第XIII部

移動体通信環境

第 13 部
移動体通信環境

第 1 章　はじめに

PDA などの携帯端末の性能の向上と、移動通信機器の普及に伴い、移動先から携帯端末を利用したインターネットへのアクセス、いわゆるモバイルコンピューティングが活発に行われ、またその利用方法も多様化している。

Rover は、このようなネットワークを利用した移動計算機環境について研究するグループである。我々が研究の対象とする移動計算機は主にノート型 PC 等の十分な処理能力を持つ計算機であるが、PDA 等の比較的能力の低い計算機も視野に入れた議論を行っている。より具体的には、インターネット移動体通信機構 (LIN6, Mobile IP) の開発、ad-hoc NFS の開発、地理情報システムに関する議論、移動に伴う動的適応の枠組の開発などの活動が行われている。

本年度は、これまで Rover で議論してきた研究項目のうち、WIDE メンバーが積極的に開発した、アプリケーションによる動的適応をおこなうための枠組である CEIPS (Computing Environment Information Providing System) の提案と設計、そしてその通信環境に特化した実装である CM1 (Communication Monitor 1) の実装と評価について述べる。

第 2 章　CEIPS: 動的適応するアプリケーションのための枠組

2.1 概要

計算機システム上でプログラムが動作する際には、動作中に利用可能なハードウェアや通信路の状況などの環境、すなわち実行環境に影響を受ける。アプリケーションの実行環境の変化に対して、ミドルウェアを含むシステムによる変化の隠蔽をおこなっても、その適用範囲には限界があり、適用できないアプリケーションに関しては、各アプリケーションによる適応が不可欠となることが知られている [185]。しかしながら、現在の計算機は実行環境に関する情報をアプリケーションが利用することを考慮していないため、実行環境を取得する機構が存在しないか、たとえ存在しても特定のハードウェアに依存するなど一般性がない。このことが、実行環境に対して動的適応するアプリケーションを作成する際の問題となる。また、たとえ環境に関する情報を取得することができても、その環境への対応方法やアプリケーション間の協調などが問題となる。

本章では、アプリケーションによる動的適応をおこなうための枠組として、動的適応に必要となる多段階の情報を一元的に提供する、CEIPS (Computing Environment Information Providing System) の提案と設計、そしてその通信環境に特化した実装である CM1 (Communication Monitor 1) の実装と評価について述べる。CEIPS は通信路、電源、温度、利用者位置、物理的位置などさまざまな実行環境に対して動的適応するための汎用の枠組である。CEIPS では計算機の実行環境に関する情報を収集しそれを一元的にアプリケーションに提供する。その際、収集した無加工の情報、それを複数まとめて一般化した情報、さらに利用者の利用ポリシを反映した情報の 3 種類の情報を用意し、アプリケーションが必要なレベルの情報を一元的に利用可能にする。この機構により、アプリケーションによる環境適応が容易になり、かつ、アプリケーション間で利用者のポリシの共有が可能となる。

2.2 背景

計算機システム上でプログラムが動作する際には、動作中に利用可能なハードウェアや通信路の状況などの環境、すなわち実行環境に影響を受ける。そして、従来の計算機システムは、その実行環境がプログラム動作中にはごく限定された変化しかしないことを想定している。しかしながら、近年のモバイルコンピューティングや活線挿抜が可能なハードウェアの普及により、プログラムの動作中に、利用可能な補助記憶や通信インターフェース、通信能力など

の実行環境が劇的に変化するようになってきた。このため、実行環境の変化を想定していない従来の計算機システムやプログラムは環境変化に追従できず、その動作効率の低下や動作不良などを引きおこしてしまうという問題が生じている。

計算機上で利用されるプログラムは、そのプログラムの目的に応じてさまざまな項目で評価される。通常は複数の評価項目が存在し、どの項目をどの程度優先するかどうかや、最低限満たすべき要求などは、利用者や、その時点でのプログラムの利用目的によって異なる。最適な動作とは、与えられた実行環境の中で評価を最大にする動作であると定義できる。

たとえば、遠隔ログインプログラムでは主に応答性が評価尺度となる。このような対話型のシステムでは応答時間が 100ms から 200ms を越えると利用者は不快に感じることが知られており [81]、この応答時間を短くすることが高い評価となる。同様のことはインターネット電話でもいえる。インターネット電話では音声の明瞭さと遅延が大きな評価尺度となる。これらは音声のサンプリングレートと伝送による損失、および伝送や符号化、復号に要する遅延時間に定量化できる。一般にはこの二つの要素の充足は相反する関係にあり、明瞭さを高くするためにサンプリングレートを高くすると、転送データ量が増大して遅延時間が増加する。さらに、転送データ量が伝送路の限界を越えると遅延時間がいっそう長くなり、かつデータの損失なども生じるため、かえって明瞭さが低下する。したがって、利用者が許容可能な遅延時間を保ちながら、最大限の明瞭さを実現することが高い評価となる。

従来の、実行環境の変化がない、もしくは変化の少ない計算機システムにおいては、プログラムは設計時、もしくは実行開始時に与えられた環境を調査して実行環境を想定し、その環境に合わせた動作をおこなうことで最適な動作をおこなっていた。しかしながら、今日の実行環境の劇的な動的変化により、プログラムが実行開始時には最適な動作をおこなっていても、その後の環境変化によってそれが不適切な動作となり、評価が低くなってしまうという問題が生じている。

たとえば、狭帯域な無線によるネットワーク接続下で WWW ブラウザアプリケーションを快適に利用する技術として、WWW ページに含まれる画像データの品質を低下させ、転送データ量を削減する方法が提案されている [191]。しかし、この計算機が物理的な移動によって広帯域な LAN に接続された場合、十分な通信帯域があるにもかかわらずデータの品質を落とすことになり、最適な動作とはいえなくなる。

2.3 CEIPS の設計方針

2.3.1 CEIPS の目的

動的適応するアプリケーション、すなわち適応型アプリケーションは、利用者の要求を最大限に満たすように、変化する環境の中で常に動作を選択するアプリケーションである。たとえば、ネットワーク経由でビデオサーバからビデオ受信をおこなうアプリケーションの場合。利用者の要求は高画質、高フレームレートの画像を見ることであるといえる。この場合、クライアントの動画再生能力と、サーバとの間のネットワークの実効通信速度が問題になる。単純には動画再生能力の範囲で再生可能で、かつ通信速度を越えないデータ量のビデオを再生すれば良い。このとき、データ量を変化させるために画質を変化させるには、画像の符号化パラメタを変化させるだけではなく、一般にはその画質に適した符号化形式を用いることが効果的であり、そのためには適切な閾値によって符号化アルゴリズムの切替が必要になる。さらに、フレームレートと画質のどちらを優先するべきかなどの判断は、最終的には利用者のポリシによる。

計算機の実行環境には、動作中に利用可能なハードウェアや通信路の状態などの計算機資源に関するもののほか、計算機の物理的な位置や時刻などの情報も含まれる。適応型アプリケーションは、得た環境情報によって適応する。適応方法には、得られる数値を動作のパラメタとして利用する方法と、得られる数値や状況を、それに適した内部状態や利用アルゴリズムの選択要因として用いる方法がある。また、実際の適応には、一般には複数の環境要因が関係するため、複数の情報を組み合せて判断する必要がある。いずれの場合にも、利用者のポリシが最終的な判断材料となる。したがって適応動作は以下のようにモデル化できる。

1. 動作に影響を与える実行環境の変化を検知する
   - 必要な実行環境に関する情報を複数収集する
   - 収集した多くの情報を少数の情報に抽象化する

2. 利用者のポリシと得られた情報から次の動作を決定する
3. 動作を変更する
4. (動作の変更が環境に影響を与える)
5. 1 番目の動作に戻る

現在の計算機システムでは、これらの処理のうち、特に実行環境の変化の検知と、情報の収集をおこなう手段がほとんど存在せず、存在しても一元化されていないため利用しにくいことが問題である。したがって、実行環境が変化した際に、アプリケーションに対し必要な情報を提供する機構が存在すれば、適応型アプリケーションを作成することが容易になる。

また、利用者のポリシは利用者ごとに大きく異なるが、同一の利用者が利用する複数のアプリケーション間ではほとんどの場合は共有可能である。利用者がすべてのアプリケーションに対してポリシを設定するのは繁雑であり、望ましくないため、ポリシを共有する方法も必要となる。

そこで、CEIPS ではアプリケーションに対し、実行環境の変化の通知、実行環境に関する情報の提供、そしてポリシを共有する機構の提供をおこなうことを目的とする。

2.3.2 CEIPS に対する要求

CEIPS は以下のような要求を満たす必要がある。

特定の実行環境に依存しない

CEIPS がある種の実行環境下で動作しないと仮定すると、CEIPS を利用するアプリケーションはその実行環境下では適応不能になってしまう。したがって、CEIPS は特定の実行環境に依存せず、目的とする計算機システム上で適応型アプリケーションが動作する限りは、CEIPS も動作する必要がる。

アプリケーションからの利用が容易

適応型アプリケーションから CEIPS を利用することが、容易でなくてはならない。

拡張性がある

適応に利用される実行環境に関する情報は、計算機の応用分野の拡大や新たなハードウェアの登場とともに増加しつづけることが予想される。そのため、CEIPS には十分な拡張性が必要となる。

ポリシの判断に柔軟性がある

一般に人間の要求は計算機にとって複雑な要求となるので、利用者のポリシは時として非常に複雑になる可能性がある。したがって、利用者のポリシを判断する処理はさまざまな要求に対応できるよう、十分に柔軟な機構を持つ必要がある。

2.4 基本設計

2.4.1 基本動作

情報の選択方法

適応型アプリケーションが適応のために必要な情報はアプリケーションごとに異なる。CEIPS からアプリケーションに実行環境に関する情報を提供する際には、CEIPS の持つすべての情報をアプリケーションに提供し、アプリケーションに必要な情報を選択させる方法と、アプリケーションからどの情報が必要であるかを CEIPS に対して登録し、CEIPS からは選択された情報のみを提供する方法がある。

すべての情報をアプリケーションに提供する方法では、アプリケーション側で自由に情報を加工することができる。また、アプリケーションから情報を指定するためのオーバーヘッドが生じない。しかしながら、CEIPS が扱う情報が増加した際に、その情報が不要なアプリケーションに対しても送られ、CEIPS とアプリケーションとの間の通信量が増加するという問題点がある。

一方、アプリケーションからどの情報が必要であるかを CEIPS に対して登録する方法では、アプリケーションはどの情報を利用するかを明示的に指定しなければならない。また、アプリケーションから情報を指定するためのオーバーヘッドが生じる。しかしながら、CEIPS が扱う情報量が増加しても、その情報をアプリケーションが必要としなければ、同じ通信量で情報を伝達することが可能となる。

CEIPS が扱う情報量は今後とも増加しつづけることが予想されるため、CEIPS では、アプリケーションからどの情報が必要であるかを CEIPS に対して登録する方法を採用する。

通知方法

実行環境が変化した場合には、アプリケーションは何らかの方法でこれを検知しなければならない。

特に連続的に変化する実行環境に対し、動作を変更する必要のある閾値はアプリケーションによって異なるため、アプリケーションごとに異なる閾値を越えた時点で変化とみなす機構が必要となる。この実現方法としては、アプリケーションが CEIPS に対して定期的に環境の確認をおこなう方法、CEIPS からアプリケーションに対していかなる変化でも通知する方法、アプリケーションが CEIPS に対して条件を設定し、その条件が満たされると CEIPS からアプリケーションに対して通知される方法がある。

アプリケーションが CEIPS に対して定期的に環境の確認をおこなう方法では、CEIPS 側は単純に要求された情報を提供するだけで良く、アプリケーションが自由にその情報を組み合わせて条件を確認することができる。しかしながら、確認の周期が短いと CEIPS に対する負荷と通信量が増加し、周期が長いと環境変化に対する反応が遅くなるという問題がある。

CEIPS からアプリケーションに対していかなる変化でも通知する方法では、アプリケーションが自由に情報を組み合わせて条件を確認することができ、かつ環境変化に対して迅速な適応が可能となる。しかしながら、特に変化の激しい環境情報を利用している場合には、CEIPS とアプリケーションの間に不必要に多い通信が発生するという問題がある。

アプリケーションが CEIPS に対して条件を設定し、その条件が満たされるとアプリケーションに対して CEIPS から通知される方法では、CEIPS とアプリケーションの間の通信量は必要最低限となり、かつ、環境変化に対して迅速な適応が可能となる。しかしながら、アプリケーションでの判断条件が CEIPS に設定可能な条件に限定されるという問題点がある。

CEIPS では、アプリケーションが CEIPS に対して条件を設定し、その条件が満たされると CEIPS からアプリケーションに対して通知される方法を採用する。さらに、「変化した場合に通知する」という条件を付加することで、適切な条件が設定できない場合にはアプリケーション側で条件判断することを可能とする。

CEIPS の分散化

CEIPS が管理すべき実行環境に関する情報の扱いには、複数の計算機間で共有する方法と、各計算機でそれぞれ管理するという方法がある。

複数の計算機間で共有する方法では、全体としての情報取得に必要なコストが下がり、複数の計算機による観測による精度の向上も期待できる。しかしながら、計算機間の通信が可能かどうかによって CEIPS で処理できる情報が変化してしまうという問題がある。

各計算機でそれぞれ管理する方法では、各計算機単体で動作が完結するため、計算機間の通信状況などに影響されずに動作することが可能となる。しかしながら、全体としての情報取得に必要なコストは上昇してしまう。

CEIPS では環境によらずに動作することが求められているため、各計算機でそれぞれ情報を管理する方法を採用する。ただし、情報を取得するために通信することを妨げるものではない。

2.4.2 基本動作モデル

以上の議論をまとめた CEIPS の基本動作モデルを図 2.1 に示す。まず、CEIPS を利用するアプリケーションは必要な環境情報と、その変化の通知を受けるべき条件を CEIPS に登録する。CEIPS は設定された条件が満たされるとアプリケーションに登録された環境情報を通知する。アプリケーションはこの通知された情報により、動作を適切に変化させることができる。

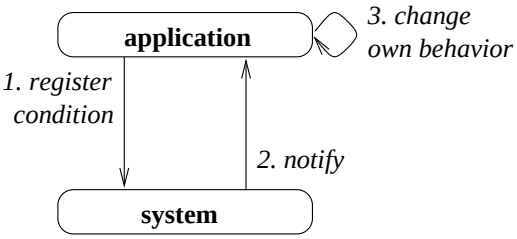

図 2.1.　CEIPS の基本動作モデル

2.4.3 実行環境情報の階層化

適応型アプリケーションが必要とする実行環境情報は多岐にわたり、また計算機の応用分野の拡大や新たなハードウェアの登場とともに増加しつづける。アプリケーションによってはこれらの非常に詳細な情報が重要な意味を持つことがある。しかし、多くのアプリケーションは詳細な情報は必要ではなく、抽象化し一般化した情報で十分である。アプリケーションが一般化した情報を利用する場合、その元となる情報の種類が増加してもアプリケーションには変更を加えることなく増加した情報を利用可能であ

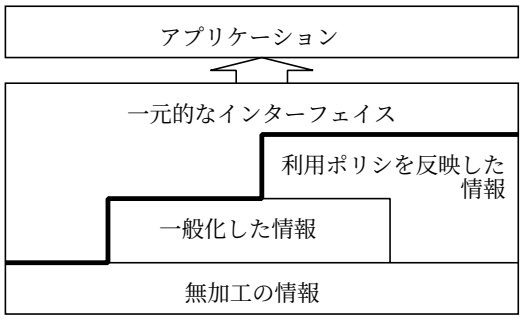

図 2.2. 3 種類の環境情報の依存関係

る。さらに、利用者のポリシも共有可能であることが多く、より高度な抽象化として利用ポリシを反映することが可能である。

そこで、CEIPS では無加工の情報、一般化した情報、利用ポリシを反映した情報の 3 種類の情報を提供する。これらの情報の依存関係を図 2.2 に示す。異なるレベルの情報を用意することで、アプリケーションは必要なレベルの情報を利用することが可能となり、適応型アプリケーションの作成が容易になる。また、利用ポリシを反映した情報を使うことで、ポリシの共有も可能になる。

2.5 CEIPS の設計

2.5.1 システム構成

CEIPS のシステムは、各種の環境情報源、情報を集中管理する環境デーモン、そして適応型アプリケーションの、3 つの要素によって動作する (図 2.3)。環境情報源は実行環境に関する情報を提供する。中央で動作する環境デーモンは CEIPS システムの中核であり、環境情報源から情報を収集し、必要な抽象化などの加工をおこない、これを一元的に適応型アプリケーションに提供する。適応型アプリケーションは必要な環境情報をサーバから受信し、その情報

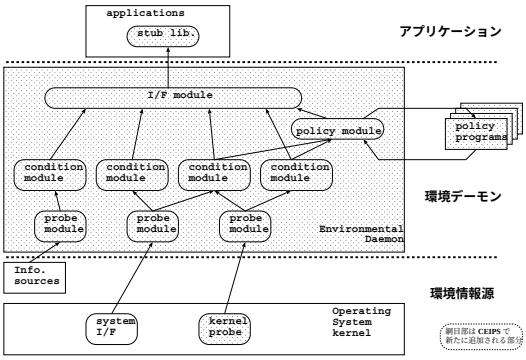

図 2.3. CEIPS のシステム構成

に従って最適な動作を選択する。

2.5.2 環境情報源

環境情報源は実行環境に関する情報を環境デーモンに提供するもので、情報の種類によって異なる。オペレーティングシステム等から情報を取得できるプリミティブが存在すれば、それらのプリミティブは環境情報源の一部となる。また、既存のオペレーティングシステムによって十分な情報が取得できなければ、カーネル内に組み込む情報取得機構、すなわちカーネルプローブを、新たにオペレーティングシステムに組み込む必要がある。

2.5.3 環境デーモン

「環境デーモン」は動的適応が必要な計算機システム上で動作するサーバプロセスであり、各種の環境情報源から情報を収集し、必要な抽象化などの加工をおこない、これを一元的に適応型アプリケーションに提供する。

環境デーモンは適応型アプリケーションと以下の 5 種類のメッセージを用いて通信する。同一の適応型アプリケーションから、複数の条件を登録できるように、登録要求を除くメッセージには登録時に環境デーモンによって一意に割り当てられる識別番号が付加される。

登録要求:

適応型アプリケーションが必要とする情報の種類と、その情報がどのような状態になった際に通知されるべきかという監視条件を環境デーモンに登録するための要求メッセージ。メッセージには、適応型アプリケーションが付加したシリアル番号が付加される。環境デーモンは、この要求を受けると必要とされた情報の監視を開始する。

登録応答:

登録要求に対する応答メッセージ。環境デーモンから適応型アプリケーションに対して、登録要求の成否と、登録要求に含まれていたシリアル番号、そして該当要求に対する識別番号を応答する。

変化通知:

登録要求によって登録された監視条件が満たされたことを、環境デーモンから適応型アプリケーショ

ンに通知するメッセージ。登録時に要求された情報の内容を、登録応答時に通知した識別番号とともに送信する。

登録解除要求:

適応型アプリケーションが、登録要求によって登録した監視条件を解除する要求メッセージ。登録応答時に通知された識別番号とともに送信する。環境デーモンは、この要求によって該当する監視条件を削除し、不必要な情報の監視を停止する。

登録解除応答:

登録解除要求に対する応答メッセージ。登録解除要求の成否を識別番号とともに応答する。

環境デーモンとアプリケーションとの間の通信路としてはコネクション型の通信路を用いる。そして、アプリケーションからコネクションを切断することによって、そのコネクションを用いて登録されていたすべての監視条件を解除することを許す。一般にアプリケーションの終了時にはすべての関係するコネクションが切断されるので、これによって自動的にそのアプリケーションが登録していた監視条件を解除することが可能になる。

環境デーモンは、I/F モジュール、ポリシモジュール群、条件モジュール群、プローブモジュール群の 4 種類のモジュールから構成される (図 2.4)。全モジュールは密接に連動して動作するために単一のプログラムに結合され、独自のスケジューラによりスケジューリングされる。各モジュール間は、上位のモジュールから下位のモジュールに対してはプロシージャ呼出しによって、下位のモジュールから上位のモジュールに対してはコールバックによって通信をおこなう。このため、I/F モジュールを除くモジュール群は追加することによる拡張が容易である。

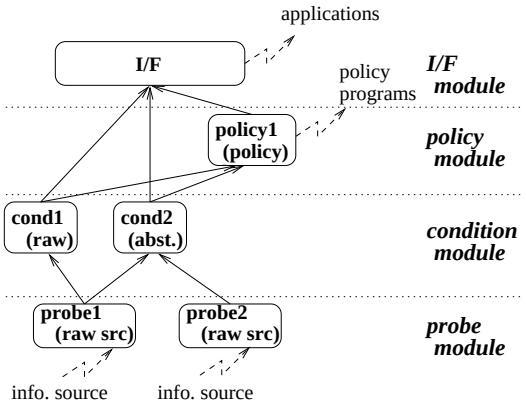

図 2.4.　環境デーモンの構成

I/F モジュール (I/F module)

I/F モジュールは適応型アプリケーションとの通信を管理するモジュールである。これは、図 2.2 中の一元的なインターフェースに相当する。このモジュールでは、適応型アプリケーションからの要求を受けつけ、各アプリケーションに設定された監視条件を管理する。この際、設定された監視条件に関する情報を扱う下位モジュールに情報の通知を依頼し、監視条件を満たすと変化通知を該当するアプリケーションに送信する。

ポリシモジュール群 (policy modules)

ポリシモジュール群は I/F モジュールの下位に属し、利用者による利用ポリシを扱うモジュールの集合である。ここで扱う情報は、図 2.2 における利用ポリシを反映した情報の部分に相当する。

I/F モジュールが、適応型アプリケーションから利用ポリシを反映した情報を要求されると、該当するポリシモジュールに対してプロシージャ呼出しによって情報提供を要求する。この際、I/F モジュールはコールバックプロシージャを設定する。ポリシモジュールは要求された情報に変化があらわれると設定されたコールバックプロシージャを呼び出すことによってこの情報を上位の I/F モジュールに通知する。

ポリシモジュールは利用ポリシ判断の元となる環境情報をさらに下位の条件モジュール群の中の複数のモジュールから取得する。そして、その情報をポリシプログラムと呼ばれる外部プログラムに転送する。実際の判断はポリシプログラムがおこない、ポリシモジュールは受信した応答に変化があった場合にのみ上位モジュールに通知する。

条件モジュール群 (condition modules)

条件モジュール群は I/F モジュールとポリシモジュール群の下位に属し、無加工の情報および一般化した情報を扱うモジュールの集合である。ここで扱う情報は、図 2.2 における無加工の情報および一般化した情報に相当する。

適応型アプリケーションが I/F モジュールに無加工の情報または一般化した情報を要求すると、I/F モ

ジュールはポリシモジュールと同様に、該当する条件モジュールに対してプロシージャ呼出しによって情報提供を要求する。この際、I/F モジュールはコールバックプロシージャを設定する。条件モジュールは要求された情報に変化があらわれると設定されたコールバックプロシージャを呼び出すことによってこの情報を上位の I/F モジュールに通知する。

条件モジュールは判断の元となる環境情報をさらに下位のプローブモジュール群の中の一つ、または複数のプローブモジュールから取得する。無加工の情報を扱う条件モジュールは一つのプローブモジュールのみを、一般化した情報を扱う条件モジュールは一つ以上のプローブモジュールを用いる。一般化した情報を扱う場合の一般化は、条件モジュール内でおこなう。

プローブモジュール群 (probe modules)

プローブモジュール群は条件モジュール群の下位に属し、環境情報源から実行環境に関する情報を収集するモジュールの集合である。ここで扱う情報は、図 2.2 の無加工の情報の元となる情報である。

プローブモジュールにおいても、条件モジュール等と同様に、上位の条件モジュールからプロシージャ呼出しによって起動され、必要に応じてコールバックプロシージャによって情報を通知する。プローブモジュールの情報取得方法は利用する環境情報源に依存する。

ポリシプログラム (policy programs)

ポリシプログラムは利用ポリシ判断を実際におこなうプログラムであり、環境デーモンから外部プログラムとして起動される。ポリシプログラムと環境デーモンとの間の通信はテキストベースのメッセージを送りあう形でおこなう。この設計により、ポリシプログラムの実装はどのような言語でおこなうことも可能になり、かつ交換も容易となるため、目標の一つである非常に柔軟なポリシ判断が可能になる。

適応型アプリケーション

適応型アプリケーションは CEIPS を用いて適応するアプリケーションである。適応型アプリケーションはその適応に利用可能な実行環境情報と、適応によって動作を変化させる際の実行環境の条件を、環境デーモンに登録する。アプリケーションは同時に複数の条件を登録をすることがでるため、適応に必要なだけの条件を登録することができる。環境デーモンから変化に関する通知を受けとると、必要ならばさらに実行環境情報を確認し、動作を変化させる。適応型アプリケーションは環境デーモンと通信するために、CEIPS が提供するスタブライブラリを利用する。

スタブライブラリ

スタブライブラリは適応型アプリケーションにリンクして用いるプロシージャ群である。このライブラリでは、環境デーモンとの通信に必要なメッセージの符号化、復号化および送受信をおこなうプロシージャ群を提供する。

2.6 CM1 の設計

CM1 は CEIPS のプロトタイプ実装である。CM1 はさまざまな実行環境のうち、特に TCP/IP 通信に関する実行環境を扱うことを目的とする。TCP/IP はさまざまな通信メディアで動作するが、通信経路上の通信メディアの特性をアプリケーションから隠蔽することにより、アプリケーションからはそれらを区別なく単一の手法で扱うことができ、そのために広く利用されている。しかしながらアプリケーションによっては、動的適応のために、隠蔽されている通信路の性能や可用性などの詳細な情報が必要なことがある。従来はそのような要求のあるアプリケーションは少数であったが、計算機の通信路の急激な変化と、計算機の利用方法の発達により、動的適応の必要なアプリケーションが増加している。このため、まず要求の多い通信に特化して CM1 を設計、実装した。

2.6.1 無加工の情報

無加工の情報は、環境情報源から取得した生の情報を利用するため、どの環境情報源から情報を取得するかに依存する。したがって、実装依存である。

現在の実装では、通信相手に対する経路情報、通信に利用するインターフェース名、インターフェースのリンク状態、TCP 層内部で測定したラウンドトリップ時間、TCP 層内部で測定したグットプット値を利用している。これらの情報源に関しての詳細は 2.7.2 節で述べる。

2.6.2 一般化した情報

実効バンド幅

通信するアプリケーション間の実効バンド幅は、そのアプリケーションが流量を制御するのに有用な情報である。アプリケーションが実効バンド幅よりも多いデータを送信した場合、データは通信路を埋めてしまい、不必要な遅延や消失の可能性が生じる。ここでアプリケーションが利用可能な実効バンド幅を知ることができれば、アプリケーションはそのバンド幅に適したデータ量を送信することができ、無用な遅延を避けることが可能になる。

ラウンドトリップ時間

ラウンドトリップ時間は通信するアプリケーション間でのメッセージの往復に必要な時間であり、通信路の遅延時間をよく表す。この情報は特に遅延が重要になる対話型アプリケーションなどに有効である。

可用性

可用性は、通信するアプリケーション間の通信がどの程度確実におこなえるかどうかを示す情報である。特にモバイルコンピューティング環境においては、移動によって通信路が利用不能になったり、無線による通信が不安定になったりすることが日常的に生じる。

アプリケーションはこの情報を利用することによって、通信できない状態で無意味な通信を試みたりタイムアウトを待つ必要がなくなる。また、通信が不安定な状態ではデータベース更新などの重要な通信をおこなわないなどの制御も可能になる。

インターフェース名

インターフェース名は、通信するアプリケーション間の通信が利用するリンクレベルの通信インターフェース名である。この情報は、通信メディアに応じた適応に利用される。

2.6.3 利用者のポリシを反映した情報

利用ポリシ

利用ポリシは、どの重要度を持つアプリケーションならば現在利用可能な通信路を利用して良いかどうかを示す情報である。この情報を用いることで、必要に応じて、重要度の低いアプリケーションの通信を抑制することが可能になる。

たとえば、計算機が携帯電話のような高価な回線を利用している場合に、回線費用を削減するために、計算機が定期的に自動実行するような重要度の低い通信は抑制することが可能になる。また、WWW 閲覧プログラムなどでは、通常は自動的に読み込まれる画像の読み込みを抑制することで、重要な本文の読み込みを高速化することが可能になる。

2.7 CM1 の実装

CM1 は現在 4.4BSD 系 UNIX の一つである FreeBSD-2.2.8R 上で実装した。本節ではこの実装について述べる。

2.7.1 環境デーモン

環境デーモンは C 言語を用いて実装した。環境デーモンとアプリケーションとの間は、常に利用可能なループバック回線を用いた TCP によって通信する。TCP はコネクション指向のプロトコルなので、適応型アプリケーションの終了時に環境デーモンがこれを検出することが可能になる。現在の実装はヘッダファイル等を含めて約 8000 行弱の規模である。

以下に細部の実装方法を述べる。

スケジューラ

CM1 の環境デーモンはプロセス自身がスケジューラを持ち、環境デーモンの各構成要素がプロセス内の情報を共有しつつ逐次実行されている。スケジューリング要素としては、通信チャネルとインターバルタイマ、そして一度だけ実行されるスケジューリング予約がある。スケジューラのエントリ一覧を表 2.1 に示す。スケジューラのメインループは以下のように動作する。

1. 予約されているスケジュールが存在すれば、その予約を実行し、実行後予約を消去する
2. 登録されているインターバルタイマの実行間隔の時間が経過していれば、該当スケジュールを実行する
3. 登録されている通信チャネルのいずれかが利用可能になるか、インターバルタイマの残り時間

表 2.1. CM1 のスケジューラのエントリ一覧

エントリ名	動作
sched_init	スケジューラの初期化をおこなう
sched_loop	メインループに突入する
sched_add_socket	ファイル記述子とコールバック関数をスケジューラに登録する
sched_rd_socket	登録済みのファイル記述子に対応する登録内容を読み出す
sched_mod_socket	登録済みのファイル記述子に対応するコールバック関数を変更する
sched_rm_socket	登録済みのファイル記述子を登録から外す
sched_add_1proc	一度だけ実行されるスケジューリング予約をおこなう
sched_add_itimer	インターバルタイマによるスケジューリングを登録する
sched_rm_itimer	登録したインターバルタイマを登録から外す

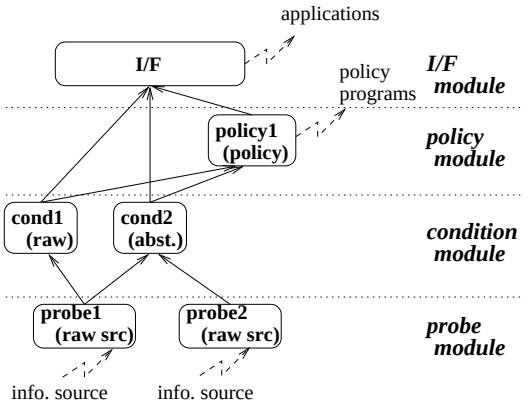

図 2.5. 環境デーモンの構成

が経過するまで停止する

4. 登録されているインターバルタイマの実行間隔の時間が経過していれば、該当スケジュールを実行する
5. 利用可能になった通信チャネルが存在すれば、対応するスケジュールを実行する
6. 1 に戻る

各モジュールの相互呼出し方法

第 2.5.3 節で述べたように、環境デーモンは、I/F モジュール、ポリシモジュール群、条件モジュール群、プローブモジュール群の 4 種類のモジュールから構成される。各モジュールの依存関係を図 2.5 に再度示す。CM1 を利用するアプリケーションから CM1 に対して必要な環境情報の登録がなされると、その登録はまず I/F モジュールで受けつけられる。I/F モジュールは要求を解釈し、要求された環境情報を提供する条件モジュールやポリシモジュールに対しコールバック関数を指定して条件を登録する。ポリシモジュールや条件モジュールは、必要に応じて情報源となるプローブモジュールに対してコールバック関数を指定して環境情報の監視内容を登録する。各モジュールは監視対象に合わせて自らの監視プロシージャをスケジューラに登録する。

スケジューラに登録された各プローブモジュールはスケジューラによって順次呼び出される。このとき監視対象が条件を満たすと、プローブモジュールは設定されたコールバック関数の呼出しをスケジューラに予約してからスケジューラに戻る。このように処理することにより、プローブモジュールは呼出しスタックを深くすることなしにコールバック関数を呼び出す。条件モジュールやポリシモジュールはプローブモジュールからコールバックを受けると、そのモジュールに登録された条件を満たしたかどうかを評価する。条件を満たした場合には、プローブモジュールと同様の方法でさらに I/F モジュールのコールバック関数を呼び出す。I/F モジュールでは、コールバックを受けた条件が含まれる登録内容に関して全ての条件モジュールとポリシモジュールが条件を満たしているかを確認する。もし条件を満たしている場合には、その登録をおこなったアプリケーションに対して現在の状況を変化通知として送信する。

ポリシモジュールと条件モジュールは、コンパイル時に表 2.2 の内容を持つ condfunc 構造体にその呼出しエントリが格納される。さらに、環境情報の種別を表す番号と condfunc 構造体の組をリンクリストとして保持し、I/F モジュールはこのリンクリストを参照して各環境情報に対応するモジュールを呼び出す。環境情報を追加する場合には、モジュールを追加し、追加したモジュールに対応する condfunc 構造体とその環境情報の番号をリンクリストに登録

表 2.2.　condfunc 構造体に含まれる呼出しエントリ

cf_init	初期化
cf_alloc	アプリケーションからのメッセージから内部表現への変換
cf_free	cf_alloc で確保した記憶領域の開放
cf_msg	内部表現からアプリケーションに対するメッセージへの変換
cf_register	監視する条件の登録
cf_unregister	監視する条件の削除
cf_get	現在の状況の取得
cf_cancel	cf_get の要求の取り消し
cf_copy	cf_alloc で変換した内部表現データのコピー

すれば良い。

2.7.2 利用可能な環境情報源

現在の実装では、以下の環境情報源を利用している。

4.4BSD routing socket

4.4BSD 標準の経路情報を参照、操作する機構であり、通信する対象に応じたインターフェース番号を得ることができる。

4.4BSD sysctl

4.4BSD 標準のカーネル内変数を参照、操作する機構であり、これを用いてインターフェース番号からインターフェース名を得ることができる。

MIBsocket[197]

FreeBSD のカーネルに組み込まれ、MIB-II[138] で定義された情報を参照、操作することを可能とする機構である。これを用いてインターフェース番号で示されるインターフェースのリンク状態を得ることができる。

cmsock

cmsock はカーネルプローブの一つであり、FreeBSD のカーネル内部に実装される。既存の環境情報源では CM1 に必要な十分な情報が得られないために、カーネル内に追加された。

cmsock とユーザ空間で動作する環境デーモンは、cmsock のために設計された独自のメッセージを用いて、新たなプロトコルファミリで区別されるソケットインターフェースによって通信する。このインターフェースは 4.4BSD 標準の routing socket に似た単純なもので、環境デーモンがこのインターフェースに接続しておけば、問合せメッセージをカーネル内に送信することと、カーネル内からの応答および通知メッセージを受信することが可能になる。メッセージは可変長で、かつすべてのデータにデータ長およびデータタイプ情報が付加されているために、扱えないデータタイプを読み飛ばすことにより、上位互換性を保ちながら拡張可能である。cmsock が用いるメッセージのメッセージヘッダ部の構造体を図 2.6 に示す。cmsock は TCP/IP のプロトコルスタックに変更を加えることで、プロトコルを通過する通信の状況を監視し、その平均と分散を得て、環境デーモンに通知する。

現在の実装では、cmsock は通信相手に対する TCP コントロールブロック [165] 内の t_srtt (Smoothed Round Trip Time) の値と、TCP 層内部で測定したグッドプット値を取得し、この平均と分散をソケットインターフェースを通じて環境デーモンに通知する。プログラムは C 言語で記述され、ソケットインターフェース部が 1000 行弱、TCP 層への変更が 100 行強の規模である。

3 種類すべてのタイプの情報と、それぞれがどのように依存しているかを表 2.3 に示す。

2.7.3 ポリシプログラム

環境デーモンとポリシプログラムとの間の通信は、一般的な標準入出力によっておこなうこととした。環境デーモンからポリシプログラムに送られる実行環境情報は設定ファイルによって選択され、選択された情報のいずれかが変化すると、環境デーモンからポリシプログラムに対して 1 情報につき 1 行づつ送信され、最後に区切りとして空行が送信される。ポリシプログラムは受信した実行環境情報を利用者のポリシと比較し、一行の応答を送信する。

現在の実装では、ポリシプログラムを perl インタープリタ言語によって実装し、利用者のポリシを

```
struct cm_msghdr {
    u_short     cm_msglen;      /* total message length */
    u_char      cm_version;     /* message version */
    u_char      cm_msgtype;     /* message type (CMM_XXX)*/
    int cm_flags;               /* flags (CMF_XXX) */
    int cm_data;                /* bitmask of data included
                                   in this message */
    pid_t       cm_pid;         /* identify sender */
    int cm_seq;                 /* for sender to identify action */
    int cm_errno;               /* why failed */
};
```

図 2.6. cmsock のメッセージヘッダ

表 2.3. 利用可能な情報とその情報源

type	情報	情報源	内容
ポリシ	利用ポリシ	すべて	通信可能な重要度
一般化	実効バンド幅	グッドプット, 経路情報	実効バンド幅
	ラウンドトリップ時間	平均 RTT, 経路情報	ラウンドトリップ時間
	可用性	リンク状態, 経路情報	可用性
	インターフェース名	ifname, 経路情報	利用されるインターフェース名
無加工	経路情報	routing socket	通信相手に対する ifindex 値
	ifname	sysctl	ifindex 値のインターフェース名
	リンク状態	MIBsocket	ifindex 値のリンク状態
	srtt	cmsock	TCP 層の t_srtt パラメタ
	グッドプット	cmsock	TCP 層で測定されたグッドプット値

プログラムとして直接記述している。

2.7.4 スタブライブラリ

スタブライブラリは C 言語で記述され、ライブラリが提供する関数の定義とともに実行環境と通知条件を記述するための構造体が C 言語のヘッダファイルとして提供されている。適応型アプリケーションはこの構造体を用いて環境デーモンと通信する。スタブライブラリには、この構造体の確保、リスト操作、解放のほか、環境デーモンと通信するための関数が定義される。

現在の実装で提供される関数を表 2.4 に示す。現在の実装では、適応型アプリケーションは環境デーモンとの通信ソケットにメッセージが届いた時、もしくは定期的に、ライブラリのメッセージ確認関数を呼び出す必要がある。これに関しては今後、環境デーモンからのシグナリングによって、適応型アプリケーションの実行とは非同期にメッセージの受信がおこなえるようにする予定である。ライブラリの現在の実装は C 言語のソースコードで 1000 行強の規模である。

2.8 CM1 の評価

通信環境に動的に適応するネットワークファイルシステムである PFS[173, 185] を適応型アプリケーションとして CM1 を利用するように変更し、CM1 の評価をおこなった。

PFS は動作中の通信環境の変化に適応するために複数の動作モードを持ち、それぞれ異なる方法でキャッシュを更新する。そして、それを環境に応じて

表 2.4.　スタブライブラリで提供される関数

関数名	動作
cm1_cond_alloc	情報一つ分の構造体を確保し、与えたデータで埋める
cm1_cond_free	確保した情報の構造体を解放する
cm1_cond_add	複数の情報を構造体のリストに結合する
cm1_clist_free	構造体のリストを解放する
cm1_open	環境デーモンとの通信を開始する
cm1_close	環境デーモンとの通信を終了する
cm1_send	環境デーモンに構造体リストを送信する
cm1_set_callback	メッセージを受けとるコールバック関数を定義する
cm1_rec_kick	明示的にメッセージの確認をおこなう
cm1_findcondbydtype	構造体のリストから目的の構造体を取り出す

動的に切り換える事によって適応をおこなう。PFS の主な動作モードは以下の 3 種類である。

NFS 互換モード:

NFS 互換モードでは、PFS はファイルにアクセスする度にファイルサーバとキャッシュとの一貫性を確認する。このモードは低遅延な通信路を利用する場合を想定している。

非同期更新モード:

非同期更新モードでは、PFS はファイルアクセスとは非同期にファイルサーバとキャッシュとの一貫性を確認する。このモードは高遅延な通信路を利用する場合を想定している。

切断モード:

切断モードでは、PFS はファイルサーバとの通信をおこなわず、キャッシュに残った情報のみで動作をおこなう。このモードでは、ユーザはキャッシュ内のファイルにのみアクセス可能となる。このモードは通信路が利用不能な場合を想定している。

2.8.1 バンド幅による適応

PFS において、非同期更新モードは低速な通信環境においても快適なファイル操作を提供する。しかしながら、この動作モードではファイルサーバとの一貫性を完全には保証できないため、通信路が十分に高速な場合にはより確実な一貫性を保つ NFS 互換モードを利用すべきである。従来の PFS では、通信路が十分に高速かどうかを PFS 自身のファイル転送スループットを測定することによって判断している。CM1 の評価のために、PFS の非同期更新モードと NFS 互換モードとの間の遷移要因として、測定した転送スループットのかわりに CM1 の実効バンド幅を利用するように PFS を変更した。

試験は、10Mbps のイーサネットで接続された 2 台の FreeBSD の動作している IBM-PC 互換機を用いておこなった。ネットワークのバンド幅を dummynet[136] を用いて変化させ、PFS がその変化を検出して動作モードが変化するまでの時間を、CM1 を利用した場合と、従来の PFS 自身で測定した場合とで比較した。試験は、バンド幅を 64Kbps に制限した状態で、非同期更新モードで開始した。最初に PFS 上にファイルを作成し、そのファイルの転送途中にバンド幅を 10Mbps に拡大した。この際の、バンド幅を拡大してから NFS 互換モードに自動遷移するまでの所用時間を測定した。

測定には 100Kbyte、200Kbyte、500Kbyte、1Mbyte の大きさのファイルを使用した。バンド幅の切替はファイル作成の 10 秒後におこなった。この切替時刻は、最小のファイルの場合でもファイル送信中であり、かつ TCP による通信が安定している時刻であることから決定した。図 2.7 に各大きさにおける、自動遷移までの最小、平均、最大所用時間を示す。

CM1 を利用しない場合には、ファイルの大きさが 100Kbyte と小さな時にはバンド幅の拡大を検出する前に転送が終了してしまい、適応がおこなえていない。また、逆にファイルが大きくなると適応までの時間が長くなっている。これは、アプリケーションからはファイルの転送中にファイルのどこまでが通信相手に転送されたかを検出することができない

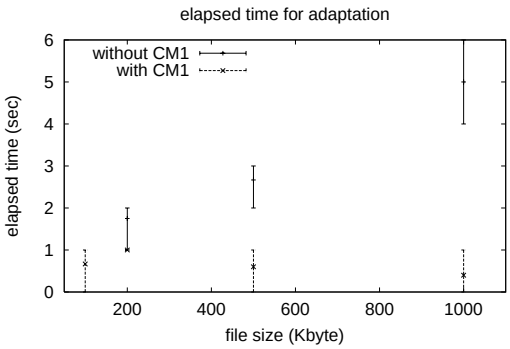

図 2.7. 適応までの所要時間

ため、ファイルの転送が終了した時点でスループットを算出しているためである。

CM1 を利用している場合には、カーネル内で精度の高い測定をおこなっているため、バンド幅拡大後、常に数秒で適応に成功している。CM1 を利用することにより、PFS は高速な回線に接続した際に速やかに NFS 互換モードに遷移することが可能になり、ファイルサーバとの一貫性をより高く保つことが可能になった。

この結果により、CM1 を情報源として利用することで、アプリケーションによる適応をより正確かつ速やかにおこなうことが可能になったことがわかる。

2.8.2 利用者ポリシによる適応

PFS では、通信路が確保できない場合と利用者が明示的に指定した場合に切断モードに遷移する。特に前者の場合には、PFS は通信を確立しようと定期的にサーバとの再接続を試みる。これは、切断モード中にはファイルサーバとの一貫性の保持ができないため、一貫性を極力保つために有効な機構である。しかしながら、この自動再接続によって、時としてダイアルアップ回線などのような通信費用の高い回線を利用者の意思にかかわらず不必要に使用してしまうことがある。

そこで、PFS が通信することを許すかどうかを CM1 の利用ポリシを用いて制御し、重要な通信のみが許される場合には、PFS は切断モードに遷移しそのまま留まるように PFS を変更した。ポリシプログラムとしては、安価で高速な通信路のときには重要度に関わらず通信を許可し、安価だが低速な通信路のときには中程度以上の重要度なら通信を許可し、高価な通信路のときには高い優先度の通信しか許さないというものを利用した。実際には、時刻と通信インターフェースから、その時点でどの重要度の通信を許すかどうかを算出する。算出された値が大きい程、重要な通信のみが回線を利用できる。表 2.5 に今回利用した条件を示す。ここで、インターフェース名 “tun” は課金されるダイアルアップ回線に、“ed” は課金のないイーサネットに接続していることを示す。また、インターフェース名 “none” は対象との通信に用いることのできる通信インターフェースが存在しないことを表す。また、23 時から 8 時までの時間は定額サービスを利用できると想定し、他の時間よりも低い優先度の通信を許すことができるとした。

PFS の実験では、PFS の重要度を 2 とし、これを越える重要度が必要とされる場合には切断モードに留まるように PFS を変更した。この変更により、通信インターフェースが利用不能な場合および定額サービス時間以外のダイアルアップ通信時には接続を試みず、イーサネット使用時か定額サービス時間のダイアルアップ通信時にのみ自動的に接続をおこなうことができた。これによって、CM1 を用いることにより、複雑なポリシをプログラムに与えることが可能であることが確認された。

2.8.3 カーネルプローブのオーバーヘッド

カーネルの TCP/IP 層に追加したカーネルプローブのオーバーヘッドを測定するために、TCP のスループットを `nttcp` コマンドによって測定した。`nttcp` コマンドは任意の大きさのデータ列を送受信し、その転送に要した時間を測定するコマンドである。`nttcp`

表 2.5. ポリシプログラムのアルゴリズム

interface	none	tun		ed
時刻	-	8:00-23:00	23:00-8:00†	-
出力	99	3	2	1
許可する通信	通信不能	重要な通信のみ	中程度以上の重要度	あらゆる通信

† ダイアルアップ回線で定額接続できる時刻を示す

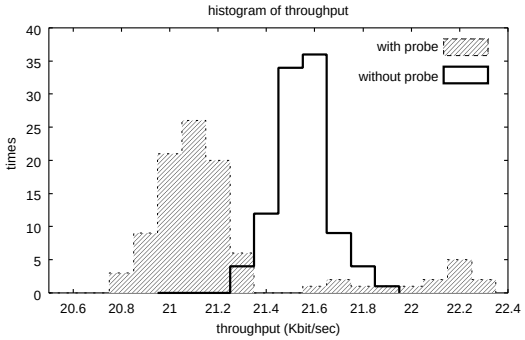

図 2.8.　スループットのヒストグラム

コマンドを送受両方で用いて、送信側から 4k バイトのデータを 20480 回 (合計 80M バイト) 送信し、その受信に要した時間からスループットを算出した。プロトコルスタックの処理時間に注目するため、ループバックインターフェースを用いて測定をおこない、外部に接続しているネットワークインターフェースを無効にすることで、外的要因による影響を最低限にした。試験には Am486DX4/75MHz をプロセッサに持ち、主記憶 20MB の IBM-PC 互換機を用いた。

測定は、カーネルプローブを組み込んだカーネルと、組み込まないカーネルで、それぞれ 100 回おこなった。その結果、スループットの平均はプローブ有りの場合は 21.2749Mbit/sec、無しの場合には 21.6018Mbit/sec となった。それぞれの場合のスループットのヒストグラムを図 2.8 に示す。ヒストグラムの形はどちらのカーネルでも同様の形をしており、プローブのオーバーヘッドは 2%以下であるといえる。この結果から、カーネルプローブのオーバーヘッドは無視できる程度であるといえる。

2.9 関連研究との比較

本節では CEIPS と、他のアプリケーション主導による適応を補助するシステムについて比較する。

2.9.1 Odyssey

Odyssey[48, 115, 114] はシステムとアプリケーションが協調して動作する適応機構の枠組である。Odyssey では、各アプリケーションは Odyssey を通じて通信する。これによって Odyssey が各アプリケーションの要求資源と全資源の利用状況を監視し、利用可能な資源量の変化をアプリケーションに通知する。通知を受けたアプリケーションはどのように適応すべきかを判断し、動作を変える。Odyssey では、各アプリケーションはそのアプリケーションが利用するデータタイプに応じた適応モジュールを Odyssey 内部に組み込むことにより、扱うデータタイプに特化した、より細かな適応を可能としている。

しかしながら、新たなアプリケーションやデータタイプの出現ごとに、Odyssey システムに適応モジュールを追加しなければならないため、アプリケーションの追加が容易ではないという問題がある。

2.9.2 SNMP

SNMP (Simple Network Management Protocol) [24] は本来ネットワーク管理のためのプロトコルであるが、このプロトコルを利用することによって、一元化された方法でネットワークの状況に関する情報を得ることができる。SNMP はネットワーク機器で動作する SNMP エージェントと、それを利用するアリケーションの間で利用される。SNMP エージェントでは、管理対象となる機器やネットワークに関する情報をオブジェクトとして管理し、それらのもつ値をアプリケーションに対して提供する。さらに SNMP には、あらかじめ設定された条件が満たされると、SNMP エージェントからアプリケーションに対して通知をおこなうトラップと呼ばれる機構が定義されている。

しかしながら、トラップに関してはほとんどの部分が実装依存になり一般性がない。また、トラップに利用できる条件は特定のオブジェクトの変化などのごく単純なものであり、複雑な条件は設定できない。そのため、適応型アプリケーションで利用しようとすると、判断の元となる情報は得られても各アプリケーションで一般化や複雑な条件の実現をおこなわなければならなくなる。

2.9.3 環境サーバ

環境サーバ [110] は実行環境に関する情報を適応型アプリケーションに提供する枠組であり、適応型アプリケーションから登録された実行環境条件が成立するとアプリケーションに通知する機能を持つ。アプリケーションはこの通知によって自らの動作を環境に適するように変化させることができる。

環境サーバでは、条件成立時の通知方法として Mach オペレーティングシステムの特色であるメッセージングの機構を使うなど、移植性が低い。また、無加工の基本的な情報しか扱わないため、各アプリケーションがその抽象化や利用者のポリシの適用をそ

れぞれおこなわなければならないという問題がある。

2.10 本章のまとめ

本章ではアプリケーション主導の実行環境への適応機構 CEIPS の提案と、その通信に特化した実装である CM1 の設計と実装を示した。CEIPS はプログラムの実行環境に対する動的適応に必要な、環境変化の検出、その環境下での適切な動作の判断、および実際の動作変更のうち、環境変化の検出と、各アプリケーションに共通の一般的な判断をする。CEIPS を利用するアプリケーションは、CEIPS から提供される環境情報によって環境変化を検出し、各アプリケーション固有の判断と CEIPS の判断結果を合わせて必要に応じて利用できる。実際の各アプリケーションの適応動作に関しては、各アプリケーションごとにおこなうため、CEIPS は関与しない。

アプリケーションは CEIPS を利用することによって、従来はオペレーティングシステム内部に隠蔽されていた、実行環境に関する高精度の詳細な情報を得ることができるようになる。さらに、無加工の情報、それを複数まとめて一般化した情報、さらに利用者の利用ポリシを反映した情報という複数のレベルの情報を、同一のインターフェースで利用できる。これによって、アプリケーションは必要なレベルの情報を利用することができ、容易に、かつ自由度の高い適応が可能となる。

評価として、CM1 を適応型アプリケーションとしての PFS に実際に適用し、従来の PFS よりもより高速かつ正確な動的適応が可能になることを確認し、その有効性を示した。また、オペレーティングシステム内部に CM1 のためのカーネルプローブを組み込むことによるオーバーヘッドを測定し、それが十分に小さいことを示した。

第 3 章　おわりに

本年度は、アプリケーションによる動的適応をおこなうための枠組である CEIPS およびその実装である CM1 について報告した。

Rover では、今後も定期的なミーティングを通して、移動計算機に適したファイルシステムや、移動透過性保証のためのプロトコルといったモバイルコンピューティングに関する議論を活発に行っていく予定である。なお、Rover のより詳細な活動については、`http://www.sfc.wide.ad.jp/rover/`を参照されたい。