修 士 論 文

状況に適応するアプリケーションの構築法に関する研究

指導教官　中島達夫 助教授

北陸先端科学技術大学院大学
情報科学研究科

赤木 敏和

1997 年 2 月 14 日

目 次

図 目 次

表 目 次

第 1 章

序論

現在、計算機は小型化・軽量化が進み、持ち運び可能な計算機が一般化し、どこでも計算機を使うことができるようになった。さらに PC カードデバイス [1] を利用することにより、計算機の動作中に新たなデバイスを追加したり、取り外したりすることが手軽に可能になった。また APM(Advanced Power Management) 機能により、計算機に搭載されたバッテリーの状態を把握したり、積極的な節電をおこなうことができるようになった。このような移動計算機環境では、利用できるデバイスや計算資源などの状態が刻々と変化する状態にある。

このような移動計算機環境において、利用するアプリケーションは現在の計算機の状態に応じた処理をおこなうことが望ましい。たとえば様々なネットワーク PC カードの挿入により、通信速度の異なったネットワークに対応する場合、その上で利用する WWW(World Wide Web) ブラウザの処理をネットワークの特性に応じて変化させることは有用であろう。さらに APM 機能を利用することにより、バッテリの残量や AC アダプタの接続の有無によって、アプリケーションの消費する計算資源を調節することも考えられる。

このような計算機の状態や環境の変化にともなうイベントを利用するアプリケーションを記述する場合、オペレーティングシステムが提供する機能を使用することになる。一般的にオペレーティングシステムが提供する機能は、必要最小限のインターフェイスをもった基本的なものであることが多い。そのようなプリミティブな機能を利用してアプリケーションを記述することは容易でない。たとえば、システムのあるリソースの指定を行う場合でも、カーネルのインタフェースはそれぞれの機能に特化した独自のインターフェースを持っている場合が多く、アプリケーションの記述は容易でない。

そこで本研究では、利用する計算機の環境や状況の変化によるイベントの発生をとらえ、これらの情報をユーザレベルで動作するアプリケーションが利用しやすい形で効率的に伝えることができ、さらにシステムリソースへのアブストラクションを提供することのできるアーキテクチャの提案を行う。

本研究で提案するアーキテクチャは以下のような構成からなる。

- 計算機の環境や状況の変化をとらえるための、カーネルの拡張モジュール

- カーネル内で発生したのイベントをユーザーレベルサーバに通知するためのイベントマネージャ
- アプリケーションが必要とするイベントをフィルタリングしたり、システムリソースのアブストラクションを提供する環境サーバ
- アプリケーションプログラムに対して環境サーバを利用するためのインターフェイスを提供するユーザライブラリ

まず、利用する計算機の環境や状況の変化のイベントを処理できるように、我々が開発を行っている Real-Time Mach マイクロカーネル [10] に対して拡張をおこなった。具体的には、PC カードデバイスに対応するための機構、および APM 機能を利用するためのデバイスドライバを実装した。この拡張により、PC カードの挿抜のイベントをとらえたり、現在利用可能なカードの種類の情報や、可搬型計算機のバッテリーの状態などを把握できるようになる。

そして、カーネル内で発生したイベントをユーザーレベルに伝達するためのイベントマネージャを実装した。イベントマネージャはカーネル内のイベントを効率的にユーザーレベルサーバに伝達することができる。

さらに、イベントマネージャから受け取ったイベントをアプリケーションに伝達し、リソースへのアブストラクションを提供する環境サーバを実装する。環境サーバは、さまざまなイベントの中から、あらかじめアプリケーションから指定された種類のイベントのみをフィルタリングし、それぞれのアプリケーションに配送する。さらに、カーネル内のシステムリソースを参照するためのアブストラクションを提供する。

そして、アプリケーションから環境サーバを利用するためのユーザライブラリは、環境サーバを利用して容易にアプリケーションプログラムを記述出来るように、扱うイベントの指定やシステムのリソースを容易に参照可能なインターフェイスを提供する。

本研究で提案するアーキテクチャでは、イベントの種類やリソースの指定に、path 名を使用する。例えば、PC カード関連のイベントの指定の場合の例を示すと、

```
/localhost/pccard/event
/localhost/pccard/event/inserted
/localhost/pccard/event/removed
```

のような表記ができる。たとえば、

```
/localhost/pccard/event
```

の場合は、PC カードデバイスで発生したイベント全て、という指定となる。さらに、

```
/localhost/pccard/event/inserted
```

のような指定をした場合には、PC カードの挿入イベントのみを指定することが出来る。アプリケーションは、これらの path 名で表される Tcl スクリプトを環境サーバに挿入することができる。たとえば、

```
path   : /localhost/pccard/event/inserted
script: "set ret \"inserted\"\n"
```

のような表記をすることによって、アプリケーションへのイベントの通知を受けることが出来る。さらに、システムのリソースも同様にpath名で示される。たとえば、

```
/localhost/apm/status/batt
```

は、APMデバイスのバッテリの状態を示す。このスクリプトでは、イベントとリソースの参照を同時に指定することもできる。たとえば、

```
path   : /localhost/pccard/event/inserted
script: "if { [cast /localhost/apm/status/ac-online] == 1 } {\n
             set ret \"card inserted and ac-online\"\n
         }\n"
```

のような記述を行った場合には、PCカード挿入のイベントがあり、さらにAC電源が利用可能な時にのみ通知を受けることができる。スクリプト中の`cast`というコマンドは、引数のpath名で表される、他のスクリプトを呼び出すものである。くわえて、例えばシステムへのコントロールもpath名で表すことができる。例えば、

```
/localhost/apm/control/suspend
```

というpathはAPM機能を用いたサスペンドモードへの切替えのコントロールをあらす。また、path名で表されるスクリプトを自由に挿入することにより、あたらしいpath名で表されるイベントやリソースの定義ができ、アプリケーション間のメッセージ通信にも使用することができる。

これらの機能により、様々なイベントやリソース、さらにはコントロール等を統一的に表現することができる。さらにスクリプト言語により、それらを組み合わせた指定も可能である。

本研究では、以上のようなアーキテクチャにより、移動計算機環境や計算機の状況に応じた処理をおこなうシステムにおいて、以下の問題の解決をめざす。

- カーネル内で発生するイベントを、フィルタリングなどの処理を施してアプリケーションへ伝えるための、効率的な手法の提供
- イベントやシステムが提供するリソースの、アプリケーションへの統一的なアブストラクションの提供
- アプリケーション記述言語に応じたよりよいインターフェイスの提供

本稿では、まず移動計算機などにおいての状況に適応するアプリケーションの構成に関する問題を述べる。そして、本研究で提案する環境サーバなどのアーキテクチャの説明をおこない、各コンポーネントの説明をおこなう。さらに今回実装した環境サーバなどの説明をおこない、環境サーバを利用したアプリケーションの構築例を示す。そして最期にまとめと今後の課題について述べる。

第 2 章

状況に適応するアプリケーションを構築するための問題

本章では、状況に適応するアプリケーションとはどのようなものか、またそのようなアプリケーションを構築するためには、どのようなサポートが必要であるかについて述べる。

2.1 状況に適応するアプリケーションの必要性

2.1.1 使用する計算機のもつ資源に様々な制約がある場合

移動計算機環境等のように、使用する計算機の環境や、おかれている状況が変化するような環境では、使用するアプリケーションの動作はそれにあわせて変化するのが望ましい。例えば、以下にアプリケーションに対して影響を与えるであろう計算機の要素について挙げる。

- 計算機の CPU 能力
- 計算機の搭載メモリ量
- LCD ディスプレイの表示能力
- バッテリの量
- ディスク装置などの記憶装置
- ネットワーク等のデバイス

例えば、ディスプレイの表示能力を例に挙げると、デスクトップ型計算機等のディスプレイのように、表示解像度や色数が豊富な表示装置では、ユーザーインターフェイスに凝るなど、多彩な表現力を駆使する事ができる。しかしながら、小型の可搬型計算機では、LCD ディスプレイが搭載されている事が多く、解像度や表示可能な色数が制限される場合が多い。このような制限のある

表示機構で、デスクトップ機と同様の表示を行うことは現実的でないといえる。例えば、WWW ブラウザアプリケーションの利用を考えると、移動計算機では、ページのなかに埋め込まれた画像イメージを非表示にしたり、項目を間引いて表示したりするほうが望ましい場合がある。

さらに例を挙げると、持ち運び可能な計算機で、電源アダプタを接続して使用している場合は、ディスク装置などをふんだんに使用する事ができるが、バッテリ駆動の場合には、できるだけディスク装置の使用を減らし、電源の節約をおこなったほうが有利である場合が多い。ネットワーク装置の場合も同様で、つねに高速でバンド幅の広いイーサネットアダプタのようなデバイスがつねに使用可能であるとは限らない。無線 LAN や携帯電話で PPP(Point to Point Protocol) をもちいた場合には使用できる資源は限られており、高速ネットワークで結ばれたデスクトップ機などのようなネットワーク資源を使う事はできない。このような場合にはおのずと使用するアプリケーションの特性も変わってくるであろう。

2.1.2 使用する計算機の持つ資源の変化がある場合

さらに、前述のような制約に加えて、移動型計算機では PC カード装置等を用いることにより、計算機の環境が動的に変化することができる。以下にアプリケーションの動作に対して影響を与えるであろう要素を挙げる。

- AC 電源からバッテリ駆動へ、あるいはその逆への変化
- 残りバッテリ量の変化
- メモリカードの挿抜によるメモリ量の変化
- ATA ディスクカードなどの挿抜による変化
- ネットワークカードなどの挿抜による変化
- APM 機能によるサスペンドなどの動作

このような計算機の環境の変化は、イベントの発生としてとらえられ、アプリケーションはイベントを検知して動作の特性を変更することができる。

例えば、駆動電源が AC アダプタからバッテリに変更されたというイベントがアプリケーションに伝えられると、アプリケーションはディスクアクセスを減らして出来るだけ節電するように動作ポリシーを変化させることができる。また、ディスクカードが挿入されたことにより、アプリケーションが使用できる領域が増え、アプリケーションの動作を変更することも考えられる。

2.1.3 状況に適応するアプリケーションの有用性

このように、現在使用している計算機の状態を把握し、変化のイベントを利用してアプリケーションの動作を適応させていく試みは大変有用であるといえる。[11][12]

例えばWWW アプリケーションを例にとり、計算機の状況に応じた処理をおこなうことの有用性について述べる。WWW アプリケーションでは、ネットワーク上に数多く分散して配置されたサーバーから、HTML(Hyper Text Markup Language) で記述された文章を HTTP(Hyper Text Transfer Protocol) によって転送し、画面に表示する。このようなWWW アプリケーションプログラムを移動計算機で利用する場合、利用する計算機のネットワークデバイスの構成などは常に変化する事が考えられ、アプリケーションプログラムの実行にも影響を与える。例えば、移動計算機ではいつでもネットワークが使用可能であるわけではなく、利用可能であったとしても、ネットワークバンド幅の狭い無線 LAN や、比較的高速な Ethernet など、さまざまなネットワークで接続される。このようなネットワーク環境が変化する環境で、WWW アプリケーションはそれぞれのネットワークにあわせた振舞いをする必要がある。

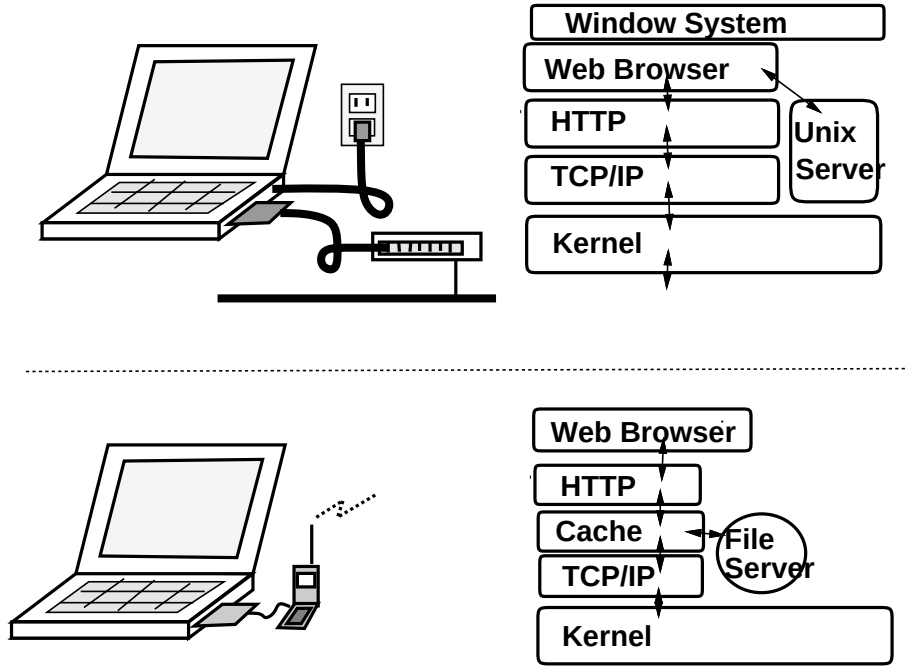

図 2.1: 移動計算機環境の例

図2.1に示したように、AC 電源駆動でしかも高速な有線ネットワークに接続されているときには、Unix サーバや TCP/IP プロトコルスタック、HTTP モジュール等の各コンポーネントをフルセットで備え、X-Window System のような高機能なウインドウシステムを使用することができる。しかしながら、バッテリー駆動でしかも無線ネットワークのようなデバイスしか使用できない場合は、WWW アプリケーションを動作させるために最低限必要なファイルアクセス機能やネットワークプロトコルスタック、HTTP モジュールを使用し、さらに高機能なウインドウシステムでなくWWW アプリケーションに特化した表示機構を備えた構成が望ましいといえる。また、バンド幅が狭いネットワークの場合には、HTTP キャッシュモジュールを使用し、積極的にバンド

幅の節約を行うほうが好ましいであろう。

このように、計算機のおかれた状況に対してアプリケーションの処理の内容を適応させる例を以下に示す。

- ネットワークデバイスの変化による、TCP/IP などのネットワークプロトコルスタックの適応
- 利用できるネットワークのバンド幅の変化による、HTML や画像データ転送のポリシーの変化
- バッテリーの残量などに依存した、節電ポリシーの変化
- 利用可能なディスプレイ装置に適応した、ウインドウシステムの表示ポリシーの変化

2.2 状況に適応するアプリケーションを構築するために必要な機構

前節で述べたように、利用する計算機の状況に適応するアプリケーションは大変有用であり、既存のアプリケーションでもこれらの情報を利用することによって、ユーザーにとってより快適に使用できるものになるであろう。以下では、このような状況に適応するアプリケーションを記述するために必要な機構について述べる。

2.2.1 カーネルなどのシステムサポート

移動型計算機などにおいて、たとえば PC カードが挿抜されたというイベントは、カーネル内部で発生する。これらのような、カーネルなどのシステム内部で発生するイベントを的確に感知する機構が必要である。さらに、発生したイベントによって、例えばカードデバイスが挿入されたときには、対応するデバイスドライバの初期化の処理など、動的な変化に対応できる構成が必要である。

しかしながら、一般的に Unix などのオペレーティングシステム等では、上述したような動的なデバイスの変化に対応するものは少ない。つまり、カーネルはシステムの起動時にのみデバイスを調べ、あとから動的にデバイスの追加や削除をおこなうことは難しい。

2.2.2 イベントの配送

カーネル内のイベントを、ユーザーレベルへのサーバ等に伝達する機構が必要である。一般的に、Unix 等のオペレーティングシステムでは signal、Mach オペレーティングシステムでは exception という機構が用いられることとが多い。しかしながら、これらの機構を用いてイベント配送の機構を記述し、アプリケーションで利用するには、機能がプリミティブ過ぎる。アプリケーションで利用しやすいようにサポートライブラリを用意することが多いが、システムの違いなどにより、インターフェイスの統合が取れていない場合が多いといった問題がある。また、システム内で発

生するイベントのうち、各アプリケーションが必要とするイベント情報だけをフィルタリングして、アプリケーションに伝える機構も必要である。

2.2.3 イベントやリソースの抽象化

アプリケーションがカーネル内の情報などの情報を参照するには、システムコール等を通して参照する。そのため、アプリケーションプログラマは取得するリソースによって、システムによって提供されるインターフェイス毎に特化した方法によってアクセスすることになる。また、このような繁雑な手間を省くために、ユーザライブラリを用意することもあるが、アプリケーション記述言語によって様々なインターフェイスを用いなければならない場合が多い。

そのため、イベントやリソースをアプリケーションプログラマが容易に参照できるように、統一されたインターフェイスが必要である。

2.2.4 アプリケーション記述のための言語的サポート

さらに、以上で述べた機能を利用して、実際にアプリケーションの動作を状況によって変化させるための言語サポートが必要である。アプリケーション記述言語は、システムから得た情報をもとに、アプリケーションの意味の変化を記述できるものでなければならない。さらに、上述のイベントやリソースを参照するための、よりよいインターフェイスを持っていることが望ましい。

2.3 まとめ

以上で述べたことをまとめると、状況に適応するアプリケーションを記述するためには、以下のような要素が必要である。

[**カーネルなどのシステムサポート**]
: 動的なデバイスの構成の変化などに対応できる機構

[**イベントの配送機構やリソースの参照方法**]
: カーネル内のイベントの発生を効率的にアプリケーションに伝達し、リソースの参照をより容易に行うための機構

[**イベントやリソースの抽象化**]
: アプリケーションプログラマがイベントやリソースの種類等を、より容易に指定できるインターフェイス

[**アプリケーション記述のための言語**]
: 状況に適応してアプリケーションの動作の変化を記述可能な言語

本研究では、特に最初の 3 項目のトピックにしぼって、移動計算機環境などにおいて、状況に適応するアプリケーションを構築するためのフレームワークを提案し、実装例を示す。

第 3 章

関連研究

本章では、移動計算機環境などの状況が動的に変化するような計算環境における、システムソフトウェアやアプリケーション記述の問題についての関連研究について述べる。

移動計算機環境等のような、状況にが動的に変化する環境についての様々な議論の中で、ここでは以下ようなトピックに付いて述べる。

- カーネルなどのシステムソフトウェアのサポート
- イベントシステム
- リソースの抽象化
- アプリケーション記述言語

3.1 動的な環境の変化に適応するシステムサポート

動的なデバイスの変化に対応するためのカーネルなどのシステムのサポートについては、様々な方法で試みられている。例えば、Unix オペレーティングシステムでは、Wildboar Porject が開発を行っている、Wildboar システム[1]等がある。これらのシステムでは、動的なデバイスの変化に対するデバイスドライバの対応のみならず、PC カードデバイスに対応する際に必要なコンポーネントである、カードサービス・ソケットサービスなどのコンポーネントがモジュール化されており、ソケットやカードのコンフィグレーションなどの操作に関しては、優れたインターフェイスを提供している。しかしながら、PC カードの挿入などによるカーネル内で発生したイベントを利用して、アプリケーションを記述しようという試みは少ない。

また、Micheal Bender らによる、"Unix for Nomads" [4] というシステムでは、移動計算機環境で電子メールなどのアプリケーションを使用する場合の問題点について挙げ、Unix オペレーティングシステムに対して拡張をおこなっている。

[1] Wildboar は BSDI 社の BSD/OS を対象とした PC カードデバイス対応のための拡張キットである。BSD/OS および BSDI は Berkeley SoftwareDesign, Inc. の商標である。

これらのシステムでは、Unix オペレーティングシステムを対象としており、カーネル等のシステムソフトウェアとアプリケーションとのインターフェースの自由な設計などが比較適難しいといえる。

3.2 イベントシステム

一般的に Unix などのオペレーティングシステムでは、カーネルなどのシステムからユーザーレベルのアプリケーションに対してイベントを伝達する場合には signal 機構をもちいることが多い。しかしながら、signal では、様々な種類のイベントをスマートに伝達することができない。

また、カーネル等では必要最低限のプリミティブな機能しか提供せず、ユーザレベルでサーバを用いて、高度なイベントの伝達の操作を提供するものもある。しかしながら、イベントのディスパッチングは、あらかじめきめられた静的な動作しか提供しないものが多い。

3.3 システムリソースの抽象化

例えば、アプリケーションがカーネル等のシステム内部のリソースを参照しようとした場合、多くのシステムでは、そのリソースを管理しているコンポーネントが違えば、参照方式が違う場合が多い。例えば、Unix オペレーティングシステムや、Mach マイクロカーネルなどのオペレーティングシステムでのデバイスコントロールなどは、デバイス名によって参照に利用する構造体が違う場合が多く、たいへん複雑なものとなっている。

また、リソース名の指定に関しては、例えば Mach オペレーティングシステムで使用できる、Environment Manager Server という実装例がある。しかしながら、これはリソース名が単一名前空間での指定に留まり、一般に言う環境変数のようなサービスしか提供出来ない。

また、B.R. Badrinath らによる、イベント配送のアブストラクションを提供するシステム [5] では、リソース名をクラスとして定義し、オブジェクト指向的アプローチによって解決しようとしている。このシステムでは、あるリソースの値が、いくつ以上、いくつ以下といった幅をもった指定が出来るといった点が優れている。

3.4 アプリケーション記述言語

アプリケーション記述言語に関しては、メタレベルアーキテクチャなどを利用して、計算機の状態に応じてアプリケーションの振舞いを変化させるアプローチが取られている例がある。LEAD という言語 [13] では、メタレベルのメソッドを状況に応じたものにディスパッチすることによって、アプリケーションの振舞いを変化させることができるものである。しかしながら、これらの研究では、アプリケーションをどのように記述するかの問題が主眼で、カーネル等のシステムレベルのサポートまで考慮されていない場合が多い。

3.5 まとめ

以上に述べたように、システムソフトウェア的なアプローチ、アプリケーション記述言語などの言語サポート等に主眼をおいたシステムは多くなされている。しかしながら、システムのサポートも含めて、状況に適応するイベント配送システム、システムリソースの参照、およびそれらのよりよいアブストラクションの提供の手法を全体的に議論し、全体としてのシステムとしてのフレームワークを提供したものは少ない。

第 4 章

状況に適応するアプリケーションを構築するための枠組

ここでは、本研究で提案する状況に適応するアプリケーションを構築するためのアーキテクチャについて述べる。このアーキテクチャは以下の要素からなる。

- カーネルサポート
- カーネル内からのイベント配送のためのイベントマネージャ
- イベントのフィルタリングやシステムリソースへのアブストラクションを提供する環境サーバ
- 環境サーバを利用してアプリケーションを記述するためのユーザライブラリ

図 4.1に、各コンポーネントの関係を示した概略図を示しておく。
以下、順に上に挙げたそれぞれのコンポーネントについて詳しく述べる。

4.1 カーネルサポート

移動計算機環境などにおいて、たとえば動作中に PC カードが挿入されたりする場合がる。このような動的なデバイスの変化に対応するためにはオペレーティングシステムのカーネルサポートが必要である。

現在、PC カードデバイスをサポートするシステムの１つに、Wildboar Project が開発を進めている、Wildboar システムがある。本研究では、この Wildboar システムをもとに、我々が研究に使用している Real-Time Mach マイクロカーネルに対して拡張をおこなった。

以下に Wildboar システムを採用した理由を挙げておく。

- PC カードコントローラ・ソケットサービス・カードサービスの各コンポーネントが明確に実装されている。

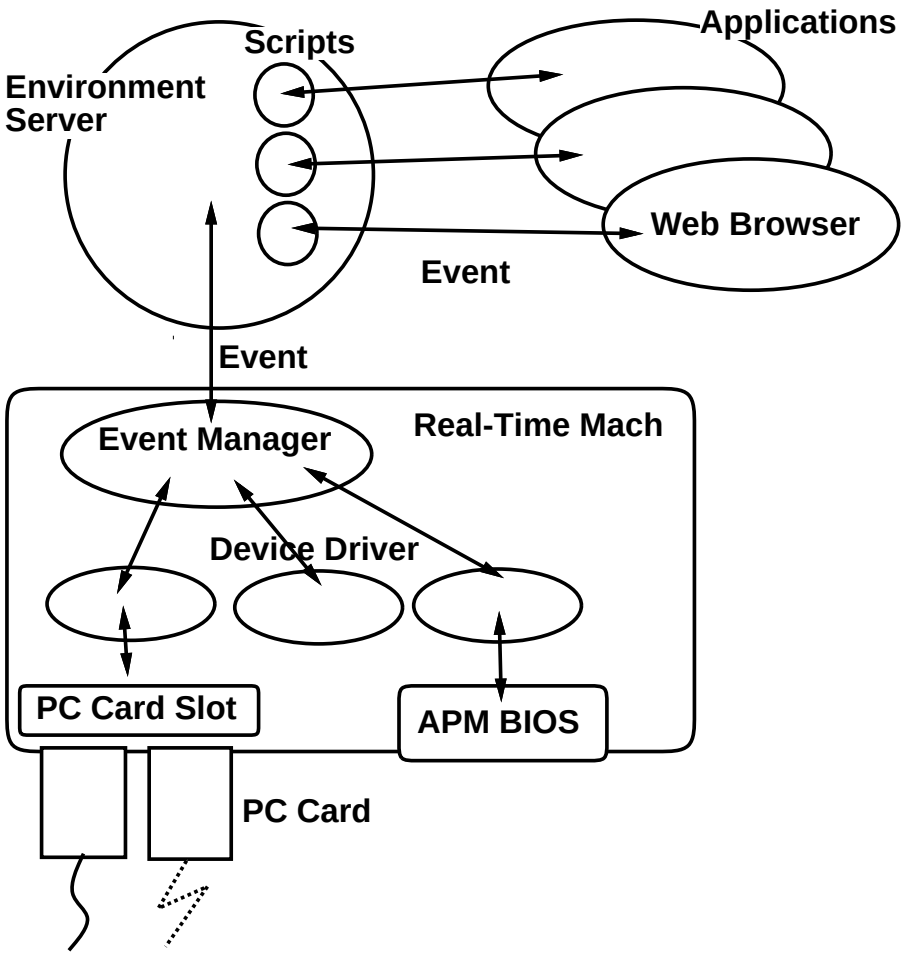

図 4.1: 提案するアーキテクチャの概略図

- 優れた APM サポート
- 移植性に優れた設計

以下に Real-Time Mach マイクロカーネルへ行った拡張を挙げておく。

- **PC カード コントローラデバイスド ライバの実装**
  PC カードコントローラの制御を行い、PC カードの挿抜のイベントの検知等をおこなう。

  カード サービス・ソケット サービスの実装
  カードサービスは PC カードデバイスの設定の際のサポートルーチンを提供し、カードの挿抜のイベントを扱う。ソケットサービスはカードコントローラのスロットに対するアブストラクションを提供する。

- **PC カードデバイスのドライバの実装**
  ネットワークカード等のデバイスの動的な挿抜に対応したデバイスドライバ。

- **APM デバイスドライバの実装**
  APM BIOS を利用して、バッテリの状態などの情報を取得する。

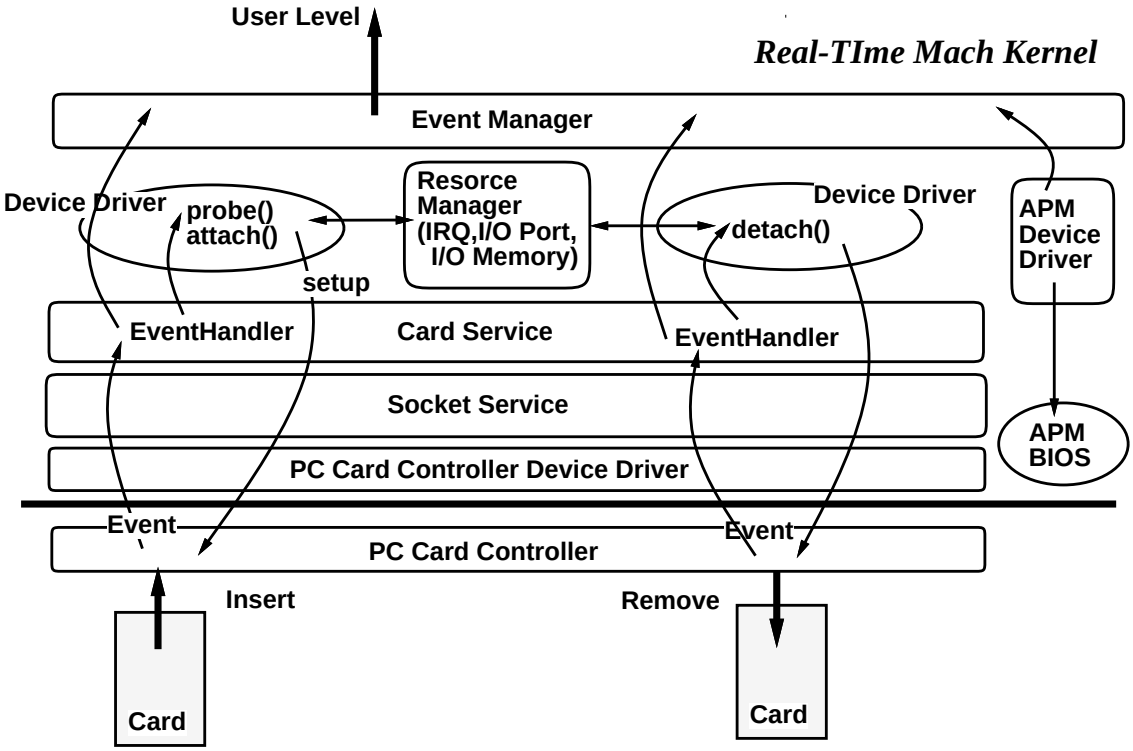

図 4.2: Real-Time Mach カーネルの拡張

図 4.2をもとに、これらのコンポーネントの関連および機能を説明する。カードの挿入のイベントは、PC カードコントローラによって検知され、PC カードコントローラデバイスドライバによって、カードサービスのイベントハンドラが呼び出される。カードサービスは、あらかじめ登録された、PC カードデバイス対応のドライバを順次呼び出し、挿入されたカードに対応するドライバを探す。対応するドライバが見つかった場合には、そのドライバは、カードサービス内に含まれるルーチンを使用し、カードの設定をおこなう。その際、必要に応じて、ハードウェアインタラプト番号、I/O ポート番号及び I/O メモリウインドウをリソースマネージャから取得する。そして、取得されたリソースの情報に従い、ソケットサービスによって抽象化されたソケットに対して、カードサービスのルーチンを使用して PC カードコントローラおよび PC カードの設定を行う。設定が終了すると、カードサービスのイベントハンドラは、イベントマネージャにカードの挿入のイベントを伝える。PC カードコントローラドライバは、設定の際にハードウェアを直接操作するための機能を提供する。

カードの排出がおこなわれた時には、PC カードコントローラデバイスドライバが検知し、カードサービスのイベントハンドラを呼び出す。イベントハンドラは挿入されていた PC カードを制御していたデバイスドライバを呼び出し、デバイスを無効化するとともに、使用していたリソースをリソースマネージャに返却する。

APM デバイスドライバは、マシンに搭載された APM BIOS を使用して AC ラインやバッテリの残量を監視する。もし変化を検知した場合は、イベントマネージャに伝える。

このような拡張により、PC カードデバイスの動的な変化をオペレーティングシステムで扱う事ができるようになり、APM 機能を利用してバッテリ状態などの情報を取得できるようになった。さらに、PC カードの挿入や電源状態の変化などの情報を取得できるようになった。

PC Card に関連するもの	
CSE_CARD_INSERTION	カードの挿入
CSE_CARD_REMOVAL	カードの抜出
APM 機能に関連するもの	
APM_EVENT_STANDBY_REQ	スタンバイモード要求
APM_EVENT_SUSPEND_REQ	サスペンドモード要求
APM_EVENT_NORMAL_RESUME	サスペンドモードからの復帰
APM_EVENT_LOW_BATTERY	バッテリの警告
APM_EVENT_POWER_STATUS_CHANGE	電源状態の変化
APM_EVENT_STANDBY_RESUME	スタンバイモードからの復帰

表 4.1: カーネル内で発生するイベント

これらの PC カードドライバ・APM ドライバからは、表 4.1 に示したようなイベントを取得することができる。これらのイベントをもとに、たとえばスタンバイモード要求のイベントが伝えられた場合、アプリケーションやファイルシステムサービスなどの各種サービスは、できるだけリソースを消費するような重い処理を控え、バッテリを節約するようなストラテジをとることができる。

4.2 イベントマネージャ

以上で述べたカーネルに対する拡張により、PC カードデバイスの挿抜によるイベントや、APM デバイスで発生するイベントをとらえることができるようになった。そこで、これらのイベントをユーザーレベルのサーバで利用できるように、ユーザーレベルプログラムに対してイベントを伝達するためのイベントマネージャを実装した。

図 4.3にイベントマネージャの概念図を示す。イベントマネージャは、Real-Time Mach のメッセージ通信機構を使用して、ユーザーレベルで動作するサーバにメッセージを配送する。カーネ

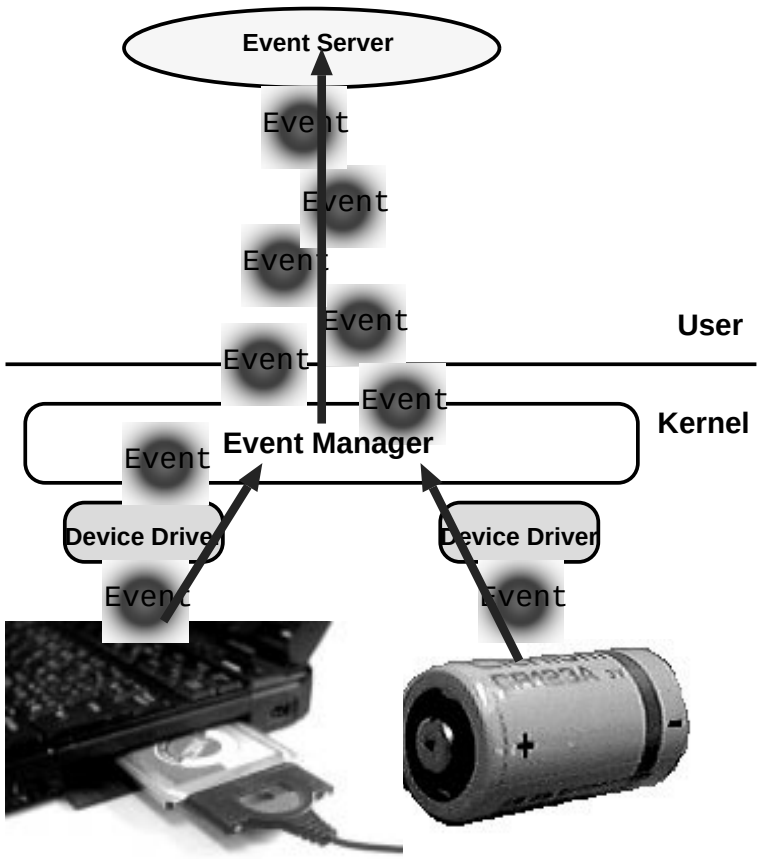

図 4.3: イベントマネージャ

ル内でイベントの発生を検知した PC カードドライバや APM ドライバなどのコンポーネントは、イベントソース ID とイベントタイプ ID をアーギュメントとして、イベントマネージャのディスパッチルーチンを呼び出す。

イベントマネージャは、Mach マイクロカーネルのメッセージ通信機構を利用して、あらかじめユーザープログラムから指定されたポートに対して、非同期メッセージを配送する。

以上で述べたファンクションの一覧を以下に挙げておく。

- カーネル内のコンポーネントが呼び出すディスパッチファンクション

```
kern_return_t eventm_dispatch_msg(int source int, type);
```

- ユーザレベルプログラムがイベントを受け取るポートを設定するためのシステムコール

```
kern_return_t eventm_set_port(mach_port_t request_port);
```

- ユーザプログラムがイベントマネージャからのメッセージを取得するファンクション

```
kern_return_t eventmanager_delivery_wait(mach_port_t port,
                                         int *event_source, int *event_type);
```

以上で述べたイベントマネージャによって、カーネル内で発生したイベントをユーザレベルのプログラムに対して効率的に配送することが可能になった。

4.3 環境サーバ

イベントマネージャを使用することにより、カーネル内のイベントをユーザーレベルのプログラムに通知することが出来るようになった。しかしながら、アプリケーションはカーネル内のイベントのすべてを使用するわけではなく、ある特定のイベントのみをフィルタリングして通知を受けたい場合が多い。また、デバイスの状態などのリソース等を参照して、より複雑な指定を行いたい場合もある。

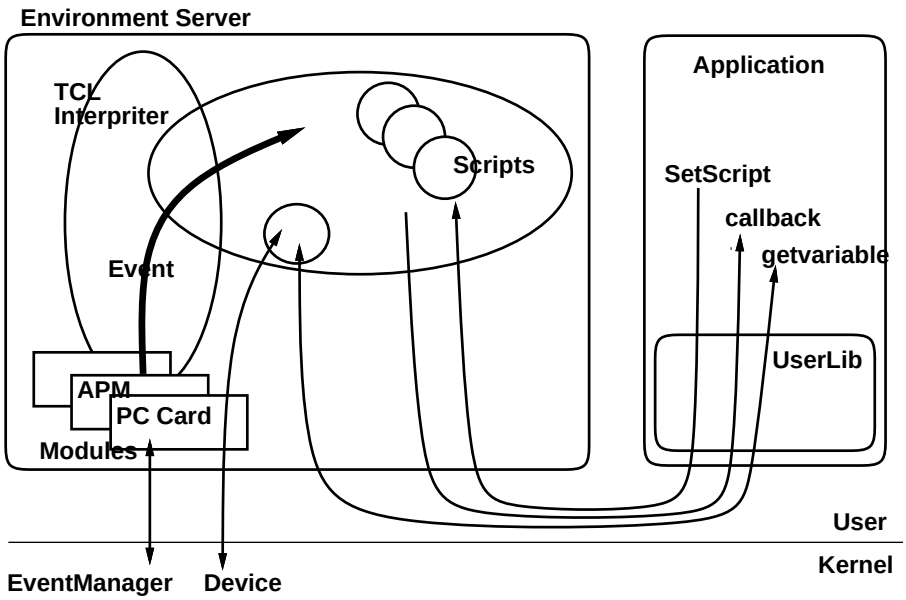

図 4.4: 環境サーバの概略図

このような機能を実現するため、環境サーバは以下のような特性をもっていなければならない。

- イベントの通知を受ける際の複雑な条件が指定可能である
- システムのリソースの参照を容易に出来る
- イベント名・リソース名に対するアブストラクションを提供する

図 4.4に環境サーバの概略を示し説明する。環境サーバ内には、Tcl のインタープリタをもった空間がある。ユーザーはイベントの通知を受けるために、スクリプトをこの空間に挿入する。また、サーバ内にはカーネル等のシステムとのインタフェースをもつモジュールが配置されている。カーネル内のイベントマネージャ等から送られたイベントは、この空間に通知されて、それぞれのアプリケーションが指定したイベントをフィルタリングして、アプリケーションのコールバックルーチンを呼び出すことによって、アプリケーションに通知される。

環境サーバについては、次章で詳しく述べる。

4.4 ユーザライブラリ

以上で述べた環境サーバーを利用して、アプリケーションプログラムを記述するために、ユーザーライブラリを提供する。ユーザーライブラリは、

- 環境サーバへのスクリプトの組込み
- 環境サーバからのスクリプトの削除
- 環境サーバ内の path 名で示されるスクリプトの呼び出し
- イベント・リソース名の path 名の検索などの操作
- 環境サーバからのコールバックハンドラの管理

の機能を提供する。

アプリケーションプログラマは、これらのライブラリを利用して、イベントやシステムのリソースを参照するようなアプリケーションを容易に記述することができる。

第 5 章

環境サーバの設計・実装

本章では、環境サーバの設計と実装に付いて詳しく述べる。環境サーバは、以下のようなコンポーネントから成っている。

- Tcl スクリプト言語のインタープリタを持つ環境空間
- アプリケーションとのインターフェイスモジュール
- カーネル等のシステムとのインターフェイスモジュール

以下、まず各コンポーネントの設計・実装について説明し、動作の原理を述べ、実際のイベントの流れかたを説明する。

5.1 環境サーバの概要

前述したように、環境サーバは単なるイベントのディスパッチングのみならず、複雑なイベントフィルタリングの要求を受け付けるものでなければならない。さらに、イベントやリソースに対してよりよいアブストラクションを提供しなければならない。そこで、本研究で提案する環境サーバは以下の特徴をもっている。

- ユーザはイベントやリソースに対する処理を Tcl スクリプトで記述する。ある種類のイベントのみを記述するフィルタや、イベントとリソースの値の参照を組み合わせた記述を行うことができる。
- Tcl スクリプトと、スクリプトの評価結果の最終的な通知先を含んだオブジェクトを環境サーバに挿入できる。
- オブジェクトは、path 名で識別される。
- オブジェクトは、path 名で表される他の オブジェクトを呼び出すことができる。

- オブジェクトのスクリプトの実行後、もしその オブジェクトが reply port をもっていたならば、クライアントへの call back を行う。

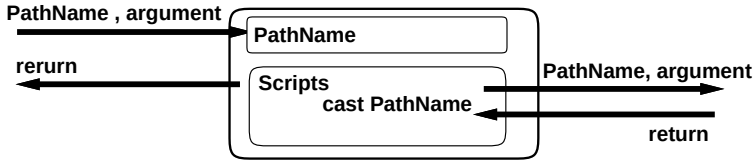

図 5.1: オブジェクト

オブジェクトの呼び出しの流れを 図5.1に示す。

5.2 オブジェクト

オブジェクトの構造は以下のような構造体で定義される。この構造体は、ユーザからのリクエストなどにも使われる。

```
typedef struct eserver_user {
    mach_port_t    port;
    int            cmd;
    unsigned char  path[ESERVER_PATH_MAXLEN];
    unsigned int   *callback
    unsigned char  scripts[ESERVER_SCRIPT_MAXLEN];
    unsigned char  arg[ESERVER_ARG_MAXLEN];
    unsigned char  ret[ESERVER_RET_MAXLEN];
} eserver_user_t ;
```

5.3 path 名

オブジェクトを識別するために path 名が用いられる。path は例えば以下のような表記が用いられる。

```
/localhost/pccard/event
/localhost/pccard/event/inserted
/localhost/pccard/event/removed
/localhost/pccard/event/slot1/inserted
/localhost/pccard/event/slot1/removed
```

```
/localhost/pccard/control
/localhost/pccard/control/up
/localhost/pccard/control/down
/localhost/pccard/control/slot1/up
/localhost/pccard/control/slot1/down
/localhost/pccard/status
/localhost/pccard/status/slot1
/localhost/pccard/status/slot1/cardname

/localhost/apm/event
/localhost/apm/event/suspend
/localhost/apm/event/suspend-resume
/localhost/apm/control
/localhost/apm/control/suspend
/localhost/apm/status
/localhost/apm/status/batt
```

5.4 システムインターフェイスモジュール

システムインターフェイスモジュールは、 スクリプト言語空間とシステムとのインターフェイスを提供する。イベントマネージャからイベントの発生の通知を受けたモジュールは、スクリプト言語空間にある 特定の path 名で表させる オブジェクトを引数を付けて呼び出す。例えば PC カードデバイスのイベントの場合には、

```
/localhost/pccard/event
```

であるし、APM デバイスの場合には、

```
/localhost/apm/event
```

である。

また、システムインターフェイスモジュールは、システムのリソースのアブストラクションも提供する。この場合は、例えばカーネルへのシステムコールなどのインターフェイスを、スクリプト言語空間の Tcl インタープリタの言語拡張として提供する。

5.5 システムスクリプト

前述のシステムインターフェイスモジュールは、スクリプト言語空間に対して、デフォルトのシステムスクリプトを挿入する。これは、path 名で表された、イベント、システムリソース、システムコントロールの抽象化されたものである。

例えば、PC カードデバイスでは、

```
/localhost/pccard/event
/localhost/pccard/event/inserted
/localhost/pccard/event/removed
/localhost/pccard/event/slot1/inserted
/localhost/pccard/event/slot1/removed
/localhost/pccard/status
/localhost/pccard/status/slot1
/localhost/pccard/status/slot1/cardname
/localhost/pccard/status/slot1/devicename
/localhost/pccard/control
/localhost/pccard/control/
```

のような表記で表される。APM デバイスでは、

```
/localhost/apm/event
/localhost/apm/event/suspend
/localhost/apm/event/suspend-resume
/localhost/apm/control
/localhost/apm/control/suspend
/localhost/apm/status
/localhost/apm/status/batt
```

のよう表される。
例えば、

```
/localhost/pccard/event
```

の場合には、

```
‘‘ if {$arg == \"1\"} {\n
        cast /localhost/pccard/event/inserted\n
    } else {\n
        cast /localhost/pccard/event/removed\n
    }\n’’
```

のようなスクリプトが挿入される。この例では、

- システムインタフェースモジュールによって、`/localhost/pccard/event` が呼び出される
- `/localhost/pccard/event` である script は、
  挿入のイベントならば`/localhost/pccard/event/inserted`、
  抜出の場合には、`/localhost/pccard/event/removed`
  という path で示される オブジェクトを呼び出す

という動作を提供する。このようなシステムスクリプトを、上述の イベント、リソース、コントロールを表す path 名の オブジェクトとして、システムの起動時に スクリプト言語空間に挿入する。

5.6 ユーザスクリプト

アプリケーション記述者は、スクリプト言語空間に対して自由に オブジェクトの挿入をすることができる。

例えば、PC カードの挿入によって通知を受けたい場合には、

```
eserver_user_t cardevent = {
        REPLY_PORT,
        ESERVER_CMD_PLUGIN,
        "/localhost/pccard/event/inserted",  /* path name */
        0,0,
        "set ret 1",                         /* scripts */
        "",""
};
```

という構造体の オブジェクトを挿入すればよい。

この オブジェクトは、図 5.2のような呼び出しをされ、アプリケーションに通知がなされる。

同様に、挿入・抜出などのいかなるイベントが発生した場合でも受け取りたい場合には、

```
eserver_user_t cardevent = {
        REPLY_PORT,
        ESERVER_CMD_PLUGIN,
        "/localhost/pccard/event",           /* path name */
        0,0,
        "set ret 1",                         /* scripts */
        "",""
};
```

という オブジェクトを挿入すればよい。このようにアプリケーションが必要とするイベントの指定が path によって柔軟に指定可能である。

さらに、システムのリソースやシステムコントロールに際しても path 名で指定される オブジェクトを呼び出すことによって、返値として取得出来る。例えば、バッテリーの状態を取得するには、

```
eserver_user_t querybatt = {
        MACH_PORT_NULL,
        ESERVER_CMD_EVAL,
```

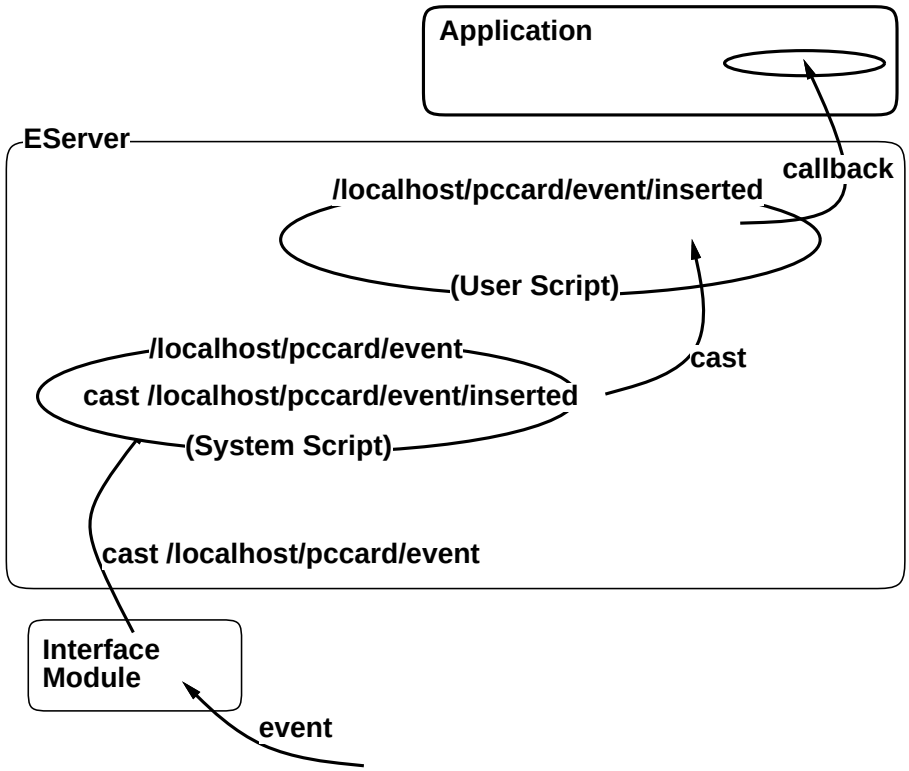

図 5.2: オブジェクトの呼び出しの例:1

```
        "",                                     /* path name */
        0,0,
        "set ret [ cast /localhost/apm/status/batt]\n",
                                                /* scripts */
        ""
};
```

という構造体を用いて、リクエストを行う。

この場合の オブジェクトの呼び出し関係を図 5.3に示す。

コントロールも同様に可能であり、

```
eserver_user_t suspendcontrol = {
        MACH_PORT_NULL,
        ESERVER_CMD_EVAL,
        "",                                     /* path name */
        0,0,
        "set ret [ cast /localhost/apm/control/suspend]\n",
```

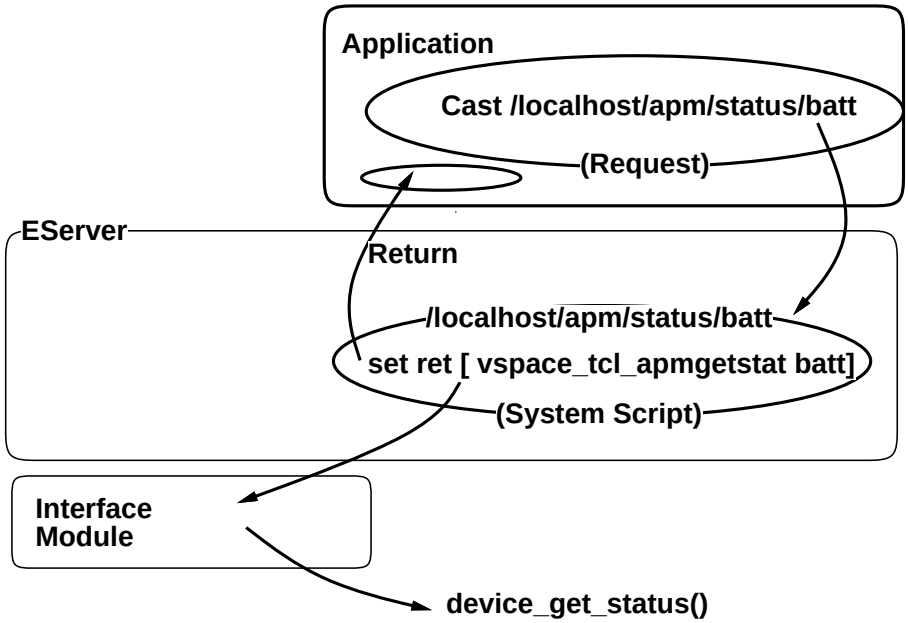

図 5.3: オブジェクトの呼び出しの例:2

```
                                        /* scripts */
        "","
};
```

というリクエストを行えば良い。

さらにユーザは、システムスクリプトのように、新しい path 名をもったオブジェクトも挿入可能である。このことにより、2 つのアプリケーション間でのメッセージ通信のような用途にも使用が可能である。

5.7 イベントやリソースの抽象化

上述したように、環境サーバの スクリプト言語空間のなかの オブジェクトは、path 名によって識別される。これは、例えばカーネルへのシステムコールや、デバイスへのコントロールの抽象化をおこなっている。

一般に、カーネルなどのシステムへのインターフェイスは、プリミティブなものが多く、ユーザーが使いやすいものではない。また、リソースを参照する先のモジュールが違うと、参照方法が変わったりするなど、大変扱いにくいのが一般的である。

そこで、環境サーバではそれぞれのシステムリソースを path 名で表される空間へ変換して提供する。

さらに、path 名指定では、ユーザーは適当な path を選ぶことにより、様々なレベルの粒度でリソースの指定が出来る。たとえば、

- `/localhost/pccard/event`
  PC カードのイベント全て。つまり、カードに対してなにかのイベントが起こった時、という指定。
- `/localhost/pccard/event/slot1/inserted`
  PC カードのイベントで、slot 1 に何か挿入された時、という指定。

のような指定が可能である。これらの機能は、システムインターフェイスモジュールのシステムスクリプトによって提供される。

5.8 ユーザライブラリ

環境サーバを利用するために、C 言語へのインターフェイスと Tcl スクリプト言語へのインターフェイスを提供するためのユーザライブラリを用意した。ライブラリは、

- ユーザスクリプトの挿入操作
- ユーザスクリプトの削除操作
- オブジェクトの呼び出し

のルーチンを含んでいる。

5.9 まとめ

以上で述べた環境サーバについてまとめると、本研究で提案した環境サーバは、

- スクリプトによるフィルタなどの記述が可能
- path 名による、イベントやリソースへのアブストラクションを提供
- ユーザスクリプトによる拡張性

という優れた性質をもつものである。

次章ではこの環境サーバを用いたアプリケーションの例を示す。

第 6 章

環境サーバおよびアプリケーションの評価

本章では、環境サーバを利用したアプリケーションの例を示し、実際にアプリケーションを記述する際に用いるスクリプトの例などを示す。

6.1 環境サーバの利用の方法

環境サーバを利用してアプリケーションを記述するために、ユーザライブラリを用意した。ここでは、環境サーバを利用したアプリケーションを記述する場合の初期化の例を示す。

ここでは、C 言語で記述する場合の例を示す。

```
#include <eserver_types.h>
#include <eserver_user.h>

ret_t client_init(mach_port_t *server_port, mach_port_t *user_port)
{
    ret_t rt;

    mach_init();
    cthread_init();

    rt = eserver_user_init(server_port, user_port);

    return rt ;
 }
```

この他に一般的な Mach の include file を include することが必要である。

eserver_user_init() は、Mach のnameserver 機能を利用して、環境サーバのport を取得し、クライアントのポートを確保する。正常終了した場合の戻値は、ESERVER_SUCCESS である。

以下に、環境サーバにスクリプトを挿入し通知を受ける場合の簡単な例を示す。なお、エラー処理などは省いてある。

```
#include <eserver_types.h>
#include <eserver_user.h>

eserver_user_t cardevent = {
    MACH_PORT_NULL,
    ESERVER_CMD_NULL,
    "/localhost/pccard/event/inserted",
    0, 0,
    "if { vspace_cast /localhost/apm/status/batt] == 100 } {\n
     set ret \"battery is high and card inserted.\"\n",
    "", ""
};

void callback();

void main(void)
{
    ret_t rt;
    mach_port_t server_port;
    mach_port_t user_port;

    rt = client_init(&server_port, &user_port);

    cardevent.cmd = ESERVER_CMD_PLUGIN ;
    cardevent.callback = callback;
    rt = eserver_user_request(&server_port, &user_port, &cardevent);

    sleep(60);

    cardevent.cmd = ESERVER_CMD_REMOVE ;
    rt = eserver_user_request(&server_port, &user_port, &cardevent);

    exit(0);
```

```
}

void callback(char *arg)
{
    printf("callback arg %s\n", arg);
}
```

この例では、バッテリ量が 100 %で、カードが挿入された時のイベントの通知を受けるスクリプトを環境サーバに挿入し、しばらく待った後、スクリプトを削除して終了する例である。

6.2 アプリケーションの例

6.2.1 環境サーバシェル

まず、一番簡単な例として、環境サーバ内のオブジェクトを呼び出すシェルの例を示す。このシェルは、Tcl インタープリタシェルの拡張として実装した。

例えば、あるオブジェクトを呼び出すには、

```
% cast PathName arg
```

のように、path 名と、パラメータを指定して、cast コマンドを実行する。例として、現在のバッテリの量を表示させる方法を示す。

```
% cast /localhost/apm/status/batt 1
```

さらに、コントロールの発効をする場合には、同様の操作で可能であり、APM 機能を用いて、スタンバイモードに切替えるには、以下のような操作で可能である。

```
% cast /localhost/apm/control/standby
```

現在では、環境サーバシェルは簡単な機能しか持っていないが、将来的には、環境サーバ内のオブジェクトの一覧表示や検索、スクリプトの操作などがおこなえるように改良を加える必要がある。

6.2.2 PC カードのイベントを対象とした例

図 6.1に PC カードスロットの情報を表示するアプリケーションの実行画面の例を示す。

この例では、単純に PC カードスロットのイベント情報を利用し、カード情報を問い合わせる方法を使用している。

- イベント取得

  path `"/localhost/pccard/event"`

図 6.1: アプリケーションの例:PC カードスロットモニタ

script `"set ret $arg\n"`

- カードスロットの情報取得

  path `""`

  script `"set ret [cast /localhost/pccard/status]\n"`

イベント取得のスクリプトでは、イベント種類を引数にしてアプリケーションに通知を行う。アプリケーションはカードスロットのステータスを問い合わせ、この場合、各スロットのフラッグと、カード ID を返す。これらの情報をもとに画面に表示を行う。

6.2.3 WWW ブラウザのコントロールを行う例

この例では、環境サーバを使用して、WWW ブラウザアプリケーションをコントロールする例を示す。

たとえば、以下の図 6.2で示した WWW ブラウザは、環境サーバに対して以下のようなスクリプトを挿入する。

path `"/localhost/app/www/control/openurl"`

script `"set ret $arg\n"`

そして、callback 先は WWW ブラウザの URL を開くルーチンになっている。

以上のような状態で、環境サーバの`"/localhost/app/www/control/openurl"`という path 名のスクリプトを呼び出すことによって、WWW ブラウザに対して指定された URL を表示させるというコントロールを行うことができる。

実行画面例では、環境サーバシェルから、URL を指定して表示させる例を示している。

さらにこれを利用して、他のイベントをトリガーにして WWW ブラウザをコントロールするスクリプトを加えて挿入することもできる。例えば、

path `"/localhost/pccard/event/inserted"`

script `"cast /localhost/app/www/openurl`
`file://localhost/etc/msgs/cardinsertmsg.html\n"`

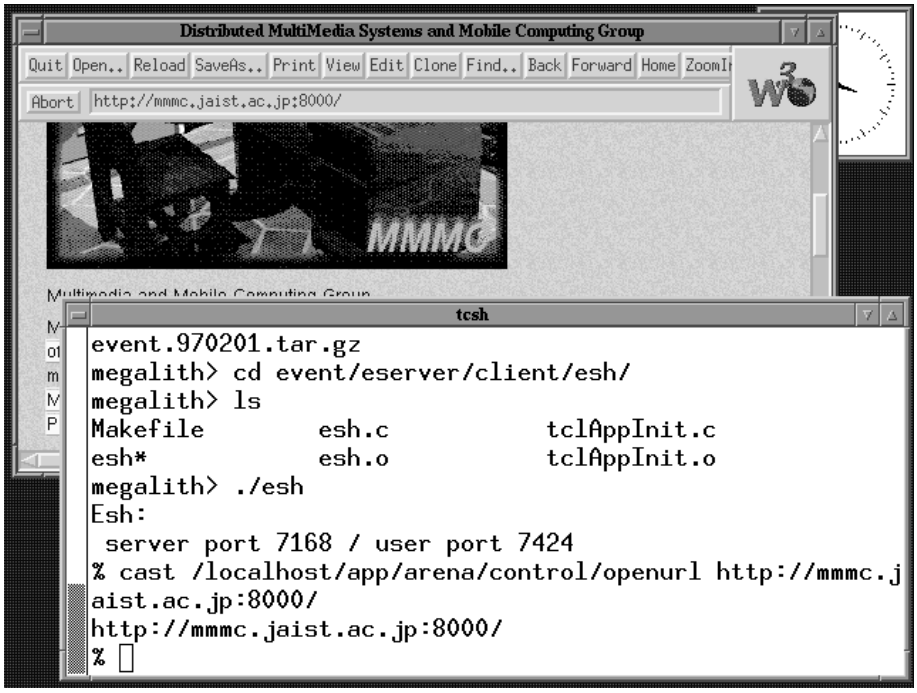

図 6.2: アプリケーションの例:esh と www ブラウザ

のように記述したオブジェクトを挿入すると、カードの挿入時に WWW ブラウザにその旨のメッセージを自動的に表示させることが出来る。同様に、WWW ブラウザの HTTP キャッシュのコントロールや、インライン画像の表示・非表示のコントロールを行うこともできる。

6.3 評価と今後の予定

本研究で提案した環境サーバを用いたアーキテクチャを利用して、いくらかのサンプルアプリケーションの実装を行ってみた。これらのアプリケーションの実装から、以下のことが示される。

- イベントの種類やリソースの指定に関して、よりよいアブストラクションが提供できた。
- スクリプトでイベントフィルタ等の指定は、大変記述性が高い。
- アプリケーション間のメッセージ通信の用途にも十分使用できる。

今後は、様々なアプリケーションに環境サーバを適応していき、さらに有効性を確かめたい。

第 7 章

まとめと今後の課題

本研究では、移動計算機環境などの計算機の状況が動的に変化するような環境において、それらの状況に適応するアプリケーションの構築法について提案し、実際にシステムのプロトタイプを実装してその有効性を確かめた。まず最初に、デバイスの変化など動的な環境変化に対応できるように、Real-Time Mach マイクロカーネルに対して拡張をおこなった。そして、カーネル内のイベントを効率的にユーザレベルサーバに伝達するためのイベントマネージャを実装した。そして、アプリケーションプログラムに対してイベントのフィルタリング等の機構を提供する環境サーバを実装し、いくらかのサンプルアプリケーションを実装し、その有効性を確かめた。

本研究で提案したシステムの特徴をまとめると、

- カーネル等のシステムのよりよいサポート
- イベントのフィルタなどのスクリプト言語による記述
- path 名表記による、イベントやリソース、コントロールの抽象化
- アプリケーション記述のためのよりよいインターフェイス

といった点が挙げられる。

現在では、状況に適応するアプリケーションを構築するためのアプローチとして、カーネルサポートなどのシステム側、そしてアプリケーションを実際に記述するための記述言語側とで、様々な試みが行われている。しかしながら、これらのアプローチを結びつけ、1 つのシステムとしてのフレームワークを提供するものは少ない。

本研究では、カーネル等のシステムサポートについての枠組と、さらにそれらのシステムリソースをアプリケーションが利用するための、イベント配送やリソースのアブストラクションの手法について議論し、プロトタイプシステムの実装した。これらの過程により、状況に適応するアプリケーションを記述することの有用な点を確かめ、それに適したシステムの提案を行うことができた。

今後は、環境サーバが提供する枠組を、より大規模なアプリケーション群へ適応させてていく試みが必要であると考える。本研究では触れなかったが、分散環境でのイベントの処理などは今

後の大きな課題である。さらに、より効率的なイベント配送の方式や、アプリケーション記述のための枠組だけではなく、システム記述のためのイベント配送システムやリソースの抽象化手法についても発展させていきたい。

Appendix A

カーネルエクステンション

本研究で開発した、PC カード・APM 機能のカーネルサポート部分は、我々の研究室で開発している Real-Time Mach マイクロカーネルの拡張機能として提供される。

A.1 PC カードデバイスインタフェイス

以下に PC カードスロットに対するカーネルインターフェイスを記述しておく。

- スロット数の取得

```
kern_return_t device_get_status(mach_port_t device_port,
                        CSC_GET_MAX_SLOT,
                        int          *status,
                        unsigned int *count);
```

status: slot 数

- スロットビットの取得

```
kern_return_t device_get_status(mach_port_t device_master_port,
                        CSC_GET_SLOT_BIT,
                        int          *status,
                        unsigned int *count);
```

status: slot bit

- スロットのカード情報取得

```
kernl_return_t device_get_status(mach_port_t device_port,
                        CSC_GET_CLIENT_INFO,
                        pccard_info_t *status,
```

```
                        unsigned int  *count)
status: slot bit
```

カード情報の取得の際に使われる構造体を以下に示しておく。

```
typedef struct pccard_info {
        int             socket,
        unsigned char   device_name[MAX_DEVICE_NAME_LEN],
        int             unit,
        int             device_type,
        int             slot_status,
        unsigned char   card_ident[MAX_CARD_IDENT_LEN],
        int             card_ident_len
}
```

さらに、device_type , slot_status に関しては、以下の種類がある。

```
device_type :
                PCCARD_DEV_TYPE_NET_ETHER
                PCCARD_DEV_TYPE_NET_WIRELESS
                PCCARD_DEV_TYPE_DISK
                PCCARD_DEV_TYPE_COM
                PCCARD_DEV_TYPE_VIDEOCAPTURE

slot_status :
                CSF_SLOT_UP
                CSF_SLOT_RUNNING
                CSF_SLOT_EMPTY
```

- スロットのアップ

```
kern_return_t device_set_status(mach_port_t device_port,
                      CSC_SLOT_UP,
                      int      *status,
                      int       count );
```

- スロットのダウン

```
kern_return_t device_set_status(mach_port_t device_port,
```

```
                                CSC_SLOT_DOWN,
                                int      *status,
                                int       count );
```

A.2 イベントマネージャ

- リプライポートの設定

```
kern_return_t eventm_set_port(mach_port_t port);
port: Mach Port
```

- メッセージの受信

```
kern_return_t eventmanager_delivery_wait(mach_port_t port,
                                int *source, int *type);
port  : 受信port
source: デバイス番号
type  : イベントタイプ
```

Appendix B

環境サーバの仕様

B.1 構造体

アプリケーションから環境サーバへのリクエストや、内部でのスクリプトなどの管理には、以下の構造体が使われる。

```
typedef struct eserver_user {
        mach_port_t          port;
        int                  cmd;
        unsigned char        path[ESERVER_PATH_MAXLEN],
        unsigned int         *callback,
        unsigned char        scripts[ESERVER_SCRIPTS_MAX_LEN],
        unsigned char        arg[ESERVER_ARG_MAX_LEN],
        unsigned char        ret[ESERVER_RET_MAX_LEN]
} eserver_user_t ;

cmd:     ESERVER_CMD_NULL
         ESERVER_CMD_PLUGIN
         ESERVER_CMD_REMOVE
         ESERVER_CMD_EVAL
```

B.2 システムインターフェイスモジュール

環境サーバ内での、システムインターフェイスモジュールの管理には、以下の構造体が使われる。

- 構造体

```
typedef struct {
```

```
        module_type_t       type,
        module_func_t       (*module_init)(),
        module_func_t       (*module_terminate)(),
        module_p_t           *next,
        cthread_t           thread,
        mach_port_t         port
} module_t ;
```

- サポートファンクション

```
ret_t module_install_scripts(eserver_user_t *request);
ret_t module_tcl_extension(vm_offset_t *initp);
ret_t eserver_request( int null, int null, eserver_user_t *request);
```

B.2.1 モジュールの追加

esrver/modules/ の下に追加し、
eserver/modules/module_conf.h にエントリを追加する。

B.3 ユーザライブラリ

アプリケーションが環境サーバを利用するためのライブラリは、C 言語用と Tcl スクリプト言語用のものを用意した。

- C 言語

```
ret_t eserver_user_init(mach_port_t *server_port,
        mach_port_t *user_port);

ret_t eserver_user_request(mach_port_t *server_port,
        mach_port_t *user_port, eserver_user_t *request);
```

- Tcl スクリプト言語

```
 % cast PathName Arg
```

Appendix C

サンプルプログラム

本文中で示したサンプルアプリケーションは、ソフトウエアパッケージ中の以下でしめされるディレクトリに格納した。

C.1 esh: 環境サーバシェル

eserver/client/esh/esh

C.2 cardmonitor

eserver/client/cardmonitor/cardmonitor

C.3 webcontrol

eserver/client/webmonitor/webmonitor

参考文献

[1] Personal Computer Memory Card International Association (PCMCIA), "PCMCIA STANDARDS Release 2", 1992.

[2] Personal Computer Memory Card International Association (PCMCIA), "PCMCIA SOCKET SERVICE SPECIFICATION Release 2.1", 1993.

[3] Personal Computer Memory Card International Association (PCMCIA), "PCMCIA CARD SERVICES SPECIFICATION Release 2.0", 1992.

[4] Micheal Bender, Alexander Davidson, Clark Dong, Steven Drach, Anthony Glenning, Karl Jacob, Jack Jia, James Kempf, Nachiappan Periakaruppan, Gale Snow, Becky Wong, "Unix for Nomads: Making Unix Support Mobile Computing", USENIX Mobile & Location-Independent Computing Symposium, 1993.

[5] B.R. Badrinath, Girish Welling, "Event Delivery Abstractions for Mobile Computing", Rutgers Univ. TR #LCSR-TR-242.

[6] Bill Schilit, Norman Adams, Roy Want, "Context-Aware Computing Applications", IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[7] Anthony D. Joseph, Joshua A. Tauber, and M. Frans Kaashoek, "Building Reliable Mobile-Aware Applications using the Rover Toolkit" Proceedings of the Second ACM International Conference on Mobile Computing and Networking(MobiCom'96),1996.

[8] John K. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley Publishing, 1994.

[9] Brian N. Bershad, Craig Chambers, Susan Eggers, Chris Maeda, Dylan McNamee, Przemyslaw Pardyak, Stefan Savage, and Emin Guun Sirer, "SPIN – An Extensible Microkernel for Application-specific Operating System Services", 6th ACM SIGOPS European Workshop, 1994.

[10] 中島 達夫, "Real-Time Mach の現状と将来", JAIST Research Report, IS-TR-94-20S, 1994.

[11] 保木本 晃弘, 中島 達夫, “適応可能なアプリケーションを構築するためのソフトウェアアーキテクチャ”, 日本ソフトウェア科学会 第 12 回大会,1995.

[12] 保木本 晃弘, 中島 達夫, “状況に依存した計算を実現するオブジェクトモデル”, ソフトウェア科学会 第 11 回オブジェクト指向計算ワークショップ WOOC95, 1995.

[13] 天野 憲樹, 渡部 卓雄, “メタレベルアーキテクチャによる移動計算機環境のための言語”, ソフトウェア科学会 第 13 回全国大会, 1995.

God’s in his heaven –
All’s right with the world.