卒業論文　　2001年度（平成13年度）

移動適応型分散通信機構の設計と実装

指導教員

慶應義塾大学 環境情報学部

徳田 英幸

村井 純

楠本 博之

中村 修

南 政樹

慶應義塾大学 環境情報学部

権藤 俊一

卒業論文要旨　2001年度（平成13年度）

移動適応型分散通信機構の設計と実装

本論文では，高度に機能分散化された機器の利用を想定し，人の移動に伴って利用する機器が変化する場合に，それぞれの場面で利用者の作業を適応させて継続する為の機構を提案する．

高度に機能分散化された機器とは，様々な目的や用途によって機能毎に分散化された，人の周囲に存在する機器である．これらの機器を利用する環境では，利用者の移動に伴う携帯端末の移動や利用者の環境自体が変化することによって，作業状態の変更が必要とされる．

既存の移動型コンピューティング環境では，常に同一の通信方式や実行環境に依存した通信状態や実行状態を持続させることを目的としている．このため，多種多様な環境に対応できず，機能高度分散型コンピューティング環境では問題が生じる．

これに対し，本論文では機能高度分散型コンピューティング環境で移動を行う移動適応型分散通信モデルを提案する．この通信モデルでは，利用者の作業状態に着目し，移動に対して利用者の作業状態を通信方式や実行環境に依存せずに継続させることを目的とする．そして，利用者の環境変化に対して作業状態を継続するために，移動適応型分散通信機構を実現する．

本論文では，移動適応型分散通信機構の設計及び，プロトタイプ実装を行った．本機構では，通信状態を抽象化して管理することにより，利用者の作業状態を移動に対して継続する．そして，実際に機能高度分散型コンピューティング環境において，作業の継続を実現できることを示した．

キーワード:

1: 作業継続　　2: 移動適応　　3: ユビキタスコンピューティング環境

慶應義塾大学　環境情報学部

権藤 俊一

Abstract of Bachelor's Thesis

A Migration Adaptive Architecture for Distributed Communications

In this paper, we assume the highly function distributed computing environment of the future, and propose a system that adapts user's work state to movement for continuation of work.

Our system assumes the highly function distributed computing environment where the user utilizes services of nearby devices. The user's movement changes the location of the mobile terminal and the surrounding environment, requiring changes in the work state.

Existing systems that support the mobile computing environment are focused on maintaining execution and communication states that are dependent on a particular protocol and runtime, respectively. Thus they cannot adapt to heterogeneous environments, and cannot be used in the highly function distributed computing environment.

For this reason, this paper proposes the mobility adaptive distributed communication model for the highly function distributed computing environment. This model focuses on the user's work state, and aims to preserve this independent of the underlying protocol or runtime. We also realize the mobility adaptive distributed communication system to preserve the user's work state during the change in environment.

In this paper, we have designed and implemented the mobility adaptive distributed communication system. We abstract and preserve the user's work state during the user's movement. We show the work state can be preserved in the highly function distributed computing environment.

Shunichi Gondo

Faculty of Environmental Information,
Keio University.

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

数々の技術的な発展に伴って，我々のコンピューティング環境は変化している．例えば，機器の小型軽量化や無線通信技術の発達によって利用する機器の移動性が向上し，モバイルコンピューティング環境が実現可能となった．また，複数のコンピュータで動作するソフトウェアが相互に通信可能となることで，分散コンピューティング環境が実現された．こうした変化を背景に今後のコンピューティング環境が考えられており，その１つとしてユビキタスコンピューティング環境 [1] が提案されている．

本節では，従来のコンピューティング環境をまとめ，ユビキタスコンピューティング環境を実現する際に生じる問題点と限界について指摘する．次節では，本論文の目的と意義を述べ，必要とされる事項や手法をまとめる．

モバイルコンピューティング環境

モバイルコンピューティング環境は，小型軽量で携帯可能な機器と無線通信技術を用いて実現される “機器が移動可能なコンピューティング環境” である．小型軽量で携帯可能な機器としては，ラップトップ型コンピュータや PDA (Personal Digital Assistant) 等の携帯端末などがある．また無線通信技術としては，携帯電話や PHS(Personal Handyphone System) など広域での利用を対象とした通信網を用いる場合と，無線 LAN(Local Area Network) [2] や Bluetooth [3] など近距離の通信システムを用いる場合がある．

また，モバイルコンピューティング環境の発展形として，装着可能なコンピュータ [4, 5] の常時携帯を想定したウェアラブルコンピューティング環境も実現されている．

分散コンピューティング環境

分散コンピューティング環境は，複数のコンピュータ上に分散したソフトウェア等が互いに通信し，協調して動作することによって目的が実現される “組み合わせのコンピューティング環境” である．分散して動作する目的は，単純な負荷分散から機能や処理毎の分散化まで多様である．また，分散の形態も目的や用途に応じて様々となる．

そして，単一のコンピュータでは目的の達成が出来ない場合，別のコンピュータに分散化された機能を用いて拡張性を実現することが可能となる．例えば，モバイルコンピューティング環境では，携帯可能な機器の制約で実現不可能な機能や過負荷となる処理をする必要がある場合，通信可能な別のコンピュータと処理を分散させている．

こうして，作業の実現可能性が携帯機器の能力に依存するという制約は解消された．

ユビキタスコンピューティング環境

ユビキタスコンピューティング環境は，様々な場所に存在する機器を用いて，“いつでも・どこでも目的を実現可能なコンピューティング環境”である．この環境では，汎用的な目的を実現可能な従来の機器に加えて，利用目的を特化し機能を単純化した機器 [6] の利用も想定される．この為，様々な機器に分散した機能を用いて目的を達成することが要求される．つまり，ユビキタスコンピューティング環境は，機能高度分散型コンピューティング環境として捉えることも出来る．そして，この環境で目的の作業を実現する為には，利用する機器を制御して協調動作させる必要がある．同時に，利用者の移動に対応して動作させる必要もある．従って，ユビキタスコンピューティング環境の実現にあたっては，前述したモバイルコンピューティング環境と分散コンピューティング環境を基盤として考慮に入れる必要がある．

一方，ネットワークを介した通信形態では，従来の Client/Server 方式に加えて Peer-to-Peer 方式 [7, 8] の普及が著しい．この場合は移動に伴って利用する機器や通信状態が複雑に変更され，既存技術の手法では対応可能な範囲に限界が生じる．例えば，ある場所で共用の機器や資源を一時的に独占して用いる場合，移動する際にその機器を解放する必要がある．この為，その機器を用いた作業を移動後の環境で継続する場合は，移動後の環境から移動前の環境にある機器を利用することは出来ない．こうした例が想定される機器としては，共用端末やオーディオ機器などが考えられる．そして，移動に伴って様々に変化する環境において，移動前後で同種の機器が利用可能となる保証はない．つまり，場所毎に利用可能な機器や利用する状況が異なる場合を想定する必要がある．こうした分散化の傾向は，Bluetooth などを用いたボディーエリアネットワーク環境が実現されることによって，より顕著なものとなることが予測される．それに伴って，前述したコンピューティング環境での手法による対応範囲の限界が問題となる．

この様に，利用環境や形態の変化と従来のコンピューティング環境での前提を変更せざるを得ない状況の発生によって，既存技術の手法では対応不可能な状況が生じている．従って，こうした状況に対応する為の新たな手法が必要とされる．

1.2 目的と意義

本論文の目的は，人が行う作業や通信の状態を移動に対し適応させて継続することである．この目的を実現することで，移動によって様々に変化する環境へ柔軟に対応可能とし，移動に対して適応的なコンピューティング環境を実現する．この為，本論文では移動適応型分散通信モデルを提案し，この機構を設計する．図 1.1 に示す本論文が想定する環境では，利用者の携帯する機器と環境に設置された機器の利用が可能である．そして，これらの機器をそれぞれの環境で組み合わせ，また切り替えて利用

する．こうした環境で作業の適応的な継続性を実現しようとする場合，前節で指摘した問題が発生する．

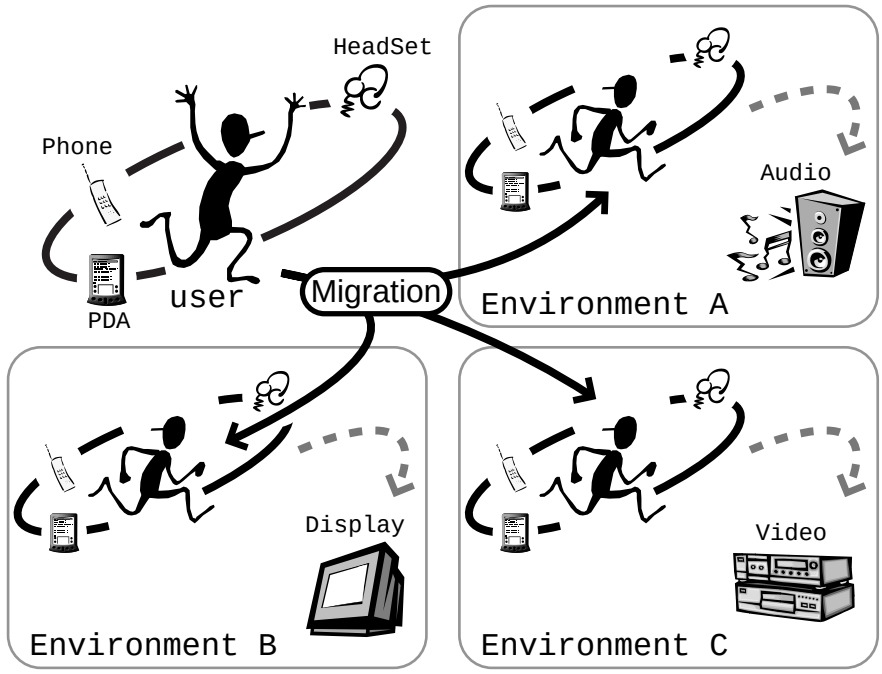

図 1.1: 本論文の想定環境．利用者が携帯する機器と環境に設置された機器を環境毎に組み合わせ，切り替えて利用可能である．つまり，移動した環境に応じて，利用者が携帯する HeadSet や PDA と移動前の環境で利用していた Audio や Video, Display 等に代って，移動後の環境に存在する Audio や Video, Display 等を利用する．

本論文で提案するシステムの目的や実現可能となることは，以下に示す通りである．

状態継続性：環境変化に左右されない作業状態の継続

移動に伴って，異なる種類の機器を利用する場合や共用の資源を用いる場合であっても，影響を受けずに利用する人の作業状態を継続可能とする．

環境独立性：特定の環境に対する依存性の排除

通常とは異なる環境で一時的に実現された作業を継続させる場合に必要である．つまり，通常は利用しない機器によって作業状態が構成された場合，これを単に持続することは合理的でなく，前節で指摘した通り，新たな問題も発生させる．

状態抽象性：状態管理による作業状態の抽象化

ある環境での作業状態を抽象化し，それを元に別環境で作業状態を再構成することを可能とする．これは，異なる環境で作業状態の継続性やその柔軟性を実現する場合に必要となる．

本論文では，機器の移動性を拡張する従来の手法に対して，利用者の作業や通信を移動に適応的とすることが重要であると考える．この適応性によって，移動に対する

作業状態の継続に柔軟性を持たせることが可能となる．これらを実現し，移動や環境の変化に関わらず作業状態の一貫性を保証し，変化する様々な環境に対して人の作業や通信の状態を柔軟に適応可能なコンピューティング環境の実現を目標とする．

1.3 本論文の構成

以下，本章を含めて全 7 章から構成される．次章では，コンピューティング環境と移動に関して，従来のアプローチと本論文でのアプローチについて論じる．続く第 3 章では，第 2 章の移動適応型コンピューティング環境を実現する移動適応型分散通信モデルを提案する．また，第 4 章では移動適応型分散通信機構の設計について述べ，第 5 章でその実装について説明する．そして第 6 章では，本論文の評価として従来のアプローチとの比較や議論を行う．最後に，第 7 章では本論文を総括し，反省や今後の課題などによって結論としてまとめる．

第2章　コンピューティング環境と移動

本章では，ユビキタスコンピューティング環境を機能高度分散型コンピューティング環境と捉え，この環境での移動について分析する．そして，前章で示した目的を実現する上で必要な，移動に対する継続性について考察する．

始めに，従来のコンピューティング環境での移動について分析し，その実現手法を機能高度分散型コンピューティング環境での移動に適用した場合に生じる問題点や限界を明らかにする．続いて，それらの問題点や限界に対して取り組むべき方針をまとめ，移動適応型コンピューティング環境として説明する．

本章の最後では，移動型コンピューティング環境を実現する既存の様々な関連技術を挙げ，移動に対するそれらのアプローチについて分析する．

2.1 従来の移動型コンピューティング環境

従来の移動型コンピューティング環境では，利用する機器がそれを利用する人に伴って移動することを前提とする．言い換えれば，移動前後で利用する機器は変化しない．この為，移動に対応する為の方法は，移動する機器自体に主眼を置いている．本節で説明する従来の移動型コンピューティング環境とは，図 2.1 で示す環境である．この環境では，実行中の作業に持続性を実現する為，利用者が用いる機器の移動に対して，その実行状態や通信作業を移動前後の環境で転送する．

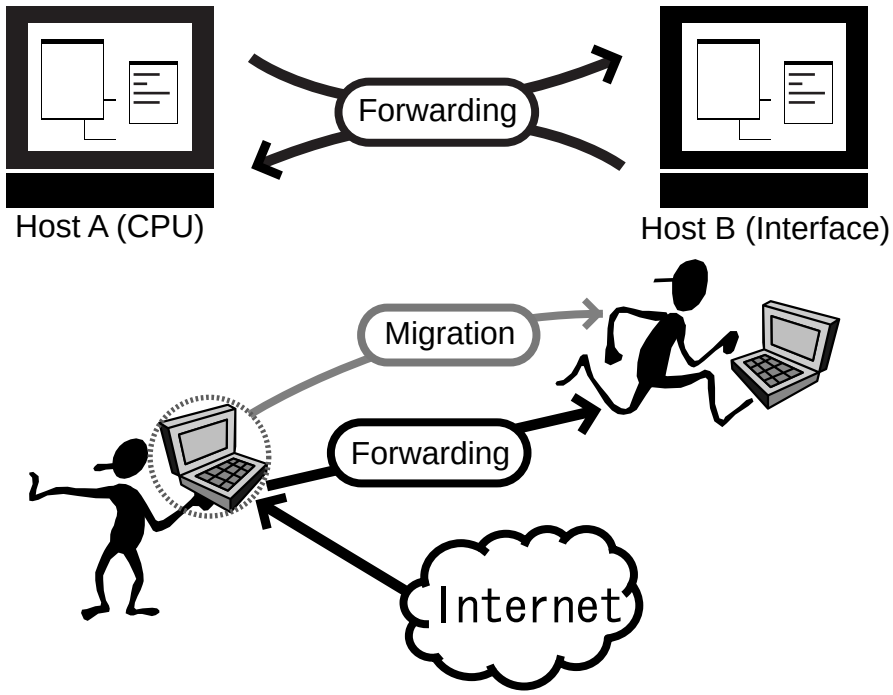

図 2.1: 従来の移動型コンピューティング環境．利用する人の移動に伴って，移動前後の環境で作業状態の持続性が実現される．上の図では，移動前の機器から移動後の機器に対して表示画面や操作インタフェースを転送している．これによって，移動前の機器に対する利用が移動後の機器を経由して持続可能であることを示す．また下の図では，移動後の作業が移動前の環境を経由して実現される場合である．これは，利用者が携帯して同時に移動する機器に対して，通信作業やアクセスの持続性を実現可能であることを示す．

2.1.1 移動への対応

従来の移動型コンピューティング環境では，機器の移動と利用する人の移動を同一視している．そして，機器の移動に際して持続的な利用を可能とする為，その移動をシステムやアプリケーションに対して隠蔽する対応方法に主眼が置かれている．例えば，無線ネットワークのローミング技術 [9] やネットワークでの転送技術 [10] によって，ネットワーク間の移動それ自体を隠蔽することが可能である．また，名前解決の更新技術 [11] などによって機器のアドレス変更を隠蔽し，移動前後で互いの到達性を

維持するアドレス解決も可能となる．図 2.2 は，移動環境での持続性を表す．

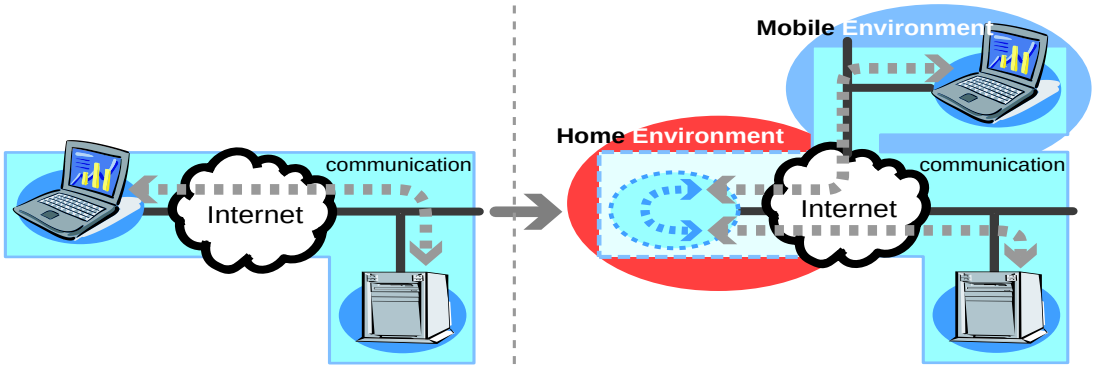

図 2.2: 移動環境と持続性．移動前の環境と移動後の環境で作業を持続する場合，作業は移動前の環境に依存する．この依存性は，データ転送など物理的な場合と，アドレス解決や実際の処理を行う機器など論理的な場合がある．つまり右側の図では，点線で示した “Home Environment” を経由することで，実線で囲んだ部分と同等の通信作業が可能となる．

ここで，上で述べた移動に対する持続的な利用とは，機器の移動に伴って利用者の作業状態を保ち続けることである．つまり，利用者が携帯する機器の移動に対して実行中の作業に移動性を実現する為，移動前後の環境で作業状態を持続する必要がある．作業状態とは，利用者が行っている作業の実行状態や通信状態を示し，主としてアプリケーションの実行状態や機器間の通信状態を示す．そして，移動に伴って環境や状況は変化する為，実行状態や通信状態を持続する様に対処する必要が生じる．ここで，移動型コンピューティング環境では持続性に主眼があることを注意し，次節で述べる移動適応型コンピューティング環境が主眼を置く継続性とは異なることに言及しておく．これについての詳細は次節において説明する．

持続性を実現する為に必要な処理は複雑であり，それぞれのアプリケーションが独自で行う場合には，アプリケーションの作成者に対して大きな負担を与える．だが，既存のアプリケーションは移動を考慮した設計とはなっていない．この為，移動に伴う複雑な処理や対応をシステムでアプリケーションに対して隠蔽し，機器の移動それ自体を見えなくすることによって，透過的に実行状態を持続させる．

しかし，ユビキタスコンピューティング環境において移動をアプリケーションに対して隠蔽することは様々な問題を生じさせる．ユビキタスコンピューティング環境においては，様々に変化する異質な環境が想定されることが挙げられる．そのため，利用者の作業は環境の変化に適応することが必要となる．しかし，機器の移動がシステムによって隠蔽されている為，アプリケーションは環境の変化を検知出来ない．機器の利用は持続されるとしても，同一環境の単純な持続ではない為，移動後の環境に対して十分に適応することが出来ない．その為，常に同一環境を持続させることが，ユビキタスコンピューティング環境での利用形態に適するとは言えない．

2.1.2　分散環境への対応

利用者に伴って移動する機器には様々な制約が生じる．例えば，機器は十分に小型軽量化されていなければ，利用者が携帯して用いることは出来ない．この為，移動する機器は据え付けの機器と比べて，実現可能な機能や性能の面で制約がある．この制約に対して，移動機器の機能や負荷を，実現可能な別の機器と通信ネットワークを媒介として分散させる利用環境が一般的となっている．これらの例を図 2.3 に示す．例えば，移動機器は高性能なサーバのクライアントとして動作し，命令の送信と処理結果の受信を行うもの [12] がある．また，移動可能な実行環境を用意し，その実行環境の間でアプリケーションを移動可能とするものも存在する．そして，演算処理を行う部分と制御を行う部分を分散させ，移動先では画面表示などの制御部分に限って受信可能とするもの [13] がある．

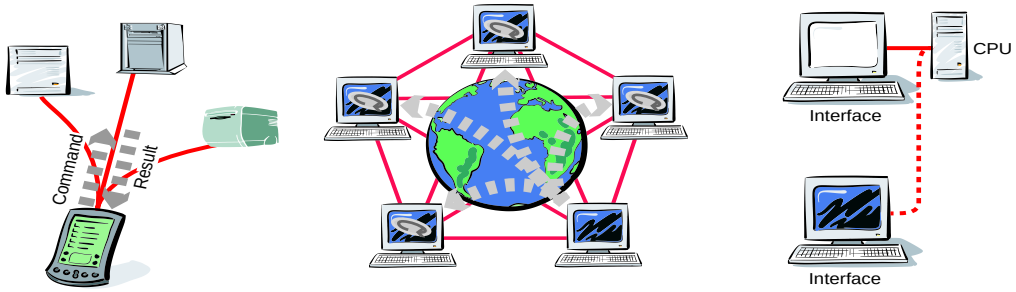

図 2.3: 様々な分散環境．分散環境として 3 つの方式の例を示す．1. 機能が異なるプログラム間で命令の送信と結果の受信を行う方式，2. 実行環境上にプログラムを移動させて目的の動作を行う方式，3. プログラムを特定の機器で動作する部分と表示や制御する部分に分離する方式．

現在，分散コンピューティング環境では，利用形態の変化による分散化が進んでいる．例えば，ネットワークを介した通信形態では，従来の単純な Client/Server 方式に加えて Peer-to-Peer 方式の発展と普及が著しい．これは，コンピュータと通信ネットワークの利用が一般的となり，個人や複数の分散したプログラムが，容易に相互で通信可能な環境が整った為である．また今後は，Bluetooth などを用いたボディーエリアネットワーク環境が実現され，コンピューティング環境の分散化が進んで行くものと考えられる．

一方で，こうした分散化の傾向が新たな問題を生じさせる．例えば，人と人の間で Peer-to-Peer な通信をする際に，その通信を移動に対して適応させる場合を想定する．例えば，移動によって Bluetooth を利用可能な範囲から離れた場合は，別の方法で通信を “継続” させなければならない．つまり，移動前の通信方式が移動後においては利用不可能となる為，同一の通信方式によって “持続” させることは出来ない．移動前後の環境で常に同一同種の機器が利用可能である保証は無く，環境毎の異質性を前提とする必要がある．同様に，共用の機器や資源を移動前の環境で一時的に独占して用いていた場合は，別環境に移動後する前にその機器を解放する必要がある．しかし，その機器を利用せずに移動前後で通信を “持続” させることは難しい．

つまり，ある環境において共用資源を一時的に独占して利用する場合や，様々な環境毎の異質性を前提とする必要がある．そして，分散コンピューティング環境では，こうした場合における移動への対応方法が未解決である．

2.1.3 問題点と限界の整理

従来の移動型コンピューティング環境の特徴は，以下の様にまとめられる．

- 人の移動と機器の移動を同一視する．
- 移動の透過性と移動前後での作業の持続性を実現する．
- 移動前後の環境を “通常環境” と “移動環境” として捉える．

従来の移動型コンピューティング環境では，作業状態の一貫性を実現するために様々な対応方法が考えられている．例えば，移動型コンピューティング環境では，利用者が携帯する機器に主眼を置き，その機器の移動に対して実行中の作業を持続することが目的である．この為，移動先においても通信や作業の “状態を持続する” という設計が前提とされる．そして，これを根拠として，利用環境を “通常環境” と “移動環境” に分類している．ここで，“通常環境” とは利用する人と機器が通常の場合に所属する環境を意味し，“移動環境” とは利用者やその機器が “通常環境” から離れている場合の利用環境を意味する．

だが，この分類方法を用いてユビキタスコンピューティング環境で移動に対応する場合，様々な問題や限界が生じる．これは，移動型コンピューティング環境が，機器とそれを利用する人の移動を同一として捉えている為である．以下，従来の移動型コンピューティング環境における問題点と限界についてまとめる．

移動を隠蔽する手法によって適応性が欠如

特定の移動方式に対する対応方法の依存性と移動の隠蔽処理は，アプリケーションから対応方法の適応性を失わせている．例えば，当初から移動を考慮して設計されたアプリケーションに対して，想定外かつ無意味な弊害を生じさせる場合がある．また，移動の単純な隠蔽では，機器の利用は持続されるとしても，移動後の環境に対して十分に適応することが出来ない．

人と機器の同一視による柔軟性の欠如

人が移動しても，利用する機器が同じであるため，作業や通信の状態を移動前後で持続させるだけで良かった．ユビキタスコンピューティング環境では，移動前後で利用できる機器が同じであるとは限らない．そのため，従来の様に状態を持続させるだけでなく，環境の変化に合わせる必要がある．つまり，柔軟性が重要となるのだが，従来の移動型コンピューティング環境ではそれが欠如している．

“通常環境” と “移動環境” の分類によって依存性が発生

それぞれの対応方法で必要とする移動透過性を実現する手法が利用できない場合には，機器の利用を持続させること自体が不可能となってしまう．つまり，“通常環境” に対する依存性が生じさせる問題として，“通常環境” において動作する機器の安定性に止まらず，“移動環境” と “通常環境” に介在するネットワークの到達性や性能に及ぼす影響がある．そして，これらの影響が予測不可能であるという点で，移動に対する持続性の信頼性を低下させる．

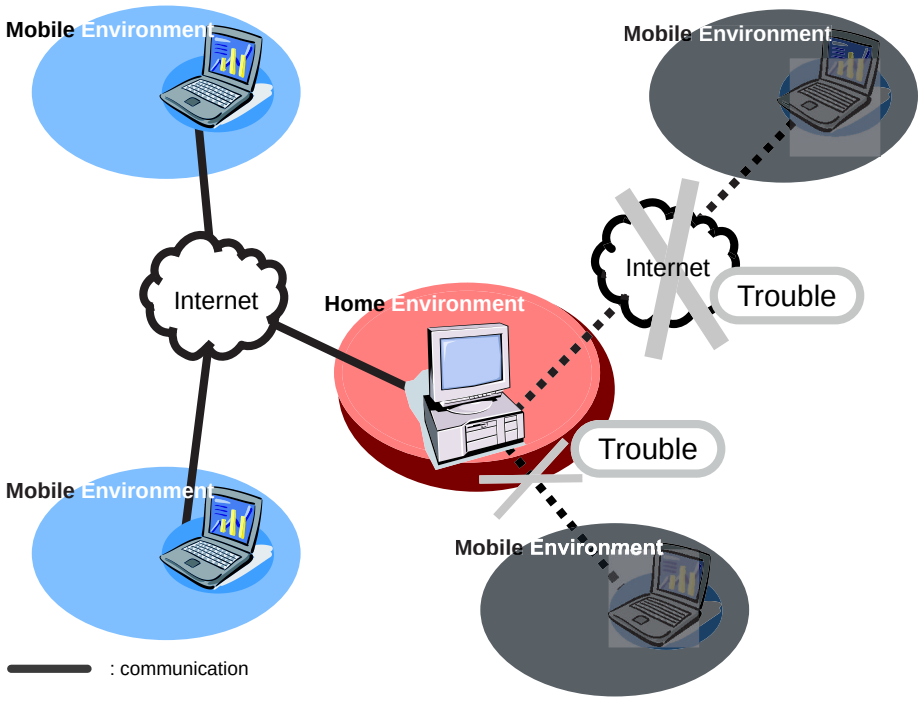

図 2.4: “通常環境” と “移動環境” という分類とその問題．特定の環境に依存した移動への対応は，予測不可能な障害に対処出来ない．また障害は，依存する環境自体に限らず，あらゆる個所で発生することを前提としなければならない．

以上の様に，ユビキタスコンピューティング環境では，従来の移動型コンピューティング環境とは異なった移動への対応方法が求められる．つまり，人の移動に作業状態を適応させると同時に，特定の移動対応方式に依存しない柔軟な移動への対応方法が必要とされている．次節では，これらの問題を解決する為に本論文で提案する移動適応型コンピューティング環境について説明する．

2.2　移動適応型コンピューティング環境

前節までにまとめた問題点や課題の解決を目的として，移動適応型コンピューティング環境を提案する．始めに，本論文で想定する環境の例を図 2.5 に示す．この環境では，利用する人の移動に対して移動前の環境で行っていた作業の継続性を実現する

為，作業の実行状態や通信作業の状態を移動後の環境に適応させている．

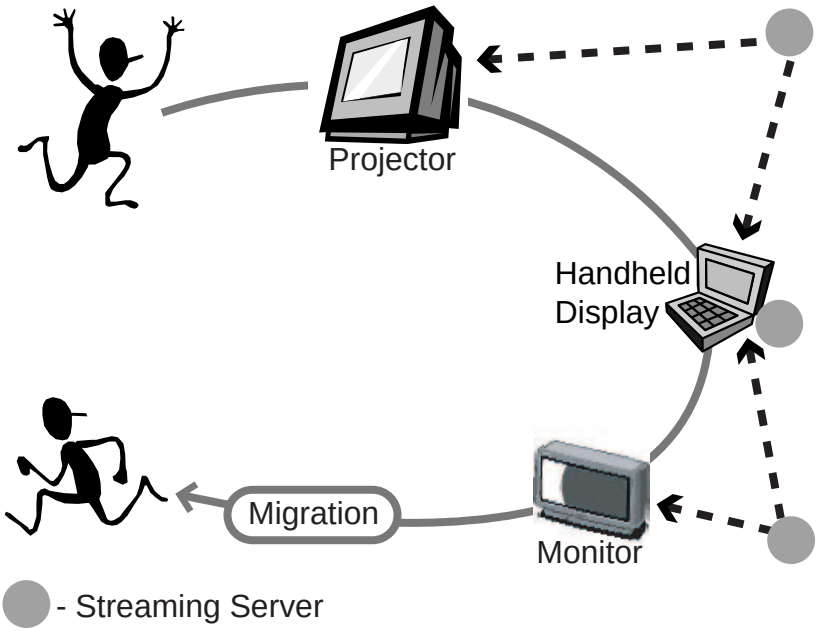

図 2.5: 移動適応型コンピューティング環境．人の移動に伴って，作業状態の状態がそれぞれの環境に適応し，移動前後の環境で作業の継続性が実現される．この図は，映像メディアを鑑賞中の利用者が，移動によって生じる環境の変化に応じて，それぞれの環境で利用可能な Projector や Handheld Display と Monitor を切り替えて利用することを示す．同時に，それぞれの環境に応じて Streaming Server も切り替えるなど，環境に対して適応的に動作しつつ鑑賞作業の状態は継続されていることも示す．

2.2.1 目的

このコンピューティング環境を実現する目的として主要な 3 点は以下の通りである．

- 利用する人の移動に作業状態の状態を適応させて継続すること．
- 特定の移動方式に依存しない柔軟かつ適応的な通信作業の移動を実現すること．
- 利用形態の変化に対応しうる新たな移動対応方式を実現すること．

2.2.2 想定する状況と課題

ここでは，環境の変化により作業状態の継続を行う必要がある状況の具体例を挙げ，解決すべき課題を指摘する．ここでは，図 2.6 において示した “映像メディア鑑賞” という作業を想定した場合を例に説明する．ここで想定する状況は，2 つの場合である．

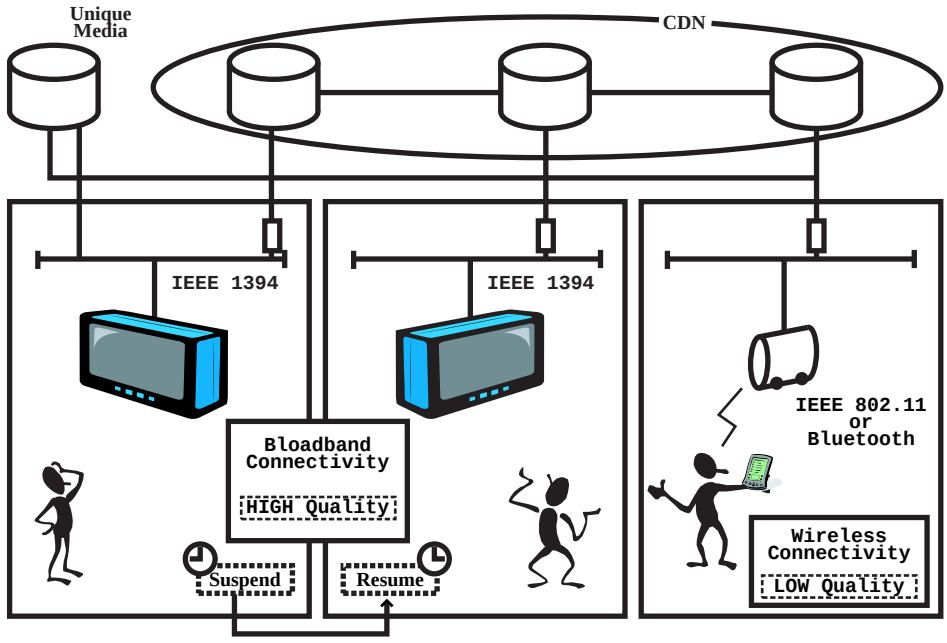

図 2.6: 想定する状況．“映像メディア観賞作業” の場合，環境の変化による機器の切り替えなどの適応処理や利用者の指示による作業の中断処理などが考えられる．

状況 1: 周辺環境への適応

移動などによって生じる環境の変化に対して，利用する機器を変更して適応する場合である．例えば，“手元の小さな無線ディスプレイに表示していた時に，大きな高解像度のディスプレイに切替えて表示する” という状況が想定される．この際，映像のフレーム・レート，解像度，フォーマットなどは，それぞれの環境に対して適応されるべきである．同時に，変更の前後で継続した鑑賞作業を実現する必要がある．

状況 2: 作業の中断と別環境での再開

利用者の意図によって任意の作業状態を保存し，後に別環境で継続的に再開する場合である．例えば，“ある特定のメディア鑑賞を一旦中断し，移動後など全く異なる環境で継続する部分から再開する” という状況が想定される．この際，環境の変化に関わらず作業を継続的に行う為，中断時に作業の状態を保存し，再開時はそれを元に中断した時点から継続して復帰することを実現する必要がある．

2.2.3　移動に対する適応性

ユビキタスコンピューティング環境は，それぞれの場所で環境に組み込まれた機器など，様々な機器を利用可能な機能高度分散型コンピューティング環境である．一般的に，これらの機器はそれぞれの場所においてのみ用いられる為，移動することは想

定されていない．この為，利用する機器が利用者に伴って移動することを前提とは出来ない．そこで，機器とそれを利用する人の移動を切り分けて考える必要がある．

また，移動前後において同一同種の機器を利用可能であることは保証されず，利用する機器の種類や実現可能な機能がそれぞれの環境によって異なることを想定する必要がある．同様に，通信相手の機器が移動や消滅する場合の想定や，共用の機器を利用する場合も考慮する必要がある．よって，様々な面で異なる上，独占して利用が出来ない環境を前提として，移動への対応方法を柔軟で適応的なものとする必要がある．

ここで，柔軟で適応的な移動への対応方法とは，利用者の作業に対して移動前後の作業状態に状態の継続性を実現することである．言い換えれば，利用する人の移動に際して，その実行状態や通信作業の状態を移動後の環境に対して適応させ，移動前の環境で行っていた作業の継続性を移動後の環境において実現することである．

例えば，図 2.5 では，移動によって利用可能な機器など環境が変化する．だが，メディア映像の鑑賞作業という作業の作業状態は，それぞれの環境で適応して継続されている．この場合の適応とは，機器やストリーミングサーバの変更に合わせて，鑑賞作業をそれぞれの環境に適した状態とすることである．

この様に，機能高度分散型コンピューティング環境では，利用者の移動に対する作業状態の柔軟な適応方式が必要とされている．これは，利用者に伴って移動する機器を前提として機器自体に対する移動への対応に主眼を置く，移動型コンピューティング環境での対応方式では実現不可能なことである．そして，移動適応型コンピューティング環境においてこれを実現する．

2.2.4 従来の移動型コンピューティング環境との相違

ここまでの説明から，前節で説明した従来の移動型コンピューティング環境と本節で提案する移動適応型コンピューティング環境の相違点を表 2.1 にまとめる．

表 2.1: 移動型コンピューティング環境と移動適応型コンピューティング環境．

	移動型	移動適応型
移動	人の移動 $=$ 機器の移動	人の移動 $\neq$ 機器の移動
対応手法の主眼	機器の 移動透過性	作業の 移動適応性
移動に対する対応手法	作業の 状態持続性	作業の 状態継続性
環境依存性	隠蔽 によって 依存	適応 によって 独立

次節では，移動適応型コンピューティング環境を実現する為に求められる前提と要件についてまとめる．

2.2.5　求められる前提と要件

移動適応型コンピューティング環境では，様々に異なる環境間で移動する際に，コンピューティングをそれぞれの環境に対して柔軟に適応させることが要求される．また，利用する機器として，利用する人が携帯する機器に限らず，環境に埋め込まれた機器や他人が携帯する機器の場合も想定する必要がある．そして，特にユビキタスコンピューティング環境では，“見えないコンピュータ [1, 14]” という概念も重要となっている．この為，アプリケーションや利用者に対しては，移動を意識させてはならないが，隠蔽することも望ましくない．前者に関しては，移動に対応する為の複雑な処理を利用者やアプリケーション開発者が行う場合は，“見えないコンピュータ” の実現に対する障害となる為である．また，後者に関しては，アプリケーションが移動に対して柔軟に適応することを困難とする為である．

ここで，移動適応型コンピューティング環境の要件は以下の様にまとめられる．

- アプリケーションの性格や利用形態に応じて移動に適応させて継続すること．
- 継続方法は，特定の移動対応方式に対して依存しないこと．
- 移動後，移動前に利用していた環境に負担を残さないこと．

移動適応型コンピューティング環境では，利用者の移動に伴った適応的な継続機構の実現を目的とする．この為，アプリケーションの性格や利用形態に応じて，利用者の作業状態を環境に対して適応させる必要がある．特に，通信中の作業状態を継続させる場合，通信方式の変化や障害の発生などによって，通信方式を変更する必要がある．一方で，同一の通信方式を利用可能な場合はそれを用いることも可能である．つまり，様々な環境に応じて環境の制限や特性への適応が必要である．また，ユビキタスコンピューティング環境では移動後の環境が不明である為，特定の移動対応方式に対して依存することは出来ない．この為，特定の機器や移動対応方式に対する依存性をなくす必要がある．最後に，移動に対する継続処理を行う際，他者が利用する移動前の環境に負担を残さないことは必須である．これが保証されなければコンピューティング環境のスケーラビリティは実現されず，移動適応型コンピューティング環境とは言えない．

2.2.6　基盤技術と研究領域

移動適応型コンピューティング環境を実現するにあたって，図 2.6 で想定される状況において必要とされる必須技術を以下にまとめ，それぞれ順に説明する．

- 移動による変更を検出する為の技術．
- 異なる環境に適応して対応可能とする為の技術．
- 同一の基盤環境を実現する為の技術．

まず，移動による変更を検出する為の技術として，通信環境や利用状態の変化を検出し，適応機構に対して通知する機構 [15] が必要とされる．これによって，機器とシステムやアプリケーションが移動を意識し，適応的に動作することが可能となる．そして，移動する為に異なる環境に対して適応する必要が生じる．ユビキタスコンピューティング環境では，異なる環境において同一の作業状態を継続する為に必要な機器や環境を探し出す機構 [16] が必要とされる．同様に，探し出した機器を利用可能とする機構 [17] も必要である．こうして利用可能となった機器を用いて作業状態を継続させる場合，機器の違いによって継続性に限界が生じる．この為，適応方法としては，単一の移動型通信方式や移動型アプリケーションを越えた柔軟で汎用的な移動対応方式が必要とされる．そして，この移動対応方式によって移動に対する基盤環境を構築する．

この様に，移動適応型コンピューティング環境の実現にあたっては様々な技術が必要とされる．本論文では特に通信管理に注目し，移動適応型コンピューティング環境における適応的な通信の継続に主眼をおく．そして，この為に用いる移動適応型分散通信機構について論じる．

2.2.7　想定される応用分野

移動適応型コンピューティング環境の実現によって想定される応用分野として，以下に示す幾つかの例が考えられる．

まず，映像メディアの鑑賞作業など，リアルタイム性を必要とする処理の場合に，異なる機器に渡って作業状態を継続することが想定される．同時に，移動処理に時間的な連続性を実現することも考えられる．例えば，ある機器で行われていた映像の再生を，ある時点で別の機器で再生する場合が想定出来る．また，移動に伴って映像メディアの取得元を再生中に切り替えることも考えられる．この場合に，鑑賞中の映像が途切れること無く処理することを移動処理の時間的な連続性と言う．

そして，組み込み機器などを利用して作業状態を継続する場合にも優位点がある．例えば，組み込み機器を作成する場合，各機器が外部と連携するインタフェースを提供することのみで利用者の作業状態を継続可能となる為，各機器は自らの機能に対して制限を受けずに実装可能となる．ここで，各機器が外部と連携するインタフェースとは，現在の機器においてリモートコントローラで制御する部分とインジゲータやディスプレイに表示されている情報である．また，機器自体は自らの環境に閉じて動作する為，機器の外部から悪意のある移動プログラムなどによって機器が破壊されたり機能停止する危険性は無い．つまり，移動に対する適応的な処理と同時に，障害に対する安全性や信頼性の両立を図ることが可能であると考えられる．

2.3　移動と関連技術でのアプローチについての分析

本論文と関連技術をそれぞれの関連性について，移動に対して変化する部分の有無で分析すると，表 2.2 の様に示すことが可能である．この表では，縦軸が変化するも

のであり，横軸は対応方法が位置づけられるレイヤである．

既存の対応手法では，人と機器が同時に移動することを前提として，特定の移動対応方法によって実行環境の継続を実現している．しかし，こうしたアプローチでは，様々に変化するヘテロジニアスな環境において，柔軟で適応的な通信の継続を実現することは出来ない．つまり，通信作業の継続性を実現する対応方法が必要である．

本節では関連システムとして，通信プロトコルの OSI7 層参照モデル [18] を参考に分類した対応レイヤ別の視点から，移動に対応する従来のアプローチについて説明する．各システムの分類は，その手法が変更を必要とするレイヤで分類した．

表 2.2: 移動と関連システムでのアプローチについての分析．

Task Core										
HOST										
Data Format										×
Application									×	IMAP, VNC
Middleware								×	Wapplet	｜
Run-Time						MobSock			｜	｜
									｜	｜
Socket						×			｜	｜
Session (RTP)					×				｜	｜
TCP/UDP				×					｜	｜
Network (IP)			×	TCP-R					｜	｜
MAC (NIC)		×	Mobile IP	｜					｜	｜
PHY (Cable)	×		｜	｜					｜	｜
(Radio)	×	Roaming	↓	↓					↓	↓
	PHY	MAC	IP	TCP/UDP	Session	Socket		Middelware	Application (Environment)	Application (Protocol)

ネットワーク層によるアプローチ

このアプローチでは，通信ネットワークのレベルで移動への対応を行う．一般的に，ネットワーク層はホスト上のオペレーティングシステムでカーネル内に実装されている．このため，移動処理はアプリケーションに対して隠蔽することが可能であり，完全に透過的に行われる．よって，既存のアプリケーションに対しても変更を加えずに，移動に対して通信を持続させることが可能となる．

しかし，移動処理はネットワーク層の通信方式に強く依存してしまう．また，実装がカーネル内であるため，システムレベルで対応が行われていない場合はユーザがこの対応方法を利用することが出来ない．このため，移動に対して適用可能な範囲は同一のネットワーク層で移動可能な部分に限られてしまう．

以下では，具体例として Mobile IP [10] について説明する．

Mobile IP

Mobile IP は，ネットワーク層でのルーティングによる対応を基本として，IP アドレスでのホスト移動透過性を実現している．このためホストにおいて移動は完全に透過であり，アプリケーションが移動に対して特別な処理を行う必要がない．

一方，現時点においても様々な問題を内在させている．その一つとしてホームエージェントを必要とすることがあり，これが非効率なルーティング [10] を発生させる．この問題に対して様々な解決方法 [19] が提案されているが，どれも抜本的に解決しているとは言えない．また，利用する環境を移動に関して “通常のネットワーク”(Home) と “移動先のネットワーク”(Foreign) に分けて考えることが設計上の基本となっている．このため，上記の非効率なルーティングやパケット内パケット [20] などが設計上の潜在的なオーバーヘッドとなる．同時に，移動後においても移動前のアドレスを占有してしまい，他者がそのアドレスを利用することは出来ない．これは共有された機器を利用する場合に大きな問題となる．そして，IP を前提としているが，異なるバージョンの IP [21, 22] 間において移動透過性を実現することは現時点において不可能である．

トランスポート層によるアプローチ

このアプローチでは，通信を行っている双方のホスト上で対応を行う．一般的に，トランスポート層はホスト上のオペレーティングシステムでカーネル内に実装されているため，この実装も同様の部分で行われる．このため，移動処理はアプリケーションに対して隠蔽され，完全に透過的に行われる．よって，既存のアプリケーションに対しても変更を加えずに，移動に対して通信を持続させることが可能となる．

しかし，移動処理がトランスポート層の通信方式に強く依存してしまう．このため，移動に対して適用可能な範囲は同一のトランスポート層を利用可能な部分に限られる．実装については，Raw-IP などを用いることによってユーザ空間で実装することも可能であるが，オーバーヘッドが発生し性能上の観点から見て現実的であるとは言えない．

以下では，具体例として，TCP-R [23] と Migrate [24] について説明する．

TCP-R と Migrate

TCP-R と Migrate のいずれも，ほぼ同様の手法で IP アドレスの変更に対して TCP によるコネクションを変更し，ホストの移動透過性を実現している．具体的には，TCP のオプションを利用して IP アドレス更新前後でコネクションを一致させ，そのソケットペア[1]を更新してコネクションを別の IP アドレスに変更させている．

この手法では動作中のコネクションを完全に変更するため，どのような状況においても End-to-End な通信を維持出来るという利点がある．しかし，このアプローチのみでは通信エンドの両者が同時に移動する場合に対応出来ず，また TCP 以外のトランスポート層に対応することは出来ない．また，移動中に通信不能となる場合，TCP のタイムアウトを回避するために別処理 [25] が必要となる．しかし，この処理においては，通信不能となる期間が不確定なため，スケーラビリティの問題を生じさせてしまう．

ソケットによるアプローチ

このアプローチでは，アプリケーションが利用している通信をセッションとして管理することで移動への対応を行う．一般的に，セッション層はホスト上のオペレーティングシステムにおいて，カーネル内とアプリケーションライブラリのどちらでも実装することが可能である．このため，移動に対応するための機構をライブラリとして実装すれば，ユーザの必要に応じて利用可能とすることも可能となる．

この場合，カーネル内や既存にあるライブラリの内部で実装した場合は既存のアプリケーションでも変更をせずに移動に適応させることが可能となる．一方で，新たなライブラリや既存ライブラリの拡張として実装した場合には，アプリケーションもそれに応じて書き直す必要が生じてしまう．

この具体例としては，MobileSocket [26] などがある．以下では，MobileSocket について説明する．

MobileSocket

MobileSocket は，Java [27] によるユーザレベルの拡張ソケットライブラリである．これによって，Java モバイルアプリケーションやエージェントに対し，ライブラリーに基づいたセッションレイヤーとして，アプリケーションの移動性と継続性をサポートする．これによって，持続的な通信を保証することが可能となる．しかし，このアプローチのみでは通信エンドの両者が同時に移動する場合に対応出来ず，また Java 以外のアプリケーションに対応することは出来ない．

[1]通信中のノードで両エンドの IP アドレスとポート番号 <dst_addr,dst_port,src_addr,src_port>.

ミドルウェアによるアプローチ

このアプローチでは，共通の実行環境を用意することによって移動への対応を行う．共通の実行環境は，移動用ライブラリや同一の実行環境を提供することによって実現している．移動用ライブラリを用いる場合は，適応型アプリケーションと移動型アプリケーション，そして両者の組み合わせの場合として，３つに分類することが可能である．最初のものは，様々な環境に移動する機器上で移動による環境などの変化に対して適応することを目的としている．また，次のものは，異なる機器間を移動して実行状態を保つことを目的としている．最後のものは，この２つの目的を実現する．また，同一の実行環境には移動プロセスや，次項で説明するモバイルエージェントを実現するための基盤環境も含まれる．

このアプローチによる具体例としては，Wapplet [17] や Mogul [28] などが挙げられる．以下では，Wapplet と Mogul について説明する．

Wapplet

Wapplet は，ユーザの周囲に偏在するネットワークに接続されている計算機資源やデバイス，アプリケーションをユーザの移動に伴って協調動作させる枠組みとして提案された．ユビキタスコンピューティング環境では，ユーザの移動と伴にユーザの周囲の環境及び，利用可能なサービスが変化する．この環境変化に対応する枠組みとして Wapplet では，オブジェクト移送機能，位置透過的メソッド呼び出し機能，(デバイスの抽象化のための) デバイス非依存なインタフェース定義機能の 3 つの機能からデバイス，アプリケーションの協調動作を実現している．オブジェクト移送機能によってユーザの移動に適応したデバイスの切り換えを実現し，また異なる器機間を移動した際にも移動前と同一の実行環境の提供を実現している．位置透過的メソッド呼び出し機能によってデバイスの位置をユーザやアプリケーション提供者から隠蔽することで，アプリケーションの位置透過性を実現している．だが，通信作業などを行う場合には，移動する環境下で適応的に切り替える処理等が実現できていない．

Mogul

Mogul は，Java で実装された，アプリケーションをホスト透過的にオブジェクト単位で移送するモデルである．このモデルではホストに依存せずに動作する全てのオブジェクトが，ホスト間での移送機能を持つ．これに対しホストに依存して動作するオブジェクト（以下ホスト依存オブジェクトと呼ぶ）に関しては，それ自身は移送されず，代替として当該オブジェクトのスタブオブジェクトが動的に生成され移送される．同オブジェクトへのメソッド呼び出しは，移送元ホスト上の対応するホスト依存オブジェクトへリダイレクトされる．以上の機能により，Mogul は既存アプリケーションコードへの変更を伴わずにホスト透過的なアプリケーション移送を実現した．しかし，移送対象アプリケーションがホスト依存オブジェクトを参照する場合，同アプリケー

ションは移送元ホストに依存して動作するため，単一ホスト上での動作と比較して障害発生点が増加するという問題点を有する．また Java 以外のアプリケーションに対応することはできない．

アプリケーション層によるアプローチ

このアプローチは，それぞれの対応方式による違いから 2 種類に分類することが可能である．それは，サーバとインタラクティブに通信しながら動作する方式と実行画面のみを転送することによって移動先でも利用可能とする方式である．

前者の場合は，サーバとインタラクティブに通信しながら動作するクライアントを用いて，サーバとクライアントの通信方法を規定している．このため，規定に沿ったクライアントであれば，どのようなものでも利用可能となる．これによって，移動前後でも作業を継続することが可能となる．

後者の場合は，実行画面が手元の機器に転送されて利用可能となる方式である．この場合は，必ずしも物理的に機器が移動するとは限らない．しかし，X Window System [29] と同様に捉えれば，分散コンピューティング環境による移動への対応であると見なすことも出来る．X Window System では，画面の表示部分 (X Server) とプログラムの実行部分 (X Client) が分離されている．そして，こうした場合に画面表示を行う部分だけをユーザの手元に移動させている場合と同等に捕らえられる．

どちらの場合も，固定されたサーバ側が全ての情報を保持しており，クライアントは単純なビューアとして働くという特徴がある．

このアプローチによる具体例としては，IMAP [30] と screen [31] や emacsclient [32] と VNC [13] などが挙げられる．以下では上記の分類で，前者の例として IMAP について，後者の例として VNC と emacsclient について説明する．

IMAP

移動を想定したメイルアクセス環境を提供する操作手順を規定している．基本的に，IMAP サーバ側に全てデータとそのステートが保存されており，IMAP クライアントはそれにアクセスするビューアとして動作する．このため，複数のクライアントから同時アクセス可能であり，それは同一の機器上であっても別々の機器上であってもよい．

screen

UNIX システムの仮想端末に状態保持機能を実現するアプリケーションである．このアプリケーションを用いることによって，別システムの端末から同一システムの全く異なった仮想端末を経由して，以前に利用していた仮想端末の利用を継続することが可能となる．これによって，移動やネットワーク状態の変化によって生じる通信の切断に対して，切断前の状態を利用することが出来る．

VNC と emacsclient

どちらも画面表示とキー入力などの操作インタフェースを移動先の機器に転送して，移動先の機器で移動前に行っていた作業の継続を可能とするものである．しかし，両者には決定的な違いがある．画面表示に関して，VNC はデスクトップ全体の表示画面を対象とするが，emacsclient は Emacs という特定のアプリケーションに依存する．つまり，機器全体で対象とする場合と個別のアプリケーションを対象とする場合の相違である．また，移動前の機器上で転送するためのサーバとして常時動いているアプリケーションが必要である．このため，移動前の機器を常に占有することとなる．

エージェントによるアプローチ

エージェントと呼ばれる自律的なモジュールが，あらかじめ用意された移動可能なプラットフォーム上を自ら移動して，目的の作業を実現する

特に移動エージェントは，ネットワーク上の機器を移動しながら，ユーザの目的を代行して処理するソフトウェアであり，具体例として，MOLE を挙げて説明する．

MOLE

Mole には抽象的な Place という概念があり，それぞれのエージェントは Place 間を移動 (マイグレーション) する．マイグレーションが起きて新しい Place に行くと，Agent の状態や実行するコードが変わる．Place は一つのノードに一つ以上存在し，訪れたエージェントとローカルのサービスを安全に実行できる環境を提供する．エージェントにはそれぞれ Global に固有な ID が振られ，その ID によって位置透過性を実現する．

2.4 まとめ

本章では，始めに従来の移動型コンピューティング環境と移動適応型コンピューティング環境における移動性ついて説明した．続いて，移動と関連システムでのアプローチの関連性について考察した．

次章では，移動適応型コンピューティング環境での通信作業を適応的に継続する為の移動適応型分散通信モデルを提案して説明する．

第3章　移動適応型分散通信モデル

移動適応型コンピューティング環境の概要とそれを実現する必要性について，前章で説明した．

本章では，移動適応型コンピューティング環境を実現する移動適応型分散通信モデルを提案する．移動適応型分散通信モデルでは，移動によって様々に変化する環境において，利用する人の移動に対して柔軟で適応的な通信作業の継続を実現可能とする．

本章では，本論文で想定するシナリオを示した後，移動適応型分散通信モデルの概念について説明する．続いて，本機構の特徴や性質と要件や必要機構についてまとめる．最後に，本機構の関連研究との比較を行う．

3.1　移動適応型コンピューティング環境と通信作業の適応性

始めに本論文で想定するシナリオを示し，通信作業の適応性について分析する．その後，移動適応型分散通信モデルの概念について説明し，要求される機能をまとめる．

3.1.1　想定するシナリオ

本論文で想定する環境の具体的なシナリオとして，ビデオストリーミング映像を鑑賞中に移動する場合が挙げられる．このシナリオのイメージを図 3.1 に示す．

ビデオストリーミング鑑賞作業

継続した
ビデオストリーミング鑑賞作業

図 3.1: 想定するシナリオの例．左上図の環境でストリーミング映像を鑑賞中に，移動に伴って図が示す状態で鑑賞作業は中断される．右下図の環境に移動後，鑑賞再開の命令によって，移動前の状態に続く鑑賞作業が環境に適応して再開される．こうして，ビデオストリーミングの鑑賞作業は，環境の変化に対して適応的に継続されている．

まず，左上図の環境は，共有のスクリーンに投影されたストリーミング映像を同時に複数人で鑑賞している状態である．この時点で，利用者が移動など何らかの理由によって鑑賞作業を中断した場合をこのシナリオでは想定する．そして，右下図の環境は，左上図の環境で中断した鑑賞作業を移動後の異なる環境に適応させて，同じ時点から継続させて再開した状態である．ここでの移動に伴って生じる注意すべきことは 2 点ある．それは，物理的な位置とネットワーク的な位置の変化や実行環境などの変化と，鑑賞作業を中断して再開するまでに経過する時間や移動後の環境利用可能な機器などの不確定性である．こうした移動に対応する為には，変化に対する適応性と不確定性に対する柔軟性が必要とされる．移動適応型分散通信機構ではこれらを実現する．

3.1.2　通信作業の適応性

前節のシナリオでも示した通り，ユビキタスコンピューティング環境に代表される機能高度分散型コンピューティング環境では，利用者がその時点に存在する環境で利用可能な機器や資源を用いて作業を実現する．この為，利用者の移動や周辺環境自体の変化によって，利用する機器の変更が必要とされる．

この時，通常の環境と通常とは異なる環境や移動中の環境などにおいて，常に同種の機器が利用可能であると想定することは現実的でない．そして，この場合にはそれぞれの環境で適したサービスが利用可能であることが望ましい．例えば，音楽鑑賞という作業の場合には以下のようなことが言える．静かで良質のスピーカが利用出来る環境では，出来る限り高品質な観賞作業を実現することが求められる．これに対して，頻繁に環境が変化する移動中などには，移動や環境の変化に適応する為のコストが最も低い作業状態を実現することが望ましい．例えば，ヘッドホンや小電力な機器を用いることが合理的である．

図 3.2 に，前節のシナリオで発生している通信作業の適応性を示す．ここでは，利用者の移動に伴って観賞作業が適応的に継続される場合を表している．この図で注意すべきことは２つある．１つは，それぞれの環境で適したストリーミングサーバを利用していることである．そしてもう１つは，移動前の環境に対して依存せずに移動後の観賞作業を継続することが可能であるということである．

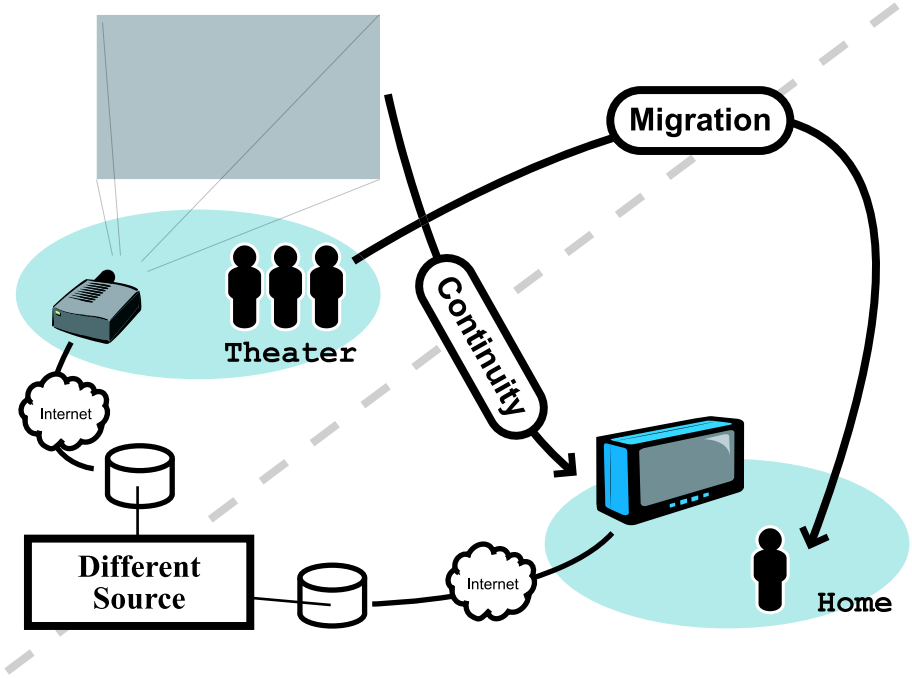

図 3.2: 通信作業の適応性．

ここで述べた様に，異なる環境を移動する際に作業の継続性を実現する為には，それぞれの環境間で利用者の作業状態を継続させる機構が必要となる．次節では，通信作業においてこれを実現する移動適応型分散通信モデルを提案する．

3.1.3　移動適応型分散通信モデル

本論文で提案する，移動適応型分散通信モデルの動作概念図を図 3.3 に示す.

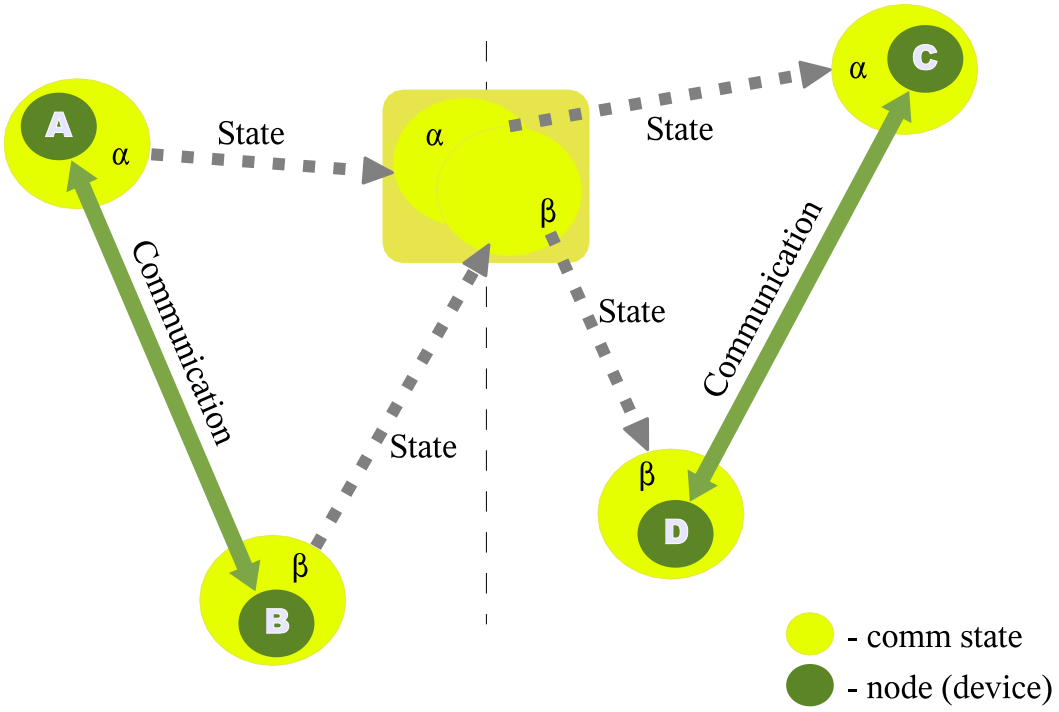

図 3.3: 移動適応型分散通信モデルの動作概念図.

この図では，始めに図の左側でノード A とノード B で行っていた通信を図の右側でノード C とノード D の通信として継続させる場合を示している．この結果，移動に対して継続される部分は，それぞれのノードを結ぶ同一の波線で示した通信作業となる.

前述の通り，移動後の環境には不確定要素が存在する場合がある．この場合，ノード A とノード C やノード B とノード D で同一の機器や実行環境が提供されることを前提とは出来ない．また，仮に同一の機器を利用する場合であっても，複数の機器を用いて機器間で移動に対応する適応機構が未整備であるという問題がある．同様に，同一の実行環境によって移動アプリケーションを利用可能な場合であっても，その利用環境における適応的な通信管理の問題は以前として未解決である．そして，前述の通り，同一の機器や利用環境を利用することが，必ずしも移動後の環境に適した状態であるとは言えない.

特に，移動に伴って生じる変化に対する適応性と不確定性に対する柔軟性が必要とされている．これを実現する為，通信作業に関連する各ノードの状態を管理し，この状態を移動に対して継続することが必要とされる．ここで状態とは，図 3.3 で各ノードを覆う円 α と β で示した部分であり，通信作業を構成する各部分に関する情報である．また，状態の具体的な内容については後述する.

3.2　移動適応型分散通信モデルに要求される機能

本節では，前節で提案した移動適応型分散通信モデルにおいて要求される特性についてまとめる．これらは，以下に示す４つが挙げられる．

3.2.1　環境適応性

移動によって生じる様々な環境の変化に対して，それぞれの環境で適した動作とする為に作業を変更することが必要である．例えば，利用する機器とアプリケーションの変更をする場合や通信する相手を変更する場合などが作業の変更として想定される．

この為，移動を十分に意識しつつ，環境の変化に対して適応的に動作する機構が必要である．この機構によって，移動に対して適応した動作をすることが可能となる．

3.2.2　運用柔軟性

様々に異なる機器や環境で利用することを前提として，移動後の環境に対する不確定性も考慮すべき項目である．この不確定性を生じさせる原因としては，移動前後で同一同種の機器やアプリケーションを利用出来ない場合に限らず，通信作業を中断し再開するまでの時間的な間隔なども含まれる．

こうした環境での移動を適応的に行う為には，特定の環境に対して依存すること無く環境の変化に対応する必要がある．つまり，通信作業の環境独立性と特定の機器やアプリケーションに依存しない適応機構の実現が求められる．これによって，移動と継続の際，特定の環境から非依存な適応性と環境変化の不確定性に対する柔軟性を実現出来る．つまり，移動に対する継続の手法に柔軟性を実現出来る．同時に，構成変更の自由度や耐障害性などの不確実性に対しても柔軟な適応をすることが可能となる．

3.2.3　環境独立性

機能高度分散型コンピューティング環境の特徴や他者と共用する機器を利用する環境である点を前提とする為，通信の移動適応性を実現する機構は利用中の環境内にある機器や資源のみを用いて実現可能である必要がある．つまり，従来の移動型コンピューティングで実現されている，移動前の環境に依存して持続する移動への対応手法とは異なる移動適応性を実現する必要がある．

移動後の環境を移動前の環境に対して独立したものとする為に，移動前の環境に対して依存性を持たない移動適応機構が必要である．また，これによって，移動後の環境に適応する際の柔軟性や移動前の環境へ依存することに起因する障害の影響を受けずに，適応的な継続動作を実現可能となる．

3.2.4 状態継続性

利用する環境が変化する前後で何らかの繋がりを実現できなければ，移動とは言えない．つまり，異なる環境間で作業の内容に共通する部分が必要である．既存の対応手法では，移動前の環境に対して何らかの持続性を保つことによって，移動前後の変化する環境で通信を移動に対応させている．

ここで，通信作業とその移動の状態に注目し，移動に対して通信作業を利用環境に適応させることが考えられる．そして，移動前後で通信作業の状態を共有することが出来れば，移動に対して通信の状態継続性を実現することも可能となる．この為，移動した環境に適応後，移動前の環境での作業状態を継続させることが要求される．そして，これを実現することによって，環境に対して独立に作業状態を継続することが可能となる．

3.3 関連研究

本節では，本論文の関連研究を挙げ，前節までにまとめた要求特性と必須機能に基づいた視点から考察を行う．要求特性では，移動適応性と作業継続性，及び環境独立性の実現度に注目する．また，必須機能では，通信切替と通信状態抽象化・管理の達成度に注目する．ここでは，関連研究として以下に示す３つを挙げて説明する．

3.3.1 M-TCP: Migratory TCP

M-TCP [33] は，クライアント・サーバ間の通信において，TCP の状態転送によって耐故障性と負荷分散を実現するものである．クライアント側からの命令を元に複数のサーバ間で TCP コネクションの状態を転送することによって，クライアント・サーバ間の通信をアプリケーションに対して透過的に継続させる．

ここで，変更可能な部分はサーバ側に限られ，クライアント側が変更されることは想定されておらず，環境適応性が十分に実現されているとは言えない．一方，同一のセッションを持続可能であるという点で，作業継続性を実現している．しかし，TCP という単独のトランスポートプロトコルのみに依存している為，環境独立性は実現されていない．M-TCP では，通信の切替を TCP によって行うことは可能となっている．だが，通信状態を抽象化しているとは言えず，その管理も行っていない．つまり，TCP というトランスポート層のプロトコルに強く依存する為，移動に対して適応的な動作を実現することが出来ていない．

3.3.2 Stream Redirection Architecture

Stream Redirection Architecture[34] は，データ送信サーバ “server” とデータ受信シンク “sink” の間にシンクプロキシ “sink proxy” という機構を介在させ，利用者の移動に

対してデータ転送の持続性を実現するものである．機器を利用する場合には，利用者が携帯するコントローラ “Controller” を通じてシンクプロキシに対してコマンドを発行し，シンクプロキシによって全ての通信動作が実行される．ここで，シンクプロキシは各環境毎に用意されていることが想定され，サーバやシンクと同様に移動しない．

利用者が別環境に移動した場合，コントローラが移動を検知して新たに利用可能なシンクプロキシを発見する．ここで，利用していたシンクプロキシでの通信を中断し，接続状態を格納するオブジェクトを同シンクプロキシからコントローラに転送する．そして，別環境で引き続いてデータ転送を行う目的で，コントローラは新たなシンクプロキシに先ほど受信したオブジェクトを転送して接続状態を復元する．こうして，シンクプロキシは同一のサーバを用いて同一なバイトオフセット[1]からデータ転送を持続して行うことが可能となる．

だが，Java という単一の実行環境を用いており，持続されるデータ転送は送受信されるデータの内容や形式に強く依存する．同時に，サーバやシンクが常に固定機器という前提や，同一のデータ送信サーバを用い続けることが，通信の適応手法から柔軟性を失わせる．つまり，移動に対して適応的な動作を実現することは出来ていない．

3.3.3 VNC: Virtual Network Computing

前述の通り，VNC [13] は，画面表示とキー入力などの操作インタフェースを転送することによって，遠隔にある別のコンピュータを操作可能とするものである．ネットワークに繋がった外部から常時アクセス可能なコンピュータで予め転送用のプログラムを起動しておき，遠隔のコンピュータから制御用のプログラムを用いることによって，操作可能となる．

VNC では，画面サイズや入力デバイスなどに合わせた適応動作は行われない為，転送用と制御用のコンピュータの環境に相違がある場合は問題となる．また，状態継続性は単純な操作インタフェースの部分に限って実現可能である．つまり，操作インタフェースに関しては，ほぼ同等の作業が継続可能であるが，音楽鑑賞などコンピュータの周辺機器を直接利用している場合には対応できない．そして，常に移動前に利用していたコンピュータとその環境に依存し．共有の機器や資源を利用することを想定する場合は適用できない．一方，利用者がクライアントアプリケーションを起動することによって，操作インタフェースの転送が行われ，通信の切替とは言えない．また，通信状態の抽象化と管理は移動前のコンピュータの状態と同一である為，敢えて抽象化して管理することが想定されていない．言い換えれば，常に全ての状態を移動前の環境が保持していると言える．前章で関連システムとして挙げた screen や IMAP も同様にして状態管理を行っているが，これらの方式では常に移動前のコンピュータやその環境に強く依存してしまう．この為移動適応性が十分に実現できず，全てのアプリケーションをこの手法で実現することは出来ない．

[1] データ転送開始時から転送されたバイト数.

3.4 まとめ

本章では，移動適応型コンピューティング環境を実現する為の移動適応型分散通信モデルを提案した．始めに本論文で想定するシナリオを示し，本機構の概念について説明した上で，本機構の特徴や性質と要件や必要機構についてまとめた．これを元に関連研究を挙げて比較を行い，既存の手法では移動適応型コンピューティング環境の実現が困難であることを示した．移動適応型分散通信機構は，様々に変化するヘテロジニアスな環境において，利用する人の移動に対して柔軟で適応的な通信作業の継続を可能とする機構である．次章から続く 2 章では，本章で示した通信モデルとその特性に基づいて，これを実現する移動適応型分散通信機構の設計と実装について説明する．

第4章　設計

前章で，移動適応型コンピューティング環境の実現を目的とする移動適応型分散通信モデルを提案した．移動適応型分散通信モデルによって，利用者の移動に伴って様々に変化する異質な環境で，人の移動に対して柔軟で適応的な通信作業の継続性を実現することが可能となる．

本論文では，以上を実現する機構を移動適応型分散通信機構と名付け，本章でその設計の詳細について述べる．

4.1 概要

始めに，前章で提案した移動適応型分散通信モデルを実現する移動適応型分散通信機構の設計方針や全体構成と基本動作について述べる．続いて次節以降では，本機構を実現する各部分について順に説明する．

4.1.1 設計方針

本論文の目的は，移動に伴って様々に変化する環境で，人が行う作業や通信の状態を適応させて継続することである．本機構の設計方針として４点を以下にまとめる．

環境に対して独立であること

本論文が想定する環境では，第 2.2 節でも述べた通り，人の移動に伴って作業状態の状態をそれぞれの環境で適応して目的の作業を実現する必要がある．だが，従来の移動型コンピューティング環境で用いられる対応方法では，対応方法が特定の環境に依存する為，様々な環境に対応出来ない．また，常に特定の環境を持続する手法は，単一の環境に対して依存性を生じ，様々に変化する環境での移動に適しているとは言えない．この為，特定の環境に対して依存しない移動適応機構が必要である．これによって，環境に適応する際の対応方法に柔軟性を実現出来る．そして，環境変化に左右されない作業状態の継続が可能となる．

作業状態を抽象化すること

様々に異なる環境下で作業の適応的な継続性を実現する為には，特定の作業状態を単に持続することは出来ない．この為，作業状態を抽象化し状態を管理することによって，適応方法や継続方法の環境非依存性を実現する必要がある．ここで，異なる種類の機器を切り替えて利用者の作業状態を継続する場合，その利用者の作業状態を取り扱う上で不可欠なことが 2 つある．1 つは，ある時点での通信環境や実行環境に依存せずに，利用者の作業状態を形式化して記述することである．そして，この際に問題となることは管理すべき情報とその記述方式である．また，もう 1 つは，すでに記述されている作業状態の表現形式から利用者の作業状態を復元して，復元された機器を利用者に利用可能とすることである．この際は，取得した情報から作業環境を再構築することが重要となる．これらによって，移動に対する作業状態の継続に柔軟性を持たせることが可能となる．

適応動作の隠蔽と適応的な処理を両立すること

機器の開発者や利用者の観点から見れば，移動に対する複雑な適応動作は隠蔽され単純化されている事が望ましい．一方で，第 2.1.3 項でも述べた通り，単純に移動を隠蔽する手法はその対応方法に環境依存性を生じ，結果的に機器やアプリケーションから対応方法の柔軟性や適応性を失わせてしまう．つまり，本機構が汎用的な機構として成立する為には，複雑な適応動作を隠蔽する必要があるが，それ故に適応性が失われる結果とならない様に注意して設計する必要がある．

適応処理が適したものであること

様々に異なる環境下で移動に対応する場合，それぞれの状況で必要な適応処理は異なる．一方，単一の移動対応方式のみを用いる場合には，対応可能な範囲が限られてしまう．また，従来の手法では移動対応処理が透過的に行われる為，適応処理を行っているかどうかが判断出来ない．つまり，移動に対する対応手法を単一に限定することなく，それぞれの状況下で適した手法を用いることを可能とする様に設計する必要がある．

安全であること

インターネットの現状を見れば，ネットワークに接続される機器の危険性は指摘せずとも明らかである．この為，機器は当初から安全性を十分に考慮して設計する必要がある．つまり，利用者の作業に関わる適応動作や継続動作を安全に行うことが出来なければならない．例えば，利用する機器に対して権限を持たない者からの不正な要求に対する保護が必要である．つまり，本機構が提供する機能によって，悪意を持つ第三者が，正当な利用者の作業を無効とすることや妨害することが無い様に注意して設計すべきである．

4.1.2 全体構成

前項の設計方針を元に，本機構の全体構成と基本動作について述べる．本機構は以下に示す５つの部分から構成され，それぞれの部分は 図 4.1 で示す関連がある．

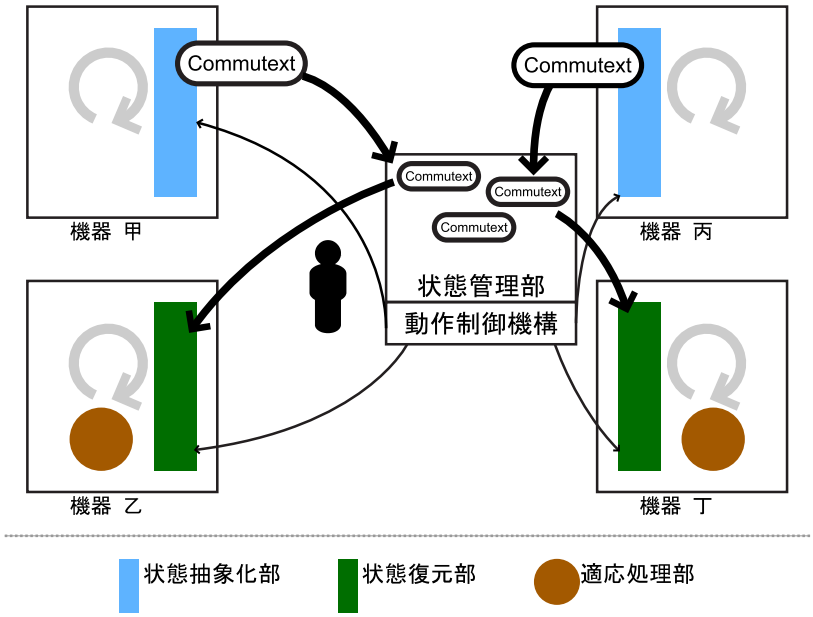

図 4.1: 本システムの全体構成図．

ここで，状態抽象化部，状態復元部，適応処理部は，それぞれの環境で機器毎に存在する．状態管理部は，利用者の作業状態を管理し，実際に状態の転送を行う部分である．また，動作制御部は，利用者が他の部分を制御する為に用いる部分である．そして，状態管理部と動作制御部は利用者から利用可能な範囲に１つ以上必要である．以下，それぞれの機能について解説する．

動作制御部

動作制御部は，他４つの部分を統制して制御する部分である．同時に，本機構の利用者や利用プログラムに対してコマンド制御を可能とする機能を提供する．

状態抽象化部

利用者の作業状態を形式化して記述する部分である．ここで抽象化される情報を元に他の部分が動作することによって，特定の通信環境や実行環境に依存しない利用者の作業状態を継続可能とする．

状態管理部

状態抽象化部で抽象化する利用者の作業状態を管理する部分である．具体的には，状態抽象化部から状態を取得し，状態復元部に反映する．これよって，異なる機器間での作業状態の転送を実現する．

状態復元部

状態復元部は，移動後の環境に利用者の作業状態を復元可能とする部分である．これによって，移動前の環境で行っていた作業状態の抽象化された記述を元に作業状態を復元し，利用者の作業状態を移動前の環境から継続的なものとする．

適応処理部

適応処理部は，実際の移動に対応した適応動作を行う部分である．適応動作としては，実行状態や通信状態の継続性を実現する機構が必要である．具体的な例として，通信の切替を行って通信状態を継続する場合が考えられる．

4.1.3　基本動作

これらの機構は次の様に動作する．動作の流れを 図 4.2 に示す.

1. 動作制御部が，状態抽象化部に対して作業状態の抽象化を命令する．

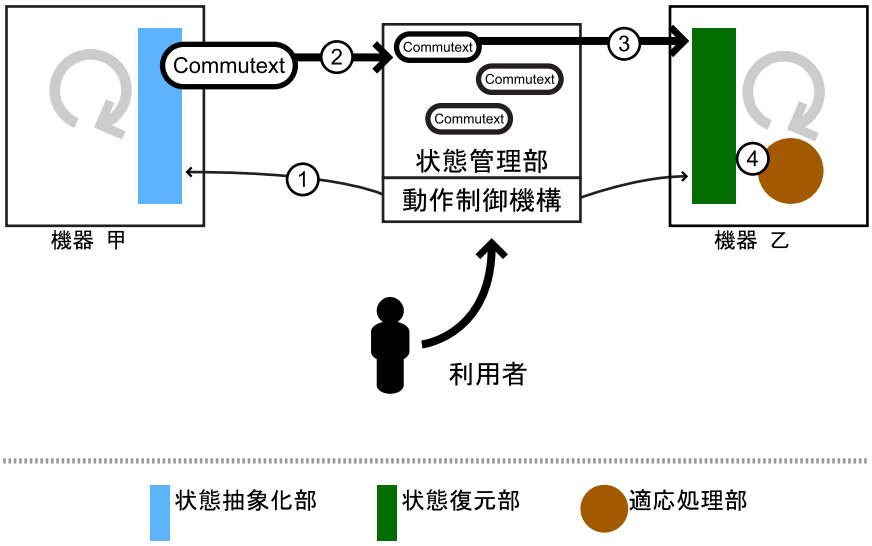

図 4.2: 本システムの基本動作図.

2. 状態抽象化部で作業状態が抽象化され，状態管理部に保存されて次の命令を待つ.
3. 動作制御部の命令で，作業状態が状態管理部から状態復元部に転送される.
4. 適応処理部が実際の適応動作を行い，利用者の作業状態が適応的に継続可能となる.

次節以降では，本項で述べた本機構を構成する各部について設計の詳細を説明する.

4.2 動作制御部

動作制御部は，それ以外の部分を統制して制御する部分である．同時に，本機構の利用者や利用プログラムに対してコマンド制御を可能とする機能を提供する.

移動に対する適応動作は．利用者や本機構を利用する別プログラムからのコマンドを元に行う．始めに動作制御部では，移動前の環境で利用していた機器の状態抽象化部から Commutext を取得し，状態管理部に保存する．そして，移動後の環境で作業を再会する場合には，保存された Commutext を状態管理部から状態復元部に転送する．こうして，システム全体の動作を制御可能とする.

4.3 状態抽象化部

状態抽象化部は，利用者の作業状態を形式化して記述する部分である．状態抽象化部では，利用者の作業を実現する為にその機器内で動作しているアプリケーションなどの動作状態を形式化して記述し，機器の内部や外部から取得することを可能とする．

4.3.1 状態の抽象化

様々に異なる環境下で適応的な継続動作を実現する場合には，ある時点での通信環境や実行環境に依存せずに，抽象化された作業状態を元に動作することが不可欠である．これによって，特定の通信環境や実行環境から独立した柔軟な適応処理が可能となる．前節でも述べた通り，これを設計する上で注意すべきことは，利用者の作業状態を形式化して記述する際の問題であり，記述する情報とその記述方式である．

記述する情報

抽象化する作業状態は，作業の定義や属性とその進行状態である．これによって，特定の作業状態を選択した上で，抽象化した作業状態として取り扱うことを可能とする．

ここで，定義や属性としては，以下に挙げるものがある．

- 作業状態の識別子
- 作業の目的
- 機器の構成情報や定義
- ネットワークの通信特性

また，進行状態としては，以下に示すものがある．

- 機器やアプリケーションの実行状態
- ネットワークプロトコルの通信状態

記述方式

本論文では，利用者の作業状態を形式化して記述したものを “Commutext”[1]と名付ける．Commutext の記述方法としては，構造を形式化して記述可能な記述方式である XML [35] を用いる．また XML は，全ての内容でテキストを用いて表現する為，様々な環境下で伝達可能な記述方式である．つまり，環境非依存なデータ記述方式としても適している．そして，第 4.5 章で説明する状態復元部によってこれが用いられ，移動後の環境で利用者の作業状態が復元される．そして，状態の継続性が実現される．

[1] “Commu”nication Con“text”

4.3.2　状態抽象化部の方針

状態抽象化の手法や状態抽象化部の設置単位について本節では述べる．

状態抽象化の手法

状態抽象化の手法としては，次に示す２通りがある．

1. プロキシモデル
2. インタフェースモデル

前者は，アプリケーションなどの動作や通信に割り込ませて状態を抽象化する手法である．後者は，アプリケーションに対して状態保存用のインタフェースやライブラリを提供して，状態の抽象化を可能とする手法である．だが，特に進行状況を取得する場合，プロキシモデルではプロキシ機構にも通信データの意味解釈をする機能が必要となり，処理の重複を生じさせる [36]．この為，効率性の面で現実的ではない．また，アプリケーション実行状態の全てを抽象化可能にすることは出来ない．一方，インタフェースモデルでは，既存のアプリケーションに対する適用性が問題となる．

切替処理を透過的に行うためには，通信状態は

抽象化部の設置単位

機器内で抽象化に用いる情報として，実行状態などアプリケーションレベルの情報と通信状態などシステムレベルの情報がある．本機構では，これらの情報を単一のアプリケーションに関してまとめて取得可能とし，形式化して記述する．この為，状態抽象化部は機器に単一のものとする．

この状態抽象化部は，動作制御部からソケット通信によって制御される．

4.4　状態管理部

状態管理部は，状態抽象化部で抽象化された利用者の作業状態を管理する機能を提供する部分である．

状態の管理

通信状態の管理とは，Commutext を機器から抽出して管理することであり，それを機器に対して反映することである．例えば，異なる機器間で通信を切り替える場合には，その機器間で通信の状態を共有する必要がある．この時，移動前の機器から Commutext を抽出し移動後の機器に転送することによって，通信状態の共有が可能となる．つまり，Commutext を管理することで，環境に非依存な適応性と環境変化の不確定性に対する柔

軟性を実現可能となる．この部分の具体的な動作として，状態抽象化部から Commutext を取得し，状態復元部に反映する機能がある．以下では，一連の動作を実現する機能について説明する．

状態の取得

状態抽象化部から利用者の Commutext を取得する機能である．ソケットを用いたプロセス間通信で取得する．

状態の保存

状態抽象化部から利用者の Commutext を保存する機能である．ここで，取得した Commutext は，それが取得された周辺環境ではなく，利用者に伴う機器に保存されるべきである．これは，次の 2 つの理由による．第 1 に，保存された Commutext が意味を持つ環境による．通常，これは利用者周辺の環境である為，利用者に伴って Commutext が移動することは合理的である．第 2 に，移動前の環境で Commutext の保存コストを無くす為である．

状態の反映

移動後の環境で，状態管理部で管理する Commutext を利用する機器の状態復元部に反映する機能である．利用者が別環境に移動後，以前の環境での作業状態を継続させる為に Commutext を通知する．そして，次項で説明する状態復元部によって，この Commutext を元に作業状態を継続する様に機器の動作が設定される．こうして，異なる環境で作業状態の共有を可能となり，利用者の作業状態を継続出来る．
ソケットを用いたプロセス間通信で反映する．

4.5　状態復元部

状態復元部は，移動後の環境に利用者の作業状態を復元可能とする部分である．これによって，移動前の環境で行っていた作業の Commutext を元に状態を復元し，利用者の作業状態を移動前の環境から継続的なものとする．

状態の復元

状態の復元とは，移動前後の異なる環境下において利用者の作業を継続させる為に機器の動作状態を設定することである．

復元の柔軟性

移動後の環境で利用する機器に存在する移動へ適応する機構を用いて作業の継続性を実現する．この為，復元方法に柔軟性を持たせることが可能となる．

耐障害性と耐故障性

対障害性とは,．一方，耐故障性は機器自体に対する対応が必要である為，本機構のみでは実現することは出来ない．

従来の移動型コンピューティング環境では，“通常環境” への依存性が，媒介するネットワークや依存環境の常時安定性を前提としている．このため，それらに関する障害が発生した場合は，作業の持続が達成出来なくなる．

4.6　適応処理部

通信の切替とは，利用環境の変化に応じて適応的に通信の切替処理を行う機能である．切替処理の方法として，大きく分けて２つの場合が存在する．１つは，同一機器やそれと通信中の機器で発生した変更に対して，その機器内で対応することである．もう１つは，環境の変化に対して通信形態を変更することを目的として，通信中の機器を変更して複数の機器で対応することである．つまり，環境の変化に対して通信を切り替えることによって，それぞれの変化に応じた作業状態の適応性を実現可能となる．

通信の切替

利用者の移動に関わらず作業状態を継続させる為に，通信切替で必要とされることは３つある．第１は，適応的な通信切替である．つまり，別のホストや機器の中継を受けず，移動に対して自律的に通信の切替を行うことである．これは，異なる環境下で適応的に通信切替を行う為に必要である．第２は，動的な通信切替である．これは，時間的連続性を必要とする切替において必要となる．第３は，切替処理の透過性である．これにより，アプリケーションや利用者に対して，複雑な切替処理を隠蔽することが可能となる．図**??**に通信切替機構を示す．

適応的な通信切替

本稿で想定する環境では，ある作業状態を構成している機器の組み合わせがある環境から別の環境にまとめて移動する場合がある．この場合の機器の変更は，それぞれの機器同士で構成の変更やコネクションの張り直しを伴う．移動前後で異なる環境で適応的に利用者の作業状態を変化させることより，効果的な負荷分散やネットワークの効率的な利用を実現可能とする．

切替処理の透過性

利用者やアプリケーションが複雑な切替処理を意識せずに適応的に機器を利用可能とする為，切替処理そのものは本機構で隠蔽する．

本システムではクライアントからの通信要求をもとに利用する機器を選択し，それに対する通信コネクションを管理する．この為機器を利用する側からは，ある時点で利用している機器やそれに対する通信コネクションは隠蔽され，通信状態を持続させる機器の利用を透過的に行える．

第5章　実装

本章では，移動適応型分散通信機構のプロトタイプ実装として，Trancerシステムの実装について説明する．前章に基づいて，本システムは５つの部分から構成される．これは，状態抽象化部，状態管理部，状態復元部，適応処理部，及び，動作制御部である．また，これらの各部分を実装する上で，前節で述べた設計方針に留意する必要もある．設計方針は，環境に対して独立であること，作業状態を抽象化すること，適応動作の隠蔽と適応的な処理を両立すること，及び，安全であることである．

5.1 Trancerシステムの実装

本章では，移動適応型分散通信機構のプロトタイプ実装である Trancer システムの実装について述べる．以下，第 4 章で示した設計に基づき，それぞれの部分について説明する．次節で実装の概観を示し，続いてそれぞれの部分についての実装を説明する．

実装概観

以下の節で説明するプログラムの構成は，図 5.1 で示す通りである．実装部分は，ユーザレベルデーモンとカーネルモジュールに加えて制御プログラムから構成され，表 5.1 に示す環境で FreeBSD オペレーティングシステム上に C 言語を用いて行った．

表 5.1: 実装環境．

項目	説明
オペレーティングシステム	FreeBSD 4.4-stable (2001/10/01)
実装言語	C 言語 gcc version 2.95.3 20010315 (release) [FreeBSD]
ハードウェア	Intel Pentium III 600MHz, 800MHz, AMD Athlon 1GHz

また現在の実装では，対象とする状態を TCP/IP による通信状態のみに限定している．状態抽象化部と状態復元部は，ユーザレベルデーモン csmd とカーネルモジュール csm によって構成される．また，適応処理部はカーネルモジュール csm と TCP プロトコルスタックへの変更で実現される．そして，状態管理部と動作制御部はユーザプログラム csmc で実行される．csmd と csm は各機器に 1 つずつ必要であり，csmc は利用者に 1 つ必要である．

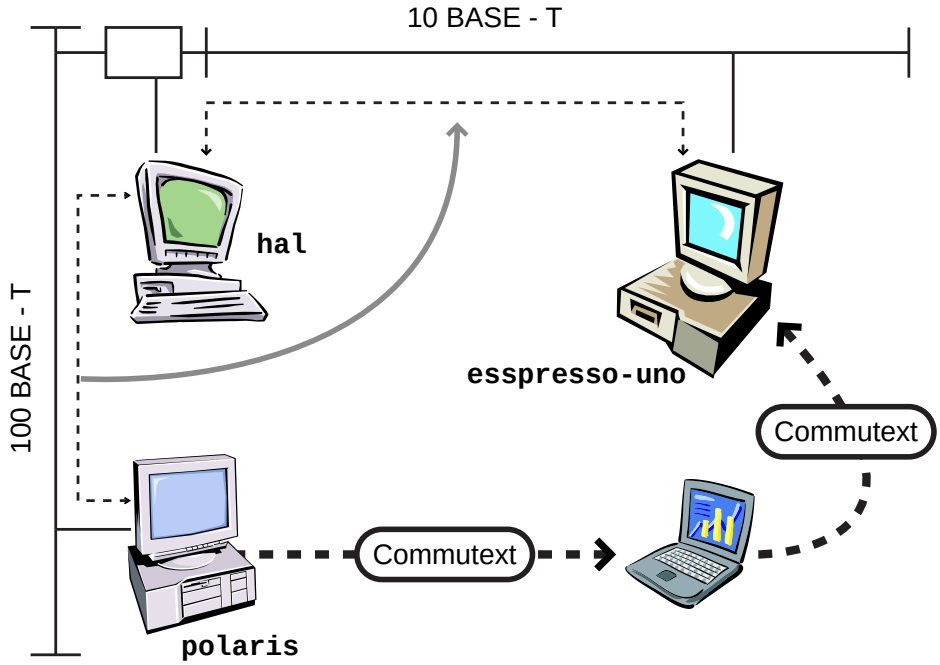

図 5.1: Trancer システムの実装概観図．

5.2 動作制御部の実装

動作制御部は，状態抽象化部と状態復元部の外部インタフェースに対してコマンドによる制御を行い，状態管理部に対して Commutext の取得・反映を行うことを可能とする．また，利用者に対して本システムを利用するためのインタフェースを提供する．

5.3 状態抽象化部の実装

状態抽象化部では，実行中の作業状態を形式化して記述することが必要である．Trancer システムでこの部分は，他の部分と連係して動作するユーザレベルデーモンとシステムレベルの情報を取得可能とするカーネルモジュールの組み合わせで構成される．この構成によって，通常はカーネル内に隠蔽されている通信状態などの情報を読み出し，ユーザプログラムで取り扱うことを可能とする．また，機器の外部から機器内部の状態を抽象化することが容易に実現出来る．

5.3.1 ユーザレベルデーモン csmd によるカーネル内部の状態取得

カーネル内に隠蔽された通信状態を取得して第 4.3.1 節で述べた Commutext として抽象化する為，ユーザレベルデーモン csmd を用いてカーネルモジュールと情報を交換する．この際，情報の交換用として，図 5.2 に示す commutext 構造体に格納された sys_commutext 構造体を用いる．ここで，sys_commutext 構造体は Commutext として必要なカーネル内の情報を格納し，commutext 構造体が Commutext を格納する．sys_commutext 構造体の各メンバについての詳細は，次節で説明する．

```
struct commutext {
        struct sys_commutext    syscomtxt;
};

struct sys_commutext {
        struct socket_pair      sp;
        struct tcpstate         tcpst;
};
```

図 5.2: commutext 構造体 - net/csm.h．

そして，ユーザレベルデーモン csmd は，デバイスファイル/dev/csm を通じて，ioctl() システムコール経由でカーネルモジュールと情報を交換する．図 5.3 は，ユーザレベルデーモン csmd 内で，システムレベルの Commutext を取得する部分の実装である．get_syscommutext() の引数は，第 1 引数の csm_fd が/dev/csm に対するファイルディスクリプタであり，第 2 引数は syscomtxt 構造体を指し示すポインタである．

```
get_syscommutext(csm_fd, &comtxt->syscomtxt)

#define get_syscommutext(IO_CSM_FD, IO_COMTXT) \
        ioctl(IO_CSM_FD, CSMIOCG_COMMUTXT, IO_COMTXT)
```

図 5.3: カーネルとのインタフェース - csmd.c.

次節では，この部分に対応するカーネルモジュール側の処理について述べる．

5.3.2 カーネル内部での通信状態の取得

システムレベルの Commutext を取得する為，csm をカーネルモジュールとして実装した．csm は，デバイスファイル/dev/csm を通じて，ユーザレベルデーモン csmd と情報の交換が可能である．前述の通り現在の実装では，対象とする状態を TCP/IP による通信状態のみに限定している．この為，図 5.2 で示す sys_commutext 構造体では，TCP を用いて通信しているコネクションの状態を格納する場合に限って考える．

TCP/IP を用いた通信では，それぞれの通信はコネクションとしてソケットペアで識別される．ソケットペアは，第 2.3 項でも述べた通り，通信中のノードで両エンドの IP アドレスとポート番号で示される．即ち，これは < 宛先 IP アドレス, 宛先ポート番号, 発信元 IP アドレス, 発信元ポート番号 > の組み合わせである．このソケットペアは，図 5.4 で示した socket_pair 構造体に格納する．この構造体のメンバ fsas と lsas は sockaddrs 型の共用体として定義され，通信プロトコルの多様性に対応する．

```
union sockaddr_set {
      struct  sockaddr        sa;     /* general version */
      struct  sockaddr_un     sun;    /* UNIX family */
      struct  sockaddr_in     sin;    /* INET family */
      struct  sockaddr_in6    sin6;   /* INET/IPv6 family */
};
#define sockaddrs       union   sockaddr_set

struct socket_pair {
      sockaddrs       fsas; /* foreign sockaddrs{} */
      sockaddrs       lsas; /* local sockaddrs{} */
};
```

図 5.4: socket_pair 構造体 - net/csm.h.

csm では，この socket_pair 構造体のメンバ fsas と lsas を用いて，カーネル内で TCP コントロールブロックを管理する TCP コネクションハッシュテーブルから，Commutext で指定されたコネクションの TCP コントロールブロックを特定する．そして，この TCP コントロールブロックから通信状態を取得可能となる．対応する TCP コントロールブロックは，図 5.5 に示す in_pcblookup_hash() 関数を用いて検索する．ここで検索した結

果 TCP のコネクションが特定された場合，inp はそのコネクションの TCP コントロールブロックを指し示すポインタとなる．

```
struct socket_pair  sp;
struct inpcb        *inp;

inp = in_pcblookup_hash(&tcbinfo,
                        sp.fsas.sin.sin_addr,
                        sp.fsas.sin.sin_port,
                        sp.lsas.sin.sin_addr,
                        sp.lsas.sin.sin_port,
                        0 /* no-wildcard */, NULL);
```

図 5.5: in_pcblookup_hash() 関数の引数 - /sys/net/csm.c．

この後，図 5.6 で示すマクロを用いて，TCP コントロールブロックから通信の状態を取得する．具体的には，マクロ TCPCB() を用いて inp が指し示す TCP コントロールブロックのメンバにアクセスし，マクロ TCPGST() を用いて図 5.7 に示す tcpstate 構造体に存在するメンバに対してその値を代入する．ここで用いる情報は TCP コントロールブロック内のシーケンス番号に関する変数の値である．これについては，第節で適応処理部の実装について述べる際に詳細を説明する．

```
#define TCPCB(x)  ((struct tcpcb *)inp->inp_ppcb)->x
#define TCPGST(x) \
        ((struct sys_commutext *)data)->tcpst.x = TCPCB(x)

        TCPGST(snd_una);
        TCPGST(snd_max);
        TCPGST(snd_nxt);
        TCPGST(snd_up);
        TCPGST(snd_wnd);
        TCPGST(iss);
        TCPGST(irs);
        TCPGST(rcv_nxt);
        TCPGST(rcv_adv);
        TCPGST(rcv_up);
        TCPGST(rcv_wnd);
```

図 5.6: TCP コントロールブロックからの状態取得 - /sys/net/csm.c．

以上で示した様に，csm ではカーネル内部の状態を示す変数を sys_commutext 構造体に格納する．こうして，csmd はカーネル内の状態を抽象化することが可能となる．

5.3.3 状態の形式化と記述

csmd では，図 5.3 で示した ioctl システムコールが成功すると，sys_commutext 構造体のメンバ tcpst には，図 5.6 で保存した TCP コントロールブロックの状態が格納され

```
struct tcpstate {
        tcp_seq snd_una; /* send unacknowledged */
        tcp_seq snd_max; /* highest sequence number sent;
                          * used to recognize retransmits */
        tcp_seq snd_nxt; /* send next */
        tcp_seq snd_up;  /* send urgent pointer */

        tcp_seq iss; /* initial send sequence number */
        tcp_seq irs; /* initial receive sequence number */

        tcp_seq rcv_nxt; /* receive next */
        tcp_seq rcv_adv; /* advertised window */
        u_long  rcv_wnd; /* receive window */
        tcp_seq rcv_up;  /* receive urgent pointer */

        u_long  snd_wnd; /* send window */
}
```

図 5.7: tcpstate 構造体 - net/csm.h．

る．この為 csmd は，commutext 構造体の内容を形式化して記述し，状態管理部に転送する．この際，Commutext を構造化して記述する言語として XML を用いる．図 5.8 に，XML によって Commutext を記述した例を示す

5.3.4 外部インタフェース

状態抽象化部の機能を外部から利用するためのインタフェースについて説明する．csmd はポート番号 8000 番でソケットを開き，外部からのコマンドに対して動作する．外部から Commutext を取得する場合のコマンドは，図**??**の通り定義される．ここで，GETCTXT がコマンド名であり，*PF* はプロトコルファミリを示し，*FHOST:FPORT* が宛先ホスト名とポート番号を示し，*LHOST:LPORT* で発信ホスト名とポート番号を示す．

```
GETCTXT  PF  FHOST:FPORT  LHOST:LPORT
```

例えば，宛先のホスト名が hal でポート番号が 5555 番と発信ホスト名が polaris でポート番号が 1024 番のソケットペアとして特定される通信で，状態の取得を要求するコマンドは以下の通りである．

```
GETCTXT  2  hal:5555  polaris:1024
```

このコマンドの出力結果は，図 5.8 で示す XML 形式によって取得可能である．

```
<?xml version=‘‘1.0’’>
<!DOCTYPE commutext SYSTEM ‘‘commutext.dtd’’>
<Commutext>

   <SysComutext>
      <SocketPair>
         <Foreign>
            <proto>NETINET</proto>
            <addr>hal.ht.sfc.keio.ac.jp</addr>
            <port>5555</port>
         </Foreign>
         <Local>
            <proto>NETINET</proto>
            <addr>polaris.ht.sfc.keio.ac.jp</addr>
            <port>1103</port>
         </Local>
      </SocketPair>
      <TCPCB>
         <snd_una>1539593644</snd_una>
         <snd_max>1539593644</snd_max>
         <snd_nxt>1539593644</snd_nxt>
         <snd_up>1539593644</snd_up>
         <iss>1539593521</iss>
         <irs>4243862162</irs>
         <rcv_nxt>4243862763</rcv_nxt>
         <rcv_adv>4243880139</rcv_adv>
         <rcv_up>4243862753</rcv_up>
         <rcv_wnd>17376</rcv_wnd>
         <snd_wnd>17376</snd_wnd>
      </TCPCB>
   </SysComutext>
</Commutext>
```

図 5.8: Commutext の XML による記述例.

5.4 状態管理部の実装

状態管理部では，状態抽象化部で形式化して記述された Commutext を管理する．Trancer システムでは，この部分を csmc というユーザプログラムで実装した．

Commutext は状態抽象化部から動作制御部を通じて取得され，本機構によって管理される．そして，再び動作制御部を通じて状態復元部に転送されて機器に反映される．

それぞれの Commutext は，図 5.9 で示す cmsession 構造体に格納される．ここでは 1 つの cmsession 構造体が単一の通信作業を表す．この構造体で MAX_COMTXT の値は，2 点以上の通信に対応していないため現状は 2 としている．また，全ての cmsession 構造体はダブルリンクドリストを形成する．

```
struct cmsession {
        struct socket_pair  sp[MAX_COMTXT];
        char                *comtxt_xml[MAX_COMTXT];
        cmsession           *next, *prev
};
```

図 5.9: cmsession 構造体とそのリスト - net/csm.h．

5.5 状態復元部の実装

状態復元部では，状態管理部から転送された Commutext に基づいて利用者の作業状態を復元する．Trancer システムでこの部分は，状態抽象化部と同一のプログラムとして実装している．この為，プログラムの構成は第 5.3 節で述べた状態抽象化部と同一である．また，本節の説明で用いる構造体も，第 5.3 節で説明したものと同一である．

5.5.1 ユーザレベルデーモン csmd によるカーネル内部への状態復元

図 5.3 は，システムレベルの Commutext を復元する為，csmd 内でデバイスファイル /dev/csm を通じた ioctl() システムコールを発行する部分を示している．set_syscommutext() の引数は，第 1 引数の csm_fd がデバイスファイル/dev/csm に対するファイルディスクリプタであり，第 2 引数は syscomtxt 構造体を指し示すポインタである．状態復元部では，状態管理部から取得した Commutext の要素を commutext 構造体の対応するメンバに代入した後，このシステムコールによって csm を呼び出し，カーネル内部に対する状態の復元を可能とする．

```
set_syscommutext(csm_fd, &comtxt->syscomtxt)

#define set_syscommutext(IO_CSM_FD, IO_COMTXT) \
        ioctl(IO_CSM_FD, CSMIOCS_COMMUTXT, IO_COMTXT)
```

図 5.10: カーネルとのインタフェース - csmd.c．

次節では，この部分に対応するカーネルモジュール側の処理について述べる．

5.5.2 カーネル内部での通信状態の復元

csm は，csmd からシステムレベルの Commutext を復元するという要求を受け，カーネル内部の通信状態が渡された sys_commutext 構造体に基づいて復元される．この際，復元される機器では，それまでとは異なる発信元アドレスとポート番号を用いて通信が継続される．この為，新たな TCP コントロールブロックを用いて，発信元アドレスとポート番号の変更に対して対応する必要がある．これに関する実装の詳細は，第 5.6 章で適応処理部の実装を述べる際に説明する．

そして，図 5.6 で示したマクロ TCPGST() とは逆方向のマクロ TCPSST() によって，図 5.7 の tcpstate 構造体から TCP コントロールブロックのメンバに状態が反映される．この結果，適応処理部の動作が終了した後から，このマクロを用いて移動前の通信状態を復元することが可能となる．

```
#define TCPCB(x)  ((struct tcpcb *)inp->inp_ppcb)->x
#define TCPSST(x) \
        TCPCB(x) = ((struct sys_commutext *)data)->tcpst.x

        TCPSST(snd_una);
        TCPSST(snd_max);
        TCPSST(snd_nxt);
        TCPSST(snd_up);
        TCPSST(snd_wnd);
        TCPSST(iss);
        TCPSST(irs);
        TCPSST(rcv_nxt);
        TCPSST(rcv_adv);
        TCPSST(rcv_up);
        TCPSST(rcv_wnd);
```

図 5.11: TCP コントロールブロックへの状態反映 - /sys/net/csm.c．

以上で示した様に，csm では sys_commutext 構造体からカーネル内部の状態を示す変数を格納する．こうして，反映された情報を元に通信状態が復元される．

5.5.3 外部インタフェース

状態復元部の機能を外部から利用するためのインタフェースについて説明する．

第項の場合と同様に，csmd はポート番号 8000 番でソケットを開き，外部からのコマンドに対して動作する．この時，外部から Commutext を復元する場合のコマンドは，以下の通り定義される．ここで，SETCTXT がコマンド名であり，*PF* はプロトコルファミリを示す．そして，次の *FHOST:FPORT* と *LHOST:LPORT* が新たな宛先ホスト名とポート番号に発信ホスト名とポート番号を示し，続く *RFHOST:RFPORT* と *LHOST:LPORT* が以前の宛先ホスト名とポート番号に発信ホスト名とポート番号を示す．最後の COMTXT が XML で記述された Commutext の実体である．

SETCTXT *PF FHOST:FPORT LHOST:LPORT PF RFHOST:RFPORT RLHOST:RLPORT COMMUTEXT*

例えば，宛先のホスト名が hal でポート番号が 5555 番に対して，発信ホスト名が polaris でポート番号を 1024 番で行っていた通信を発信ホスト名が espresso-uno でポート番号を 1026 番で行う場合，状態の復元を要求するコマンドは以下の通りである．

SETCTXT 2 hal:5555 espresso-uno:1026 hal:5555 polaris:1024

5.6 適応処理部の実装

適応処理部では，実際の移動に対応した適応動作を行う．

TCP のコネクション移動による継続処理

前述の通り現在の実装では，対象とする状態を TCP/IP による通信状態のみに限定している．この為，TCP で通信しているコネクションで，移動に対してトランスポート層での状態情報を利用して継続させる場合を考える．

適応処理部は，状態復元部での設定によって動作する．これは，csm で行われる図 5.12 で示す部分である．まず，csmd から指定されたアドレスのうち，相手の機器と切り替え前に通信していた機器のアドレスとポート番号を保存する．ここで保存する部分は，図 5.13 でプロトコルコントロールブロックに対して新たに追加した項目である．

```
struct socket_pair      rsp;

/* copy addr from data to sockp */
bcopy((char *)&((struct sys_commutext *)data)->rsp.fsas,
      (char *)&rsp.fsas, sizeof(rsp.fsas));
bcopy((char *)&((struct sys_commutext *)data)->rsp.lsas,
      (char *)&rsp.lsas, sizeof(rsp.lsas));

TCPCB(t_inpcb->inp_rladdr) = rsp.lsas.sin.sin_addr;
TCPCB(t_inpcb->inp_rlport) = rsp.lsas.sin.sin_port;
TCPCB(t_flags) |= TF_REQ_RDRT;
```

図 5.12: 移動前のアドレスとポート番号を保存 - /sys/net/csm.c．

また，5.12 で設定している TF_REQ_RDRT フラグは，tcp_output() 関数が適応処理を行うために用いる．これらの追加したフラグを 5.14 に示す．

tcp_output 関数では，5.12 で設定された TF_REQ_RDRT フラグによって REDIRECT パケットを送信する．通信相手の機器では，このパケットを受け取った場合に移動前

```
struct inpcb {
...
        u_short inp_rfport;             /* foreign port */
        u_short inp_rlport;             /* local port */

        union {
                /* local host table entry */
                struct  in_addr_4in6 inp46_local;
                struct  in6_addr inp6_local;
        } inp_rdependladdr;
...
}
#define inp_rfaddr      inp_rdependfaddr.inp46_foreign.ia46_addr4
#define inp_rladdr      inp_rdependladdr.inp46_local.ia46_addr4
```

図 5.13: プロトコルコントロールブロックへの追加項目 - /sys/netinet/in_pcb.h．

```
#define TF_REQ_RDRT_PM  0x0040000 /* have/will request RDRT_PM */
#define TF_REQ_RDRT     0x0080000 /* RDRT_PM request */
#define TF_RCVD_RDRT_PM 0x0100000 /* other side has requested RDRT_PM */
#define TF_RCVD_RDRT    0x0200000 /* a RDRT request was received */
```

図 5.14: 追加したフラグ - /sys/netinet/tcp_var.h．

の別機器で行われていたコネクションのデータを受け取ったものであると認識することが出来る．このパケットは，SYN フラグと図 5.15 で示すオプションで構成される．

この為，新たな TCP のオプションを定義した．これを図 5.17 に示す．

```
if (len && (tp->t_flags & (TF_REQ_RDRT_PM|TF_RCVD_RDRT_PM)) &&
    (tp->t_flags & TF_REQ_RDRT)) {
        /* REDIRECT/SYN packet */
        flags |= TH_SYN;
        flags &= ~TH_ACK;

        /* clear flag */
        tp->t_flags &= ~TF_REQ_RDRT;

        len = 0;

        /* redirect option of packet header */
        opt[optlen++] = TCPOPT_NOP;
        opt[optlen++] = TCPOPT_RDRT;
        opt[optlen++] = TCPOLEN_RDRT;
        opt[optlen++] = 0;

        /* Old Address */
        *(u_int32_t *)&opt[optlen] = tp->t_inpcb->inp_rladdr.s_addr;
        optlen += 4;
        *(u_int32_t *)&opt[optlen] = (u_int32_t)tp->t_inpcb->inp_rlport;
        optlen += 4;
        /* New Address */
        *(u_int32_t *)&opt[optlen] = tp->t_inpcb->inp_laddr.s_addr;
        optlen += 4;
        *(u_int32_t *)&opt[optlen] = (u_int32_t)tp->t_inpcb->inp_lport;
        optlen += 4;
}
```

図 5.15: REDIRECT パケットのオプション - /sys/netinet/tcp_output.c:tcp_output()．

```
#define TCPOPT_RDRT             16              /* Experimental */
#define    TCPOLEN_RDRT                 19
```

図 5.16: 追加した TCP オプション - /sys/netinet/tcp.h．

また，このオプションを用いた場合，図 5.17 に示すフォーマットで TCP ヘッダに新たなオプションが追加される．

適応処理部の通信相手が REDIRECT パケットを受信すると，図 5.17 に示したオプション部分に格納されたアドレスで自らの TCP コントロールブロックを更新して，新たなアドレスからの通信を可能とする．

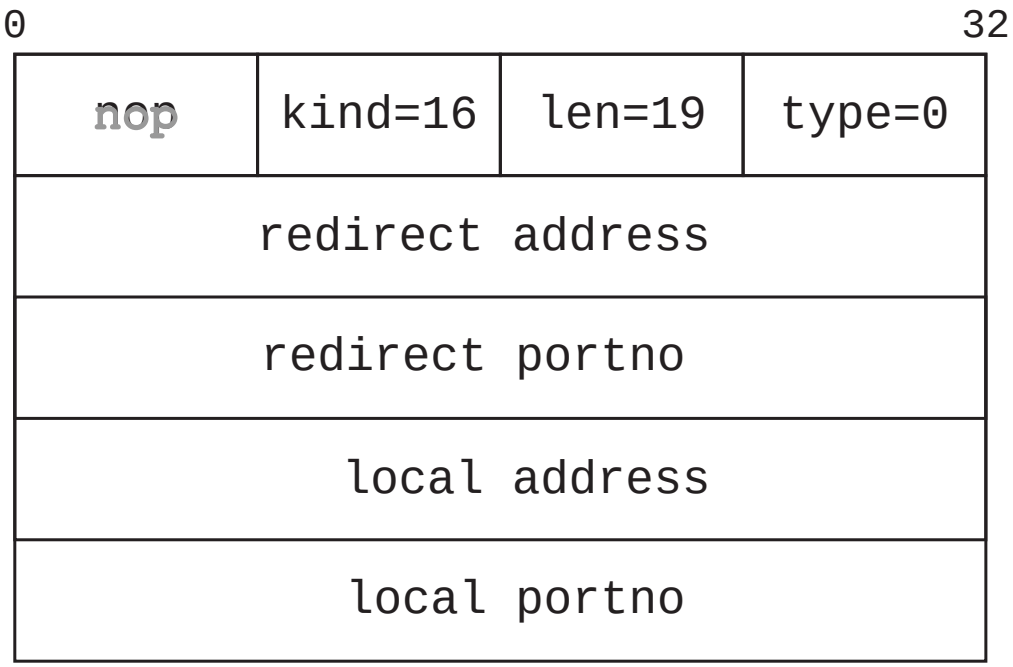

図 5.17: REDIRECT パケットの TCP ヘッダ追加部分.

第6章　評価

本章では，移動適応型分散通信モデルとそれを実現する機構のプロトタイプ実装である Trancer システムについて，評価を行う．まず，本システムの目的と設計について考察し、本システムの実際の動作を検証する．

6.1 設計の考察

第 1.2 節で示した本システムの目的に対して設計を考察する．移動適応型分散通信モデルの目的は，ユビキタスコンピューティング環境での移動する際に，状態継続性，環境独立性，状態抽象性の 3 点を実現することであった．これらを順に分析する．

状態継続性：環境変化に左右されない作業状態の継続

種類が異なる機器や共用する機器を利用する場合であっても，利用者の作業状態を移動に対して継続可能とすることが目的であった．本論文では，異なる機器間で通信の移動を可能とした．また，利用者の移動と利用する機器の変化に対して，通信作業の状態を継続させることで利用者の作業状態を継続可能であることを示せた．よって，状態継続性は実現出来たと言える．

環境独立性：特定の環境に対する依存性の排除

移動時の作業が，特定の環境における作業や環境それ自体に依存すること無く独立して動作可能とすることが目的であった．本論文では，利用者が移動後に用いる機器や通信は，移動前の機器の動作に依存しないという点で，環境独立性を実現している．つまり，通信移動後のホストは移動前のホストから影響を受けない．よって，環境独立性は実現出来たと言える．

状態抽象性：状態管理による作業状態の抽象化

ある作業状態を抽象化することで，それによって別環境で作業の継続を可能とするが目的であった．本論文では，状態抽象化部によって通信状態を形式化して記述し，別環境でその通信状態を再構成することを実現出来た．だが，実装では単一のプロトコルのみに限定しており，様々な通信環境に適応可能とすることは今後の課題として挙げるべき点である．

6.2 動作検証

第 5 章で述べた Trancer システムのプロトタイプ実装を用いて，移動適応型分散通信機構の動作を検証した．検証に用いた環境は，図 5.1 で示した構成と同一である．

まず，このシステムの検証時に用いる単純なサーバプログラムを作成した．このプログラムは，クライアントプログラムからの入力に対して，１回の入力毎に１ずつ増加させた値を返すものである．このサーバプログラムは同時に複数のクライアントに対して通信可能であり，返す値はそれぞれのクライアント毎に独立である．このサーバプログラムをホスト “hal” で動作させ，これに対するクライアントプログラムがホスト “polaris” と “espresso-uno” で動作する．

ここで，図 6.1 は，ホスト “hal” と “polaris” の通信がホスト “hal” と “espresso-uno” の通信に移動する場合において，それぞれのホスト上で動作するクライアントプログラムの表示例を示している．この検証では，移動後の異なるホスト上でサーバの返す値に移動前のホストから一貫性を確認出来た．この為，Commutext として抽象化した状態の情報を元に機器が異なる場合にも作業の状態を継続可能であることが示せた．

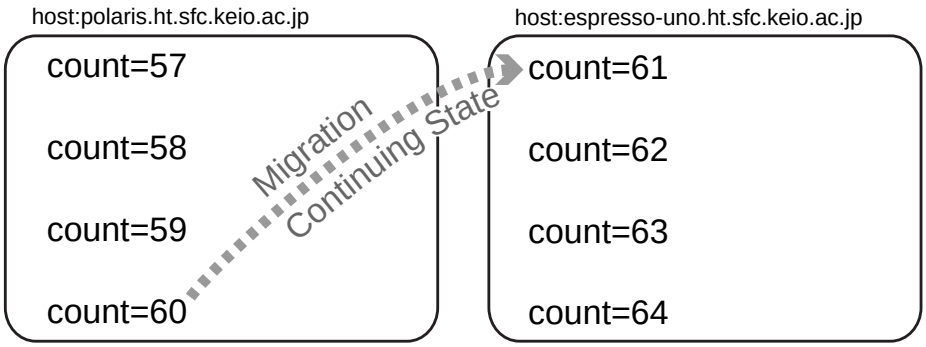

図 6.1: Trancer 動作検証時の画面イメージ．

図 6.2 では，ここで生じた通信の適応処理を図示する．

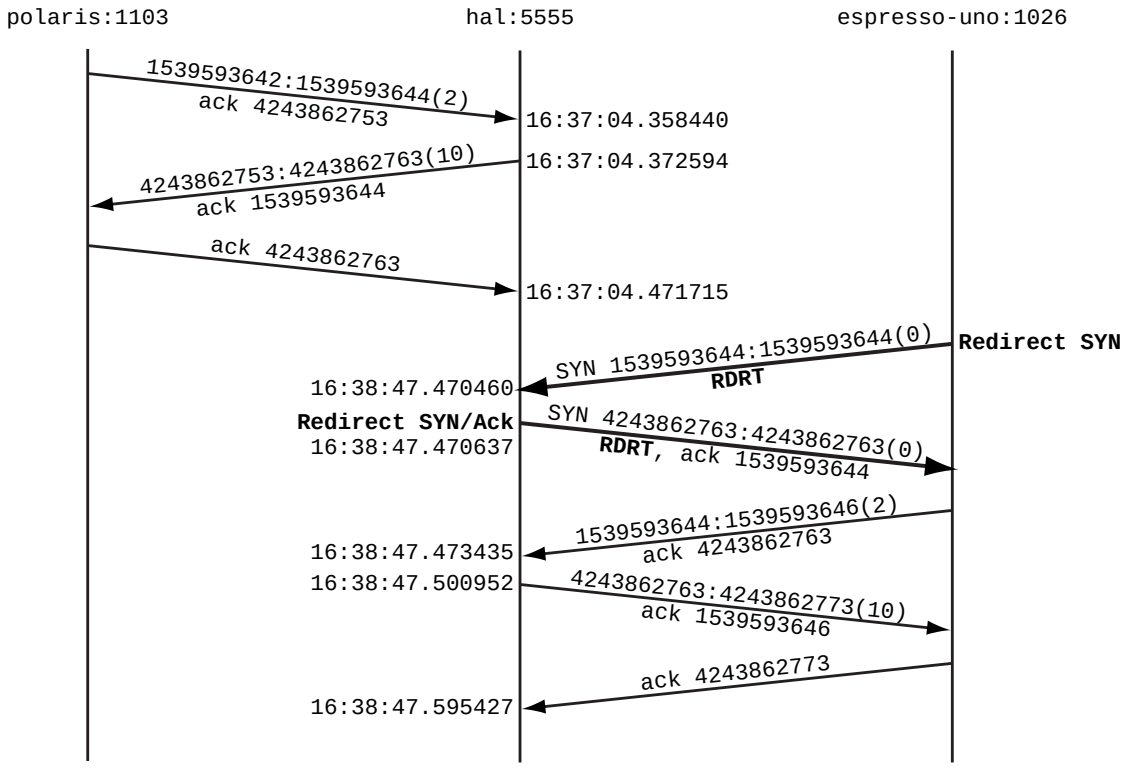

図 6.2: Trancer 動作検証時，クライアント側の移動．

6.3 議論

本節では，本論文で実現したシステムと既存の関連研究とを比較する．

6.3.1 関連研究との比較

前節までの本システムのプロトタイプ実装と第 3.3 節で述べた関連研究との比較を表 6.1 にまとめる．それぞれの項目を満たす場合は “yes” で示し，満たさない場合には “no” で示した．

表 6.1: 関連研究による実現性の比較．

	M-TCP	SRA	VNC	Trancer
環境適応性	no	yes	no	yes
運用柔軟性	no	no	no	yes
環境独立性	yes	no	no	yes
状態継続性	yes	no	no	yes

注: 便宜上，Stream Redirection Architecture を SRA として示した．

以下，要求される特性とそれぞれのシステムについて比較した結果を述べる．

環境適応性

環境適応性の有無は，環境の変化に対して適応的に動作するかどうかで決定する．M-TCP は，サーバ側の負荷分散に視点を置くものであり，利用者の環境変化に対しては適応的に動作しているとは言えない．また，VNC も単純なユーザインタフェースの転送を行っているに過ぎず，適応的に動作しているとは言えない．だが，SRA では，シンクプロキシを通じた環境後との適応的な継続を実現している．Trancer では，作業の状態管理による適応性を実現する．

運用柔軟性

運用柔軟性の有無は，特定の環境から非依存な適応性と環境変化の不確定性に対する柔軟性を実現可能であるかどうかで決定する．M-TCP は，全て TCP のレベルで実装している為，TCP 以外の通信プロトコルを用いて柔軟に対応することが出来ない．また，SRA は，シンクプロキシという機構を常に必要とする．そして VNC は，常に VNC の操作プログラムを必要とする．この点から，これら 2 つの機構は運用柔軟性を持つとは言えない．Trancer では，作業の状態管理による運用の柔軟性を実現する．

環境独立性

環境独立性の有無は，移動前の環境に対して依存性を持たずに移動後の環境を実現可能かどうかで決定する．

M-TCP は，移動前後の環境で状態を交換する必要がある為，状態を転送する際には互いに通信可能でなければならない．また，SRA では，それ自身で構築する環境に依存性を有する．Trancer では，通信作業を行うプロセスと管理するプロセスを分離し，移動後には移動前に利用していた機器は解放される．

状態継続性

状態継続性の有無は，移動など環境が変化する前後において通信作業の状態を共有することが可能かどうかで決定する．SRA や VNC では，状態が実行作業の中に隠蔽されている為，作業の状態のみを対象として取り扱うことは出来ない．M-TCP では共有可能な状態を TCP の通信状態に限っている．Trancer では，抽象化した利用者の通信状態を管理することによって，移動前後で状態を共有可能としている．

6.3.2　持続性と継続性

本論文では，移動型コンピューティング環境を持続的な手法と捉え，移動適応型コンピューティング環境を継続的な手法として説明した．利用環境が様々に変化するユビキタスコンピューティング環境では，持続的な手法では限界が発生する．この為，従来の移動型コンピューティング環境による手法では対応出来ない．

移動に対する継続性を実現する為には，利用者の作業状態を抽象化し，状態管理によって移動前後の環境で状態を共有する必要がある．この為，本機構では状態管理によって移動に対する通信の継続性を実現した．

6.3.3　適応性と柔軟性の実現

利用環境が様々に変化するユビキタスコンピューティング環境において利用者の作業を継続させる為には，作業状態を環境に適応的なものとする必要がある．また，利用環境の様々な状況の変化を前提として，適応方法の選択に柔軟性が必要である．

本機構では，状態管理機構と適応機構を分離し，適応機構が場面に応じた対応を行うことを可能とする．これによって適応方式の柔軟性が実現出来る．

第7章　結論

本論文では，機能高度分散型コンピューティング環境において利用者の移動に対して通信作業の適応的な継続を実現する為の移動適応型分散通信モデルを提案した．また，その通信モデルを実現する機構として Trancer システムを設計し実装した．

本論文では，始めに従来の移動型コンピューティング環境における移動性について分析した．そして，本論文で想定する機能高度分散型コンピューティング環境において移動について分析した．その結果，従来の環境で用いられている移動に対応する為の手法を用いた場合には，問題が発生することを明らかにした．

この為，本論文では移動適応型コンピューティング環境における通信作業に注目し，移動適応型分散通信モデルを提案した．このモデルにより，利用者の移動に対して通信作業をそれぞれの場面に適応させて，作業を継続することが可能となる．

移動適応型分散通信モデルの特徴は，利用者の作業状態を抽象化して記述することにより，状態管理によって通信作業を移動に対応させることである．

本論文では，移動適応型分散通信モデルを実現する Trancer システムの設計及びプロトタイプ実装を行った．さらに，このシステムに対する評価を行い，本システムによって，状態継続性，環境独立性，状態抽象性の 3 点が実現されることがわかった．これは，ユビキタスコンピューティング環境において利用者が移動する際に，必要とされる要件を満たしている．

本論文を踏まえて，今後の課題を以下にまとめる．

複数の通信プロトコルへの対応

現在のプロトタイプ実装では，対応可能な通信プロトコルを TCP に限っている．しかし，移動適応型コンピューティング環境を実現する為には，他にも様々な通信プロトコルに対応する必要がある．

両エンドが同時に移動する場合の対応

現在のプロトタイプ実装では，両エンドが同時に移動する場合に対応出来ていない．これは，ピアー・ツー・ピアな通信において移動適応性を実現する場合に問題となる．

通信切り替えの実時間性

本論文で例として挙げたビデオストリーミング観賞作業の継続を実現する場合は，移動に対する適応処理や通信の切り替え処理にも実時間性が要求される．そのため，本システムの処理速度向上及び実時間性を実現する為の設計の見直しが必要である．

謝辞

本研究を進めるにあたって貴重な御指導を賜りました，慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します．また，的確かつ重要な御助言を頂きました，慶應義塾大学政策・メディア研究科助教授西尾信彦博士の御指導に感謝します．

慶應義塾大学徳田・村井・楠本・中村・南研究室の皆様には，折りに触れ貴重な示唆や御助言，御指導を頂きました．特に，徳田研究室の先生方や諸先輩方，及び，Mobile Computing next generation(MCng) 研究グループの皆様には，深く感謝の意を表します．岩本健嗣氏，永田智大氏，村瀬正名氏，梅染充男氏，若山史郎氏には，本論文の執筆に当たって絶えざる御指導や励ましを頂きました．また，青木俊氏，大嶋ひとみ氏，鈴木源太氏，田丸修平氏，守分滋氏の協力に感謝します．

最後に，研究会の活動を通じて数多くの経験や刺激を受ける機会を頂きましたことに心からの謝意を表し，謝辞と致します．

2002 年 1 月 31 日

権藤 俊一

参考文献

[1] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, Vol. 36, No. 7, pp. 74–84, July 1993.

[2] IEEE 802.11 Wireless LAN Working Group. http://grouper.ieee.org/groups/802/11/.

[3] Bluetooth SIG, Inc. http://www.bluetooth.org/.

[4] Xybernaut K.K. Mobile Assistant. http://www.xybernaut.co.jp/.

[5] IBM Research and Citizen. WatchPad. http://www.trl.ibm.com/projects/ngm/.

[6] Donald A. Norman. INFORMATION APPLIANCE: *The Invisible Computer*, chapter 3. MIT Press, 1998.

[7] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems Principles and Paradigms*. Prentice Hall, 2002.

[8] Peer-to-Peer Working Group. http://www.p2pwg.org/.

[9] 松下温, 中川正雄（編）. ワイヤレス LAN アーキテクチャ. 共立出版, 1996.

[10] C. Perkins. IP Mobility Support. RFC 2002 (Updated by 2290, 2794) PROPOSED STANDARD, IETF, 1996.

[11] Ed. P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136 (Updates 1035) (Updated by 3007) PROPOSED STANDARD, IETF, 1997.

[12] NTT DoCoMo, Inc. i モード対応 Java. http://www.nttdocomo.co.jp/p_s/imode/java/.

[13] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, Vol. 2, pp. 33–38, 1998.

[14] Donald A. Norman. *The Invisible Computer*. MIT Press, 1998.

[15] 西尾信彦, 徳田英幸. EISS: 環境情報サーバスイートを用いたシステムの状況適応. 情報処理学会 第 9 回 コンピュータシステムシンポジウム論文集, 11 1997.

[16] 永田智大, 西尾信彦, 徳田英幸. 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 第 11 回 コンピュータシステム・シンポジウム論文集, 1999.

[17] 村瀬正名, 若山史郎, 権藤俊一, 永田智大, 西尾信彦, 徳田英幸. 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. 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.84, pp. 31–38, 沖縄, 5 2000.

[18] W. リチャードスティーヴンス/篠田陽一. UNIX ネットワークプログラミング Vol.1 ネットワーク API:ソケットと TLI, 第 1.3 章. ピアソン・エデュケーション, 1998.

[19] C. Perkins and D.B. Johnson. Route Optimization in Mobile IP. draft-ietf-mobileip-optim-11.txt WORK IN PROGRESS, IETF, 2001.

[20] C. Perkins. IP Encapsulation within IP. RFC 2003 PROPOSED STANDARD, IETF, 1996.

[21] J. Postel. Internet Protocol. RFC 791 Standard, IETF, 1981.

[22] S. Deering and R. Hinden. Internet Protocol Version 6 (IPv6) Specification. RFC 2460 Draft Standard, IETF, 1998.

[23] Daichi Funato, Kinuko Yasuda, and Hideyuki Tokuda. TCP-R: TCP Mobility Support for Continuous Operation. In *IEEE International Conference on Network Protocols 97*, Atlanta, October 1997.

[24] Alex C. Snoeren, Hari Balakrishnan, and M. Frans Kaashoek. Reconsidering Internet Mobility. In *8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau/Oberbayern, Germany, May 2001.

[25] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *ACM Transactions on Computer Systems*, Febuary 1992.

[26] Tadashi Okoshi, Masahiro Mochizuki, Yoshito Tobe, and Hideyuki Tokuda. MobileSocket: Session Layer Continuous Operation Support for Java Applications. 情報処理学会 論文誌, Vol. 40, No. 6, pp. 10–14, 2000.

[27] Sun Microsystems Inc. The Java Language Specification, 1996.

[28] 中澤仁, 望月祐洋, 徳田英幸. ホスト透過型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, pp. 2573–2584, 1999.

[29] The Open Group. X Window System. http://www.x.org/.

[30] M. Crispin. Internet Message Access Protocol - Version 4rev1. RFC 2060 (Obsoletes RFC1730) PROPOSED STANDARD, IETF, 1996.

[31] SCREEN - the terminal multiplexer. http://www.math.fu-berlin.de/ guckes/screen/.

[32] GNU Emacs: EMACSCLIENT. http://www.gnu.org/software/emacs/.

[33] Kiran Srinivasan. MTCP: Transport Layer Support for Highly Available Network Services. Master's thesis, Rutgers, The State University of New Jersey. Department of Computer Science, 2001.

[34] Jorge Rafael Nogueras. A Stream Redirection Architecture for Pervasive Computing Environments. Master's thesis, Computer Science and Engineering at the Massachusetts Institute of Technology, 2001.

[35] W3C. Extensible Markup Language (XML) 1.0 (Second Edition), 6 2000. http://www.w3.org/TR/REC-xml.

[36] 権藤俊一, 永田智大, 岩本健嗣, 西尾信彦, 徳田英幸. ウェアラブルネットワークにおける移動適応型分散通信機構. 情報処理学会マルチメディア, 分散, 協調とモバイル (DICOMO2001) シンポジウム, 鳴戸, 7 2001.