技術レポート

# サーバールームセキュリティ管理テストシステム

## ― センサーネットワーク実応用実験から ―


技術研究部
ユビキタス研究グループ
伊藤　強

技術研究部
ユビキタス研究グループ
佐々木達也


## 1．はじめに

センサーネットワークとは、センサーを持った小さな（通常多数の）コンピュータが無線ネットワーク越しにデータを収集するシステムである。センサーネットワークによって実空間のさまざまな状況に関する情報をコンピュータネットワーク内に取り込むことができる。

現在のコンピュータネットワーク上に存在する情報は、その多くが人間が入力した古い情報である。しかし、センサーネットワークが普及すると、現実の世界の各所に設置されたセンサーが（例えば気温などの）各種の物理量をリアルタイムで自動的に収集するようになる。そしてそれらの「新鮮な粒度の細かい」データにユーザーがアクセスできることになるのである。これによって、コンピュータが現実の状況をより詳細に把握することが可能になり、従来では不可能だったさまざまなアプリケーションが生まれる可能性がある。

現在、技術研究部（以下技研）ユビキタス研究グループでは、センサーネットワークを応用して実際に役立つアプリケーションを構築することに焦点を当てて研究を進めている。今回は、特に最近の個人情報保護の重要性の高まりなどに鑑み、重要拠点のセキュリティ管理の自動化を目的として、センサーネットワークの技術を応用したテストシステムを構築した。

このシステムは、CACの本社移転に伴い移転先で実験環境を確保することができなかったため、当初の計画より早い段階で終了しなければならなくなった。しかし、これまでの実験によってすでに多くの成果を得られているので、今回それを紹介する。

## 2．テストシステム概要

テストシステムのユーザーインタフェースはWebアプリケーションとしてインプリメントされている。ブラウザでこのアプリケーションを開くと図1の画面が表示される。

この画面の右上に見えるのが旧飯田橋事業所新館2階にあったサーバールームである。入り口と中央付近に網かけの（本来は色付きである）楕円形のアイコンが表示されている。

入り口にあるのがドアの状態を表示するアイコンである。ドアが施錠されているかどうか、また、ドアが開いているかどうかを知ることができる。

サーバールーム中央付近にあるのが、モーションセンサー（人間が出す赤外線を検知するセンサー）の状態を表示するアイコンである。このセンサーによってサーバー

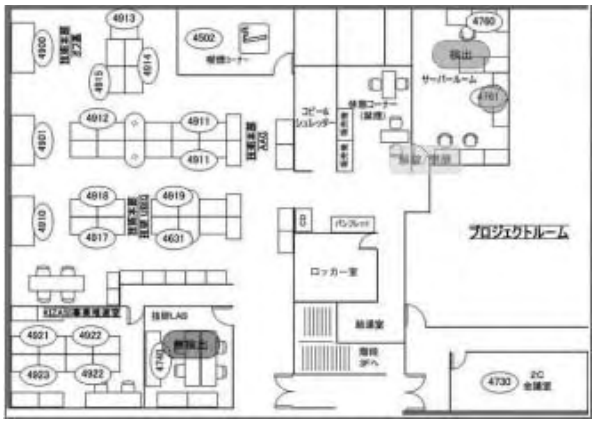


**図1　技術研究部のレイアウト**

アプリケーションは当然のことながらカラー画面であるためモノクロに変換した。そのため非常に見にくくなってしまったが、ご容赦願いたい。

ルームの中に人がいるかどうかを知ることができる。

また旧飯田橋事業所には技研実験スペース（技研LAB）にも同様のモーションセンサーが取り付けられていた。このセンサーの状態を表示するのが画面下方にあるアイコンである。

### 2.1　ドア状態検知

物理セキュリティの基本はドアの状況把握である。本システムではドアの開閉状態、施錠状況をWebページで確認することができる。

アイコンは次の3つの状態をとる

- 「施錠」（青色のアイコン）
- 「解錠/閉扉」（黄色のアイコン）
- 「開扉」（赤色のアイコン）

「施錠」状態以外の状態が一定時間続くと警告ダイアログがポップアップして危険を知らせる。

さらに、ドアの状態の詳細情報を見る場合、ドアのアイコンを右クリックし、コンテキスト・メニュー（図2）から「施錠/開閉状態を表示」を選ぶと、図3のダイアログがページ中に開く。このダイアログには現在の状態（ここでは開扉状態）になった時刻と経過時間を表示している。

またコンテキスト・メニューから「状態遷移履歴を表示」を選ぶと、ドアの状態が何から何に、いつ変わったかという情報の履歴を一覧表示することができる（図4）。

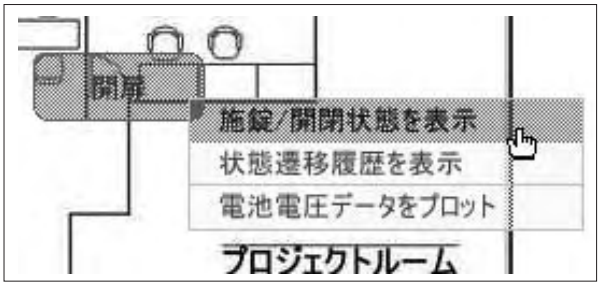


図2　ドアセンサー・メニュー


図3　施錠／開閉状態ダイアログ


図4　施錠／開閉状態ダイアログ

### 2.2　人の検知

サーバールームの物理セキュリティで二番目に重要なのが、人の侵入を検知することである。本システムではサーバールーム内の人の有無を検知して表示する。モーションセンサーのアイコンは下記の2つの状態をとる。

- 「無検出」（青色のアイコン）
- 「検出」（赤色のアイコン）

ドアセンサーのアイコンと同様に右クリックによってコンテキスト・メニューを呼び出し詳細な状態を表示できる（図5）。

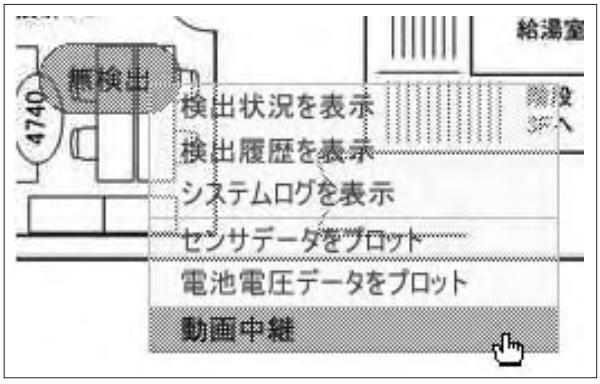


図5　モーションセンサー・メニュー

各センサーノードはIDタグの読み取り機能も実装しているため、タグを身につけた人がサーバールームに入ると、自動的に記録される（図6）。

またタグを身につけた人でない場合は「未登録」という形で記録される。セキュリティレベルの設定によってこの


図6　モーションセンサーダイアログ


図7　人検知履歴ダイアログ


図8　スナップショットダイアログ〉挿入

「未登録」人物の入室が検知された場合、ドアが一定時間開いていた場合と同じようにアラートダイアログを表示することもできる。実際の実験では、サーバー室ではアラート表示をするように、技研実験スペースではアラート表示をしないように設定されていた。

またカメラを設置することで、モーションセンサーのデータをカメラで撮影した画像データの撮影時刻と照合し、履歴ダイアログ（図7）からそれぞれの画像を呼び出すことができる（図8）。ライブ映像によってカメラが設置された場所を監視することも、もちろん可能である。

## 3．テストシステム構成

本システムは、大きく分けて下記の3層から構成されている（図9）。

●センサーネットワーク層：センサーで各物理量を測定し無線で送信する

●ミドルウェア層：センサーネットワーク層からデータを取得しデータベースに記録・保存する

●アプリケーション層：データベースから読み込んだデータをGUIでユーザーにわかりやすく表示する

それぞれの層について、以下順を追って説明する。

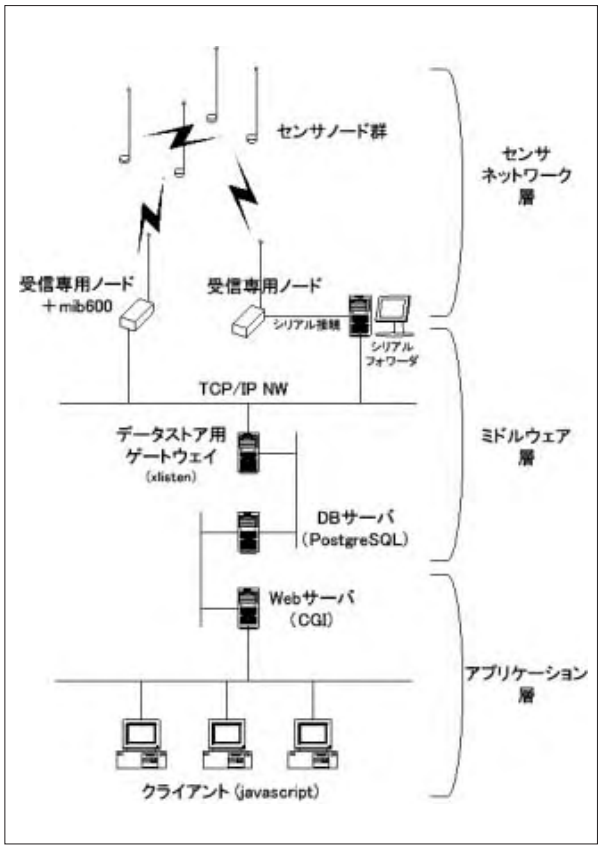


図9　システムを構成する3層

### 3.1　センサーネットワーク層

#### 3.1.1　Mote

センサーネットワーク層は、無線通信機能を持ち電池で駆動される小さなコンピュータであるMote（塵の意、小さなものでは500円玉大のものもある（図10））が基本構成要素である。このMoteはカルフォルニア大学バークレー校によって開発されたセンサー・デバイスであり、TinyOSという専用の基本ソフト上で動作するプログラムによって制御可能である。例えば、センサーの出力値を取得する時間間隔、取得データを無線パケットで送信するタイミングなど、あらゆることを自由に決定することができる。

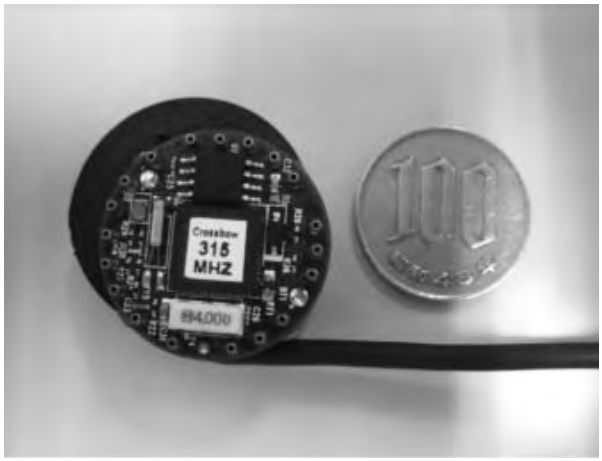


**図10　Mica2dot（500円玉大のMote）**

Moteに汎用センサー基板を取り付け、各種センサーからのリード線をこの基板上の適切なスルーホールに接続することで、1つのセンサーノードが構成される（図11）。

各センサーノードから無線で送信される測定データパケットはデータ受信専用ノードで受信される。このノードも受信専用にプログラムされているだけでセンサーノードと同じMoteが使われている。データ受信専用ノードはシリアルポート接続デバイスか、またはEthernet接続デバイスに接続することによって、PCなどから受信データにアクセスすることができる。

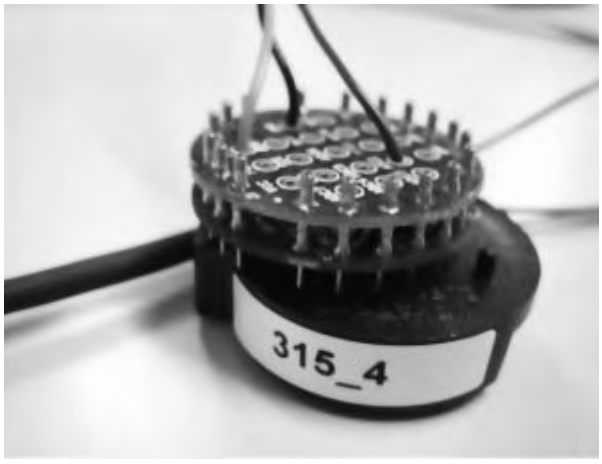


**図11　Mica2dotに汎用センサーボードを載せたところ**

Moteは、実際に使用していると電池の消費が予想以上に早く、本システムが稼動し始めた当初は、単3アルカリ乾電池2本で10日程度の寿命しかないという大きな問題点があった。

電池消費の調査を行い始めた頃には、この問題の原因をCPUでの電力消費が大きいため、という予想を立てていた。しかし、実際は無線通信コンポーネントの部分が大きな電力を消費していたことが明らかになった。

初期に用いていた無線通信コンポーネントはtos/CC1000RadioAck/ディレクトリ以下のものであったが、後期では tos/CC1000RadioPulse/ディレクトリ以下の省電力仕様のコンポーネントを使用した。この省電力コンポーネントは、簡単に言うと動作中の大部分の期間を寝て過ごし、たまに起きて仕事をするという戦略で消費電力の大幅な削減を実現するものである。

CC1000RadioPulse以下のコンポーネントを使用した現在稼動実験中のノードは、本稿執筆時点で連続稼動日数140日を超えており、電圧は2.2V程度を維持している。あとどの程度持つかは、現時点では予測が大変難しい状況である。

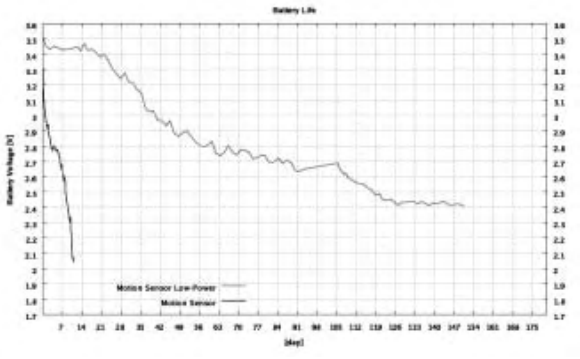


**図12　通常コンポーネント使用時と省電力コンポーネント使用時の電池電圧の推移**

図12のグラフ中の左側の点線が初期のモーションセンサーノードの電池電圧の推移である。11日程度で電圧が約2.05Vで動作しなくなっていた。一方、右まで伸びる線は後期のモーションセンサーノードの電池電圧の推移であり、156日を過ぎた現在でも2.2V前後の電圧を保っておりなお稼動中である*1。

*1）本稿執筆後に起動より184日間で停止した。

### 3.1.2　ドアセンサー

ドアの開閉状態を検知するためにホールICを使用し、施錠状態を検知するための接触型のON/OFFスイッチを作成した。

まず、ドアの施錠状態を見るON/OFFスイッチの仕組みを説明する。このスイッチにはMote付属の電池ボックスのうち、不要になったものからマイナス端子を引き抜き再利用した。この端子2つが並んで固定されるように基盤にはんだ付けし（図13）、これをドアの鍵受けのくぼみにセットした。それぞれの端子は電気的に離れており、鍵がかかっていない状態ではOFF状態になる。ドアの鍵をかけることにより、鍵の金属ブロックが接触して電気的に短絡（ショート）しON状態になる。極めて原始的な発想をそのまま具現化したものであるが、このようなものでも施錠状態を完全にモニタリングすることができている。

**図13　ON/OFFスイッチ**

鍵が開いていると、スイッチがOFF状態で固定抵抗に電流が流れないのでADCポートでの電圧測定値がほぼ0Vになる。鍵が閉まってスイッチがON状態になるとMote給電ポートから3V程度の電圧が固定抵抗にかかり、ADCポートでの電圧測定値は給電ポートの電圧値に等しくなる。つまり、測定値がほぼ0Vなら鍵が開いていて、3Vなら閉まっている、と判断できる。

次に、ドアの開閉状態を見るホールICの仕組みを説明する。ホールICとは簡単にいえば磁気センサーの一種である。移動する荷電粒子が磁場の影響で進行方向を曲げられる性質（フレミングの左手の法則）を利用して磁場を検知する。磁場が一定値以下なら数百mVの不定な電圧値を出力し、磁場が一定値を超えると完全に0mVの電圧値を出力する。この著しい非線形性によって、磁場の有無を鮮明に検知することが可能である。

このホールICをドアの枠に固定し、永久磁石をドア側に取り付けた。ドアを閉めると永久磁石がホールICに接近するようになっており、ホールICに大きな磁場がかかる。逆にドアを開くとホールICにかかる磁場は小さくなる。この差は前述のように大きな電圧差として測定できるので、結果としてドアの開閉状態を検知することができる。

これらのON/OFFスイッチとホールICを1つの500円玉大のMoteであるMica2dotに接続し、ドア状態検知ノードとして動作させている。Mica2dot汎用センサー基盤ではADC2～ADC7まで合計6ポートのAD変換ポートが使用可能なので[*2]、1つのMoteに6種類までセンサーを接続することができる。

ここで1つ重要なのは、ON/OFFセンサーもホールICも、ドアの状態検知以外にもさまざまなモノの状態検知に応用することができる、ということである。特にホールICと永久磁石の組み合わせは、物理的な接触個所が皆無なので、高い信頼性を達成できる。

### 3.1.3　モーションセンサー

サーバールーム内への人の侵入を検知するために市販のモーションセンサーを入手し、Moteに接続した。今回使用したモーションセンサーは松下電工製のNaPiOnである。これは、焦電効果（電気的な自発分極が電磁波により乱されると、分極電圧が変化する効果）を利用した赤外線センサーであり、簡単にいうと人の体から放射される赤外線を検知するセンサーである。NaPiOnの監視範囲内で人が動くと、NaPiOnの駆動電圧に等しい電圧を出力し、検知されない場合は出力端子はオープン状態になる。この差は極めて大きいので、人の有無を鮮明に判別可能である。

このNaPiOnの端子とリード線を基盤にはんだ付けし、PC固定用の台に貼り付けた。そのリード線をMoteに接続し、モーションノードを構成したものを、サーバールーム中央のロッカーの上に、ドアの方を向くように設置した。置き場所をかなりいい加減に選んだにも関わらず、サーバールームへの人の侵入を完璧に検知できたことについては、NaPiOnの性能の高さに驚かされた。

### 3.1.4　IDタグ

IDタグは、ID送信機能だけをプログラミングしたMoteを用いている。これは、現在広まりつつあるアクティブ型RFIDタグの動作を模倣したもので、ゆくゆくはMoteを使った代用ではなく、安価なRFIDタグへ置き換えることができると期待している。

標準状態のMoteでは電波パケットの送信能力が高すぎるので、次ページ図14のようにアンテナを曲げることに

＊2）他にADC0、ADC1というポートがあるが、こちらは自由に使うことができない。

よって電波の出力を抑制している。電波の到達範囲が適度に限定されていると、タグの位置把握に効果的になる。現時点で本システムにはタグの詳細な位置把握機能は実装されていないが、現在でも各受信ノードでタグからの電波強度を測定／記録しているので、作り込みを行えば、電波強度を利用してタグが存在している位置を推測することも可能である*3。

**図14　IDタグノード**

### 3.2　ミドルウェア層

ミドルウェア層におけるデータの流れの概要を図15に示す。

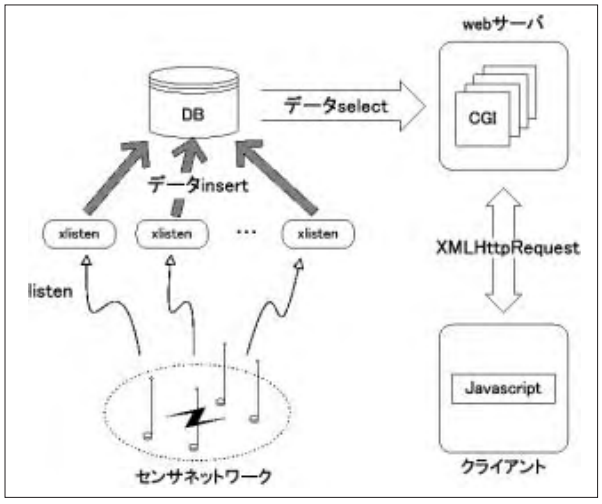


**図15　ミドルウェア層におけるデータの流れ概要**

#### 3.2.1　データ取得／データストア

センサーネットワークからのデータ取得とデータベースへのデータのストアはMote開発セットに付属しているXListenというデータモニター用ツールを独自に拡張したものを用いている。XListenはもともと、各センサーノードの情報を、PCとつながれたシリアルケーブル接続デバイスによって接続されたデータ受信専用ノードからモニターすることによって、センサーボードが適切に作動しているかどうかを確認するためのツールである。ただし、このツールのソースコードを覗くと、LAN経由で接続されたデバイスと通信するモジュールと、さらに別のPCにシリアルケーブル接続されたデバイスともSerialForwarderというツール経由で通信するためのモジュールを作成しようとした残骸があった*4。そこでこのモジュールを完成させ複数のデータソースを選べるようにした。また、このツールはもともと取得したデータパケットを16進形式でダンプしたり、センサー値をデコードして人にわかりやすい数値を表示するなど、さまざまな出力形式をサポートしている。このため出力モジュールの追加も簡便に行えるように設計されており、このツールを拡張して用いることで、新しいセンサーが追加された場合も容易にそのセンサーのための出力モジュールを追加できることも、このツールを利用した理由の1つである。

今回XListenに行ったモジュールの残骸の修正と機能追加は下記の通り。

- ●LAN経由でのMoteとの接続モジュールの修正
- ●SerialForwarder経由でのMoteとの接続モジュールの修正
- ●データベース（PostgreSQL）への接続モジュールの修正
- ●データベースへのデータストアモジュールを含む、各センサーノード、IDタグノードの出力モジュールの追加

本システムではサーバールームからの情報をサーバールーム内に設置したLAN接続デバイスから取得し、技研実験スペースからの情報は実験スペースと近接している技研部員の机上に設置されたシリアル接続デバイスから取得している。

#### 3.2.2　データベース

本システムではデータベースに、代表的なオープンソースRDBMS（Relational DataBase Management System）の1つであるPostgreSQLを用いている。

センサーネットワークのデータのような大量のデータを扱わなくていけない場合の問題点の1つとして、データの蓄積が進んだ場合のパフォーマンスの問題がある。

PostgreSQLはINSERTに最適化されたRDBMSでありINSERTの処理はテーブルにデータが増えてもほとんど変わらない。逆にSELECT処理はデータ量の増加にともない

＊3）実際に先行研究では誤差1〜2mの精度にてタグの位置を把握することができた。

＊4）最新のCVS版ではこの残骸は完成されているが、さらにアプリケーション固有といくつかの汎用の機能追加も行っている。

パフォーマンスが落ちていくことになる。また、PostgreSQLではDELETE処理は削除フラグが付加されSELECTの対象から外されるだけで実際にデータが削除されるわけではない。同様にUPDATE処理もDELETE→INSERTという処理がなされるためデータ量が増えるので要注意である。本システムでも最新データのみを保持するテーブルを実現するために大量のUPDATE処理を行っている。そのせいで、見かけ上Moteノードの数の行しかないテーブルに対しての問い合わせに数秒かかるような事態になった。これを回避するためXListenのデータストアモジュールにおいて、削除フラグのついたデータを実際に消すSQL文（VACUUM）を一定間隔で発行している。

また、アプリケーションでは表示するデータによって、時間の経過とともに表示するための時間がかかるものがあった。この問題は取得した大量のセンサーデータをすべて貯め込んだテーブルからSELECT問い合わせを行ってデータを抽出するときに時間がかかることが原因である。この問題に対処するために最初に考えたのが、センサーデータをすべて貯めていくテーブルとは別にアプリケーションに必要な期間のデータ（たとえば過去1日間）のみを持つテーブルを作成する方法である。しかし、この方法では最新データを保持するテーブルと同様の、削除フラグがついているデータを実際に消す処理が必要であり、この処理を行う間隔を調整するのは最新データテーブルのように簡単ではない。このため、別の対処方法を考慮する必要があった。

本システムが実際に採用した方法は、ドアの開閉状態の変化やモーションセンサーの非検出状態から検出状態への変化をイベントと考え、このイベントのみを記録するテーブルを作り、このテーブルからデータを抽出することで問い合わせ時間の短縮を図るというものである。

イベントテーブルへのデータの挿入にはPostgreSQL独自の機能であるルールという機能を用いて行っている。ルールは他のDBMS（DataBase Management System）における（そしてPostgreSQLにも存在する）トリガーという機能に非常に良く似ている。両者の大きな違いは、トリガーが条件にしたがって前もって定義されたプロシージャを起動するのに対し、ルールでは条件にしたがって追加のSQL問い合わせが行われる点である。プロシージャ呼び出しのオーバーヘッドがないため、ほとんどの場合にルールはトリガーより高速に動く。

このイベントテーブルを用いる前は約100日間のデータがあった場合、サーバールームに入室した人が検出されたかどうかをアプリケーションで表示するためには約25秒の時間がかかった。イベントテーブルを用いた後に同じデータに対して同じ操作をした場合は1秒かからずにデータを表示できるようになった。

### 3.3 アプリケーション層

#### 3.3.1 Ajax

本システムでのアプリケーションのユーザーインタフェースにはAjaxの手法を用いて、画面遷移のない比較的リッチなWebアプリケーションを実現している。

AjaxとはAsynchronous Javascript＋XMLの略で、簡単にその仕組みを説明すると、ユーザーが使うブラウザに読み込まれたjavascriptから非同期でサーバーに対してGET/POSTを行い、javascriptによってその結果をブラウザ上に表示するものである。この手法を駆使すれば、Webブラウザ上でデスクトップアプリケーション並の表現をすることも可能となる。

本システムでは、マップ上のノードの表示を一定間隔でリフレッシュしたり、ブラウザの中にダイアログを開いて直近1時間のデータをプロット表示する、といったことを行うためこの手法を用いている。

#### 3.3.2 ユーザーインタフェース

本システムのユーザーインタフェースには一度大幅なバージョンアップが施されている。もともとは1つのノードに対し1つのダイアログが表示され、タブによって表示するデータを切り替えていた。このインタフェースは直感的でわかりやすい反面、タブを切り替えるごとに（データの更新を必要としない場合でも）新しいデータを取得しなければならず*5、切り替えたタブを表示するのに時間がかかるという欠点を持っていた。このような欠点を解消するため、バージョンアップ時に基本的な操作はすべてコンテキスト・メニューを基本としたものに変更した。また、javascriptに疑似クラスを導入することで、すべてのユーザーインタフェースやノードの情報などのオブジェクトを、javascriptでは通常行うことのできないクラスベースのオブジェクト指向にもとづいて、一から設計しなおした。

以下に本システムで特徴的なインタフェースについて説明する

●コンテキスト・メニュー

多くのウィンドウシステムやアプリケーションと同様に右クリックによってコンテキスト・メニューを（用意されていれば）呼び出せる。例えばドアセンサーのノードアイ

*5）なぜ、常にタブの切り替えとともに新しいデータを取得しなければならないのかという疑問を持たれるかもしれない。これはタブを切り替えるとデータを取得するのではなく、新しいデータを取得するにはタブを切り替えるタイミングしかないためである。このためタブの切り替えとデータ取得は同じアクションによって起こる。

コンを右クリックすると図2のようなメニューが開く。このメニューから必要な情報をクリックすることによってその情報を表示するダイアログを開くことができる。

●ポップアップ・ヒント

センサーノードのアイコンの上にマウスカーソルを乗せることによって、ポップアップ・ヒントを呼び出し、そのノードがどのような役割を持っているか、直近にデータを受信した時刻など基本的なデータを見ることができる（図16）。

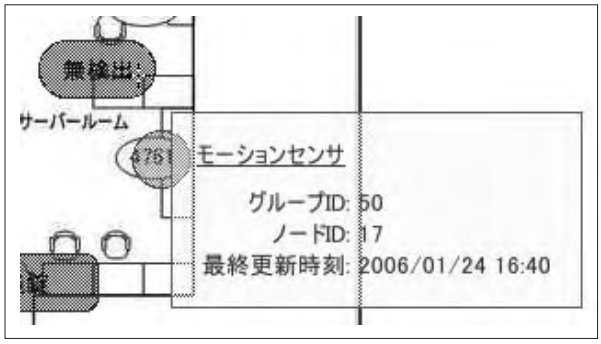


**図16　ポップアップ・ヒント**

また、プロットダイアログではマウスポインタのある点の値と時刻を表示する。

●ダイアログ

ブラウザ中に開く疑似ダイアログボックス。アクティブなダイアログはフレームの太さで区別することができる。

シェードボタンを押し下げすることによってタイトルバーだけ形にして、ダイアログの裏に隠れて表示されているものを見ることが可能である。

### 3.3.3　カメラ

本システムではカメラを設置することによって、モーションセンサーのデータと連動させ、人が部屋にいる間の画像を保存する機能を有している。カメラは高機能のネットワークカメラと安価なWebカメラの両方の設置実験を行った。高機能のネットワークカメラは、それ単体で動画を配信するサーバー機能を持っており、ファイルを他のサーバー等に転送する機能も持っている。一方、安価なWebカメラはPCにつないでPC上のアプリケーションを使用しないと機能しない。今回の実験ではWebカメラをLinuxの動いているPCに接続しffmpegというストリーミングサーバーソフトを用いて動画配信、画像の切り出しを行い、自作のRubyスクリプトによってアプリケーションサーバーへファイル転送を行った。

アプリケーション内ではカメラの情報はモーションセンサーの情報の1つとして表示される。カメラの情報を持つモーションセンサーのコンテキスト・メニューを開くと「動画中継」というメニューが選択可能になっている。このメニューはカメラ情報を持たないモーションセンサーでは選択不可になる。

このメニューをクリックすると図17のダイアログが開きライブ映像を見ることができる。システム稼動時、サーバー室には2台の高機能ネットワークカメラが設置されていたため、ダイアログ中に画面が分割されて2ヵ所の映像が表示される。


**図17　サーバー室ライブ画像**

技研実験スペースのカメラは安価なWebカメラであるため画像が粗いが人の判別は十分可能である（図18）。


**図18　技研実験スペースライブ画像**

モーションセンサーのコンテキスト・メニューから「検出履歴を表示」を表示を選択すると、検出されていた期間と検出された人の情報が表示される。

カメラの情報を持つモーションセンサーではこのダイア

ログの各行がクリック可能になっている（P.63図7）。この行をクリックすることによって、その行に表示されている期間の1秒ごとの連続写真をサムネイル表示するダイアログを開くことができる（P.63図8）。

さらにこのダイアログ中のそれぞれのサムネイル画像はそれぞれクリックで拡大表示される。

サーバー室に設置している高機能ネットワークカメラの画像はかなり鮮明であり、顔を隠すような行為をしない限り入室した人物の特定は容易である。

安価なWebカメラからの画像はサムネイル画像でみるとかなり粗く感じる（図19）が、これはサーバー室のサムネイル画像と同じ大きさの画像を拡大しているせいであり画質的には高機能ネットワークカメラのものとほとんどかわりがない。


**図19　技研実験スペースのスナップショットダイアログ**

しかし、拡大画像では安価なWebカメラの画像は高機能ネットワークの画像に比べてかなり画質が悪いように感じる。これは静止しているものは完全に判別可能であり、動いている人間がブレていることから考えると、解像度の問題ではなくレンズの明るさの問題であると考えられる。

これらの静止画像は1秒ごとにサーバーに送られて来るものを一定間隔でモーションセンサーデータと照合し、撮影されてから1日以上たっているものでセンサーに反応がない期間の画像を消すという作業を行っており、無駄な画像でストレージを圧迫することはない。

## 4．まとめ

我々は、このテストシステムの構築／運用によって以下の知見が得られたと考えている。

- ●代表的なセンサーネットワークデバイスの1つであるMoteの使用法
- ●Moteの消費電力の問題とその解決策
- ●Moteへの市販センサー/独自センサーの取り付けと取り扱い
- ●センサーネットワークのデータの保存/蓄積法
- ●これまでシステムでは扱うことがなかった大量データを扱った場合のDBMSの問題点
- ●大量データを扱う場合の一対処案
- ●入退室セキュリティにおける考慮点
- ●ネットワークカメラ、Webカメラの特性
- ●AjaxによるWebアプリケーションの構築法

今後、今回得られたこれらの知見をもとにさまざまなセンサーを用いた新たなシステムを構築する予定である。

また、解決できていない問題として、センサーノードをどのように人にとって目立たない存在するかというものがある。Moteなどの無線で通信する電池駆動のノードは、壁や天井に固定しなければならなかった今までのセンサーと違い、センシングしたい場所に置くだけで設置でき、非常に簡単に配付／配置することができる。これは裏をかえせば、電池駆動のノードは固定センサーのように壁や天井に埋め込んだり、カバーで目立たないように覆う作業に工夫が必要であることを意味する。なぜなら、目立たないように工夫していないセンサーネットワークのノードは人の目に奇異に映ってしまう可能性が高い。この事柄が実際にセンサーネットワークのシステムを人の近くに導入する妨げになってしまうかもしれない。これは今後の課題としてかなり重要度の高い問題の1つである。

さらに新たな試みとして、センサーネットワークを用いたシステムの中にData Stream Management System（DSMS）を導入する予定である。DSMSは広範囲に設置されたセンサーネットワークのデータのように、常に流れ続ける大量のデータを処理することに優れたシステムである。このため、うまく組み合わせることで、DBMSだけでは難しい処理（パフォーマンスの問題など）も効率よく実現することが期待できる。

## 〈参考文献〉

1. Paul Horowitz, Winfield Hill：*THE ART OF ELECTRONICS（Second Edition）* Cambridge University Press, 1989