SQLクエリによるIoTデバイスの遠隔制御インタフェースの統合

木下 舜[1]　空閑 洋平[2]　中村 修[3]

概要：本研究では、インターネットに接続された複数のデバイスに対して、SQL クエリを利用した制御を可能とする、UDQ(Unified IoT Device Query) を提案する。現在、コンピュータの小型化と低価格化によりインターネットに接続され遠隔操作が可能なセンサやアクチュエータが増加しているが、多くの場合固有のリソース制限や API を持っているため、アプリケーションはそれぞれ個別に対応する必要がある。そのため、センサ値に応じて操作を変えるような、複数デバイスを横断したアプリケーション開発が難しくなっている。そこで、広く利用されている SQL の文法を利用した問い合わせにより、固有のインタフェースを意識しなくともアプリケーションからデバイスを制御できるシステム、UDQ を構築した。UDQ システムはアプリケーションが発行した SQL クエリを解釈し、条件に合致するデバイスに対してそれぞれに適切な方法で命令を行なう。このため、複数のデバイスを組み合わせたアプリケーションを作成することが容易になる。実装では、UDQ システム上に土壌の湿度を監視し、それを LED により通知するアプリケーションを作成し、システムの特徴を示した。

Unifying Interfaces of IoT Devices Using SQL Query

Shun Kinoshita[1]　Yohei Kuga[2]　Osamu Nakamura[3]

1. はじめに

近年、インターネットに接続し様々な役割を果たすデバイス (IoT デバイス) が次々と出現し、私達の生活へ進出してきている。これらは今まで遠隔から操作できなかった装置をインターネット経由で利用する事を可能とするものの、それぞれに異なる用途を想定して製作されていることが多く、異なる通信経路や API を持つため、専用のアプリケーションに強く依存している。一方、インターネットを介して多くのデバイスが接続可能な状態においては、多数のセンサから状態を判断したり、複数のデバイスが同時に処理を行なう事でより高度なアプリケーションを構築できる。その上ではデバイス固有の実装を隠蔽し、統合されたインタフェースにより命令を行なうことが望ましい。それにより、アプリケーションから複数のデバイスをまたいだ条件の指定や、動作の変更が容易に実現できる。しかし、デバイスの実装に使用されているコンピュータは、OS の無いマイコンや Linux の動作するシングルボードコンピュータなど、複数の種類が存在し、それぞれのリソースによりデバイスの機能が制限されているため、デバイス側に統一されたインタフェースを期待することは現実的でない。

本研究では、複数デバイスが連携したアプリケーション開発を容易にするため、SQL クエリを使用し、統一されたインタフェースを利用して複数のデバイスへ問い合わせや命令を行なうことを試みる。SQL はセンサデータを始めとしたデータベースの参照において一般的に使われており、O/R マッパを始めとしたライブラリ資産が存在しているので、操作インタフェースとして適切と考えた。UDQ の概要を図 1 に示す。アプリケーションは各々のデバイスに対して、データに対して行なうのと同じように問い合わせを行えばよく、その間に行われる通信や個別のインタフェースについて意識する必要はない。具体的な通信経路や通信

[1] 慶応義塾大学総合政策学部
Keio University Faculty of Policy Management
[2] 慶応義塾大学院政策・メディア研究科
Keio University Graduate School of Media and Governance
[3] 慶応義塾大学環境情報学部
Keio University Faculty of Environment and Information Studies

内容は、UDQ システムが隠蔽する。このことにより、あるセンサの現在の状態に応じて、他デバイスに接続されたアクチュエータの動作を変更するような横断的アプリケーションを容易に作成できる。

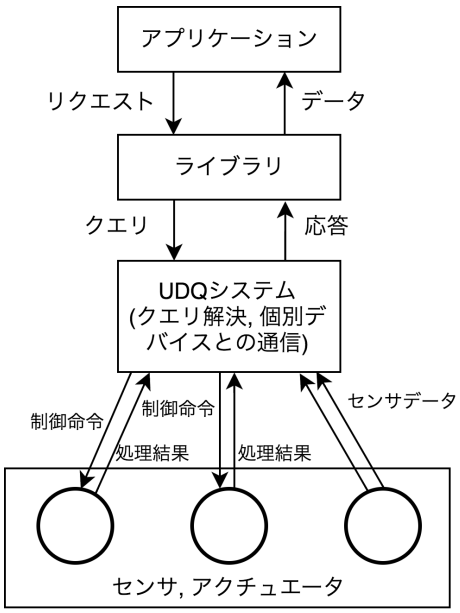

図 1: UDQ システムの概要

実装として、UDQ システムを用い、土壌の湿度を監視しその結果を LED の点灯にて表現するアプリケーションを作成し、デバイスを横断したアプリケーションが作成可能であることを示す。

2. 関連技術

プログラミング技術が発展するにつれて、多くのフレームワークやライブラリが誕生してきた。フレームワークの有名なものには、Ruby on Rails や、CakePHP などの Web アプリケーションフレームワークがあげられる。現在、このようなフレームワークは多くの Web サイトのバックエンドに使用されている。多くの場合、フレームワークは、O/R マッパ機能をもつ。これは、プログラム内で定義したオブジェクトの持つデータから SQL 文を生成し、RDBMS に保存する事で永続化する機能である。この機能を利用すれば、アプリケーションプログラマは複雑な SQL 文を記述する必要がなくなる。

小型かつ安価なコンピュータである Raspberry Pi [6] や Arduino [1] の登場により、自作 IoT デバイスを制作することも困難ではなくなった。さらに小型のコンピュータでも、インターネットに接続可能なものもある。例えば、ESP8266 [2] というマイコンがある。この製品は単体で無線 LAN に接続でき、TCP/IP での通信が行える。小型で省電力であるため、インターネットに接続されたセンサ, アクチュエータ製作に適している。前述した 3 つのデバイスにはそれぞれ特性があり、IoT デバイスを作成しようとするとき、その制限により提供できる API が変化する。Raspberry Pi は Linux が動作し、負荷に耐え、多彩なインタフェースを持てる一方で、ESP8266 は非力で、負荷の大きい API を提供することは難しい。Arduino は直接インターネットに接続できないので、何かしらの通信機器を経由しなくてはならず、通信方式を意識した API を用意しなくてはならない。このように、複数のインタフェースをもった、インターネット接続デバイスを横断的に利用することのできるプラットフォームは現在存在しない。

同様にインタフェースが分断された領域では、SQL もしくは SQL 類似言語を利用した抽象化層の研究が行われている。無線センサネットワーク (WSN) の領域では、Madden 等による TinyDB に関する研究 [4] がある。そこでは、専用に用意したハードウェアとその中で動く OS を利用し、ストリーム処理やイベント監視のための拡張を行った SQL 拡張言語である TinyDB 言語よる呼び出しをセンサネットワークに対して行なうプラットフォームを作成している。TiniyDB は、ノード間のネットワーク状態を監視しつつ配送コストを意識することで、消費電力が小さくなるような経路でセンサデータを収集する。

Web の分野では、Web 上にあるリソースを記述する Resource Description Framework(RDF) に対して、クエリによる問い合わせを行なう SPARQL [5] がある。RDF は、それまではプログラムから認識しづらく検索しづらかった Web のリソース同士の関係情報を、プロパティとしてとらえる。この考え方に沿ってデータベースを作成することで、Web リソースの関係性を元に情報が検索可能となる。SPARQL では RDF のトリプルと呼ばれるデータを対象に問い合わせを行なうため、SQL とは異なり、主語, 術後, 目的語の概念を利用してデータ動詞の関係を記述する。

コンピュータ構成, 状態の SQL を利用した抽象化としては、osquery [3] という製品がある。このソフトウェアは、OS ごとに異なるコマンドによって呼び出されるコンピュータの構成情報を SQL のデータベースとして抽象化し、別々の OS においても同一のインタフェースで呼び出せるようにする。ユーザやアプリケーション開発者は、利用しているコンピュータ環境の差異を意識せずとも必要な情報を取得, 変更することができる。加えて、コマンドラインでは複雑になってしまう複数の情報を組み合わせた条件検索を SQL を用いて記述できる。

3. UDQ システム

3.1 システムの概要

本研究ではデバイスを横断したアプリケーションを容易に作成できるようにするため、デバイス固有のインタフェースを抽象化するシステムを設計, 作成した。インタフェースとして、アプリケーション作成時に既存のライブ

ラリ資産を利用可能な、SQL を用いた。

システムは入力された SQL クエリを解釈し、保存されているデータベースから条件に合致したデバイスを特定した後、それぞれのアダプタを通じて、各々のデバイスに適した方法でデータ取得, 入力値送信を行なう。アダプタは各デバイスの API や通信方式の情報をもち、直接デバイスとやり取りを行なう役割を担う。このときの様子を図 2 に示す。

通信やデバイスに問題が有り、操作に失敗した場合はデータベースの内容は変更されない。参照されるセンサデータは最新の値が保存されたテーブルから参照される。それにあたり、計測されてからの時間経過が問題となるため、後述するタイムアウトを設ける。最終的に取得した情報や命令結果は、通常の SQL への問い合わせと同様の形でアプリケーションへと返却する。

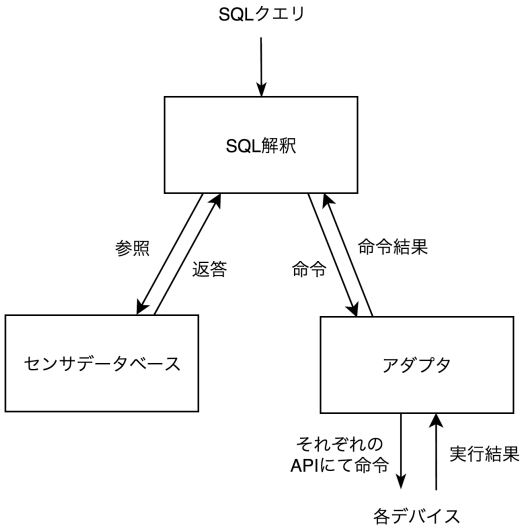

図 2: SQL 実行時の挙動

3.2 設計

既存の SQL 関連ライブラリを最大限に活用するため、SQL 構文への直接の改変は行わず、それぞれのキーワードに意味を追加する形で UDQ システムを設計した。センサ, アクチュエータの種類は、データベースのテーブルとして表現し、カラムとしてデバイスの IP アドレス, 待ち受けているポート番号の情報を持たせる。このことにより、リレーショナルデータベースの機能を利用し、センサ, アクチュエータが接続されたデバイスを参照することができる。加えて、センサの値や、アクチュエータの状態を示すカラムを持たせ、それらに対して SELECT 文や UPDATE 文を用い情報の入出力を行なう。

個々のセンサもしくはアクチュエータは行として表現し、現在の値に応じて操作を決定するため、最新のデータのみを保存する。これらの情報を保存するテーブルにおける、カラムの抽象化例を図 3 に示す。例としてセンサのテーブルを使用したが、アクチュエータにおいても、デバイスに接続されたアクチュエータを表現する部分は同様である。その場合、センサ値に代わって、現在の動作状態を反映するカラムがテーブルに存在する。これらの他にも、リレーションを作成する補助的なテーブルを UDQ システム上に作成することで、任意の条件に当てはまるデバイス全体に対して問い合わせを行なうことが可能となる。

タイムスタンプ	IPアドレス	Port番号	センサID	センサ値	タイムアウト
2016/1/1	aaa.aaa.aaa.aaa	9876	1	ddd	400
2016/1/1	xxx.xxx.xxx.xxx	1234	1	yyy	56
2016/1/1	xxx.xxx.xxx.xxx	5678	2	zzz	56

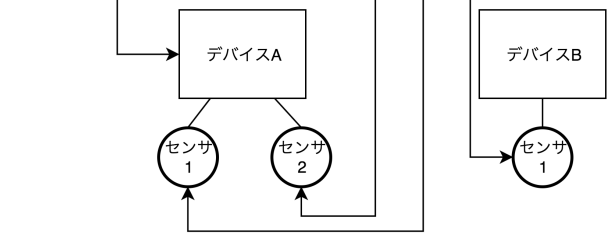

図 3: テーブルを利用したデバイスの参照例

以上の設計から、アプリケーションは参照や更新を通常の SQL データベースへの問い合わせと同様の形で行なうことができるようになる。問い合わせ結果を受け取ることで操作対象となったデバイスがどれであるのかや、命令が成功したか否かを判断できる。

3.3 SQL との対応

SQL はデータベースへの問い合わせや更新を行う言語であるため、これをセンサやアクチュエータに対しての問い合わせに利用するにあたり、表 1 のように幾つかの構文に意味を加えた。全て標準的な SQL と同じ意味を持っているが、SELECT 節と UPDATE 節においては、デバイスとの通信が生じることがある。SELECT ではデバイスの状態やセンサの値を取得でき、UPDATE によりテーブルの

表 1: SQL との対応

SELECT	取得するデータを指定 SELECT sensor_value SELECT status
UPDATE	デバイスに値を送信 UPDATE led SET status = '1'
FROM	デバイスの種類を指定 FROM moisture FROM wall_switch FROM led
WHERE	状態の範囲を指定 WHERE place_label = "RoomA" WHERE sensor_value > 100

特定カラムに書き込みが行われた際には、該当デバイスに対し動作命令を行なう。

3.4 タイムアウト

インターネットに接続されたセンサデータをデータベースに保存するにあたり、データを必要となるたびに取得するか、一定期間ごとに取得し続けるかの 2 種類の手段が考えられる。必要になるたびに取得する方法では、常に最新のデータを期待できるものの、複数のセンサを同時に利用した際、データ取得にかかる時間が大きくなってしまう。しかし、一定期間ごとにデータを取得する手法では、センサの種類によって変化する頻度が異なるため、全てのセンサから同じ間隔でデータ取得すると、取得された情報が現在も意味を持った値であるかがわからない。そこで、センサの特徴によって情報の有効期間が違うことに着目し、タイムアウトの概念を導入する。情報が意味を持つ期間について、気温, 湿度計のような値の変化が劇的で無いセンサデータの寿命は長くなり、加速度計など値の変動が大きいセンサでは寿命は短くなる。それぞれのセンサごとにその情報が価値を持っている期間を設定し、前回取得した情報の有効期間が終了する際にデータを再取得するようにした。データを送り続けるプッシュ型のセンサと、問い合わせに応じてデータを返答するプル型のセンサそれぞれについて、タイムアウトを導入した際の挙動を図 4 に示す。タイムアウトの概念を導入し、センサの特徴に合わせて適切なタイムアウトを設定することで、不必要な問い合わせが削減される。その結果、複数のデバイスに同時に問い合わせる場面が減り、システムにかかる負荷を軽減する。

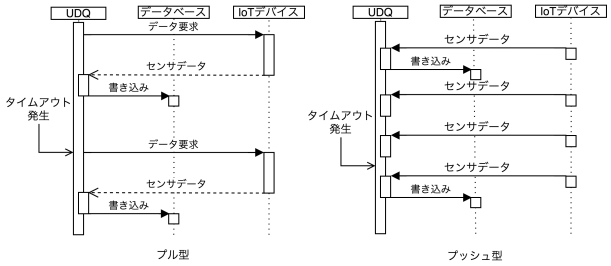

図 4: センサの種類とタイムアウト

4. 実装

4.1 UDQ システムの実装

本実装は SQLite の virtual table 機能上に構築されており、システム上で抽象化されたデバイスは通常の SQL テーブルと同様に振る舞う。SQL クエリの解析は SQLite の機能を利用した。クエリが解析された後、virtual table が呼び出された時に実行される、デバイスとのやり取りを担当するアダプタ部分を python を利用して作成した。タイムアウトが訪れるごとに、センサデータを更新する機構については、Ruby を利用して実装した。新しいデバイスを本実装に登録するためには、virtual table 機能を利用しデバイスへの具体的な操作方法を定義したアダプタを作成し、他のアダプタが動作する python スクリプトから参照できるようにする。

4.2 アプリケーションの概要

UDQ システムのモデルケースとして、土壌の湿度を計測しつつ、一定の値よりも値が大きかった場合、LED の点灯によりユーザに通知するアプリケーションを作成した。インターネットに接続された土壌湿度センサと、LED を用意し、それらに対して UDQ システムから問い合わせを行なう。これらのデバイスは、サーバとして動作し、アクセスを受けると渡されたパラメータに応じて動作を変えるよう実装した。作成したアプリケーションの構成と処理は図 5 の通りである。

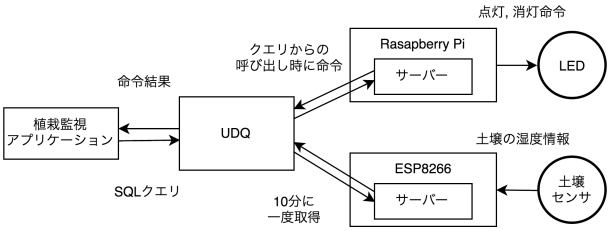

図 5: アプリケーションの構成と処理

異なるコンピュータに接続された 2 つのデバイスに対し、UDQ により問い合わせを行なう。植物栽培時に水をやるべきかを判断できるよう、土壌を監視するアプリケーションを想定し、一定の湿度以下となったときに、LED を点灯させる命令を下す。

土壌の湿度は頻繁に変化するものではないので、センサデータのタイムアウト時間は 10 分間に設定した。

4.3 データベース構造

本アプリケーションでは、土壌湿度センサを表す moisture テーブルと、LED を表す led テーブルを作成した。それぞれのテーブルの内容は表 2 と表 3 の通りとした。それぞれのサーバを指定するために、IP アドレスとポート番号を保持させ、センサである moisture テーブルには、センサ値を保持するカラムと、タイムアウト設定を保存するためのカラムを、アクチュエータとなる led テーブルには点灯の有無を表現するカラムを設けた。また、呼び出し範囲をよりアプリケーションからわかりやすくするために、場所ラベルとそれに対応する IP アドレスを保持するテーブル、place を作成した。

表 2: moisture テーブルのカラム

カラム名	ip	port	sensorvalue	timestamp	timeout
内容	デバイスの IP アドレス	デバイスのポート番号	計測されたセンサ値	データが取得された時間	タイムアウト時間

表 3: led テーブルのカラム

カラム名	ip	port	led status	timestamp
内容	デバイスの IP アドレス	デバイスのポート番号	LED の状態	データが更新された時間

4.4 クエリ

土壌の湿度監視アプリケーションを UDQ システム上で作成するにあたり、問い合わせのための SQL クエリを設計した。今回使用したクエリを図 6 に示す。

```
UPDATE led
SET status = '1'
WHERE ip = (
  SELECT led_ip
  FROM (
    SELECT ip
    FROM moisture
    WHERE sensor_value < 260
  )
  INNER JOIN place ON place.sensor_ip = ip
  WHERE place_label = 'RoomA'
);
```

図 6: 湿度を元に LED を操作するクエリ

この SQL クエリでは、最初に土壌センサの値が一定以上の物を moisture テーブルから捜索し、それを元に場所ラベルとのリレーションから操作対象となる LED のサーバを特定する。その後、led テーブルの status カラムを更新する。このとき、システム内部で LED のサーバへのアクセスが発生し、LED への命令が行われる。status カラムに 1 が書き込まれた際は LED が点灯し、0 が書き込まれると消灯する。成功した場合は led テーブルの status が更新され、失敗した場合は、更新されない。これを監視し、アプリケーションは、現在 LED が点灯しているのか消灯しているのかを知ることができる。

5. まとめ

本研究では、様々なインタフェースが存在する IoT デバイスに対して、デバイスの種類をまたいだアプリケーション開発を行いやすくするために、SQL クエリによる統一インタフェースを提案した。その手段として、SQLite の拡張機能として UDQ システムを実装した。UDQ システムを経由することによって、アプリケーションはデバイスごとに異なる実装を意識すること無くデータの取得や制御を行なうことができる。また、豊富なライブラリ, フレームワーク資産を活かすことが可能である SQL をインタフェースとして選択することで、様々なアプリケーションに対応する。

SQL により複数のデバイスを制御するために、テーブル内の行として各デバイスを抽象化し、接続先の情報や、デバイスの状態を保持させた。問い合わせを処理するにあたり、必ずしも常に最新のデータが必要でないことに注目し、センサデータの取得を呼び出されるたびに行なうのではなく、タイムアウトを設定しデータ収集を事前に行なう事で負荷を軽減した。

UDQ システム上で植物を栽培している土壌の湿度を監視するサンプルアプリケーションを設計, 実装し、複数の IoT デバイスを統一的に操作することが可能であることを示した。ここでは異なるデバイスを使用してセンサとアクチュエータを設置し、それらを関連付けるテーブルを利用することで、SQL の利点を活かしたクエリ発行が可能であることを確認した。

5.1 今後の課題

UDQ システムにおいて、デバイスの操作に必要な全ての情報は仮想テーブルのカラムとして表現されなくてはならない。このことにより、インターネットに直接接続されていない大規模なセンサネットワークや、複雑なパラメータ設定が必要なデバイスに対して問い合わせを行う際、多くのカラムが必要となってしまう。カラムの増加は、発行しなくてはならないクエリの長さに大きく影響を与え、アプリケーション作成を難しくする。この問題を解決するためには、システム内で個別のデバイスとやりとりをするアダプタ部分での工夫が必要となる。

操作するデバイスを新規に追加する時は、仮想テーブルを増やし、実際に処理を行なうアダプタを作成する。しかし、現在の実装ではプログラムを大きく変更しなくてはならないため、作成が困難となっている。そこで、外部に設定ファイルをもたせ、そこへの記述によりデバイスを追加できる仕組みを導入することを予定している。

現在の UDQ システムは SQLite の python 用ライブラリである apsw 上で実装されており、SQLite をデータベースシステムとして利用するライブラリ群から直接使用する事はできない。SQLite を対象としているライブラリからプログラムを変更せずに UDQ を呼び出すには、C 言語により再実装し、SQLite の静的ライブラリとしてビルドする必要がある。アダプタも含めてビルドする必要が有るためこの方法はデバイス新規追加の容易さとトレードオフの関係にある。

参考文献

[1] Arduino. https://www.arduino.cc/.

[2] ESP8266. https://espressif.com/en/products/hardware/esp8266ex/overview.

[3] Facebook, Inc. osquery. https://osquery.io/.

[4] Sam Madden, Joe Hellerstein, and Wei Hong. Tinydb: In-network query processing in tinyos. 2003.

[5] Eric Prud'hommeaux and Andy Seaborne. SPARQL

query language for RDF. W3C recommendation, W3C, January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[6] Raspberry Pi. https://www.raspberrypi.org/.