モーバイルコンピューティング 3－12
（1997. 12. 5）

適応可能アプリケーションの構築を支援する環境サーバ

中島 達夫, 会津 宏幸, 小林 勝, 嶋本 堅司
石川県能美郡辰口町旭台 1-1
北陸先端科学技術大学院大学

いつでもどこでも移動計算機を介して様々なサービスにアクセスすることを可能とするモーバイルコンピューティングやあらゆる場所に計算機を埋め込むことで様々な物体をプログラマブルにするユビキタスコンピューティングは, 従来実現することができなかった新しいアプリケーションを構築することを可能とする. このような環境では, 環境が提供する様々な情報を利用することにより, アプリケーションの振舞を実行環境に適応する必要がある.

本論文では, 適応的なアプリケーションを構築するための基盤ソフトウエアとして重要なコンポーネントである環境サーバに関して述べる. 環境サーバは, 計算機環境に関する様々な情報をアプリケーションが統一的にアクセスすることを可能とする. そのため, 環境サーバを用いることにより, 適応的なアプリケーションをよりシステマティックに構築することが可能となる.

Environment Server: An Infrastracture for Distributed Computing

Tatsuo Nakajima, Hiroyuki Aizu, Masaru Kobayashi, Kenji Shimamoto
Japan Advanced Institute of Science and Technology

Various new types of applications can be available in future computing environments such as mobile computing environments and ubiquitous computing environments. In these environments, computing environments can be changed dramatically, and applications may be migrated among computers that have drastically different hardware configurations. Therefore, these applications should be adapted to various computing environments that may be have dramatically different characteristics for their efficient executions, and the adaptation requires that the application can access information about computing environments through a uniform interface.

In this paper, we propose an *environment server* that is important as a basic infrastructure for building adaptive applications for future computing environments. The environment server manages various information about computing environments in an integrated fashion, and applications can access these information through its well defined interface. This makes it possible to build adaptive applications with a systematic framework.

1 はじめに

いつでもどこでも移動計算機を介して様々なサービスにアクセスすることを可能とするモーバイルコンピューティング [3, 13] やあらゆる場所に計算機を埋め込むことで様々な物体をプログラマブルにするユビキタスコンピューティング [1, 11] は, 従来実現することができなかった新しいアプリケーションを構築する可能性を提供する.

例えば, モーバイルコンピューティング環境において位置情報などの環境情報を用いることにより, ユーザが自分の近くの好みのレストランを検索したり, 友人との適切な通信手段を友人の現在の状況に応じて選択することができるようになる [8, 9]. また, 環境に埋め込まれた計算機を利用することで, ユーザに関する様々な情報を自動的にモニタリングすることが可能となる. これにより, ユーザが今何をやっているかに従いアプリケーションの振舞を動的に変化させることが可能となる. つまり, ユーザの位置情報やユーザの振る舞いをモニタリングすることで獲得した情報を用いることによりユーザの意図が明確になり, ユーザが何をしようとしているかをアプリケーションが予測することで実行を最適化することが可能となる.

このような計算機環境は, 様々な局面でユーザの仕事をサポートしたり, 情報をユーザの好みや現在の状況に応じてパーソナル化することを可能とする. つまり, あらゆるものを計算機を利用することで拡張することが可能となる. このような概念として, オフィス内の様々なものに計算機を埋め込むことで, 実世界と計算機の仮想世界を融合するアクティブオフィスが提案されている [4]. この考え方を拡張し, 世界中のあらゆるものを計算機を用いて拡張する (アクティブ化する) ことにより, アクティブホーム, アクティブユニバーシティ, アクティブシティなど様々な環境を拡張することが可能となる. そのためには, 計算機環境に関する情報の統一的な管理が重要なものとなる.

一方, ハードウエアやネットワーク技術の飛躍的な向上にともない, 機能や性能の異なる様々な計算機が多様な通信メディアを介してネットワーク上に接続されるようになってきている. また, モーバイルエージェントやモーバイルアプリケーションなど, 様々な構成を持った計算機間をプログラムが移動したり [2], 動的にハードウエア構成が変化するシステム上のプログラムの実行について考える必要性を生じてきた [5, 13, 7, 12].

このような環境に対処するためには, プログラムを実行するハードウエア環境などの実行環境に関する情報やシステムの負荷などの様々な計算機環境に関する情報を取得し, アプリケーションを実行環境に適応することが必要となる. 例えば, CPU の性能, メモリ容量や使用可能なデバイスの機能や性能といった計算機のリソースに直接関わる情報や, 移動計算機環境でよく利用される位置情報や通信回線使用料などリソースに間接的に関わる情報などを取得することができれば, アプリケーションは計算量やデータの転送量などを調整することにより, アプリケーションの実行を最適化することが可能となる.

しかし, 従来のオペレーティングシステムは計算機環境に関する情報を非常にアドホックな形で提供してきた. 特に, リソースに関する情報へのアクセスは, オペレーティングシステムによってそれぞれ固有のプリミティブを用いて実現されている. より悪いことに, リソースの種類が異なると, 異なるインタフェースが提供されているため, 計算機環境に関する情報を利用するアプリケーションをシステマティックに構築することが非常に困難なものとなってしまった. その結果, 環境に適応可能なアプリケーションを構築することが困難であった.

本論文では, 適応的なアプリケーションを構築するための基盤ソフトウエアとして重要なコンポーネントである環境サーバに関して述べる. 環境サーバは, 計算機環境に関する様々な情報をアプリケーションが統一的にアクセスすることを可能とする. そのため, 環境サーバを用いることにより, 適応的なアプリケーションをシステマティックに構築することが可能となる. 我々は, 環境サーバはのプロトタイプを, Real-Time Mach[10] 上に構築し, 適応的なモーバイルアプリケーションを構築するためのツールキットであるサービスプロキシツールキットなど, 適応的なアプリケーションを構築するプロジェクトの重要なコンポーネントとして用いている.

2 環境に適応するソフトウェア

2.1 動的に変化する計算機環境

インターネットに代表される分散環境の発達は, 従来の計算機環境に大きな変革をもたらした. グループ会議システムなどの分散環境上のアプリケーションでは, 相手に最も近い計算機や現在使用している計算機にデータを送信する必要があるため, 相手や計算機の位置情報のようなリソースに間接的に関わる外部の環境情報の取得が必要である. また, 時間制約を保証するようにデータを正しく送信するためには, その時点で使用している通信メディアの実効バンド幅や遅延時間, パケット損失率といった計算機のリソースに直接関わる環境情報の取得が必要である. 以上のように, 計算機環境は時事刻々と大きく変化し, アプリケーションはその変化を考慮しないと実行効率が大変悪いものとなってしまう.

このようなアプリケーションの実行に必要な計算機環境に関わる情報には, 通信メディアの実効バンド幅やバッテリー容量などの計算機のリソースに直接関わる情報や, 位置情報や通信回線使用料などリソースに間接的に関わる外部の情報がある. これらの情報には計算機が起動中全く変化しない静的な情報と環境に応じて変化する動的な情報がある. 上記のグループ会議システムの例でも見られるように, 分散環境上では動的に変化する情報が増えつつある. 本論文では, 以上のようなアプリケーションの実行に影響を与えうる情報を計算機環境に関する情報と呼ぶ. 計算機環境に関する情報が動的に変化する環境では, それらの計算機環境に関する情報をアプリケーションが取得して, その情報を用いてアプリケーションの実行効率を最適化することが必要となる.

2.2 一貫した環境情報の扱いの必要性

前節で述べたように, 動的に変化する計算機環境に対応していくためには, 計算機環境に関わる多様な情報を取得し, それらをアプリケーションに提供する必要がある. しかし, 従来の方法では, 個々の計算機上で環境に関する情報をどのように表現するかに関して全く統一されていなかったため, 情報を取得するためのインタフェースも個々の計算機やリソースの種類に依存してしまっていた. そのため, 新たな計算機環境に関する情報を考慮する場合は, 関係するすべてのコンポーネントを考慮して実装を変更する必要が生じ, シ

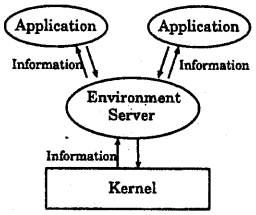

図 1: 環境サーバの構成

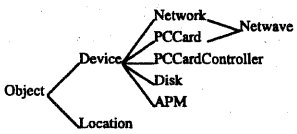

図 2: 環境情報のクラス階層の構成例

ステムの拡張が大変困難なものとなってしまった. また, 環境情報の取得方法に一貫性がない場合, 情報の種類毎に扱いを変更する必要があるため, アプリケーションの作り方がアドホックなものとなってしまった. そのため, 環境の変化にシステマティクに対応する適応可能アプリケーションを構築することが困難であった.

そのため, 動的に変化する環境に適応可能なアプリケーションは計算機環境に関係する情報を統一的に扱うことができるようなシステムコンポーネントを必要としている. このコンポーネントを用いて, アプリケーションは, 現在使用中の通信メディアの実効バンド幅や通信コストを問い合わせたり, バッテリー残量が少なくなったことを通知してもらうことが可能となる. また, このコンポーネントは, 計算機環境に関する情報の管理を統一的な方法で行なうため, 新しい環境情報をコンポーネントに追加することが容易となる. その結果, すべての環境情報を統一したインタフェースによりアクセスすることが可能となるので, システマティックなフレームワークを用いて適応可能なアプリケーションを構築することが可能となる.

3. 環境サーバの設計

3.1 概要

環境サーバは計算機環境に関する情報を統一的に管理するデータベースである. 環境サーバに対するインタフェースは統一化されており, 様々な計算機環境に関する情報を共通のインタフェースによりアクセスすることを可能とする. 我々のアプローチでは, 図 1に示されるように, アプリケーションは直接カーネルから計算機環境に関係する情報を取得せずに, 常に環境サーバを介してアクセスする.

環境サーバでは, 統一的なインタフェースを提供するため, すべての計算機環境に関係する情報を属性と属性値のペアとして表現する. 環境サーバでは, それぞれ環境に関する情報を複数の属性名と属性値のペアを持ったクラスとして表現し, それらのクラスを階層的に定義することを可能としている. また, 各情報は階層的なネーミングを用いて名前付けされる. そして, 名前階層とデータベースの検索を融合することにより, 属性の検索を簡単に行なうことができるようになる.

3.2 属性の定義

環境サーバは多様な情報を扱う必要があるため, 類似の情報をグループ化することにより全体を整理することが好ましい. 環境サーバでは, 計算機環境に関する情報を階層を持ったクラスとして定義することを可能とすることにより, 情報全体が矛盾しないように管理することを可能とする. クラスの定義は図 2に示すようにデバイスや位置情報などの情報の種類によって階層化される. 各クラスはクラス名と, 属性名と属性値の組からなる複数の属性を持つ. 我々の方式の特徴は, 以下に示すように, 属性値を直接指定するのではなく, 属性値を取得するための関数（属性取得関数）, 変更するための関数（属性変更関数）, 初期化のための関数（属性初期化関数）などの属性値を操作するための関数を定義する.

```
class クラス名： スーパクラス名1, ....
   (属性名1  (取得関数1  変更関数1  初期化関数1))
   (属性名2  (取得関数2  変更関数2  初期化関数2))
   (属性名3  (取得関数3  変更関数3  初期化関数3))
```

環境サーバでは, 上に示したように, 必要に応じて複数のクラスを継承することにより新しいクラスを作ることができる. クラス継承により継承されるのは属性名だけである. つまり, 取得関数, 変更関数, 初期化関数などは継承されずに, 新しいクラスを定義する毎にすべての属性名に対して取得関数, 変更関数, 初期化関数を定義する必要がある. この方式は, CORBA などで用いられているインタフェースの継承と同様, 多重継承に関する様々な問題を避けることが可能である.

以下に, クラスの定義例として無線ネットワークの PC カードである Netwave を定義するクラスを考える. Netwave クラスは PCCard クラス, Network クラスを順に継承することにより作られる. つまり, Netwave クラスは図 3に示すように PCCard クラスを継承することで Tuple 属性を継承する. また, PCCard クラスが Network クラスを継承することで, MACaddress, Bandwidth, ErrorRate の各属性を継承している. この図では, 取得関数, 変更関数, 初期化関数の定義は省略してあるが, これらの関数は前に述べたようにクラス継承により継承されないため, 各クラスの各属性毎に独立に定義する必要がある.

図 4は, Netwave クラスをインスタンス化した MyNetwave オブジェクトを表している. この図で示された属性値は Netwave クラスで定義された各属性名毎の属性取得関数を用いて呼び出すことができる. また, 属性変更関数を用いることで変更することができる.

3.3. 属性のネーミング

インスタンス化された計算機環境に関する情報を表現するオブジェクトは, 図 5で示すように階層的なネーミングを用いてアクセスすることを可能とする. 環境サーバが提供するネームスペースは通常の階層型ネームスペースと異なり, ネームスペースと問い合わせを融合することにより属性の問い合わせを扱いやすいものとしている.

図 5に示す Device, PCslot, Netwave, EtherLink, SunDisk はインスタンス化されたオブジェクトであ

```
class Network : device
    MACAddress      // MAC アドレス
    Bandwidth       // バンド幅
    ErrorRate       // エラー率

class PCCard : device
    Tuple           // PC カードのタプル情報

class Netwave : Network,PCCard
    DeviceName      // デバイス名
    Type            // デバイスタイプ
    Tuple           // PC カードのタプル情報
    MACAddress      // MAC アドレス
    Bandwidth       // バンド幅
    ErrorRate       // メディアのエラー率
```

図 3: Netwave クラスの構成

```
MyNetwave : class Netwave
  DeviceName      xnw0
  Type            "Xircom Netwave"
  Bandwidth       1Mbps
  MacAddress      xx-xx-xx-xx-xx-xx-xx-xx
  Tuple           "xxxxx"
  Error Rate      xx
```

図 4: Netwave オブジェクト

るとする. この例では, ネットワークカードの Netwave と EtherLink, フラッシュメモリカードである SunDisk が PC カードスロット PCslot に挿入されていることを示している. ここでは, PC カードをスロットから抜き差しすることにより, PCslot の下位階層のオブジェクト名が動的に変化する. この例では, 現在 3 つのカードが PCslot に挿入されているが, EtherLink カードを抜くことにより PCslot の下位に存在する EtherLink オブジェクトは消滅する. また, 同じカードを挿入することにより PCslot の下位に EtherLink オブジェクトが現れる.

図 5において Netwave オブジェクトの属性を問い合わせる場合は/Device/PCslot/Netwave に問い合わせる. 例えば, /Device/PCslot/Netwave に対してバンド幅属性を問い合わせることにより, Netwave オブジェクトのバンド幅を知ることができる. また, 以下に述べるように, 属性名への問い合わせを階層的ネームスペースと融合することにより, 詳しいオブジェクト名を知らなくても属性を問い合わせることが可能となる. 環境サーバでは, 階層型ネームスペースの葉だけではなくノードも属性を持つオブジェクトである. この階層型名前空間では, 下位に存在するすべてのオブジェクトの持つ属性が上位のノードの属性として集約される. 例えば, 図 5に示す例では, /Device/PCslot は, Netwave, EtherLink, SunDisk の 3 つのオブジェクトが持つすべての属性を集約している. つまり, /Device/PCslot に対して size 属性を問い合わせると, SunDisk オブジェクトの size 属性の属性値が返される.

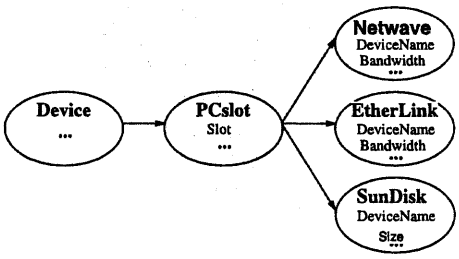

図 5: 問い合わせ名前空間の例

しかし, 集約をサポートした環境サーバの問い合わせ機構は, 1 つの問い合わせに対して複数のオブジェクトの同一名の属性がマッチする場合を考慮する必要がある. 例えば, 図 5に示す例では, Bandwidth 属性を/Device/PCslot に対して読み出すことを問い合わせると, Netwave と EtherLink の両方の Bandwidth 属性の属性値が返えされることになる.

環境サーバはアプリケーションに返す属性が複数存在する時, QueryType を問い合わせ時に指定することにより適当な属性を選択することを可能としている. QueryType には All:すべての属性が返る, Best:最も良い値の属性が返る, Worst:最も悪い値の属性が返る, Any:属性のうちの任意の 1 つが返る, の 4 つのオプションから選択することが可能である.

図 5を用いて QueryType を使用した例を以下に示す. ここでは, /Device/PCslot に対して Bandwidth 属性を問い合わせた場合を示している.

All: ((Netwave (Bandwidth 1Mbps))
(EtherLink (Bandwidth 10Mbps)))

Best: (EtherLink (Bandwidth 10Mbps))

Worst: (Netwave (Bandwidth 1Mbps))

Any: (Netwave (Bandwidth 1Mbps)) or
(EtherLink (Bandwidth 10Mbps))

属性値を比較することができない場合は, ここで示した QueryType のうち, Best と Worst を用いることはできない. そのような属性に足して, Best や Worst を指定して属性値をアクセスした場合は, エラーが返される.

3.4 属性の操作

アプリケーションは環境サーバが提供する属性をアクセスするための統一的なインタフェースを用いてオブジェクトの属性のアクセスを行なう. アプリケーションは, 各オブジェクトの属性を [属性名@オブジェクト名] というシンタクスを用いてアクセスする. ここで, 属性名は値を知りたい属性の属性名であり, オブジェクト名は前節で説明したオブジェクトを示す階層的な名前である. オブジェクト名が階層ネームスペースのノードである場合は, 前節で説明したように下位のオブジェクトが持つ属性は上位のオブジェクトに集約される.

環境サーバのインタフェースは, 属性をアクセスするため, 以下に示す 3 つの関数を提供する.

属性値 = **Get**(属性, QueryType)
エラー値 = **Put**(属性, 属性値)
エラー値 = **Register**(条件, handler, type)

ここで, type は以下のパラメータを指定できる.
条件を満足している間に通知する
条件を満足するようになった時に通知する
条件を満足しなくなった時に通知する

Get 関数は指定された属性の属性値取得関数を呼び出すことでオブジェクトから属性値を取り出す. Put 関数は属性の属性値変更関数を呼び出すことで, 属性値をアーギュメントとして与えられた値に変更する. Register 関数は環境サーバによって呼び出されるコールバックハンドラを登録する. 登録されたコールバック関数は, アーギュメントにより指定された条件が type で指定された状態になった時に呼び出される. Get と Put 関数で用いられる属性値や Register 関数で指定される条件はすべてストリング列として定義される.

次に, Register 関数で指定される条件式がどのように指定されるかに関して述べる. 我々のシステムでは, [属性名@オブジェクト名] と<, <=, >, >=, ==などの条件オペレータ, &や || などの論理オペレータを使用することができる.

以下に, 条件式の例を示す.

(1) [Bandwidth@/Device/PCslot/Netwave] <= 1Mbps

(2) [Bandwidth@/Device/PCslot] <= 1Mbps

(3) [BatteryCapacity@/Device/APM] < 10

(4) [Power@/Device/APM] == Battery

(1) は Netwave のバンド幅が 1Mbps 以下という条件を定義している. (2) は, PC カードに挿入されているネットワークのバンド幅が 1Mbps 以下という条件を定義している. 次の, (3) は, バッテリ残量が 10%以下である条件を示している. 最後の (4) は, 移動計算機がバッテリ駆動しているという条件を定義している.

イベントの発生を定義することを許すと, 2 つの条件を&を用いて結合した場合にイベントの発生時間を考える必要があるため, &の成立についての定義が難しくなる. そのため, 我々のシステムでは, 属性値として PC カードの挿入のようなイベントを指定することを許していない. 例えば, PC カードのスロット 1 への挿入は, " [slot1@/Device/PCCard] != empty" の条件で, Register のアーギュメントの type を" 条件を満足する時に通知" にすることにより表現することが可能となる.

4 環境サーバの実装

4.1 環境サーバの構成

図 6は環境サーバの構造を示している. 環境サーバは以下の 4 つのモジュールから構成される.

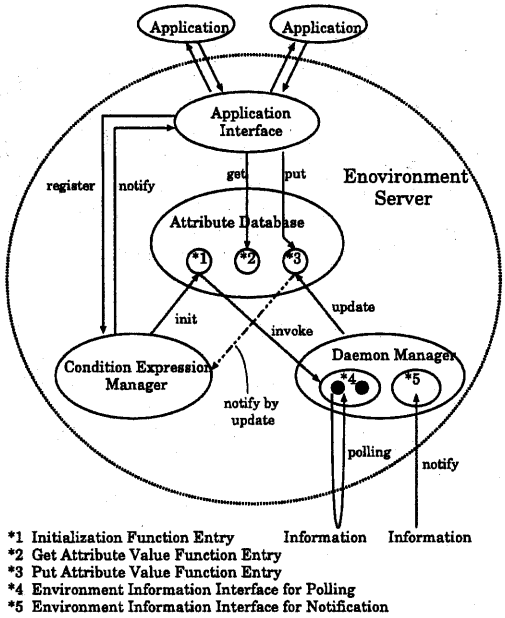

図 6: 環境サーバの構造

(1) アプリケーションインタフェースモジュール

条件式の登録, 属性値の取得・変更といったアプリケーションからの要求を受け付けるインタフェースモジュール.

(2) 属性データベースモジュール

各オブジェクトの属性名と属性値をデータベースとしてメモリ内に管理するモジュール. 各属性ごとに用意された初期化関数, 属性値取得関数, 属性値変更関数を環境サーバ内部で属性を操作するために保持している.

(3) 条件式管理モジュール

アプリケーションから Register 関数により登録された条件式を管理するモジュール. 条件式が真となった時に, Register 関数で定義されたコールバックハンドラが呼び出される.

(4) デーモン管理モジュール

計算機環境に関する情報を定期的に問い合わせて取得するデーモンと他のコンポーネントから送られたイベントを管理するデーモンを管理するモジュール. これらのデーモンは各属性の初期化関数により起動され, 定期的にシステムの状態をポーリングしたり, 他のコンポーネントからイベントを受け取ることにより環境サーバが管理する属性値を変更する.

4.2 属性データベース

環境サーバが起動されると, 現在環境サーバがサポートしている計算機環境に関する情報を定義している初期化ファイルが読み込まれ, 属性データベースが構築される. 初期化ファイルには, クラス名の階層や各クラスの属性, オブジェクト名の集約階層, 環境情報の情報源となる場所（デバイスならデバイスドライバの場所）などが記述されている. なお起動時は静的な情報しか参照できず, 動的な環境情報の取得は初期化関数の要求に基づきデーモンが起動されることにより初期化される. 本節で述べた処理は属性管理モジュールにより行なわれる.

4.3 属性値の取得・変更

属性値を取得する場合, アプリケーションは" 属性名@オブジェクト名", QueryType を引数に Get 関数を呼び出す. 要求を受け取ったアプリケーションインタフェースは, 引数とともに属性データベース内にある指定された属性名@ オブジェクト名に対応する属性値取得関数を呼び出す. もし指定された属性がそのオブジェクト自身に存在しない場合, 集約しているオブジェクトをチェックする. そして, もし複数のオブジェクトに指定された属性が存在する場合, アプリケーションに返すべき属性値を QueryType に従って選択する.

また, 属性値を変更したい場合は, 属性名@オブジェクト名, 新しい属性値をアーギュメントとして Put 関数を呼び出す. これにより, 属性名@オブジェクト名に対応する属性値変更関数が呼び出され, 指定されたオブジェクトの属性値が変更される. Put 関数に指定されたオブジェクトに指定された引数がない場合は, 集約は用いられずエラーが返される. Get, Put 関数から取得関数, 変更関数への変換はアプリケーションインタフェースモジュールによりおこなわれる. また, オブジェクト名からインスタンスオブジェクトへのポインタへの変換や, 名前空間内のノードオブジェクトに対する問い合わせの管理は属性管理モジュールによりおこなわれる.

4.4 アプリケーションへの通知

アプリケーションが計算機環境に関する情報に変化が起きた時に通知される必要がある場合, 条件式, コールバック関数, 通知タイプを引数に Register 関数を呼び出す. これにより, 条件が満足した時, アーギュメントで指定されたコールバックハンドラが呼び出される. 登録された条件式は, コールバックハンドラとペアにされ条件式管理モジュール内に格納される. type は 3.4 節で述べたように 3 種類定義されている. 条件式は条件式管理モジュールにより管理されており, オブジェクトと属性名からそれを含む条件を簡単に見つけることができるように, オブジェクト名と属性をキーにハッシュテーブルに登録されている.

初期化関数により起動したデーモン管理モジュール内のデーモンが環境情報の変化を認識した場合, 属性変更関数が呼ばれる. また, 属性変更関数はアプリケーションから Put 関数を呼び出すことによっても起動される. この時, 変更する必要がある属性の属性名をアーギュメントにして条件式管理モジュールを呼び出す. 条件式管理モジュールは属性名をハッシュ関数のキーとしてその属性を含む条件を検索する. 次に, 変更があったオブジェクトの属性名を含むすべての条件の評価を行なう. そして, 条件が真となる条件が存在する場合は, Register の type に従いコールバックハンドラを呼び出す. もし, type パラメタの指定が条件が" 満足する間定期的にコールバック関数を呼び出す" という指定になっている場合は, 新しいデーモンを起動し, デーモンは定期的にコールバックハンドラを起動する. このデーモンは属性の変化により条件式を満足しなくなった時に消去される.

5 結論と今後の課題

本論文では, 計算機環境に関する情報を統一的に管理する環境サーバに関して述べた. 環境サーバは, モーバイルコンピューティング環境やユビキタスコンピューティング環境の実現に必要な適応可能なアプリケーションをシステマティックに構築するための基盤ソフトウエアとなる. 本論文では, 計算機環境に関する情報の表現法, 情報の名前づけ, 問い合わせのためのインタフェース, 実装法に関して述べた. 環境サーバは現在 Real-Time Mach オペレーティングシステム上でプロトタイプが稼働している. また, 第 5 節で述べたようなアプリケーションにより使用されている.

今後の課題としては, Register 関数により登録する条件式の記述力の強化, 同時に複数の条件が満足する場合のハンドラの呼びだし順序などに関して検討する予定である. また, 大規模なシステムにおける環境サーバの情報の扱いなどに関しても検討する予定である.

参考文献

[1] "Special Issue on Computer-Augmented Environment", Communications of the ACM, Vol.36,No.7, 1993.

[2] F.Bennett, T.Richardson, A.Harter, "Teleporting - Making Applications Mobile", In the Proceedings of Workshop on Mobile Computing Systems and Applications, IEEE, 1994.

[3] G.H.Forman and J.Zahorjan:, "The Challenges of Mobile Computing", IEEE Computer, Vol.27, No.4, 1994.

[4] A.Harter, and A.Hopper, "A Distributed Location System for the Active Office", IEEE Network, Jan/Feb 1994.

[5] Akihiro Hokimoto, Kuniaki Kurihara, Tatsuo Nakajima, "An Approach for Constructing Mobile Applications using Service Proxies", The 16th International Conference on Distributed Computing Systems(ICDCS'96), May, 1996.

[6] H. Ishii, and B. Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms", In Proceedings of CHI'97, ACM, March 1997.

[7] B. N. Schilit, M. M. Theimer, and B. B. Welch: "Customizing mobile applications", In *Proceedings of the 1st USENIX Symposium on Mobile & Location-Independent Computing*, August, 1993.

[8] B.Schilit, N.Adams, R.Want, "Context-Aware Computing Applications", In the Proceedings of Workshop on Mobile Computing Systems and Applications, IEEE, 1994.

[9] M.Spreitzer, and M.Theimer, "Providing Location Information in a Ubiquitous Computing Environment", In Proceedings of 14th ACM Symposium on Operating System Principles, 1993.

[10] H.Tokuda, T.Nakajima, P.Rao, "Real-Time Mach: Towards a Predictable Real-Time System", the USENIX Mach Workshop, October, 1990.

[11] M.Weiser, "Some Computer Science Issues in Ubiquitous Computing", CACM, Vol.36, No.7, 1993.

[12] G. Welling and B. R. Badrinath: "A Framework for Environment Aware Mobile Applications", In *the Proceedings of the 17th ICDCS Conference*, Baltimore, Maryland, May, 1997.

[13] 中島 達夫, "モーバイルコンピューティング" 無線 LAN アーキテクチャ, 第 5 章, 共立出版, 1997.