システムソフトウェアと
オペレーティング・システム 84－5
（2000．5．25）

計算機環境に応じて サービス提供方式を適応させる 通信用ツールキット

村瀬正名[1] 若山史郎[1] 権藤俊一[1] 永田智大[2] 西尾信彦[3] 徳田英幸[1 2 3]
[1] 慶應義塾大学 環境情報学部 [2] 慶應義塾大学大学院 政策・メディア研究科 [3] 慶應義塾大学 SFC 研究所

本論文では，ウェアラブルコンピュータのような可搬性が高く，計算機資源の乏しい端末に適したアプリケーションフレームワークを提案する．本システムは，従来のアプリケーションフレームワークと異なり，アプリケーションの一部または全部がウェアラブルコンピュータとプリンタやディスプレイなどサービスを提供するホストに移動し，連携動作することが可能である．また，WC が移動すると，それに合わせてより近くにあるサービスを提供するホストにアプリケーションの一部を移動させ，負荷分散を実現することができる．ネットワークにさえ接続できれば，計算能力が非力であるウェアラブルコンピュータでも高度なアプリケーションを実行することができる．本稿では，以上のアプリケーションフレームワークのプロトタイプを設計，実装し，その性能評価を行なった．

An Adaptive Communication Toolkit which Alternates Service Provision Policy Responding to Computing Environments

Masana Murase[1] Shirou Wakayama[1] Shunichi Gondou[1]
Tomohiro Nagata[2] Nobuhiko Nishio[3] Hideyuki Tokuda[1 2 3]

[1]Faculty of Environmental Information, Keio University
[2] Graduate School of Media and Governance, Keio University
[3] SFC Research Institute, Keio University

This paper proposes an application framework for devices with high mobility and limited computing resources, such as Wearable Computers. The system, unlike conventional application frameworks, allows the part or the whole of the application to move to the WC and nearby host computer(s), cooperating to offer services such as printing and display. Furthermore, as the WC moves, the application migrates to host(s) nearby, distributing the computing load. With the presence of a network connection, even a device with limited computing resources can run complex applications. In this paper, a prototype of the above application framework was designed, implemented, and evaluated.

1 はじめに

近年，計算機の小型化が進み，ノート PC や PDA(Personal Digital Assistant)，i-Mode といった種々の携帯端末が普及している．端末の小型化の流れにあわせて，ユーザが常に着用する形で運用できるウェアラブルコンピュータ (以下 WC) といった研究 [1] も始められている．WC 環境は，従来研究されていたモバイル計算機環境と同一視されがちであるが実は様々な違いをもっている．

その中でも大きな違いは，モバイル環境における計算機ホストはノート PC のように一般の PC を小型化したもので，機能的な違いはバッテリ駆動，中断/ 再開機能，ネットワークとの一時的な接続などを除くと近年はストレージやプロセッサの高機能化のおかげでほとんどなくなりつつある．これに比較して WC 環境では端末の極端な小型化のために，同程度レベルの I/O デバイスはほとんど期待できない．つまり WC 環境では，貧弱な I/O デバイス環境下での最適化したユーザインタフェースを考案するか，ユーザの近傍にあり利用可能な資源やデバイス (これらを我々はサービスと呼ぶ) を協調的に利用する手法を開発することになる．IBM の VisionPAD [2] などは前者の手法である．我々は後者の協調分散的な手法をもとにすでに WC 環境におけるソフトウェアアーキテクチャの研究として，ユーザの意向を WC にとりこみ適切な環境変化を

トリガーに実行するミッション機構 [3] と，環境により利用可能状況の変化に対応するサービスルックアップ機構 [4]，種々のセンサデバイスから環境情報を抽象化して統合的に扱う機構 [5] を開発してきた．本稿ではこのような協調分散環境でのアプリケーション構築のためのフレームワークとして Applet for Wearable Computers(以下 Wapplet) フレームワークを提案し，そのプロトタイプの構築について報告する．

我々の仮定する WC 環境は分散協調スタイルであり，ユーザが装着するホスト自体には高度な入出力機能や重い計算処理機能を期待せず，その代りに周りの環境には様々なサービスを提供することを期待している．またユーザはその環境の中を刻々と動きまわるので，時々ユーザにとっての利用可能なサービスは変化する．このような環境におけるアプリケーションは，常に周りの環境変化を認識し，可能なかぎりユーザの意向に合せて適応的に動作すべきであろう．

例えば，音声と動画からなるマルチメディア情報を受信するアプリケーションを WC 環境で稼働する例について考えてみる．ユーザが屋外にいる場合で手元の WC 以外に利用可能なサービスが得られなければ，受信するマルチメディア情報の出力は WC 上の貧弱な I/O デバイス経由とならざるを得ない．場合によっては，カラー動画がモノクロの静止画でしか扱えなかったり，音声しか扱えないこともありえる．しかし，ユーザがオフィスに移動するなどの環境変化にともない，通常のディスプレイ装置が WC から利用できるサービスとして発見されると，受信情報の出力先をリダイレクトすることによりサービスの質の向上が見込まれる．また，この場合，アプリケーションの制御のための GUI ウィジェット (Play, Stop ボタンやボリュームつまみ) や，そのための入力デバイスはオフィスにあるより適切なデバイスに切り替えられるかもしれないし，WC にあるもののままであるかもしれない．

つまり，我々の仮定する WC 環境における協調分散スタイルのアプリケーションには，環境の変化に適応してサービスの利用形態を変化できることが望まれる．そこで我々は，コンポーネント技術 [6] とモバイルエージェント技術 [7] を意識したアプリケーションフレームワークである Wapplet 機構を設計し以下で提案する．Wapplet は複数のコンポーネントから構成されるアプリケーションであり，各コンポーネントはその時々で適切なサービスを求めネットワーク上を移動する．このためコンポーネント間通信には位置透過性を与えている．

以後，第 2 節では従来の関連研究について述べ，モバイルコンピューティグ支援技術とその問題点について説明する．続いて第 3 節では本論文で提案するアプリケーションフレームワークである Wapplet フレームワークの設計を行う．第 4 節では，Wapplet フレームワークのプロトタイプについて説明を加え，第 5 節でその評価を行う．

2 従来のモバイルコンピューティング支援技術

我々の目指すシステムに類似している研究として，負荷分散処理技術では，MetaSpace[8]，Mogul[9] がある．これらは，プログラムの一部を別のホストに移動させることで負荷分散を実現している研究である．また，コンポーネント間の位置透過的な操作技術として CORBA[10] がある．CORBA は OMG(Object Management Group) の提案するシステムである．さらに，環境の変化に適応してサービスを利用する研究として Location Aware Mobile Computing[11] がある．

本節では，上記の関連技術について概要と問題点，あるいは我々の目指すシステムとの位置付けを述べる．

2.1 MetaSpace

MetaSpace は，動的分散ソフトウェアの構築を支援するツールキットである．MetaSpace では，計算機資源を提供するホスト (以下サポートホストと呼ぶ) が資源登録ホストに CPU パワーやメモリ容量，ホストの位置などの登録を行う．ユーザはこの登録ホストに問い合わせを行い，利用可能なサポートホストの情報を得る．ユーザは，アプリケーションの一部を移動プログラムとして，サポートホストへ移送させ，分散処理を行なう．

MetaSpace は Java によって実装され，プラットフォーム非依存の分散処理環境を提供している．Java RMI 利用経験のあるユーザであれば，分散モジュールの実装を比較的容易に行なうことができる．

だが，現状の MetaSpace ではサポートホストの CPU やネットワークの負荷上昇に合わせて移動オブジェクトをさらに別のホストに移動させて負荷を減らすということは困難である．利用者端末からサポートホストへの移動は可能だが，サポートホスト間の移動が不可能である．例えば，サポートホスト A の CPU 負荷が高くても，サポートホスト A に移動してきたオブジェクトは，CPU 負荷の低いサポートホスト B に移動することはできない．

2.2 Mogul

Mogul は，ユーザインタフェースを伴ったアプリケーションのホスト間移動を実現するシステムである．Mogul の特徴は，複数のホスト上を連続移送することが可能で，移動可能オブジェクトと移動不可能オブジェクトを動的に分離することができる．このため，プログラマはあらかじめホスト間移送を考慮して，アプリケーションを記述する必要がない．加えて，Java 言語を用いて実装されているのでプラットフォームに依存しない．このシステムを利用することでアプリケーションプログラマは容易に分散計算環境を構築することが可能である．

しかし，現在の Mogul では，オブジェクトを移送した後，移送元ホストから，そのオブジェクトをリ

モートコントロールすることができない．例えば，始めにホストAで音楽再生プログラムを実行し，このホスト上に現れているコントロールパネルを通して再生や停止などの制御を行う．次に音楽再生部分(デコーダ)のみをホストBに移動させ，ホストAは先程利用していたコントロールパネルからホストB上の音楽再生部を制御することは難しい．

2.3 CORBA

CORBAは，IDL(Interface Definition Language)と呼ばれるインタフェースのみが定義されており，これをC言語や，Java，C++，SmallTalkなどに組み込みこむことで，様々なプログラミング言語でCORBAを扱うことができる．また，ORB(Object Request Broker)を通じて，オブジェクトのメソッド呼び出しをリモート・ローカルの区別なく扱える枠組を提供している．そのため，サーバがネットワーク上のどのマシンに在り，どのように呼び出すかはORBが処理する．クライアントは呼び出したいオブジェクトの名前のみ指定すればよい．

すなわち，CORBAを使った分散システムでは，システム構成の変更に柔軟に対応することが可能である．

このような性質を持つことから，CORBAを利用して，我々の提案するシステムを構築することが可能である．しかし，常にCORBAの環境が全てのマシンに整ってはいないので，CORBAに特化したシステムは避けるべきである．

2.4 Location Aware Mobile Computing

Location Aware Moblie Computingは，環境を認識し，その環境の変化に合わせて通信の仕方や頻度を変化させるシステムである．このシステムでは，PADと呼ばれるデバイスをリモートコントローラとして扱い，他のマシンと協調動作して目的のサービスを得ることが可能となる．PADは，小型ディスプレイとタッチパネルのような入力機能を備えたデバイスであり，無線での通信が可能だが，計算処理能力を保持していない．

PADはLocation Serverと呼ばれるディレクトリサービスによって管理されている空間に入ると，CPUサーバと呼ばれるホストを利用して機器を制御する．例えばテレビのそばに近付くと，CPUサーバにそのテレビコントロール用のプログラムがロードされる．PADにはテレビコントロール用のインタフェースのみが表示され，タッチパネルなどによってテレビをリモートコントロールできる．しかし，このシステムでは次のような課題が残されている．

- CPUサーバの負荷上昇
- CPUサーバ依存

PADは計算処理を全てCPUサーバに任せるため，CPUサーバに処理が集中し，負荷が上昇する．しかし，このシステムでは，CPUサーバの負荷が上昇した場合の処理があらかじめ用意されていない．

また，計算処理の全てをCPUサーバで集中的に行なっているため，PADが存在する物理空間，またはネットワーク上の空間にCPUサーバが存在しないか，もしくは支障をきたしているときにはサービスを受けることができなくなる．

3 Wapplet(Applet for Wearable Computer) フレームワーク

Wappletフレームワークにおいて，我々は次のようなシナリオを想定している．複数のディレクトリサービスによって，いくつかの物理的に区切られたセグメントが形成されている．このセグメント間をWCは移動し，移動に合わせて適切なサービス利用方式が割り当てられる．その際，WCは，自身の置かれているセグメントにある，適切なサービスプロバイダ(サービスを提供するホスト)と協調動作し，可能な限りユーザの要求に合わせて品質の高いサービスを実行することが重要となる．図2にその概念図を示す．

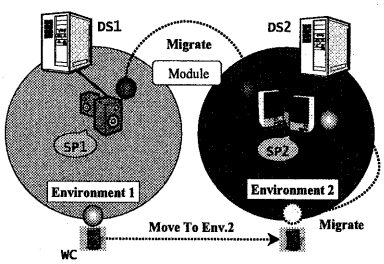

図1: WC，SP，DSにより構成されるWappletフレームワーク

それぞれの環境にはディレクトリサービスが存在し，その環境内で利用できるサービスプロバイダの情報を管理，保持している．環境により，利用可能状況の変化に対応するサービスルックアップ機能であるASAMAをWCは利用することで，WCの移動などによる環境変化が生じても利用するサービスプロバイダを変更して変化に適応できる．

また，アプリケーションデータベースと呼ばれる貯蔵庫を用意し，WCがサービスプロバイダを利用するためのアプリケーションを保管しておく．WCは，必要時にそこからダウンロードすることで，常にアプリケーションを持ち運ぶ必要がない．Wappletフレームワークでは，これをWappletデータベースと定義する．

3.1 Wappletフレームワークの目的

本研究の目的は次の二点である．

- WCの移動に伴う環境変化を認識し，その状況における最適なサービス利用を行なうことが可能な枠組を構築すること

- 単機能的であるWCがまわりにあるサービスプロバイダを利用して，複雑で多機能的な処理を行なうことが可能な枠組を構築すること

3.2 問題点とその解決法

以上の目的を達成するために，次の課題を解決しなければならない．

- スペック不足の問題
- 実装言語依存の問題
- CPU，ネットワーク負荷に関する問題
- サービスクオリティーに関する問題
- サービスの継続性の問題

スペック不足の問題とその解決法

WCの性能(CPUパワー，メモリ容量など)は貧弱であり，アプリケーションに合わせてC，JAVA，SmallTalkなど複数のランタイムを利用するのは困難である．そのため，WCはアプリケーションが必要とするランタイムなどに関する情報を管理し，必要なランタイムがない場合は，WCの周囲にある別のホストに代行を命じる機構が必要である．さらに，このアプリケーションを管理する機構は，ランタイムに依存しないものでなければならない．

本論文では，アプリケーションの動作設定ファイル(これをConfigファイルと呼び，アプリケーション毎に定義される)をXML形式で記述して，ランタイムに依存しないようにする．また，このようなファイルの記述内容を反映したプログラム(ミッションオブジェクト)を生成するミッション機構と連携させることで，本課題を解決することができる．

例えば，Configファイルに，必要ランタイムとしてAランタイムが明記されている．WCにそのランタイムが存在しなければ，ミッションオブジェクトが，Aランタイムの存在する別ホストへアプリケーションを移送させて，実行する．

実装言語依存の問題とその解決法

実装言語依存とはアプリケーションフレームワークを構築するために，特定の言語のみに縛られることを指す．実装言語非依存であれば，既存のプログラムを別の言語で記述し直す必要がなくなる．

このため，プログラミング言語を抽象化する仕組みを必要とする．本研究では，抽象化するために，Configファイルを利用する．アプリケーションプログラマは，Configファイルに必要なランタイムを記述すれば，Configファイルより生成されるミッションオブジェクトが必要なランタイムを持つホストを見付けだしてくれる．

すなわち，アプリケーションプログラマは既存のプログラムコードをWappletフレームワーク用に新たに書き直す必要はない．

CPU，ネットワーク負荷に関する問題とその解決法

WCは，負荷上昇によって適切にサービスを利用することができなくなる恐れがある．そのため，WCのようなCPUパワーの弱いマシンで，複数または高度なアプリケーションを動作させることは，非常に困難である．

また，ユーザと共に頻繁に移動するWCは，ネットワーク媒体に無線を利用するため，利用できる通信帯域が十分でない．例えば，WCでビデオストリームを受け続けるアプリケーションを実行すると，ネットワーク負荷はすぐに上昇してしまう．

この問題を解決するためには，WCのCPU，ネットワーク負荷に合わせてアプリケーションの動作を変化させる必要がある．

そこで，本システムでは，いくつかのモジュールによって構成されるアプリケーションを構築する．CPUやネットワークなどの資源を節約するために，モジュールの一部を別のホストへ移動させ，そのアプリケーションを実行する．先述の例を使えば，WCのCPUやネットワークの負荷が上昇すると，ビデオ再生のモジュールが別ホストに移動し，そこで映像が表示される．これによって，ビデオストリームはWCのネットワークを侵食せず，CPU負荷も抑えられる．

サービスクオリティーに関する問題と解決法

上述の仕組みを利用することで，この問題も解決することができる．WCは貧弱なI/Oデバイスしか備わっていない．例えば，映像を見るのにモノクロでしか表示できない場合が考えられる．しかし，周囲にカラー表示可能なディスプレイサービスプロバイダが発見できれば，映像処理に関わるモジュールをそのサービスプロバイダに移送すればよい．これにより，鮮明な映像を得ることが可能となる．

サービスの継続性の問題とその解決法

ユーザは移動しながらWCを利用するため，移動しながらでもプログラムを継続して実行することが有用となる．このため，プログラムがホスト間を移動するとき，そのプログラム内の変数などの実行状態を保持して移動することが必要となる．これにより，プログラムを構成する複数のモジュールが，各々移動したとしても，プログラムの一貫性が保たれ，継続したサービス利用が可能となる．

3.3 Wappletの構成

Wappletフレームワークで提供する各々のアプリケーションはConfigファイルといくつかのモジュールから構成される．このスタイルを持つアプリケーションをWappletと名付ける．Wappletの概念図を図2に示す．

図2の例にあるように，1つのWappletは，ConfigファイルとA，B，Cの3つのモジュールによって構成されている．モジュールAは，WCでのみ動作できるモジュールである．モジュールB，Cは，WCやサービスプロバイダ間を移動することができるプログラムである．

次に，Configファイルとモジュール群の概要についてまとめる．

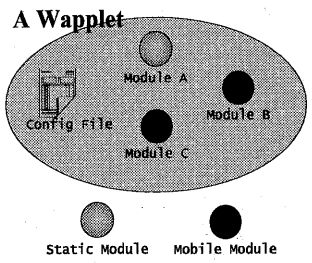

図 2: Wapplet 概念図

3.3.1 Config ファイルとその動作

Config ファイルによって，上述のようなスペック不足解消と実装言語非依存が実現される．設定ファイルである Config ファイルは，Wapplet ごとに定義され，Wapplet 作成時に自動的に生成することが可能である．具体的な記述内容を以下に示す．

- Wapplet の種類
- 動作に必要なモジュールの名前
- 各モジュールの性質 (移動可能／移動不能)
- モジュール動作のタイミング
- 必要なデバイス
- 必要なランタイム

Config ファイルは，ミッション機構と連携する．これにより，Config ファイルのランタイム非依存性が保たれ，かつ Config ファイルより生成されるミッションオブジェクトにより Wapplet の動作を規定，管理することが可能になる．

Config ファイルの動作

次に Config ファイル取得について説明する．WC は，Wapplet 使用要求を行なうと Wapplet データベースから，必要な Wapplet の Config ファイルのみをダウンロードする．

ミッション機構は，Config ファイルをもとにミッションオブジェクトを生成する．Config ファイルには上述の Wapplet 動作規定が記述されており，ミッションオブジェクトはその動作規定を管理する．例えば，どのモジュールに移動命令を下すかの判断，ディレクトリサービスから送られて来る新しいサービスをユーザへの通知を行なう．

モジュールへの移動命令などは，ソケットを通じて行なわれ，ユーザは特にモジュール移動のための手続きを必要としない．

3.3.2 モジュール群

Wapplet 構成のうち，アプリケーションの実際のプログラムの部分がモジュール群である．

モジュールには，二種類あり，移動可能モジュールと移動不能モジュールが存在する．移動可能モジュールは，負荷分散やサービスクオリティの確保のために，ローカルのホストだけでなくリモートホストにも移動できるプログラムである．また，移動可能モジュールは，ミッションオブジェクトから移動命令を受け取ると，指定されたサービスプロバイダに移送される．移動不能モジュールは始めに Config ファイルを取得したホスト (通常 WC) でのみ動作可能なものである．例えば，ローカルのファイルやデバイスを扱うモジュールがこれに相当する．

Wapplet 共通モジュール

全ての Wapplet に共通して必要なモジュールの機能をまとる．共通モジュールの機能は，メソッドの位置透過的呼び出し，全移送機能，部分移送機能，移動モジュール復旧機能の 4 つである．

メソッドの位置透過的呼び出し

メソッド呼び出しの位置透過性とは，移動モジュールがローカルにあるのか，リモートにあるのか意識することなくメソッド呼び出しをして，透過的に操作できることを言う．

全移送機能

全移送機能は，モジュール内のメソッドやコンテキストなど全てを移送させる機能である．ミッションオブジェクトからのモジュール移動命令を受けるため，ソケットを開いている．

部分移送機能

部分移送機能は，モジュールのコンテキスト (モジュール内の変数などの実行状態) のみを移送させる機能である．モジュールのコンテキストを抽出し，リモートホストにそれを移動させる．また，ミッションオブジェクトからのモジュールコンテキスト移動命令のやりとりはソケットを通じて介される．

部分移送は全移送と比べると，モジュールの転送容量を抑えることができるため，WC のネットワーク帯域に合わせて利用される．

移動モジュール復旧機能

移動モジュール復旧機能とは，移動モジュールの移動先で何らかの障害が生じ，サービスの停止を余儀なくされたときに動く機能である．この機能では，移動モジュールのコピー (チェックポイント) を保持しリモートホスト故障時にそのモジュールを最適なホストに復旧させる．ユーザは再びサービスを利用することができるようになる．

3.4 Wapplet の動作例

Wapplet の基本動作を具体例で示す．例としてビデオサービスを使用する．ビデオサービスを利用する Wapplet は次のものによって構成される．

- Config ファイル
- Filter モジュール (画像の大きさを変化させるモジュール:移動可能)
- Visualize モジュール (映像を表示するモジュール:移動可能)
- Decode モジュール (ビデオストリームを復号化するモジュール:移動可能)

- Sound モジュール (声や BGM を再生するモジュール:移動可能)
- ControlPanel モジュール (ビデオ再生などの制御パネル:移動不能)

ここで一連の流れを図 3 に示す.

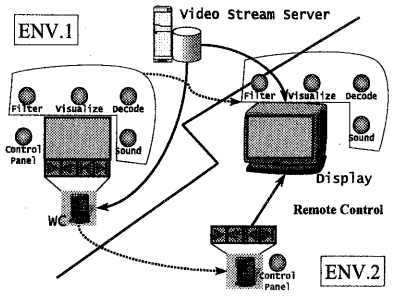

図 3: ビデオストリーム再生の動作例

WC は, 始めにディレクトリサービスに利用したいサービスの要求を送る. ここではビデオ再生を行いたいという要求である. 要求を満たすことが可能であれば, ディレクトリサービスの示す Wapplet データベースから必要な Wapplet の Config ファイルのみダウンロードする.

次に, ミッション機構によって, この Config ファイルからミッションオブジェクトが WC 上に構築される.

ミッションオブジェクトは WC 上にビデオ再生に必要なモジュール群をダウンロードし, ビデオサーバにアクセスしてビデオを再生する. ここで WC 上にロードされるモジュールは上述の全てのモジュールに当たる. また, WC ではディスプレイの質が良いとは言えないため, Filter モジュールを利用して画面の大きさやフレームレートの間引きが行われ, WC 上で最適に動作するように働く.

ここで, WC ユーザが別の環境に移動し, 新たに高画質のディスプレイサービスを受けることができるようになると, DS から WC にサービス利用可能メッセージが送られる. このメッセージはミッションオブジェクトが受け取り, ユーザにディスプレイの使用許可を表示する. ユーザがディスプレイサービスの利用を選択すると, WC 上に ControlPanel モジュールのみ残して, その他の全てのモジュールはディスプレイサービスプロバイダに移動を開始する. WC ユーザは残された ControlPanel モジュールを通じてディスプレイの映像をリモートコントロールすることができる.

今まで WC・ビデオサーバ間で行なわれていたデータのやりとりが, ディスプレイサービスプロバイダ・ビデオサーバ間に移る. これによって, WC での CPU 使用率や, ネットワーク負荷を下げることができる. また, DSP でのサービスは WC でのディスプレイサービスよりも質が良いため, より鮮明な画質を得ることが可能となる.

4 Wapplet プロトタイプの実装

以上の Wapplet サービスを実現するプロトタイプの実装を行った. 実装は Windows98 上で Sun Microsystems 社が提供する JDK1.2.2[12] を用いて行った.

プロトタイプとして, Wapplet 共通モジュールである, 全移送機能およびメソッドの位置透過的呼び出しを実現したプログラムを作成した. それぞれ, Transport モジュールと ControlWrapper モジュールである.

4.1 Transport モジュール

Transport モジュールは, RMI(Remote Method Invocation) を用いている. 移送部はおもに, モジュールの移送, モジュールの返送 (移送元にモジュールを戻す), モジュールのロードの機能を提供している. モジュールのロードは Wapplet データベースからのダウンロードに使われる機能である. 具体的なメソッド名を以下に示す.

```
void go(Object obj, String hostName);
Object come(String hostName);
Class load(String, hostName, String className);
```

図 4: オブジェクト移送部のメソッド名

4.2 ControlWrapper モジュール

ここでは, モジュールメソッド呼び出しの位置透過性実現のために下図のようなモデルを作成した.

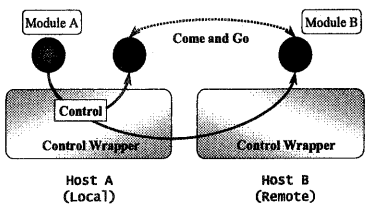

図 5: モジュールの位置透過性実現のモデル図

モジュール A は, モジュール B の存在位置を把握する必要はなく, ControlWrapper に対し, モジュール B の操作要求を行なえばよい.

この ControlWrapper モジュールが提供するメソッドを利用すれば, モジュールの位置に関係なく, そのモジュールを操作することができる. ControlWrapper はモジュールがローカル端末に存在する場合は, そのローカルにあるモジュールを直接操作する. 一方, リモート端末にある場合は, RMI 呼び出

しを利用して操作する．これらは，ControlWrapper部で完全に隠蔽しているため，Wappletプログラマは，このサブシテムが提供する位置透過的な操作を行なうメソッドを呼び出せばよい．

4.3 アプリケーション例

単純なWappletとして，ウィンドウの背景色を切り替えるプログラムを作成した．プロトタイプとして提供される2つのモジュールを利用している．

このWappletで使用されるモジュール群は以下の通りである．

- ColorPaletteモジュール(ウィンドウを出すモジュール:移動可能)
- Mainモジュール(プログラムのメイン部:移動不能)
- Transportモジュール(オブジェクト移送部:移動不能)
- ControlWrapperモジュール(移動オブジェクト操作部:移動不能)

4.3.1 WCでの処理

本Wapplet処理の一連の流れを図6に示す．

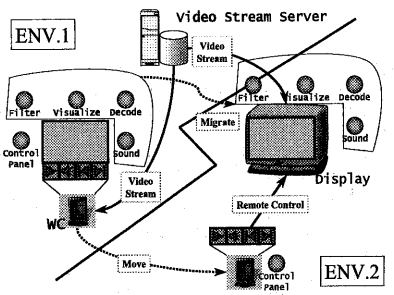

図6: 一連の流れ

このWappletは，WCと仮定したラップトップコンピュータ上にウィンドウが表示される．ユーザはWC上で本Wappletを実行すると，WCはhttpdが起動しているWappletデータベースから，ColorPaletteモジュールをダウンロードする．ダウンロードが完了すると，画面にウィンドウが表示される．ユーザは，画面に表示されたコントロールウィンドウ(背景色切り替えボタンがある)を利用して，表示されたウィンドウの色を変更する．

ユーザはサービスプロバイダにColorPaletteモジュールの移動を選択すると，ウィンドウの全てがそのホストに移送さる．移送が完了すると，ユーザは先程のコントロールウィンドウでサービスプロバイダに表示されているウィンドウを移送前と同じように操作できる．

4.3.2 サービスプロバイダでの処理

サービスプロバイダ側では，RMIを利用したリモートメソッド呼び出し可能なクラスが定義され，RMIレジストリに登録されている．クライアントであるWCからのリモートメソッド呼び出しによって，サービスプロバイダ側のControlWrapperモジュールが呼び出され，移動モジュールであるColorPaletteモジュールの制御が行なわれる．

5 評価

本節では，プロトタイプで示したメソッド呼び出しの位置透過性を実現したControlWrapperモジュールと，全移送機能を提供するTransportモジュールについて取り上げ，プロトタイプの評価を行なう．

我々の示したControlWrapperモジュールの有用性を評価するため，オーバヘッドを測定する．また，Transportモジュールの提供するモジュール移送とモジュールのロードに必要とする時間を測定し，全移送機能の性能評価を行なう．

評価に用いたアプリケーションは第4.3項で示したアプリケーション例と同じである．実行環境を表1に示す．

表1: 利用した環境

役割	CPU	主記憶	ネットワーク
WC	Pentium 266MHz	64MB	10MbpsLAN
SP	PentiumII 333MHz	192MB	100MbpsLAN
WDB	UlsraSPARC-II 248MHz × 2	512MB	100MbpsLAN

5.1 ControlWrapperの定量的評価

ControlWrapperのオーバヘッドを比較するために，次の項目を測定した．

- ローカルメソッド呼び出し
- リモートホストのRMI経由メソッド呼び出し
- ローカルホストのRMI経由メソッド呼び出し
- ControlWrapperによるローカルホストのメソッド呼び出し
- ControlWrapperによるリモートホストのメソッド呼び出し

この測定結果を表2に示す．

表2: 測定結果

オブジェクト操作	処理時間 (msec)
ローカルメソッド呼び出し	1.5
リモートホストの RMI 経由メソッド呼び出し	14.5
ローカルホストの RMI 経由メソッド呼び出し	4.6
ControlWrapper によるローカルホストのメソッド呼び出し	1.8
ControlWrapper によるリモートホストのメソッド呼び出し	14.9

ControlWrapperによるローカルホストのメソッド呼び出しとローカルメソッドの呼び出しを比較すると，値がほぼ同じである．また，ControlWrapperによるリモートホストのメソッド呼び出しとリモートホストのRMI経由メソッド呼び出しを比べると，差がほとんどない．このことから，ControlWrapperモジュールによるメソッド呼び出しのオーバヘッドは少ないことが示せた．

また，ControlWrapperモジュールと同等の機能をRMIだけで実装することも可能であるが，表2

の示すように，ローカルホストのRMI経由メソッド呼び出しのオーバヘッドが大きい．ControlWrapperモジュールによるローカルメソッド呼び出しと比べると，約2.6倍もの時間がかかっている．

よって，本システムのControlWrapperモジュールの優位性を示せた．

5.2 Transportの定量的評価

Transportモジュールでの評価では，ColorPalleteモジュールのロード時間，ColorPallete移送時間を測り，Wappletフレームワークの全移送機能を利用する際の処理時間を測定した．結果を表3に示す．

表 3: 測定結果

Transport が提供する処理	Transport 内部処理	処理時間 (msec)
モジュールロード	http 経由でのモジュールのロード	1.5
	モジュールのインスタンス生成	0.2
モジュール移送	ローカルでの移動モジュール終了	1.6
	リモートホストのルックアップ	9.3
	リモートホストへのモジュール転送	8.1

測定結果から，移送の際にはホストのルックアップに時間がかかることもわかった．しかし，移送処理トータルの時間は，19.0msecであるので，実用に耐え得る機能であることが示せた．

また，モジュールのロードは，モジュール内の変数などの状態をもたないため，モジュール移送より早く転送できることがわかった．すなわち，ネットワーク帯域が狭いときには，全移送を行なうのは効率が悪い．モジュールロードと部分移送機能が提供するコンテキスト移送を組み合わせることで，全移送と同じ機能を実現でき，帯域幅が確保できないときに，この手法は有効だと考えられる．

5.3 考察

今回のWCとしてはラップトップコンピュータを用いたが，より非力な環境でサービスの実行や環境変化への対応を実現し，実用性を検討する必要がある．また，一つのディレクトリサービスが提供するセグメント内でのモジュール移送のみの評価であったので，セグメントを跨いだ環境も考慮する必要があるだろう．

6 今後の課題

今後は，より複雑なケース，例えば複数のディレクトリサービスが提供するセグメント間をWCが移動する場合や，複数の移動モジュールが存在する場合などの実装，評価を行なわなくてはならない．また，オブジェクト移送手段もWCとサービスプロバイダ間のネットワーク負荷が大きいときには，モジュール全てを移送するのではなく，そのコンテキストのみを移送可能にするなど部分移送の機能も組み込む必要がある．これらは，Transportモジュールの拡張を行なうことで可能であると考える．さらに，本プロトタイプでは扱われなかった，モジュールの自動復旧機能を実現し，耐故障性を高めるとともに実用性のあるシステムを目指す．このためには，モバイルエージェント技術のチェックポイントのような機構を組み込む必要があるだろう．

7 まとめ

WCのような非力な計算機資源しか持たない端末でも，近傍に存在する利用可能な資源やデバイスを協調的に利用する手法により高度なサービスを得ることができる．このような分散協調環境でのアプリケーション構築のための枠組として，Wappletフレームワークを提案した．

Wappletフレームワークでは，アプリケーションをConfigファイルと複数のモジュールから構成し，Configファイルから生成されるミッションオブジェクトによってユーザに意向に合わせたサービス利用方式を提供することを解説した．

また，本システムのプロトタイプとして，全移送機能とモジュールの位置透過的なメソッド呼び出し機能を実現した．これにより，移動モジュールの転送や，返送が可能になり，モジュールが様々なホスト間を移動することが可能になった．また，ユーザは移動モジュールの位置を意識せずに，そのモジュールを操作することが可能になった．併せて，その性能評価を行ない，実用に耐え得るシステムであることが証明された．

参考文献

[1] MIT Wearable Computing Web Page http://wearables.www.media.mit.edu/projects/wearables/

[2] IBM Inc, VisionPad Web Page http://www.ibm.co.jp/News/leads/980912/

[3] 岩本，西尾，徳田，"ウェアラブルコンピュータにおけるミッション機構"，情報処理学会コンピュータシステム・シンポジウム論文集 vol.99, No.16 pp.41-48(1999).

[4] 永田，西尾，徳田，"適応的ディレクトリサービスにおけるステート管理方法"，情報処理学会コンピュータシステム・シンポジウン論文集 vol.99, No.16 pp.57-64(1999).

[5] 若山，村瀬，権藤，岩本，西尾，徳田，"ウェアラブルコンピュータにおけるデバイス非依存な環境情報の認識機構の設計と実装"，情報処理学会システムソフトウェアとオペレーティングシステム研究報告書 No.83 pp.61-66(2000).

[6] ActiveX Web Site http://www.microsoft.com/com/tech/ActiveX.asp

[7] I.Satoh, "MobileSpaces:A Framework for Building Adaptive Distributed Applications using a Hierachical Mobile Agent System", Tge 20th IEEE International Conference on Distributed Computing Systems(2000).

[8] 藤村，中澤，大越，徳田，"動的分散ソフトウェアツールキット MetaSpace の設計と実装"，情報処理学会システムソフトウェアとオペレーティングシステム研究報告 No.81 pp.119-124(1999).

[9] 中澤，望月，徳田，"ホスト透過性オブジェクト移送システム Mogul の実現"，情報処理学会論文誌 Vol.40, No.6(1999).

[10] CORBA http://www.corba.org/

[11] H.W.Peter Beadle, B.Harper, G.Q.Maguire Jr., J.Judge, "Location Aware Mobile Computing", IEEE/IEE International Conference on Telecommunications(1999).

[12] Sun Microsystems Inc.:Java2 SDK Standard Edition(1999).