場合もあり，最終的には受講生の現物を送ってもらうことで解決をしているが，意外とケアレスミスが多い．今までに集まったトラブル事例を補足資料として公開することで，最近はトラブル相談が減る傾向にある．

遠隔非同期の授業なのでトラブル時などで孤独を感じる人もいるが，ディベートを通じた情報交換がお互いを刺激し合い孤独感の解消に大変役に立っている．

8.2. ゼミナールの変遷

表 8.1 にゼミナール基本系のシステム構成の変遷について示す.

2014 年度秋学期の開始時点では，センサは温度/照度センサともにアナログ系であったが，2017 年度秋学期から温度センサをデジタルの BME280 モジュール（照度/気圧センサ内蔵）に変更した．また，2018 年度秋学期からアクチュエータに音声出力の圧電スピーカを追加した.

表 8.1　ゼミナール基本系の変遷

	年度	センサ、アクチュエータ	IoTデバイス、エリアNW	IoTゲートウェイ	IoTサーバ、プロトコル	Webサービス
M2Mをはじめる	基本系	・センサ：温度（A）、照度（A） ・アクチュエータ：LED	・Arduino Uno（Arduino IDE） ・シリアル接続/ZigBee接続	・PC（Windows、Processing）	・無料IoTサーバ ・HTTP/MQTT	・無料Webサービス
	2014秋	・温度（A：LM335Z） ・照度（A：AMS302）	・XBeeピッチ変換基板（はんだ付）	・GWでトリガ条件	・Xively（HTTP）	・Zapier/Gmail
	2016春				・M2X（HTTP）	・IFTTT/Gmailに変更
IoTをはじめる	2017秋	・温度（D：BME280：湿度・気圧も内蔵）	・XBeeモジュール（XB24Cに代替）			
	2018秋	・圧電スピーカ追加	・XBeeシールド（はんだ付無）採用			
	2019春	・照度（A：S13948-01SBに代替）				
	2019秋					・Zapier/Gmailに変更
	2020春			・GWのトリガ条件中止	・IBM Cloud（MQTT）、Node-RED（トリガ条件）、Cloudant（NoSQL）	
	2021秋				・OwnCloud（自PC）、Node-RED（トリガ条件）、標準File	・IFTTT/Gmaiに変更

IoT デバイスとして最初から Arduino Uno を，エリアネットワークとしてシリアル接続と ZigBee 接続を使用している．ZigBee を採用したのは，低消費電力でメッシュ型のネットワークに触れてもらうためである．はんだ付けが必要な XBee ピッチ変換基板を当初使っていたが，2018 年度秋学期以降に XBee シールドを採用してからはんだ付けが不要となり，初心者の負担がかなり減った.

IoT ゲートウェイとして当初から PC（Windows）版を採用して開発の効率を図った．従来，IoT ゲートウェイでトリガ条件によるアクチュエータの駆動を行っていたが，2020 年度春学期より IoT サーバ上にトリガ条件機能を移行した.

IoT サーバとして，2014 年当時最も有名であった BaaS 版の Xively（HTTP）を無料枠で使用した．その後無料サービスが中断したため，2016 年度春学期以降 Xively と互換性が高い AT&T 社の M2X（HTTP）に移行した．2020 年度に AT&T 社が無料枠のサービスを中断したことから，当時クレジットカードでの登録が不要であった IBM Cloud（MQTT）に移行した．しかし，2021 年 9 月に IBM Cloud もセキュリティ強化としてクレジットカード登録が必要になったため，急遽自分の PC 上に IoT サーバ（OwnCloud）を構築する形態に変更した.

Web サービスについては当初 Xively のトリガ機能と Zapier・Gmail 連携によるイベント通知機能を提供してきたが，2016 年度春学期に M2X に移行した時に M2X のトリガ機能と IFTTT・Gmail との連携によるイベント通知機能に変更した．2019 年度秋学期に IFTTT から Gmail 接続のサポートが中断されたのに伴い Zapier に変更した．その後 2021 年度秋学期に Zapier の無料枠の縮小に伴い，Gmail 接続が可能になった IFTTT に戻ることになった．

以上のように，電子部品の製造中止に伴う代替品への交換や，オープン系のソフトウェアやサービスの通告なしの当然の変更（中止）などが発生するので，代替策を事前に検討しておく必要がある．7 年半に渡る本演習の取り組みにおいては，すべて対策を打つことができて，大きな障害になることはなかった．

このような状況は，実際に稼働する IoT システムにおいても発生する問題であり，実際の運用・保守体制を事前に検討しておく必要がある．

8.3. ゼミナールの分析

本節では 2018 年度秋学期から 2020 年度秋学期の 5 学期分（ゼミナール修了者 46 人）についての分析内容を説明する．

図 8.2 にゼミナール修了者の内訳を示す．20 代から 80 代と広い年代の方が参加されており，それぞれの背景やスキルレベルはばらばらという状況である．最近は若い 20 代の比率が高くなる傾向にある．また，IoT に興味を持つ女性も多い．

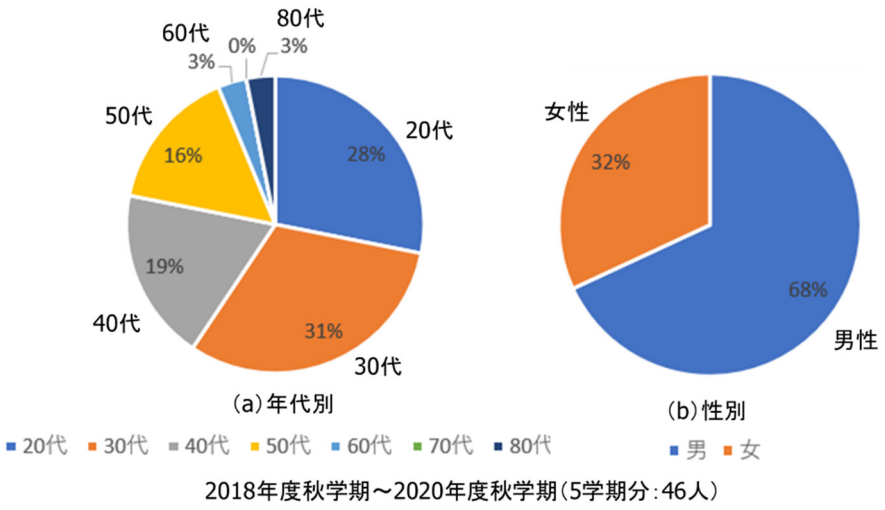

図 8.2　ゼミナール修了者の内訳

提出された期末レポートを分析してどの程度発展系に挑戦されたかを調べた結果を図 8.3 に示す.

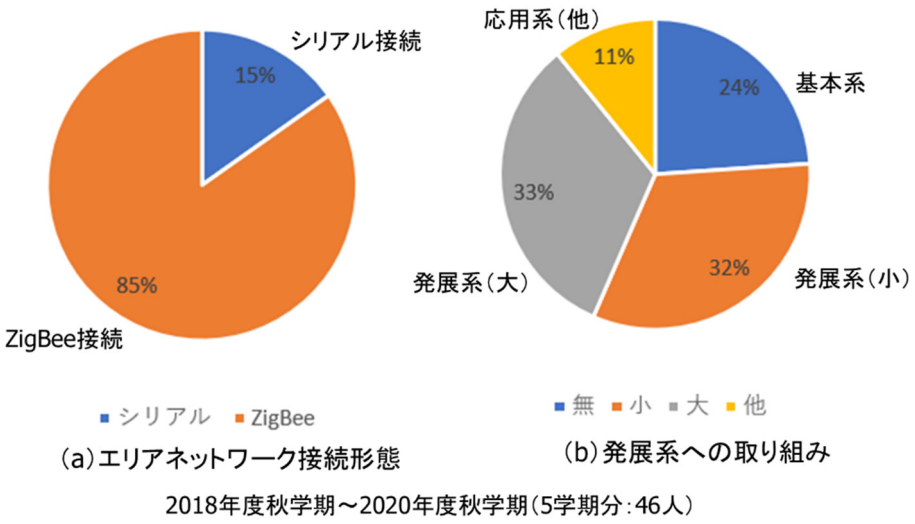

図 8.3　エリアネットワーク接続形態と発展系への取り組み

(a)のエリアネットワーク接続については，基本系として ZigBee を推奨していることからほとんどの人が挑戦しているが，設定等が面倒なこともあり，最終的に USB によるシリアル接続になるケースが 15%ほど存在する．これに対しては補足資料の充実で改善される見込みである．

(b)発展系への取り組みにおいて，基本系のみにとどまった人が約 1/4 を占めている．電子工作が初めてでプログラミングも苦手な初心者が多い．発展系（小）は，提供したプログラムの中に組み込まれた発展系のコードを活用して実現する簡単な発展系で，ある程度プログラミングができる人が多い．発展系（大）は，追加センサデータを IoT サーバまで追加していくやや難しい発展系で，過去に少し経験したレベルの人が多い．応用系（他）は基本系の他に独自のオリジナル機能の追加に挑戦したレベルの高い人が該当する．後のアンケート結果にも表れているように，基本系をベースとして発展系への挑戦を促したことで，それぞれのスキルレベルで満足度の高い結果になったかと思われる．

受講生（46 人）のアンケート結果（5 段階評価）の平均は以下のとおりで，比較的高い評価を得ている．

・授業内容の興味・関心度：4.46
・授業内容の理解度：4.33
・新たな知識や技能の修得：4.67
・授業を終えた後の満足度：4.46

また，受講生の主なコメント（自由意見）を以下に示す．全体として本来の目的を達成できていると思われるが，いくつかの問題点も指摘されており，今後の課題である．

・四部構成で進めることで IoT の仕組みや全体像を理解できた
・初めての電子工作で苦労も多かったが実際に動くと満足感が大きい
・ケアレスミスでも動かなくなるので着実に確認しながら進めるのが大事
・最後まであきらめないで取り組むことが大事である
・トラブルを解決することで，理解を深めることができた
・ディベートルームやトラブル相談室での情報（知見や工夫）や励ましが刺激になり大変役に立った（孤独感を解消できた）
・基本系は言われたとおりにやれば，比較的簡単に実行できるが，発展系でセンサを追加しようとするとプログラムの仕組みを理解しないと難しい
・トラブル等での質問のタイミングが難しく，後手後手になってしまう（困ったら早目に相談すべし）

⇒気軽に相談（Q&A ルームへの投稿など）するように指導を充実する必要がある

・資料に記載されているのだが，トラブったときにどこを調べたらよいのかを見つけるのが大変であった

⇒トラブル事例の収集が進み，補足資料なども充実してきているが，検索機能の充実が必要である

９．まとめ

2014 年度秋学期より継続して取り組んできたゼミナール「IoT をはじめる」（当初は「M2M をはじめる」）の 7 年半に渡る成果をまとめたものである．広範な技術の組み合わせである IoT システムを四部構成でステップバイステップで学びながら，安価な使いやすいオープン環境を提供することで初心者でも容易に IoT プロトタイプを構築できる実践的な演習環境を提供することができた．

受講者が文系や理系も混在し，年齢層やスキルレベルがばらばらな中で受講者の満足度を向上させるために，まず IoT システムの典型的なパターンを定義して修得すべき基本系を設定した．その上でスキルレベルに応じて発展系や応用系に挑戦できる仕組みを導入することでスキルレベルに応じた満足度を得ることができた．

本稿では，基本系の他に発展系や応用系への取り組みパターンを整理することができた．サイバー大学 IT 総合学部のゼミナール生からの改善要望や発展系の成果も活用させていただき感謝する次第である．

これまでのゼミナールでは，温度や照度などの比較的変化が緩やかでデータ量が少ないセンサデータを扱ってきたが，IoT のデータとしては時間軸に沿って高速に変化する信号やデータ量が多い映像に対する取り組みが残されている．今後は，これらの時間軸で高速に変化しデータ量の多いデータを対象に AI を使った分析までを含めた「オープン環境による IoT プロトタイプ構築」を目指して取り組んで行く予定である．

また，実際に IoT システムを構築し利用し続けていくためには，プロトタイプ構築とは異なり，セキュリティポリシーに基づいたセキュリティ対策，規模拡大や変化への拡張性，相互接続性や標準化動向に対応できる IoT アーキテクチャの柔軟性，バッテリ交換やプログラムの遠隔保守などのメンテナンス性の考慮，継続して発生するネットワークやクラウドなどのランニングコストへの対応などを十分に検討して実用化に臨む必要がある．

本取り組みの一部は，2019 年度-2021 年度/文部科学省/基盤研究（C）「IoT・AI 利活用人材育成のためのプロトタイプ構築法と分野横断実装教育の研究」（19K03151）の助成を受けたものである．

参考文献

1) NPO 法人 M2M・IoT 研究会 HP：https://www.m2msg.org/（2022/1/31 確認）
2) 電気学会第 2 次 M2M 技術調査専門委員会編：『M2M/IoT システム入門』，森北出版，(2016)
3) 辻秀一，澤本潤，清尾克彦，北上眞二：「M2M（Machine-to-Machine）技術の動向」『電気学会論文誌 C』 Vol.133, No.3, pp.520-531, (2013)
4) 清尾克彦：「M2M（Machine to Machine）技術の動向と応用事例」『サイバー大学紀要』第 5 号，pp.1-22, (2013)
5) 清尾克彦：「オープン環境による M2M-IoT システム構築の動向と取組事例」『サイバー大学 e ラーニング研究』第 4 号，pp.1-26, (2015)
6) スマート IoT 推進フォーラム IoT 人材育成分科会：https://smartiot-forum.jp/tech-strategy/jinzai（2022/1/31 確認）
7) 小泉寿男, 井上雅裕, 大江信宏, 秋山康智, 市村洋, 清尾克彦：「論説：分野別 IoT 教育カリキュラムの提案」『工学教育』66 巻 1 号，pp.46-53, (2018)
8) 5-stages-in-the-design-thinking-process：https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process（2022/1/31 確認）
9) 組込みシステム産業振興機構 第 14 回組込み適塾：【J02-02】実装演習「クラウドセンシング」：https://www.kansai-kumikomi.net/kumikomi/14th/syllabus/j02-02/index.html（2022/1/31 確認）
10) Arduino：https://www.arduino.cc/（2022/1/31 確認）
11) ESPr Developer：https://www.switch-science.com/catalog/2500/（2022/1/31 確認）
12) M5Stack：https://m5stack.com/（2022/1/31 確認）
13) obniz：https://obniz.io/ja/（2022/1/31 確認）
14) CHIRIMEN オープンハードウェア：https://chirimen.org/（2022/1/31 確認）
15) SensorTag：https://www.tij.co.jp/tool/jp/CC2650STK（2022/1/31 確認）
16) Raspberry Pi：https://www.raspberrypi.org/（2022/1/31 確認）
17) Processing：https://processing.org/（2022/1/31 確認）
18) Python：https://www.python.org/（2022/1/31 確認）
19) JavaScript：https://ja.wikipedia.org/wiki/JavaScript（2022/1/31 確認）
20) Node-RED：https://nodered.org/（2022/1/31 確認）
21) Node.js：https://nodejs.org/（2022/1/31 確認）
22) Xively：https://en.wikipedia.org/wiki/Xively（2022/1/31 確認）
23) M2X：https://m2x.att.com/（サービス停止により 2022/1/31 確認できない）
24) IFTTT：https://ifttt.com/（2022/1/31 確認）
25) Zapier：https://zapier.com/（2022/1/31 確認）
26) ZigBee：https://ja.wikipedia.org/wiki/ZigBee（2022/1/31 確認）

27） XBee：http://digi-intl.co.jp/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/index.html（2022/1/31 確認）
28） XCTU：https://hub.digi.com/support/products/xctu/（2022/1/31 確認）
29） WebSocket：https://ja.wikipedia.org/wiki/WebSocket（2022/1/31 確認）
30） MQTT：https://mqtt.org/（2022/1/31 確認）
31） Fritzing：https://fritzing.org/（2022/1/31 確認）
32） Key Trends from the IoT Developer Survey 2018：https://blog.benjamin-cabe.com/2018/04/17/key-trends-iot-developer-survey-2018（2022/1/31 確認）
33） IoT & Edge Developer Survey Report 2021：https://outreach.eclipse.foundation/iot-edge-developer-2021（2022/1/31 確認）
34） AWS IoT：https://aws.amazon.com/jp/iot/（2022/1/31 確認）
35） IBM Cloud：https://www.ibm.com/jp-ja/cloud/internet-of-things（2022/1/31 確認）
36） Google Cloud Platform（GCP）：https://cloud.google.com/solutions/iot?hl=ja（2022/1/31 確認）
37） Azure IoT：https://azure.microsoft.com/ja-jp/overview/iot/#overview（2022/1/31 確認）
38） 全部同じだと勘違いしていませんか？　4 大クラウド徹底比較（AWS, Azure, GCP, IBM）：https://goworkship.com/magazine/4-cloud-services/（2022/1/31 確認）