第4章

ユーザによる制御が可能なセンサ/アクチュエータネットワーク

4.1 はじめに

半導体技術の進歩はパーソナルコンピュータの高性能化に留まらず，さまざまな分野に大きな影響を与えている．半導体技術によって培われたリソグラフィ技術はMEMSを生み，超小型・高感度・低消費電力の特徴を持ったセンサやアクチュエータの実現を可能とした．また，SoC（System-on-Chip）はマイクロコントローラと無線通信機能を1チップ化し，RFID技術[69]や無線センサネットワーク[14]などの誕生に寄与した．さらに半導体技術とMEMSを組み合わせることで，マイクロコントローラ，無線通信機能，センサ・アクチュエータを1つのチップ上に統合することも可能になりつつある．このような超小型のセンサやアクチュエータが，将来的にはさまざまなオブジェクトに組み込まれることが予想される．

センサやアクチュエータが組み込まれたオブジェクトはさまざまな情報の入力や出力が可能となる．本論文ではセンサを「環境からの情報を取得する機能」，アクチュエータを「環境に対してアクションを起こす機能」というように広義に捉える．例えば，センサは椅子に組み込まれた3軸加速度センサといった単なるセンサだけでなく，壁のスイッチなどの入力装置もセンサとして扱う．アクチュエータは，カーテンの開閉を制御するモータだけでなく，部屋の間接照明などの情報を出力可能な装置もアクチュエータとして扱う．このようなわれわれの身の回りに存在する膨大な数のセンサとアクチュエータを連携させることで今までにないような新しいサービスが誕生する可能性がある．

本研究の目的はこのようなセンサやアクチュエータが身の回りのありとあらゆるオブジェクトに組み込まれた空間において，一般のユーザが簡単な操作でこれらのオブジェクト同士を連携させ，サービスを構築することのできる仕組みを作ることである．センサとアクチュエータによってもたらされるサービスはセンサがイベントによる環境の変化を検出するという機能を持っているがゆえにイベントドリブンの特性を持つ．また，われわれの身の回りに存在するオブジェクトは膨大な種類が存在するため，オブジェクト同士の接続の柔軟さや新しいオブジェクトの開発の容易さが重要になってくる．

このような観点から，筆者らはイベントドリブンプログラミングの仕組みを単純化したBind Control Modelを基に構築したデバイス連携技術「ANTH」を提供する．通常のイベントドリブンプログラミングではイベントからコールバック関数に対してデータを渡すため，イベントとコールバック関数との関係が静的に決められる．それに対してBind Control Modelではすべてのコールバック関数をデータを受け取らない形に統一する．このような仕組みにより，呼び出し側のプログラムから呼び出される側のプログラムへの接続が柔軟になる．また，呼び出し側と呼び出され側のプログラムをそれぞれ独立に設計することができ，開発が容易になる．一方でBind Control Modelでは単純さゆえに実現できるサービスが固定

的になり，ユーザによる操作の負担が発生する．ANTH ではセンサノード上で動作する VM を用いてセンサやアクチュエータの機能を動的に追加する機構を導入することで実現できるサービスの幅を広げる．また，操作端末を利用したいオブジェクトに近付けて操作する直感的なヒューマンインタフェースを導入することでユーザによる操作の負担を軽減する．さらに，無線通信におけるアプリケーション層から MAC 層までを縦断的に考慮することで低消費電力な無線通信を実現する．

これらの機構を筆者らが開発した無線センサネットワークのテストベッドである PAVENET[56] 上に実装した．ANTH の最小セットはプログラムメモリ 3 kbyte，データメモリ 237 byte で実現することができる．つまり ANTH の最小セットは Microchip 社の PIC16 といったメモリ 256 byte，プログラムメモリ 4 kbyte の非常に計算資源の少ない CPU 上でも実装することができる．また，VM の命令セットを ANTH で扱うサービスに特化して構築することでネイティブで実行するときに比べて 1.24 倍の実行速度の低下に留めた．さらに，ANTH の無線通信プロトコルをシングルホップで構築し，かつ電源状況に応じて複数の MAC プロトコルを切り替えることでバッテリで動作するオブジェクトでは劣化を無視した場合に単三電池 2 本で約 3 年動作の低消費電力化を実現した．

本章では，まず 4.2 で ANTH のデザインとして，具体的なアプリケーションシナリオから必要とされる機能要件を抽出する．4.3 では，4.2 で示した機能要件に基づいて設計・実装した ANTH の個々の機構について述べる．4.4 では，ANTH を用いることでどのようなサービスが構築可能になるかを述べる．ついで 4.5 で ANTH の評価を多角的な面から行う．そして 4.6 で既存の研究との差別化を行い，最後に 4.7 でまとめとする．

4.2 要件

4.2.1 目的

本研究の目的は，ユーザが身の回りに存在するオブジェクト同士をできるだけ簡単に連携させるための仕組みを提供することである．本研究ではセンサやアクチュエータが身の回りの椅子や壁，コーヒーカップ，ペットボトル，目覚まし時計，テレビなど，ありとあらゆるオブジェクトに組み込まれた空間を想定している．このような空間では，各オブジェクトが具備したセンサから取得された情報とアクチュエータによる操作を組み合わせることでさまざまなサービスが実現されると考えられる．

センサによって取得されるデータはイベントドリブンの特性を持っている．センサネットワークの分野ではこのインベントドリブン性を生かしたプログラミング言語 [8] やオペレーティングシステム [9, 16, 70, 68]，ルーティングプロトコル [30, 32]，MAC プロトコル [41] などさまざまな研究がなされている．このようなセンサのイベントドリブンの特徴により，センサとアクチュエータの連携によって実現できるサービスの多くもイベントドリブンの特性を持つ．イベントドリブンのサービスでは，センサは環境の変化やユーザの操作を「ある事象が起きた」という情報に変換する．たとえば，目覚まし時計は時間情報を「アラーム」に変換し，スイッチはユーザの操作を「スイッチが押された」という情報に変換する．アクチュエータはセンサで取得された「ある事象が起きた」という情報に基いて環境に対してアクションを起こす．たとえば，目覚まし時計は「ベルを鳴らす」ことで環境に対してアクションを起こし，室内照明は「点滅する」ことで環境に対してアクションを起こす．

このようなイベントドリブンのサービスを前提とした筆者らの目指す世界を 4.2.2 に示す．

4.2.2 アプリケーションシナリオ

Bob は出張で東京に来ており，品川のホテルに予約をとっている．ホテルの部屋にあるほとんど全てのオブジェクトにセンサやアクチュエータが組み込まれている．Bob は部屋に案内されるとまず，携帯電話のメール受信機能とベッドの脇にある間接照明の色を結びつけることにした．照明が持つ色は「赤」「黄」「青」「白」などであった．Bob は未読メール数 0 を白，1〜5 を青，6〜10 を黄，11 以上を赤で通知することとした．

Bob は夕食をまだとっていなかったのでルームサービスを頼んだ．料理が来るまで Bob はヘッドホンで音楽を聴きながら読書をすることにした．ルームサービ

スが来た事に気付かないとまずいのでBobはドアの「ノックされた」という機能とヘッドホンの「ミュート」を結びつけることにした.

読書に熱中しているとヘッドホンの音楽が突然止まり，ドアをノックする音が聞こえた．ルームサービスが来たのだ．Bobはヘッドホンをはずして夕食を食べながらテレビのニュースを見始めた．一通りのニュースが終わった．Bobがふとベッドの脇にある間接照明に目を向けると照明が赤になっている．どうやらメールが溜まっているようだ．Bobは食事を食べ終わったらメールのチェックをすることにした.

Bobは携帯電話のメールの処理をし終わったのでベッドで横になりながら先ほどの読書の続きをすることにした．Bobは朝に弱いので，携帯電話のアラーム機能を部屋のライトの「ON」，テレビの電源の「ON」，部屋据え置きの目覚まし時計のベルの「ON」，カーテンの「開く」を結びつけた．これで朝寝過ごすことはないだろう．Bobはしばらく読書をしていた．すると猛烈な眠気を感じてきた．Bobは眠りに落ちる寸前にベッドの背もたれを軽く2回ノックした．すると部屋の全ての照明が落ち，Bobは暗闇の中で深い眠りに落ちた.

4.2.3 要件

本研究では4.2.2に示した世界を実現するためにヒューマンインタフェースから無線通信プロトコルまでの総合的な観点から必要となる機能の検討を行う.

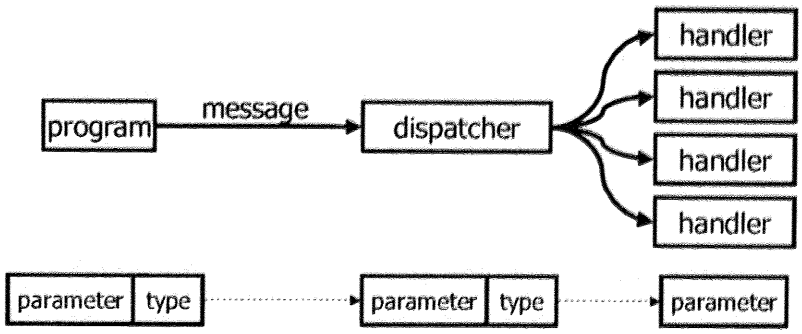

図 4.1: イベントドリブンプログラミング

4.2.2のアプリケーションシナリオから分かるように，本研究で対象としているのはヘッドホンやドア，壁などのわれわれの身の回りに存在する多種多様なオブジェクトである．このような多種多様という特徴を支えるためには，連携可能なオブジェクトが多ければ多いほどサービスの多様さが実現されるので，オブジェクト同士の接続の柔軟さが重要となる．加えて，これらのオブジェクトは機能も多

様であるので，オブジェクト自体を容易に開発できるようにする必要がある．このような観点から，筆者らはBind Control Modelと呼ばれるイベントドリブンプログラミングをより単純化したプログラミングモデルを用いる．

まず，一般的なイベントドリブンプログラミングを図4.1に示す．通常のイベントプログラミングでは送信側のプログラムがイベントの種類（type）とイベントのデータ（*parameter*）を含むメッセージを*dispather*に対して送信する．*dispatcher*はイベントの種類に応じて*handler*を選択し，*handler*に対して*parameter*を渡して実行する．つまり，既存のイベントドリブンプログラミングではメッセージの持つデータ形式と*handler*の引数の形式とが完全に一致している必要があるため，メッセージと*handler*との関係に強い依存性がある．

一方のBind Control Modelは既存のイベントドリブンプログラミングにおけるやりとりされるメッセージを*trigger*を表す識別子のみに限定し，*handler*に対して*parameter*を渡さない．つまり，Bind Control Modelでは*trigger*と*handler*を結び付けているものは識別子だけであるので*trigger*と*handler*の結合性が既存のイベントドリブンプログラミングに比べて弱い．そのため，*trigger*の識別子を変更するだけで簡単に*trigger*の配送先の*handler*を変更することができる．

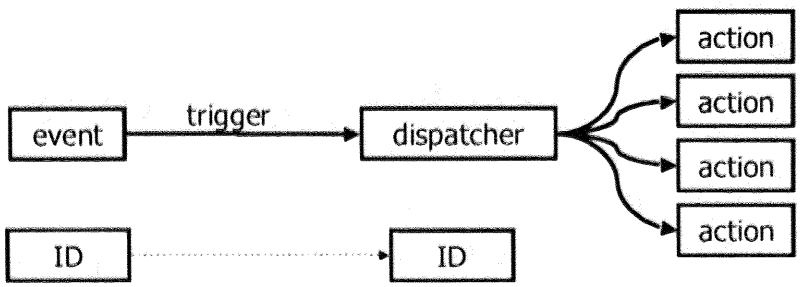

図4.2: Bind Control Model

図4.2にBind Control Modelを示す．Bind Control Modelは*event*と*action*から構成される．

*event*はオブジェクトの取得した環境の変化やユーザの操作を*trigger*に集約するための機構である．たとえばドアの持つ加速度情報の時系列データを「ノックされた」の*trigger*に集約したり，目覚まし時計の持つ時刻情報の時系列データを「アラーム」の*trigger*に集約したりする．

*action*は環境に対して出力を行う*event handler*として動作する．たとえば，間接照明の「激しく点滅させる」や，カーテンの「開く」，ヘッドホンの「ミュート」などが*action*に相当する．各*action*には識別子が割り当てられており，*trigger*を受け取った*dispather*は*trigger*の識別子と等しい識別子を具備する*action*を実行

する．

既存のイベントドリブンプログラミングではオブジェクト同士の接続性はメッセージの持つイベントの種類に強く依存していた．それに対して Bind Control Model では，*event* と *action* の接続が *trigger* に共通化されているため，全ての *event* と *action* は接続可能になる．つまり，Bind Control Model では接続の柔軟性が実現される．

また，既存のイベントドリブンプログラミングでは送信側のオブジェクトは受信側のオブジェクトがどのような種類（type）の *handler* を具備しているかを，*handler* は送信側のオブジェクトがどのようなデータ（parameter）を送ってくるかを考慮して開発する必要があった．それに対して Bind Control Model では，各 *event* には識別子が割り当てられており，*event* は *trigger* として識別子のみをネットワークに対して送出するため，受信側がどのような *handler* を持っているかを考慮せずに *event* を開発することができる．また，*action* は，*parameter* を受け取らないのでどのようなデータが送られて来るかを考慮して設計する必要がなく，一連の独立した処理の集合として記述されるので開発の容易性が実現される．

Bind Control Model の動作例としてドアの「ノックされた」という *event* とヘッドホンの「ミュート」という *action* を接続して「部屋のドアをノックされたらヘッドホンの音量をミュートにする」というサービスを実現する場合を考える．まず，ドアの *event* に対してある識別子「0x1234」を設定する．次にヘッドホンの *action* に対しても同じ識別子「0x1234」を設定する．ドアがノックされると，ドアの *event*「ノックされた」がネットワークに対して *trigger* として「0x1234」を送信する．ネットワークに送信された *trigger* は，「0x1234」を持つ *action* へと配送され，ヘッドホンの *action*「ミュート」が実行される．

このような Bind Control Model の単純さにより，接続の柔軟性と開発の容易性の高さのメリットがある一方で単純さゆえに考慮しなければならない点も存在する．

まず，Bind Control Model では実現されるサービスが *event* と *action* に強く依存するのでサービスの内容が固定的になりがちになる．これに対し，筆者らは，*event* や *action* をオブジェクトに対して動的に追加する仕組みを導入することで実現できるサービスの幅を広げる．また，このような単純なモデルでも少なくとも 4.2.1 に示したサービスは全て実現できる．なお，本研究で実現可能なサービスに関しては 4.4 で改めて議論する．

次に，Bind Control Model ではユーザの負担が増加する．Bind Control Model では *event* と *action* をユーザが自由に接続したり *event* や *action* を追加することでさまざまなサービスを作成することができる．しかしながらそのような自由度がある一方で，Bind Control Model ではユーザが明示的にどの *event* とどの *action* を接続するかを支持しなければならない．このような自由度とユーザの負担はト

レードオフの関係にあるが，Bind Control Model では *event* と *action* を簡単に接続したり追加したりするための使いやすいヒューマンインタフェースが必要となる．

さらに，本研究ではバッテリで動作するようなオブジェクトをも対称としているため，実際の無線通信の消費電力を削減しなければならない．表 4.1 に MICA Mote[7] や Particle Computer[71] などのセンサノードで頻繁に用いられている無線モジュールである CC1000 の消費電力を示す．無線通信ではパケットを送信しているとき，受信しているとき以外の受信待機時にも電力を消費する．つまり，無線通信における消費電力を下げようとする場合にはいかにして無線のスリープ状態を増やすかにかかわる MAC プロトコルが重要になる．

RF の状態	消費電流
受信	7.4 mA
受信待機	7.4 mA
送信 (10 mW 時)	10.4 mA
スリープ	0.2 μ A

表 4.1: CC1000 の消費電力

以上の議論より，本研究では Bind Control Model を中心に

- オンデマンドな機能の追加が可能なミドルウェア
- 使いやすいヒューマンインタフェース
- 低消費電力な無線通信プロトコル

を同時に満たすようなデバイス連携技術を実現を目指す．

4.3 ANTH

筆者らは，4.2での議論を踏まえてデバイス連携技術「ANTH」の設計と実装を行った．ANTHの全体像を図4.3に示す．ANTHは低消費電力性と省資源性を実現しつつも多様なサービスを提供するために，ヒューマンインタフェースから無線通信プロトコルまでを統合的に構築する．具体的にはBind Control Modelを中心に，オンデマンドな機能の追加が可能なミドルウェア，使いやすいヒューマンインタフェース，低消費電力な無線通信プロトコルの3つを提供する．

実装はPAVENETモジュール[72]上で行った．計算資源としてはマイコンであるPIC18LF4620[65]を用い，通信資源には315 MHz帯の微弱無線であるCC1000[63]を用いている．

オペレーティングシステムには筆者らが開発したPAVENET OS[56]を使用した．PAVENET OSは無線センサノード用のハードリアルタイムOSであり，地震モニタリング[49][11]やコンテクストアウェアなコンテンツ配信システム[54]などのセンサネットワークのアプリケーションなどの研究にも使用されている．プログラミング言語は無線通信プロトコルと基盤システムに関してはANSI C準拠のHI-TECH Software社のPICC-18コンパイラを用いた．

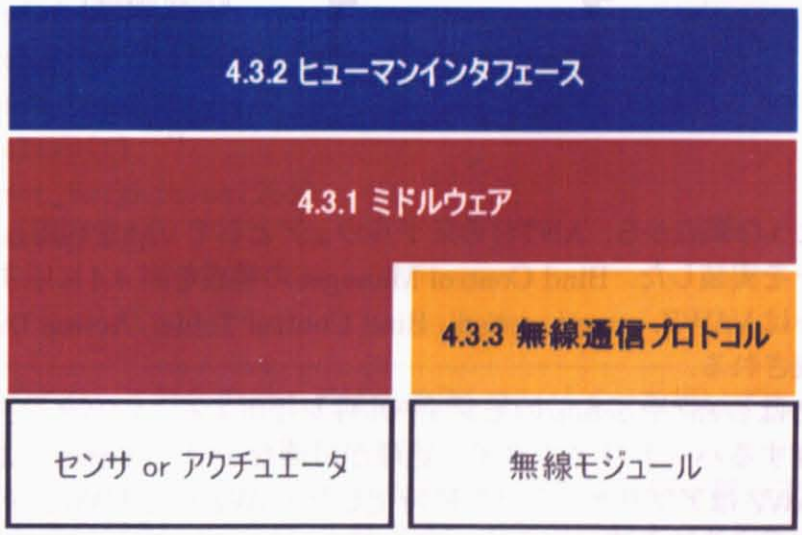

図 4.3: ANTH

4.3.1 ミドルウェア

ANTH ではサービスの多様性を実現するためにオブジェクトの *event* や *action* を動的に追加可能なソフトウェアモジュールとして実現する．*event* や *action* はオブジェクトのデバイスドライバ的な動作をするのでハードリアルタイム性が必要となるような状況も想定される．また，ソフトウェアモジュールをオブジェクトに追加する際に，ソフトウェアモジュールのサイズが大きい場合には転送に時間がかかり，電力を多く消費する原因になる．ソフトウェアモジュールを保存する場合にも計算資源の制限からできるだけソフトウェアモジュールは小さいほうが望ましい．さらに，ソフトウェアモジュールにバグがあった場合でもシステム全体に影響を与えないような仕組みが必要となる．

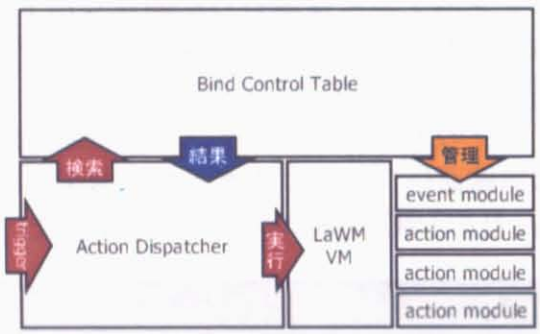

図 4.4: Bind Control Manager

このような観点から，ANTH のミドルウェアとして VM を利用した Bind Control Manager を実装した．Bind Control Manager の構成を図 4.4 に示す．Bind Control Manager は VAWZ，*event*, *action*，Bind Control Table，Action Dispatcher の 5 つから構成される．

VAWZ は *event* や *action* のモジュール性を提供する．VAWZ は，PAVENET OS 上で動作するハードリアルタイム処理が可能な VM （Virtual Machine）である [57]．VAWZ はアプリケーションに特化した VAWZ-C と VAWZ をシステム作成時に自動生成することでソフトウェアモジュールサイズの削減しつつも処理性能の低下を抑えることができる．さらに，メモリ保護機能を持たない CPU 上で安全にソフトウェアモジュールを実行することができる．なお，ANTH では，VAWZ と VAWZ-C を ANTH と各オブジェクトの機能に特化して実装した．具体的には，*event* を送信する命令やセンサから値を取得する命令，アクチュエータを操作するための命令などを VAWZ 上の機械語の 1 命令として実現した．

event, *action* は VAWZ 用に作られた言語「VAWZ-C」を用いて開発する．VAWZ-C は C 言語に似たセマンティクスを持っている．目覚まし時計の「アラーム」の

*event*は

```
void main(){
    id1 = add_event("アラーム");
    while(1){
        if(state == 0){
            sig_wait();
        }else{
            anth_emit_event(id1);
            state = 0;
        }
    }
}
```

のように記述する．ライトを1秒毎に点滅させる*action*は

```
name = "点滅させる";

void main(){
    while(1){
        set_brightness(0);
        sleep(1);
        set_brightness(255);
        sleep(1);
   }
}
```

のように記述する．これらのコードはLaWMコンパイラを用いてLaWM VMで実行するための機械語へと変換される．

Bind Control Tableは Bind Control Managerの中核を担う部分であり，*event*や*action*の管理を行う．Bind Control Tableの各項目を表4.2に示す．*function descriptor*はノード内で*event*や*action*を一意に識別するための固有値であり，*event*や*action*を追加するときにシステムが割り当てる．*function identifier*はネットワーク内の*event*や*action*の接続を示すための識別子であり，4.3.2で述べるヒューマンインタフェースによって自由に書き換えることができる．*function name*は*event*や*action*の名前であり，ユーザがオブジェクトの機能を操作端末に表示する時に用いられる．*function type*は機能の属性であり，ソフトウェアモジュールが*event*か*action*を区別するものである．*entry point*は*event*や*action*の実行開始場所で

あり，モジュールが保存されている領域の開始アドレスになる．

function descriptor	function identifier	function name	function type	entry point
1	0xA234	アラーム	event	0x4000
2	0x3B56	ベル（1秒）	action	0x4100
3	0x98C6	ベル（5分）	action	0x4200
4	0x543D	ベル停止	action	0x4300

表 4.2: Bind Control Table

Action Dispathcer は受け取った *trigger* に応じて *action* を実行する機構である．Action Dispatcher が *trigger* を受信すると，*trigger* が持つ識別子を鍵として Bind Control Table を検索する．*trigger* の持つ識別子と等しい識別子を持つ *action* が Bind Control Table に存在した場合にはその *action* を LaWM VM 上で実行する．見つからなかった場合にはその *trigger* は破棄される．

ANTH では *event* や *action* の作成はオブジェクトの開発者や，ボランティアのユーザなどが行い，インターネット上でさまざまな *event* や *action* が公開されることを想定している．ユーザはインターネット接続可能な携帯電話や PDA を用いて自分の好みの *event* や *action* をインターネット上からダウンロードする．そして 4.3.2 で述べるヒューマンインタフェースの仕組みにより，携帯電話や PDA を *event* や *action* を追加したいオブジェクトに近付けてソフトウェアモジュールをロードする．オブジェクトにロードされた *event* や *action* は Bind Control Table に対して追加され，LaWM VM 上で実行される．

4.3.2 ヒューマンインタフェース

ANTH は，身の回りのありとあらゆるオブジェクト同士を連携させることを想定しているため，対象となるオブジェクトの数が膨大になる．そのため，ユーザがどのオブジェクトのどの機能と，どのオブジェクトのどの機能を連携させるかを効率よく指定する仕組みが重要となる．

オブジェクト同士を連携させるためのヒューマンインタフェースとしては [73] や [74] などのようにビジュアルプログラミングの手法を用いたものが存在する．これらの手法では PC の GUI 上にオブジェクトのアイコンを表示し，ユーザがマウスを用いて「これとこれを接続する」のように指定する．この操作では，本来であれば「目の前にある」目覚まし時計を GUI 上の目覚まし時計のアイコンに対応させるという行為が直感的な要求の伝達の妨げになる．さらに，このような GUI では身の回りに存在するオブジェクトの数が増えれば増えるほど画面上のオブジェ

クトも増加し，ユーザに対して混乱を与えてしまう．特に本研究では部屋にあるありとあらゆるオブジェクトを対象としており，オブジェクトの数が多くなるため，この問題は顕著になる．

それに対して，ANTH では，Proximity-based User Interface という近接情報を利用したヒューマンインタフェースを提供する．Proximity-based User Interface では，連携するオブジェクトと連携を指定する操作端末を分離し，操作端末をオブジェクトに近付けて操作を行うことで連携に使用するオブジェクトの指定を直感的かつ効率的にする．

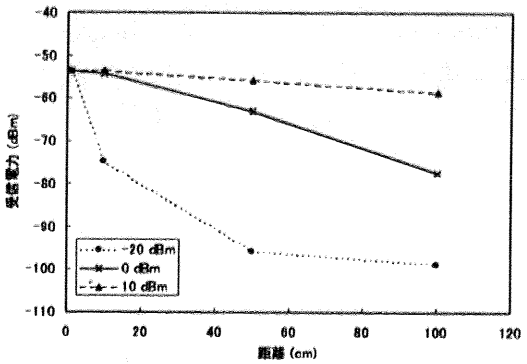

図 4.5: 距離に応じた受信電力

ANTH の Proximity-based User Interface では近接情報の検出にオブジェクト同士の通信に使用する無線の電波強度を利用することで，特別な機構を新たに用意することなくヒューマンインタフェースを実現することができる．電波を利用した位置の検出は数多く研究されている [75, 76, 77]．これらの研究では，電波による位置検出の精度は数 m である．しかしながら，送信電力を状況によって弱くすることで受信電力を基準に「近接しているか」「近接していないか」の 2 択の検出は比較的容易に行う事ができる．図 4.5 に送信電力を-20 dBm，0 dBm，10 dBm と変化させたときの距離と受信電力の関係を示す．実験では Chipcon 社の無線モジュール，CC1000 を用いた．図 4.5 から明らかなように，送信電力が弱いときほど距離による受信電力の減衰が激しい．このような電波の特性を利用し，操作端末からオブジェクトに対する通信開始を意味するパケットを送る時のみ送信電力を弱くし，さらにオブジェクト側で受け取ったパケットの受信電力をある任意の閾値で通信対象を制限することで至近距離にあるオブジェクトのみと通信を行う．なお，筆者らは操作端末とオブジェクトの距離が 10 cm 以内であるときにオブジェクトに対して操作可能なように実装した．実装は PDA 上で C#.NET Compact Framework を用いて図 4.6 のような GUI を作成した．

PDA と ANTH 間でやりとりされるコマンドを表 4.3 に示す．Proximity-based User Interface では，PDA や携帯電話などの操作端末を連携するオブジェクトに近付け，検索ボタンを押す．すると PDA から *search* コマンドが送信される．*search* コマンドを受け取ったオブジェクトは，受信電力がある閾値以上だった場合に *hostid* を返信する．PDA は取得した *hostid* を用いて ANTH に対して *get_hostname* コマンドを送信し，オブジェクトのデバイス名を取得する．次に PDA は *hostid* を用いてオブジェクトに対して *list* コマンドを送信する．*list* コマンドを受け取ったオブジェクトは自分の持っている *event* や *action* の *function descriptor* のリストを返信する．PDA は受け取った各 *function descriptor* に対して *get_funcname* コマンドを送信し，*event* や *action* の名前を取得し，画面に表示する．ユーザは PDA を操作して利用したい機能を選択することで，識別子の設定（*set* コマンド）を行う．また，操作端末をインターネットに接続し，新しい *event* や *action* をダウンロード

して追加（*add*）することもできる．追加にはファイルを指定して行われる．

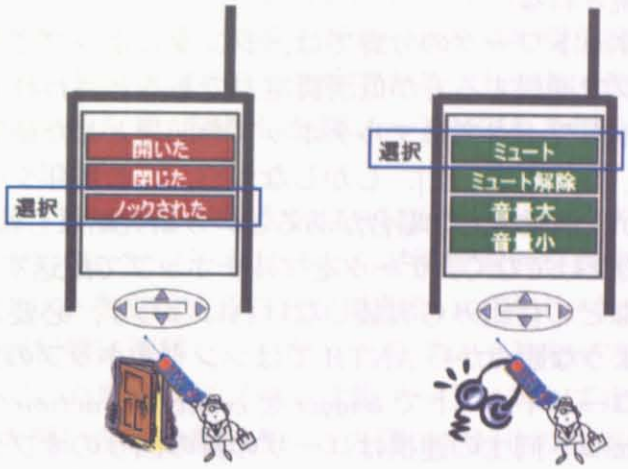

図 4.7: Proximity-based User Interface

動作例として図 4.7に「ドアをノックされたらヘッドホンの音量をミュートにする」というサービスを実現する場合を示す．まず，ユーザはドアに携帯電話を近付ける．すると携帯電話にドアの持っている機能がリストアップされる．ユーザはその中から「ノックされた」を選択する．次にユーザは携帯電話をヘッドホンの近くに持っていく．今度は携帯電話にヘッドホンの持っている機能がリストアップされる．ユーザはリストの中から「ミュート」を選択する．以上の操作によりドアの「ノックされた」と「ミュート」が関連付けられサービスが構築される．

4.3.3 無線通信プロトコル

ANTH では Bind Control Model によりオブジェクト同士の連携は 4.2.3 に示したように *event* から *action* へ *trigger* を送信することで実現される．そのため，無線通信プロトコルでは，*event* の送信した識別子（*trigger*）を同じ識別子を具備する *action* へ配送する仕組みが必要となる．また，ANTH では，身の回りのオブジェクト同士を連携させることを目的としている．つまり，基本的には家の中のオブジェクト同士の通信であり，通信距離は最大でも数十メートルを想定している．さらに，ANTH では室内照明や家電など家庭用電源で動作するオブジェクトや，ぬいぐるみやコップなど電源供給機能を持たないオブジェクトを対象にしている．電源供給機能を持たないオブジェクトに組み込むセンサやアクチュエータはバッテリで動作するため，これらのオブジェクトにおける無線通信の低消費電力化が必須である．無線通信の低消費電力化としては，アプリケーション層やネッ

トワーク層における通信量，MAC層における無線資源利用のスケジューリングなどによって実現される．

無線センサネットワークの分野では，シングルホップでデータを配送するよりもマルチホップで通信する方が低消費電力であると言われて来た [78]．そのため，無線センサネットワークではマルチホップを前提とした研究が多くなされている [52, 79, 45, 43, 33, 20, 80, 31]．しかしながら，ここ数年シングルホップでデータを配送した方が低消費電力な場合があるという研究結果も報告されている [47, 48]．また，消費電力だけでなく，データをマルチホップで配送する場合にはルーティングプロトコルなどの仕組みも実装しなければならず，必要とされる計算資源が増加する．このような観点から ANTH ではシングルホップの無線ネットワークを構築し，無線ブロードキャストで *trigger* を *event* から *action* へ配送する．ANTH におけるオブジェクト同士の連携はユーザの身の回りのオブジェクト同士の連携であるので長くても数十メートルの到達距離があればよいのでシングルホップの無線通信でも十分に対応可能である．

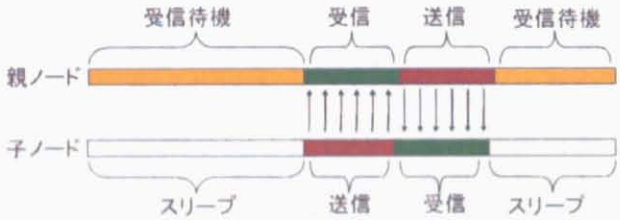

図 4.8: ANTH MAC

さらに，ANTH では無線通信におけるさらなる低消費電力化を実現するために MAC プロトコルとしてイベントドリブンに特化した A-MAC（ANTH MAC）を提供する．ANTH では室内のオブジェクト同士の連携を想定しているため，家庭用電源で動作するオブジェクトやバッテリで動作するオブジェクトなどさまざまな電源環境を持ったオブジェクトが存在すると想定できる．A-MAC は各オブジェクトの電源環境に応じて複数の MAC プロトコルを使い分けることでバッテリ駆動のオブジェクトの消費電力を削減する．

A-MAC では，家庭用電源で動作する全てのノードは親ノードになる．一方でバッテリ動作するノードは，まず，隣接する親ノードがいるかどうかの確認を行う．もし親ノードがいた場合，バッテリ動作するノードはそのノードを親ノードとした子ノードになる．もし親ノードがいない場合，そのノードは流浪ノードとなる．

親ノードは常に無線モジュールを受信待機状態にしているため，いつでもパケッ

トを受信することができる．また，親ノードに子ノードの登録があった場合，子ノードからの問い合わせがあるまで受信した *trigger* パケットを保存しておく．

流浪ノードも親ノードと同様に無線モジュールを常に受信待機状態にしているため，いつでもパケットを受信することができる．しかし，流浪ノードは親ノードと違って子ノードを持たず，隣接した親ノードが現れた場合には直ちにその親ノードの子ノードになる．

子ノードの無線モジュールの状態を図 4.8 に示す．子ノードは通常時はスリープ状態にある．すなわち，子ノードはほとんど消費電力を消費しないかわりに他のノードからのパケットを受信することができない．子ノードは，*event* が発生したときは直ちに送信状態に遷移し，パケットをブロードキャストする．子ノードが *action* を具備していた場合，子ノードは親ノードに対して定期的にパケットが到着していないかの問い合わせを行う．この問い合わせの間隔は *action* の種類によって異なる．たとえば,「任意の時刻になったらカーテンを開ける」といったサービスを考えた場合,「カーテンを開ける」という *action* はリアルタイム性は重視されないので問い合わせ間隔が長くてもサービスのクオリティとしては低下しない．もし親ノードがいなくなった場合には直ちに流浪ノードへと遷移する．

以上のような仕組みにより，子ノードは問い合わせ時以外は無線モジュールをスリープ状態にするので消費電力を大幅に削減することができる．たとえば CC1000 を具備した子ノードが 5 秒おきに親ノードに対して問い合わせをしている場合，平均 40 μ A の消費電流になる．A-MAC を用いないノードでは平均 13 mA 程度であるので A-MAC を用いたノードでは約 300 倍近く消費電力を削減することができる．単 3 電池 2 本を用いたセンサノードの場合，電池の劣化を無視すれば約 3 年の駆動が可能となる．

ANTH の無線通信プロトコルを PAVENET OS の無線プロトコルスタックを用いて実装した．ANTH のパケットフォーマットを図 4.9 に示す．ANTH のパケットは MAC プロトコルヘッダ 6 byte，アプリケーションプロトコルヘッダ 2 byte ペイロード 6 byte の計 14 byte から構成される．プロトコルタイプは，MAC 層のパケットの種類を表す．ANTH では，ユーザがサービスを構築し，実際にサービスが実行されるまでの各段階によって MAC プロトコルに対する要件が異なるそのため ANTH では複数の MAC プロトコルを使い分けている．

プロトコルタイプには BORADCAST，SEARCH，RET，AMAC の 4 種類が存在する．BROADCAST は *trigger* の配送に使用される．同じシーケンス番号を持つ *trigger* を複数回送信することで *trigger* の配送の信頼性を高めている．SEARCH は近接するオブジェクトの検索に使用する．SEARCH を受け取ったオブジェクトは受信電力をチェックし，再送制御付きの MAC プロトコルで返信する．RET は再送制御付きの CSMA の MAC である．再送制御付きの CSMA ではまずキャリアセ

ンスし，キャリアがビジーでなかった場合にパケットを送信する．そして一定時間 ACK を待つ．ACK が帰ってこなかった場合は最大3回まで再送を行う．AMAC も RET と同様に再送制御付きの CSMA であるが，*trigger* の問い合わせのみに使用される．AMAC ではまず，子ノードから親ノードに送るときにはキャリアセンスし，キャリアがビジーでなかった場合にランダムバックオフをしてパケットを送信する．パケットを受け取った親ノードは ACK も兼ねてキャッシュしてある *trigger* を返信する．

パケットタイプは再送制御付き MAC の時の通常のデータパケットと ACK パケットの区別や，AMAC の親ノードか子ノードか，流浪ノードの区別をするために使用する．宛先アドレスはパケットの送信先のアドレスであり，ソースアドレスはパケットの送信元のアドレスである．宛先アドレスとソースアドレスとシーケンス番号のセットで重複して受け取ったパケットの削除を行っている．CRC は誤り検出を行うための領域である．

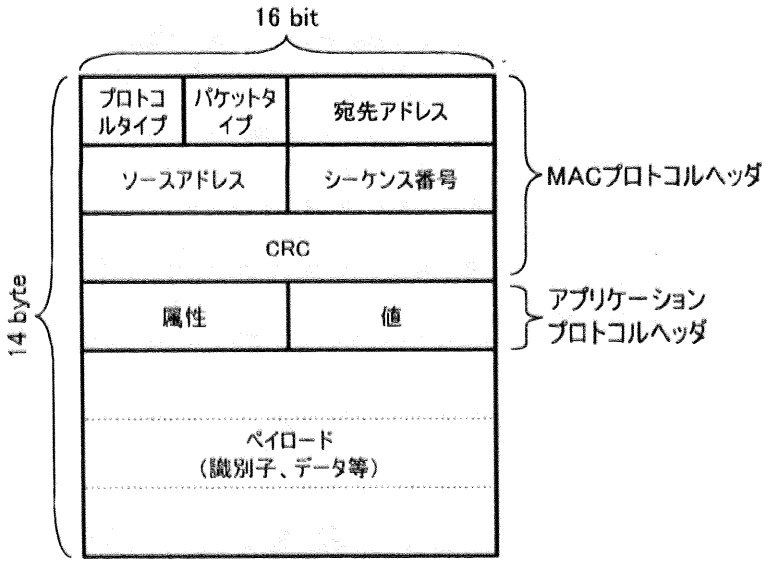

図 4.9: パケットフォーマット

アプリケーションヘッダは *trigger* 用のパケットなのかヒューマンインタフェース用のパケットなのかを区別するための領域である．ペイロードには *trigger* 用のパケットの場合には識別子が，ヒューマンインタフェース用のパケットの場合には *filename* などが入っている．

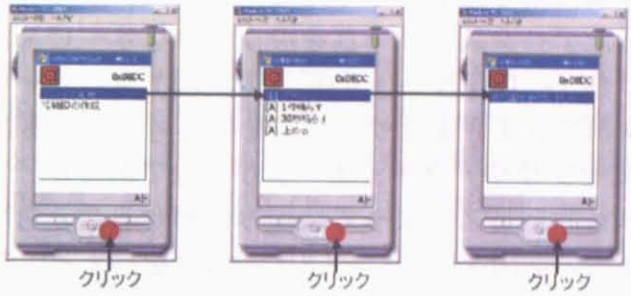

図 4.6: 操作端末のスクリーンショット

コマンド名	意味
search	至近距離にあるオブジェクトの hostid を取得
get_hostname	オブジェクトの名前を取得
list	fd のリストを取得
get [*fd*]	*fd* に関連付けられた fid を取得
set [*fd*] [*fid*]	*fd* に *fid* を設定
exec [*fid*]	*fid* に関連付けられた *action* を実行
get_funcname [*fid*]	*fid* に関連付けられた *fname* の取得
add [*filename*]	*filename* をオブジェクトに追加する

fd = function descriptor
fid = function identifier
fname = function name

表 4.3: コマンドリスト

4.4 アプリケーション

ANTHは「IF ～ THEN ～」の形式であらわすことのできるサービスを対象としたデバイス連携技術である．そのため，パソコンからプリンタに対して論文を出力するといったデータストリームを扱うようなサービスとは相補的な技術となる．ここでは，「IF ～ THEN ～」の形式で実現できるサービスを列挙し，ANTHはアイディア次第でさまざまなサービスを構築可能であることを示す．

ANTHは主にセンサやアクチュエータが対象であるが，プリンタなどの機能の豊富なデバイスもサービスによっては組み合わせることもできる．まず，プリンタに対して「建物の地図を印刷」という *action* を与える．このとき *action* には建物の地図のデータが含まれている．次に自動ドアに「ドアが開いた」という *event* を与える．そして「ドアが開いた」という *event* と「建物の地図を印刷」という *action* を結びつけることで「人が来たら地図を印刷する」といったオンデマンド地図印刷サービスを作ることができる．

ANTHを用いると，日曜大工的な感覚でセキュリティシステムを構築することもできる．まず，庭に人感センサをいくつか設置し，人感センサに「反応した」という *event* を与える．次にユーザのベッドの脇にある目覚まし時計の *action* である「ベルを鳴らす」と庭に設置してある照明の *action* である「10秒間点灯する」を結びつける．次にテレビゲームのコントローラの *event* である「右」「左」と庭に設置してある首振りカメラの「右」「左」を結びつける．これにより，庭に誰かが侵入したら目覚まし時計のベルがなると共に庭の照明をつけ，ゲームコントローラで首振りカメラを制御して侵入者がいないかを確認するといったインスタントセキュリティシステムを構築することができる．

ANTHを用いると郵便受けのチェックも簡単になる．まず，郵便受けに重さセンサを組み込み，*event* として「0グラム」「0～50グラム」「51～100グラム」「101グラム以上」を与える．次にリビングの間接照明の *action* 「暗い」「やや暗い」「やや明るい」「明るい」をそれぞれの *event* と結びつける．これにより，ユーザは郵便受けまで見に行かなくても新聞や手紙が届いているかどうかをリビングの間接照明を見るだけで知ることができる．

ANTHを用いるとエアコンを温度だけでなく別の基準で制御することもできる．たとえば臭いセンサの *event* である「無臭」「やや臭い」「臭い」「猛烈に臭い」とエアコンの「OFF」「設定温度22度・微風」「設定温度20度・中風」「設定温度18度・強風」を結びつけることで，部屋に人が充満して汗をかいて部屋が臭くなってくるとエアコンのスイッチを入れるといったサービスを構築することができる．

4.5 評価

4.5.1 プログラムサイズ

表 4.4 に ANTH のプログラムサイズを示す．データはデータメモリのサイズであり，プログラムはプログラムメモリのサイズである．PAVENET OS は，ハードリアルタイム処理を提供するためのタスクスケジューラのみのコードサイズである．Wireless Protocol Stack は PAVENET OS が提供する無線プロトコルスタックのコードサイズである．PAVENET OS では，物理層，MAC 層，ネットワーク層，トランスポート層をレイヤリング化し，各層のプロトコルを柔軟に入れ替えることのできる仕組みを提供している．Debug Library は PAVENET OS 上でアプリケーションを開発するときに使用するデバッグ用のライブラリである．LaWM VM は *event* や *action* のモジュール性を提供する．ANTH は，ANTH のコア部分であり，ヒューマンインタフェースの機能やオブジェクト同士を連携させるための機能が実装されている．ANTH MAC は ANTH の MAC プロトコルである．

表 4.4 から分かるように，ANTH のコア部分はデータメモリ 237 byte，プログラムメモリ 2968 byte と非常に少ない計算資源で実現できる．ANTH では，ハードリアルタイム処理を提供する PAVENET OS の機能や，*event* や *action* のモジュール性を提供する機能は必須では無くオプションとして扱うことができるように設計されている．つまり，ハードリアルタイム処理が不要で *event* や *action* のモジュール性が無くても大丈夫なオブジェクトの場合には ANTH のコア部分を実装するだけで他のオブジェクトと連携ができる．そのため，例えばデータメモリが 352 byte，プログラムメモリ 14 kbyte で 8 bit の命令長，スリープ時 100 nA 以下，2.0 V で 1 MHz での動作時に 100 uA の Microchip 社の PIC16LF917[81] などの単純な CPU 上でも実装することができる．

モジュール	データ（byte）	プログラム（byte）
PAVENET OS	47	1536
Wireless Protocol Stack	628	930
Debug Library	955	40873
VAWZ	1698	287
ANTH	237	2968
ANTH MAC	18	2059
合計	1935	48702

表 4.4: プログラムサイズ

4.5.2 VMを用いた場合によるオーバヘッド

ANTHでは，バッテリで動作するオブジェクトはオブジェクトの電源寿命を最大限に延ばすために通常はCPUをスリープ状態にし，定期的にCPUを起こすといった動作をする．このような動作において，*event*や*action*をVM上で実行することによって著しく消費電力が多くなることは望ましくない．このような観点から，筆者らは*event*や*action*をネイティブで実装したときとVMで実装したときのCPUのアクティブな時間の計測を行った．

表4.5にネイティブで実行した場合とVM上で実行した場合の*event*と*action*のCPUのアクティブな時間を示す．*event*は，ソフトウェアモジュールが*trigger*を発行してから実際にオブジェクトがパケットを送信し終わってスリープするまでの処理時間である．*action*は，オブジェクトがパケットを受け取ってからLEDを点灯させる*action*を実行した後にスリープするまでの処理時間である．

4.5から分かるように*event*や*action*をネイティブで実行した場合とVM上で実行した場合には最大で1.24倍の処理時間の増加になる．LaWM VM上で実装した*event*や*action*の方がネイティブに比べて遅い．この処理時間の差はVMで実行する時にVMのバイトコードをネイティブの機械語に変換しながら実行する時のオーバヘッドである．

モジュール名	ネイティブ (ms)	VM (ms)	性能比
event	5.70	5.90	1.04
action	2.51	3.10	1.24

表 4.5: *event*と*action*の処理性能

4.5.3 無線通信プロトコルの性能評価

ANTHの無線通信プロトコルの評価を行った．まず，全てのノードが家庭用電源につながっている場合に，ノード数と*event*が*trigger*を発行する間隔に応じてパケット到達率がどのように変化するかの確認を行った．*event*と*action*の数はネットワーク内で50%づつにした．図4.10にパケット到達率のグラフを示す．Sは1つの*trigger*に対してMAC層でパケットを何回送っているかを表す．ANTHでは，*trigger*の到達確率を上げるために1つの*trigger*に対してパケットを多重化して送っている．Tは，*event*がパケットを送信する間隔の平均の秒数である．パケットの送信間隔の平均が1秒 (T = 1) で，1つの*trigger*に対して1つのパケットを投げている場合 (S = 1) にはノード数に応じてパケット到達率が急激に低下

している．これはノード数が増えた時に *trigger* が衝突することに起因する．一方でパケットの送信間隔の平均が 1 秒 (T = 1) の場合でも 1 つの *trigger* に対して 3 つのパケットを投げている場合 (S = 3) にはノード数が 32 ノードの時でもパケット到達率をほぼ 100%に保っている．ANTH では，1 つの空間内のノード数は数十〜数百ノードを想定しているが，各 *event* は頻繁に *trigger* を発行するようなオブジェクトは対象としていない．例えば目覚まし時計は 1 日に数回 *trigger* を発行する程度であり，壁のスイッチは多くても一日に数十回 *trigger* を発行する程度である．つまり，図 4.10 の結果におけるパケット到着率は十分であると言える．

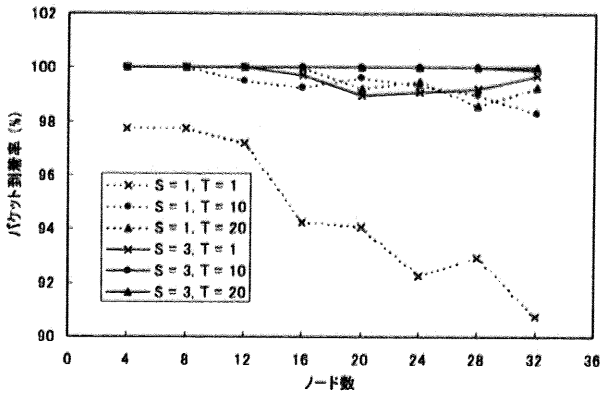

図 4.10: ノード数に応じたパケット到達率

次に，全ノード数を 32 ノードに固定し，バッテリ駆動のノード数の比率を変えたときのパケット到着率を図 4.11 に示す．また，家庭用電源に接続されているノードは *trigger*8 個を保存可能なバッファを持っている．T は，*event* がパケットを送る間隔であり，M はバッテリ駆動のノードが親ノードに対して問い合わせをする間隔である．

まず，バッテリ駆動のノードの比率が増加すると，バッテリ駆動のノードが親にパケットの問い合わせをするトラフィックが増加するのでパケット到達率が低くなる．また，バッテリ駆動のノードが親ノードに問い合わせをする間隔 (M) が短くなった時も問い合わせのトラフィックが増加するのでパケット到達率が低くなる．

さらにグラフでは，T の値と M の値が近いときに p ケット到達率が低くなっていることが分かる．これは親ノードが具備している *trigger* を保存するバッファが溢れていることに起因する．つまり，パケットの送信間隔が非常に短いようなオブジェクトが多数存在するようなオブジェクトを想定する場合には親ノードの *trigger* 保存用バッファを多めに用意しておくか，効率的にバッファを利用するための仕組みを用意しなければならない．しかしながら，先ほども述べたように，ANTH

では各 *event* は頻繁に *trigger* を発行するようなオブジェクトは対象としていないので *trigger* 保存用のバッファはそれほど多くなくても大丈夫である.

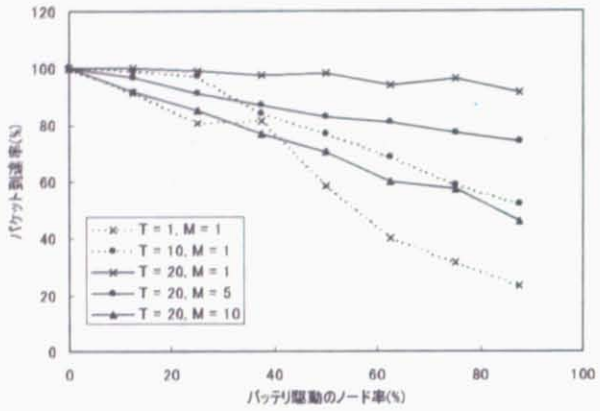

図 4.11: バッテリ駆動のノード数に応じたパケット到達率

4.5.4 アプリケーション

図 4.12 に ANTH を組み込んだオブジェクトを示す. ぬいぐるみ, 携帯音楽プレーヤ, 目覚まし時計などを実装した. これらのオブジェクトを利用してぬいぐるみを振ると音楽を再生するといったぬいぐるみを携帯音楽プレーヤの操作端末として使うようなサービスを構築することができる.

図 4.12: ANTH

4.6 関連研究

Smart-Its[62] は，身の回りのオブジェクトに組み込むためのコンピュータのプラットフォームを提供することを目的としている．Smart-Its は無線通信機能を持っており，オプションとしてセンサボードを外部接続可能なように設計されている．それに対して ANTH は，身の回りのオブジェクトにコンピュータが組み込まれているという前提で，これらのオブジェクト同士を連携させる仕組みを提供している点が Smart-Its と異なる．

本研究と最も近い研究として Ubiquitous Chip が挙げられる [82]．Ubiquitous Chip は身の回りのオブジェクトにコンピュータが組み込まれた環境において，コンピュータ同士を連携させることでさまざまなサービスを構築することを目的としている．Ubiquitous Chip は，Active Database[83] で用いられている ECA ルールを入出力を扱える形に拡張したものを用いることでオブジェクトの機能を単純な ECA ルールで記述可能にしている．しかしながら，Ubiquitous Chip は ECA ルールを省資源なデバイスで実現する手法について主眼が置かれており，デバイス同士の連携時における通信の方式，低消費電力化，ヒューマンインタフェースは提供していない．特に Ubiquitous Chip はデバイス同士の接続は有線であり，無線通信により接続の柔軟性を実現しつつも低消費電力性を提供する ANTH とは異なる．

デバイス連携という観点からは，UPnP[84]，Jini[85]，Salutation[86]，STONE[87]，Dragon[73]，ECHONET[88] など，さまざまな技術が存在する．これらの技術は多様なサービスに対応可能なように機能を抽象化して規格化されている．そのため，実際にソフトウェアをどういう風に設計し，実装するかということは開発者に任せている．それに対して ANTH では低消費電力性や省資源性を目指して機能を具体化し，ヒューマンインタフェースから無線通信プロトコルまでを定めることで，低消費電力性や省資源性を実現している．つまり ANTH は，UPnP などの既存の技術ほど多様なサービスは提供できないものの「IF ～ THEN ～」で実現可能なサービスを低消費電力，省資源で実現可能であるので既存の技術とは相補的であると言える．

無線センサノードにおけるソフトウェアのモジュール化という観点からは VM*[89]，SOS[68]，Mate[90] などさまざまな技術が存在する．ANTH では実装を PAVENET OS 上で行ったため，PAVENET OS 上で動作する LaWM VM を用いた．VM* や SOS，Mate がリアルタイム処理をサポートしていないのに対し，LaWM VM は PAVENET OS の機能を利用してハードリアルタイム処理を実行可能である．

デバイス同士を連携させるためのヒューマンインタフェースとしては Dragon[73] や Carp@[74]，VoodooIO[91] などが存在する．Dragon や Carp@は PC 上の GUI を用いたビジュアルプログラミングの手法でデバイス同士の連携を指定する．これらの技術では，対象とするオブジェクトの数に応じて画面上のアイコンの数が増

大し，一般のユーザが使用するには向いていない．Pin&Play[92] の派生技術である VoodooIO は，notice board と呼ばれる電源機能と通信機能を持つボードに対してセンサやアクチュエータを接続して連携させるためのヒューマンインタフェースを提供する．これにより，センサやアクチュエータは電源の心配は不要となるが，notice board に設置したデバイス同士の連携しかできないので設置の自由度が失われる．

無線センサネットワークの MAC プロトコルは数多く研究されている [52, 43, 44, 42, 45, 45]．ANTH がシングルホップの無線センサネットワークを対象としているのに対し，これらの研究はマルチホップの無線センサネットワークを対象としている．さらに，ANTH では各オブジェクトの電源環境はさまざまであるが，既存の無線センサネットワークの研究は電源環境が均一だという前提のものが多い．そのため，既存の無線センサネットワークのプロトコルを ANTH で利用した場合には遅延が大きい割にはバッテリ駆動のオブジェクトは A-MAC ほどの低消費電力性を実現できない．また，Z-MAC[42] は CSMA と TDMA を動的に切り替えることで低消費電力性を実現しているが，A-MAC では親ノードが常に受信待機状態であるので特別な同期が無くても送信時と受信時以外はスリープ状態にすることができる．

4.7 むすび

本章では，われわれの身の回りのオブジェクトにセンサやアクチュエータが組み込まれた環境において，これらのオブジェクト同士をユーザが簡単な操作で連携させることを目指したANTHの設計と実装について述べた．ANTHでは，まず，Bind Control Modelと呼ばれるプログラミングモデルを用いることで多種多様なデバイスの柔軟な接続性と開発の容易性を提供した．Bind Control Modelではそのモデルの単純さゆえに実現可能なサービスが固定的になりがちになるという問題や，ユーザの負担が増加するという問題が存在する．このような問題に対してANTHではVMを用いたソフトウェアモジュール機構と近接情報を基にしたヒューマンインタフェースを提供する．さらに，通信におけるアプリケーション層からMAC層までを統合的に扱うことで無線通信の低消費電力性を実現した．

ANTHを実際に普及させるためには*event*を解析することによるユーザのプライバシの問題や外部から*action*を実行したりといったセキュリティの問題が存在する．プライバシとセキュリティの問題は利便性とトレードオフの関係になることが多いため，今後，実証実験を通してどのような形でプライバシやセキュリティを提供していくかを考えていく必要がある．また，現在，ANTHの応用を広げるために，ANTHのMACプロトコルやVMなどの機能をLSI化してデバイス的な観点からの低消費電力化や省資源化の実現を目指している．