アプリケーションの状態移送による移動時の作業継続に関する研究

卒業論文 1998 年度 (平成10年度)

慶應義塾大学 環境情報学部

三輪修嗣

指導教員
村井純
徳田英幸
楠本博之
中村修

概要

現在の計算機環境では、ネットワークの普及により移動して利用する機会が増加している。そのため、移動して計算機上で作業する機会が増加した。本研究では、移動を計算機を持たずにネットワークに接続された計算機間での移動とする。従来のアプリケーションは、移動しての作業を継続することを考慮していない。特にファイルなどの外部的な要素がある点と、移動前後で作業している状態が変化する点が問題である。このような問題を解決するためには、移動前と移動後で画像を移送したり、状態を移送するといった方法がある。

本研究では、このような問題を解決し移動計算機環境を支援するために、移動前のアプリケーションの状態を移動後に移送する機構を提案し、実装及び評価を行なった。本研究により、移動計算機環境を考慮したアプリケーションを、容易に開発できる環境を提供する。そのために、作業の継続が可能となるモデルを移動前のアプリケーションの状態を移動後に移送するという方法で設計し、その枠組をライブラリ・テンプレートとして設計・実装した。

本研究で設計した機構を利用したサンプルアプリケーションをLinux2.0.35上に実装し、本モデルの性能評価をした。その結果、サンプルアプリケーションが用いたライブラリ・テンプレートにより、本モデルが汎用的に有効であることを検証できた。また、少ない転送量かつ短い動作時間で作業の継続ができたことにより、本モデルが低速なネットワークでの利用に有効であることを検証できた。

Abstract

In the computing environment, the chances which are moved and used by spread the network are incresing. Therefore the chances which are moved and worked on the computer are also incresing. Primary problems are the external elements which are file and socket and changed status which is working between before movement and after movement. To solve such problems, there are approaches which transmit graphics information or status before movement to after movement.

In this research, architecture which transmit status before movement to after movement is proposed to solve primary problems and to support the move computing environment. This research provides the environment which easily develops apprication used architecutre considered the move computing environment. Therefore using the approach which transmit the status, the model which is able to continue to work was designed and the framework as the library was designed and implemented.

The proposed architecture is implemented on Linux 2.0.35 architecture and its performanced is evaluated. By the library used by the sample application, this model is valid in the wide use. Also to be contined to work in the small mount of transmission and short movement time, it is verifed that this model is valid in the low speed network.

目次

図目次

表 目 次

第1章.　序論

本章では、本研究における背景と目的について説明する。また、本論文の構成を示す。

第1.1 節.　背景・目的

自宅や学校、会社などネットワークにつながった様々な場所に計算機が存在し、その計算機を利用して作業を行なう機会が増加している。学校や会社といった場所では、利用者は授業や仕事といった理由で移動しなければならない状況が多々発生する。そのため、常に同一の計算機で作業を行なうのではなく、移動して様々な計算機上で作業を行なうことになる。つまり、異なる計算機間を移動しながら作業をする必要性が存在する。

様々な場所に計算機がある環境では、同一計算機上で継続して作業を行なえない状況が発生する。その際、作業を継続するためにはユーザが他の計算機へ移動する必要がある。

現在のシステムでは、移動する際に移動前の作業を継続するためには次の工程が必要となる。

- 移動する前に作業を中断する
- 移動した後でシステムを立ちあげる
- 移動する前の状態を再現する

上記のように現在のシステムでは移動して作業を継続するためには多数の操作を必要とする。また、ユーザにより操作の誤りや移動前の状態の忘却が発生する。そのため、移動前の状態を再現できない、あるいは、無駄な動作のために多くの時間が浪費される。

本論文では、計算機間を移動した際、容易に作業が継続できるシステムの設計・実装を行なう。本研究により、ある計算機上で作業していたユーザが、移動後の計算機上で移動前の作業を容易に継続できるようになる。

第1.2 節.　本論文の構成

第 2 章では、本論文で論じる移動計算機環境について述べ、移動環境の現状について述べる。第 3 章で移動計算機環境について行なわれている関連研究について述べる。第 4 章では、第 2 章での議論を基に現在の移動計算機環境での問題点を整理する。また、問題を解決するアプローチを挙げ、考察する。第 5 章では、第 4 章で挙げたアプローチに基づいたモデルの設計について述

べ、そのモデルを用いたライブラリの設計について述べる。第 6 章では、第 5 章の設計に基づく実装について解説する。第 7 章では、第 5 章及び第 6 章で行なったツールキットを利用し、サンプルアプリケーションの設計・実装について述べる。第 8 章では、この実装の評価について述べる。最後に、第 9 章では本論文のまとめ及び今後の課題について述べる。

第 2 章.　移動計算機環境

本章では、本研究において論じる移動計算機環境について述べ、現状の移動計算機環境を支援するシステムについて述べる。

第 2.1 節.　移動計算機環境の定義

計算機同士を接続したインターネットや LAN などのネットワーク環境が普及した。そのため、ネットワークでの利用を前提としたアプリケーションが一般的に使用されるようになった。具体的にはメールソフト、WEB ブラウザソフトといったアプリケーションである。

また、計算機の軽量化が進み、携帯型計算機が普及した。それに伴い、自宅、会社、学校といった場所を携帯型計算機を所持したまま容易に移動することが可能となった。

そこで、本論文では、アプリケーションを利用することを作業と定義する。そして、アプリケーションを利用する人をユーザとする。移動計算機環境を「ユーザが移動し、ネットワークに接続されている計算機を用いて作業が可能となる環境」と定義する。次節では、本論文の焦点とする移動計算機環境について述べる。

第 2.2 節.　移動計算機環境

移動計算機環境を「ユーザのみが移動するモデル」と「ユーザが計算機を持って移動するモデル」の 2 つのモデルに分類し、これらについて議論する。

第 2.2.1 節　移動計算機環境のモデル

- モデル１　：ユーザのみが移動するモデル

  このモデルでは、ある計算機上で作業をしているユーザが移動し、他の計算機で作業を再開する。この概念図を図 2.1 に示す。計算機を持たずに移動するため移動前の計算機にいる時には作業が可能であり、移動している間は作業ができない。そして、他の計算機の所に移動した時に再び作業が可能となる。

計算機を持たずに移動するため、移動先に計算機を接続できるかどうか考慮する必要がない。問題点として、大きな環境の変化が挙げられる。別の計算機に移動するために、移動前と環境が同じとは限らない。そのため、作業の継続ができない恐れがある。

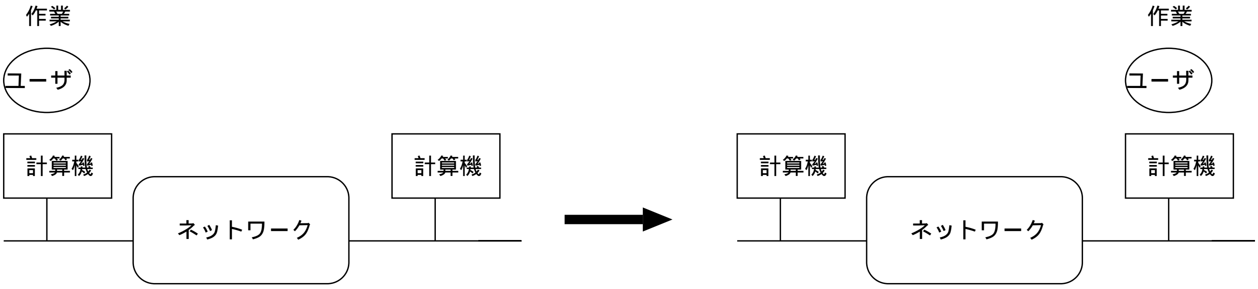


図 2.1: モデル 1

- モデル 2 ： ユーザが計算機を持って移動するモデル

  このモデルでは、ある計算機上で作業をしているユーザがその計算機を持って移動する。そして、移動先でネットワークに接続して作業を再開するモデルである。

  計算機を持って移動するため、移動先でも同じ計算機を用いる。そのため、ユーザが作業する環境は全く同一となる。問題点として、作業できる場所の制限が挙げられる。計算機を持っての移動を考慮すると、携帯型計算機上での作業を前提としなければならない。

  さらに、このモデルは使用するアプリケーションによって、2 つのモデルに分けることができる。

  - モデル 2 - 1 ： 完結したアプリケーションを持った計算機が移動するモデル

    完結したアプリケーションとは次のものを指す。

    - サーバ・クライアントモデルでないサーバを必要としないアプリケーション
    - 他の計算機にあるサーバを必要とせず、アプリケーション単体で作業が可能なサーバ・クライアントモデルのアプリケーション

    このモデルでは、ユーザは完結したアプリケーションが動作する計算機と共に移動し、移動後に作業を再開するモデルである。この概念図を図 2.2 に示す。アプリケーションはそれ単体で完結しているため移動している間でも作業が可能となる。

  - モデル 2 - 2 ： クライアントを持った計算機が移動するモデル

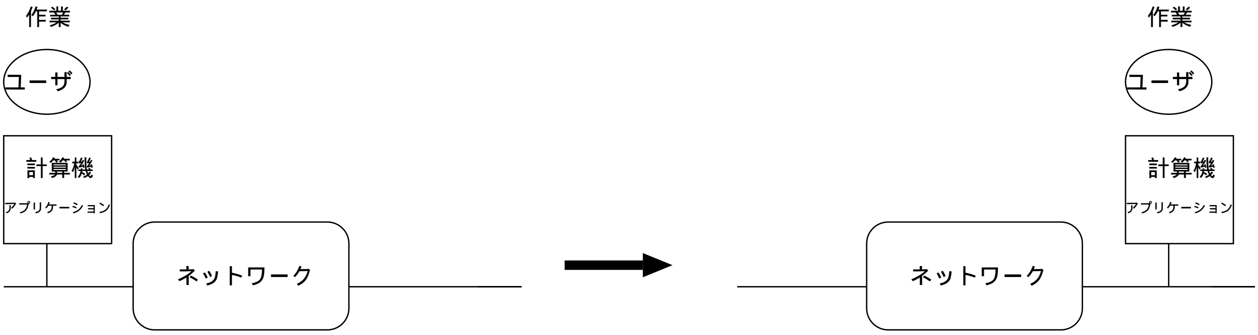


図 2.2: モデル 2 - 1

サーバ・クライアントモデルのアプリケーションで、サーバを同一計算機上に持たない。

このモデルでは、ユーザはクライアントが動作する計算機と共に移動し、移動後に作業を再開するモデルである。この概念図を図 2.3 に示す。計算機がサーバに接続されている時には、作業が可能である。ユーザが移動する時に計算機とサーバと切り離し、同一計算機上にサーバがないため、再びサーバと接続され通信できるまでの間は作業ができない。そして、サーバと接続されれば再び作業が可能となる。

作業を行なうためには、サーバとの通信が必要となる。そのため、サーバが動作している計算機と通信できる環境になければばならない。

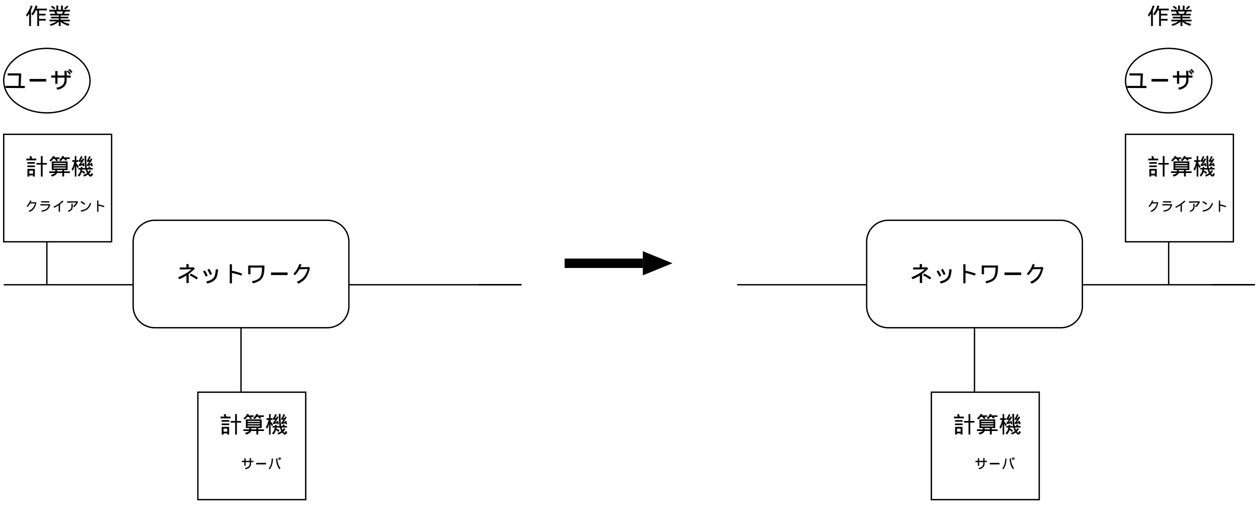


図 2.3: モデル 2 - 2

第 2.2.2 節　モデルの考察

移動計算機環境での作業の継続という点において、第 2.2.1 節で挙げたモデルについて考察する。

モデル 1 は、ユーザは計算機を持たずに移動する。そのため、移動先のネットワークに接続された計算機が存在すれば作業が再開できる。しかし、作業は別の計算機上で行なうことになり、作業環境の変化は大きい。しかも、移動先の計算機上に、移動前と同一アプリケーションが導入されていなければ作業の継続ができない。

これに対してモデル 2 は、移動前後で同一の計算機を使用するため、作業環境の変化は極めて小さい。まず、モデル 2 - 2 のサーバ・クライアントモデルのアプリケーションを用いて作業していた場合を考慮する。この場合、サーバの動作する計算機との通信が必要となる。そのため、サーバと通信できる環境でなければ、移動前の作業は継続をできない。また、計算機を持って移動するためには携帯型計算機上で作業を行なわなければならない。

現在、携帯型計算機が普及している。しかし、計算機を持って移動しても、移動先で計算機をネットワークに接続する環境はまだ整っていないのが現状である。ネットワーク環境が整っていても、外部からの計算機を接続できない環境は依然として多々ある。そのため、現在の計算機利用の形態として計算機を持って移動する機会は少ない。このような計算機を持って移動しない機会が多い実際の環境を考慮すると、モデル 1 の計算機を待たない移動は実際の環境への適合性は高い。そして、モデル 2 の計算機を持っての移動は実際の環境への適合性は低い。

以上のことを表 2.1 にまとめる。

表 2.1: 各モデルの比較

	環境の変化	継続の可能性	実際の環境への適合性
モデル 1	大きい	△	高い
モデル 2 - 1	小さい	◎	低い
モデル 2 - 2	小さい	○	低い

実際の環境への適合性が高く、作業環境を選ばないという点で、本論文ではモデル 1 の「計算機は移動せずユーザのみが移動して作業を継続するモデル」を想定する。そして、モデル 1 において作業の継続を実現するシステムの構築を目指す。

第3章. 移動計算機環境に関する関連研究

ユーザのみが移動する移動計算機環境で作業の継続に関する既存の研究として、screen[1],VNC[2] が挙げられる。本節では2つの研究について述べる。

第3.1節. screen

screenは端末エミュレータであり、1台の端末上で複数の端末を仮想的に生成する。つまり、生成された仮想端末ごとに、文字端末をエミュレートする。

仮想端末のセッションと接続することによって（図3.1(a)）接続した仮想端末で作業が可能である。セッションと接続を切り離しても（図3.1(b)）、再び切り離してあったセッションと接続することにより（図3.1(c)）元の状況で作業の継続が可能である（図3.1(d)）。

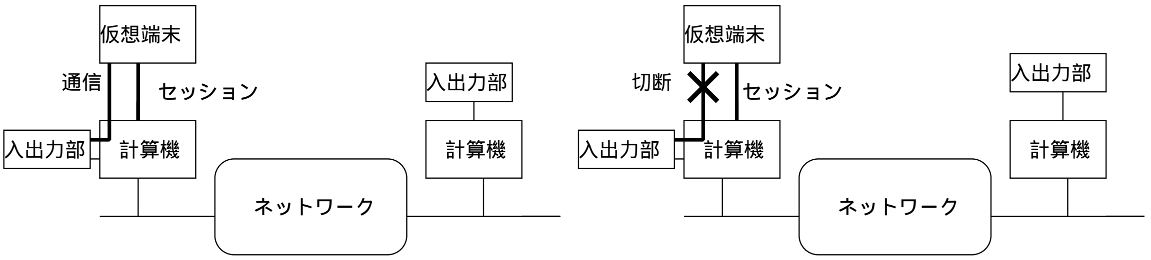
screenでは作業前のセッションを仮想端末と切り離して移動し、再び移動前の仮想端末とセッションを接続することにより、移動した時に作業の継続が可能となる。しかし、screenは文字端末をエミュレートするだけである。したがって、移動時には端末上で動作するアプリケーションしか、screenでは移動した時に作業を継続できない。

第3.2節. VNC

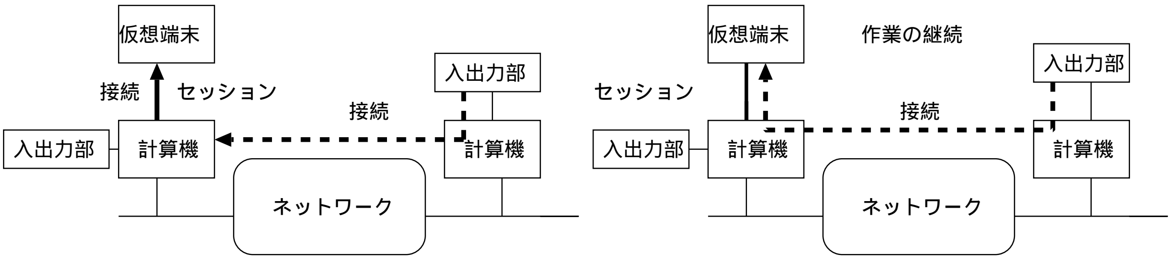
VNC(Virtual Network Computing)はORL(Olivetti & Oracle Research Lab)で開発されたシステムである。インターネット上のどこからでもデスクトップ環境を見ることができるリモートディスプレイシステムである。また、計算機のアーキテクチャにかかわらず動作する。実行画面を図3.2に示す。

VNCで使用されているプロトコルは単にサーバがクライアントに表示されたフレームバッファを更新するだけである。プロトコルがフレームバッファを元に動いているため、すべてのオペレーティングシステムやアプリケーションで動作することが可能であり、TCP/IPのような通信プロトコル上での運用が可能である。入力のプロトコルは、ユーザがキーやポインタのボタンを押した場合やポインティングデバイスが移動した場合には、常に入力イベントはクライアントによってサーバへ送られる。

サーバとクライアントでコネクションが確立されるときには、サーバがクライアントに認証を



(a) 移動前　(b) 切断



(c) 移動後　(d) 作業の継続

図 3.1: screen

求め、認証が認められればデスクトップのサイズ、ピクセルのフォーマット、使われているエンコードを取り決めるためにメッセージを交換する。その後、クライアントはスクリーンをサーバに要求し、セッションが開始される。また、セッションの終了はいつでも可能である。

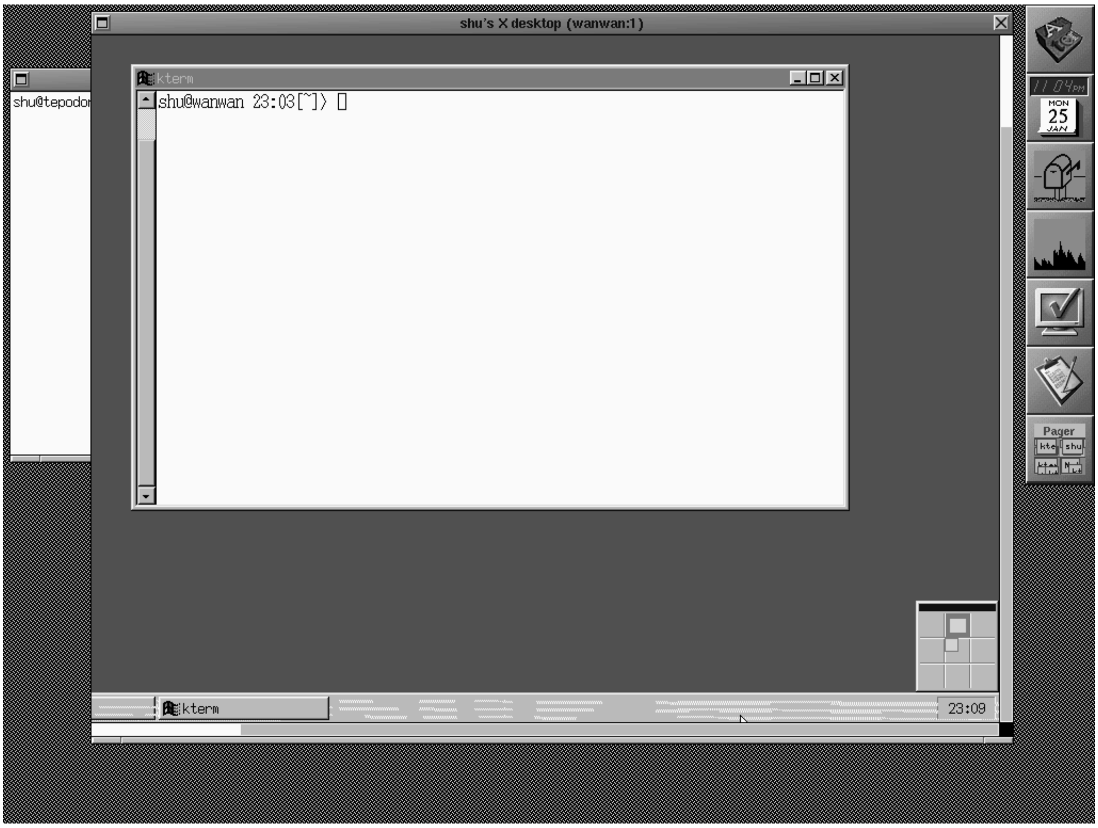
クライアントが行なうことは、信頼できる通信[1] と表示するピクセルをサーバに要求する点である。サーバが行なうことは、クライアントから要求されたフォーマットでピクセルデータを送るなどの、上記のプロトコルを必要な形に変換することである。

VNC ではサーバとなる計算機上で仮想的にデスクトップを起動し、その上で作業を行なう。移動した後に VNC クライアントを起動し、サーバに接続することで移動して作業を継続することが可能となる (図 3.3)。

この場合、問題点は実行速度である。サーバ・クライアント間では画像を転送する。そのため、高速なネットワーク上ではそれほど問題にならないが、ダイヤルアップのような低速なネットワーク上で用いると転送速度が遅いため実用的ではなくなる。また、サーバで仮想的にデスクトップを起動しているのでサーバの負担が大きくなる。

---

[1] 実装では TCP/IP を用いている



```
shu@wanwan 23:03[~]> ▯
```

図 3.2: VNC の実行画面

第 3.3 節.　まとめ

本章では、移動計算機環境の実現のための 2 つの研究について解説した。本節では 2 つの研究について、「システムの汎用性」、「必要な転送量」の 2 つに焦点を絞り、まとめる。

- アプリケーションの汎用性

  screen は文字端末をエミュレートしている。そのため、screen では文字端末上で動作するアプリケーションしか利用できない。一方、VNC はデスクトップを仮想的に起動している。そのため、デスクトップで動作するアプリケーションすべて利用できる。つまり、VNC サーバが起動している OS 上で動作するアプリケーションがすべて利用できる。

- 必要な転送量

  VNC と screen とを必要な転送量を計測し比較する。

  Linux 2.0.35 上で VNC サーバと screeen を動作させ、FreeBSD 2.2.5 上で VNC クライアントと screen と通信する文字端末を動作させる。VNC クライアントと文字端末上でそれ

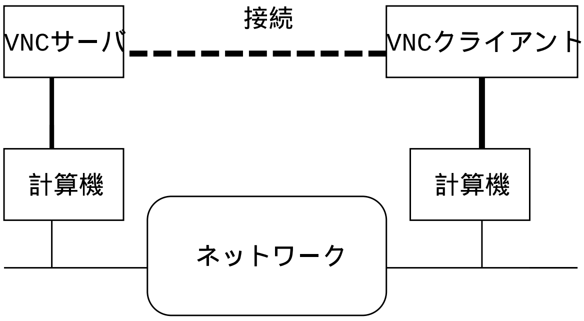


図 3.3: VNC の仕組み

ぞれ 100 文字を入力した。その結果、screen は 12 キロバイト、VNC は 118 キロバイトの転送量が必要であった（表 3.1）。

screen では転送に文字を用いて転送を行なっている。そのため、screen で必要な転送量は少ない。一方、VNC はサーバ・クライアント間で画像を用いて転送を行なっている。そのため、VNC で必要な転送量はたいへん大きい。上記の計測により検証できた。

以上を表 3.1 にまとめる。

表 3.1: 各研究の比較

	汎用性	転送量	転送量の計測値 (byte)
screen	×	◎	12408
VNC	◎	×	117948

screen は利用できるアプリケーションが少ない。そのため、汎用的な利用には向かない。しかし、利用できるアプリケーションであれば、必要な転送量が少ないため、処理速度の低い計算機や低速なネットワークでの利用が可能となる。

一方、VNC は汎用的に利用できる。しかし、必要な転送量がたいへん多く、サーバやネットワークにかかる負荷が大きい。そのため、処理速度の低い計算機や低速なネットワークでの利用は向かない。

本研究では、「システムの汎用性」、「必要な転送量」の 2 点を考慮する。そして、この 2 点を実現した、低速なネットワークでの利用が可能な、汎用性のあるシステムを設計・実装する。

第4章．現状の問題点と解決のアプローチ

本章では、現状の移動計算機環境での問題点を述べ、問題解決のアプローチについて述べる。また、そのアプローチについて検証を行なう。

第 4.1 節.　問題点

現在の移動計算機環境で、ユーザが計算機を持たず、移動して作業の継続を実現するシステムを構築する。その際、以下の 2 つの問題点が存在する。

- ファイル・ソケットの問題

  アプリケーションには作業をするためにアプリケーション本体以外のものを必要とするものがある。アプリケーション以外のものとはファイルやソケットといったものである。ファイルは情報の保存や読み込みに、ソケットは通信に必要となる。

  ファイルを必要とするアプリケーションの場合、移動する前の作業を移動した後で継続するためにはファイルも移動させなければならない。現在のアプリケーションではフロッピーディスクや ftp [4] を用いることによって移動前の計算機から移動後の計算機へとファイルをユーザが移動させなければならない。また、ソケットを必要とするアプリケーションの場合、移動した後にコネクションを張り直さなければならない。

- 作業を再開した時と移動前の終了時での状態の相違

  作業を継続するためにはアプリケーションを起動し、移動前と同じ状態にしなければならない。しかし、アプリケーションを移動先で起動する場合、アプリケーションは初期化して起動する。その時の状態は移動前に作業をしていた状態と異なっている。そのため、現在のアプリケーションで作業を継続するためには、ユーザが移動前の状態にしなければならない。

  エディタを例に挙げる。この例で移動前と同じ状態とは、その時に編集しているファイルを開いており、画面に編集している文章が表示され、書いてある場所にカーソルがある状態である。移動する前にエディタで文章を書いていた場合、移動した後で移動前の文章に「書く」という作業を継続するためには、移動した後にファイルを開き、前に書いていた場所までカーソルを移動しなければならない。

第 4.2 節.　解決のアプローチ

第 4.1 節で述べた 2 つの問題を解決するアプローチとして以下の 3 つのアプローチを挙げる。

- デスクトップ全体の画像を移送する

  移動前の計算機上にサーバを、移動後にクライアントを起動する。サーバはクライアントにデスクトップ全体の画像を移送し、クライアントはサーバに入力イベントを移送する (図 4.3)。このアプローチではアプリケーションはすべて移動前の計算機上で実行され、クライアントは画像を表示するのみである。この場合、デスクトップとは表示している画面全体を指す。

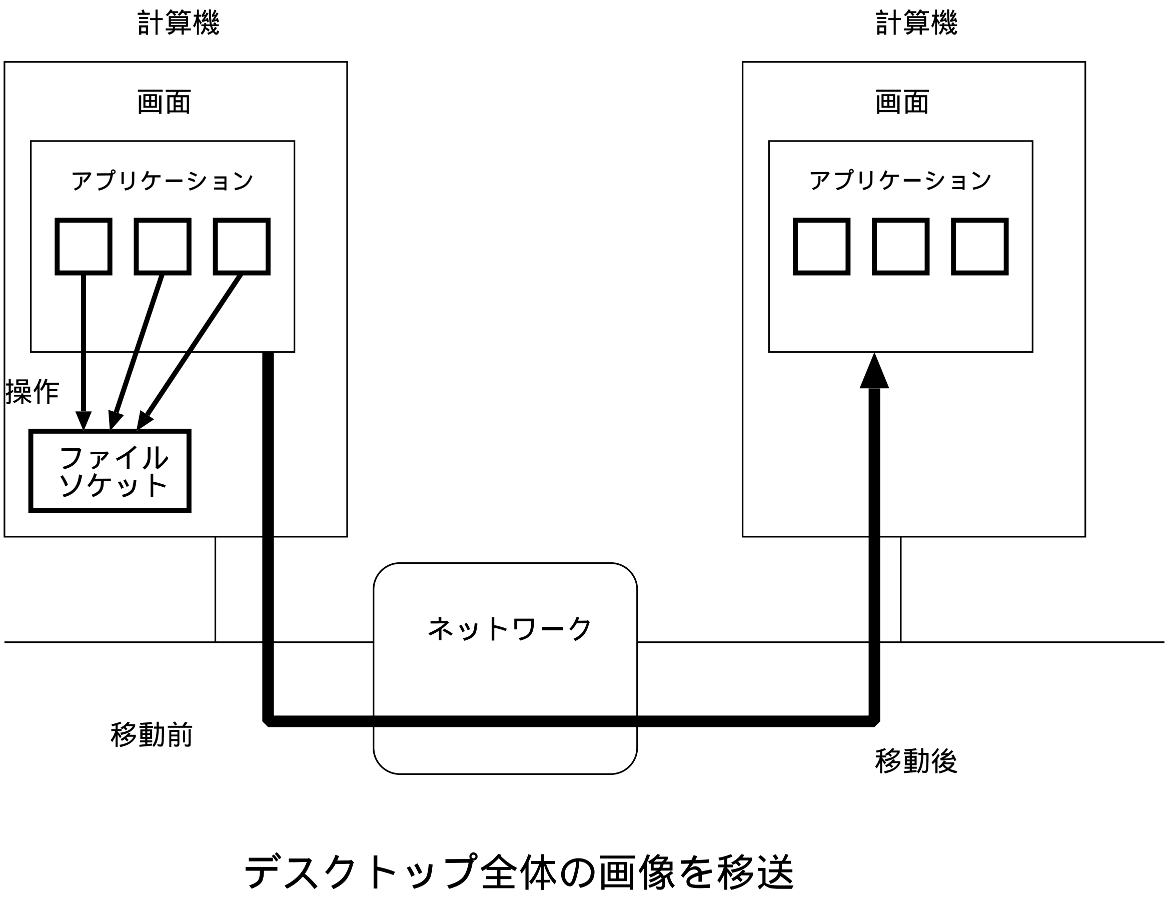


図 4.1: デスクトップ全体の画像を移送

- アプリケーションの画像を移送する

移動前の計算機上にサーバを、移動後の計算機上にクライアントを起動する。サーバはクライアントにアプリケーションの画像を移送し、クライアントはサーバに入力イベントを移送する。このアプローチでは前と同様、アプリケーションはすべて移動前の計算機上で実行され、クライアントは画像を表示するのみである（図 4.3）。

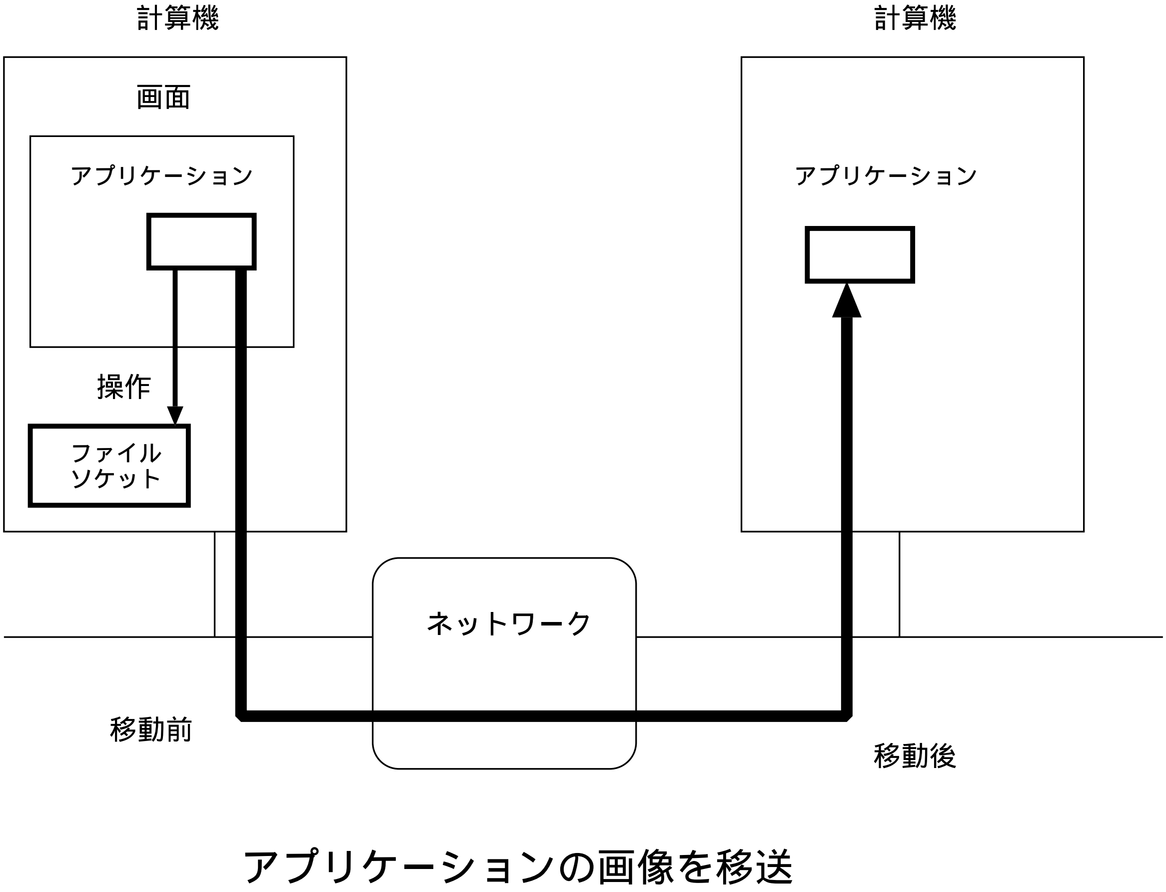


図 4.2: アプリケーションの画像を移送

- アプリケーションの画像を移送

  移動前のアプリケーションで作業していた状態を移動後のアプリケーションに移送する。この場合、移動前のアプリケーションがサーバ、移動後のアプリケーションがクライアントとなる。クライアントはサーバに入力イベントを移送する。そして、サーバとクライアントの両方で実行される。状態とはアプリケーションを実行するのに必要な情報である（図 4.3）。

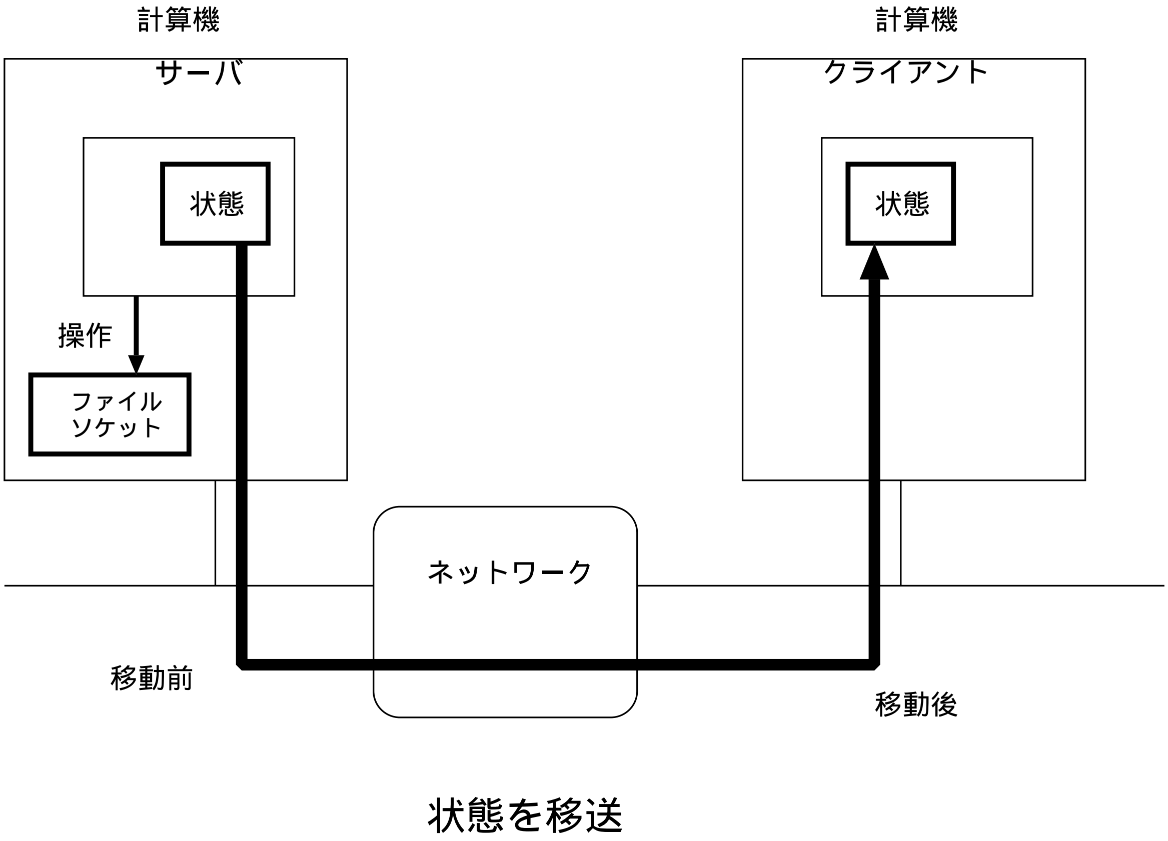


図 4.3: アプリケーションの実行状態を移送

第 4.3 節.　アプローチの考察

第 4.2 節で挙げた解決のアプローチは、画像を移送するものと、アプリケーションの実行状態を移送するもの、の 2 つに大別できる。

まず、画像を移送するアプローチでは、移動前後におけるデータの転送量が大きくなる。さらに、デスクトップ全体を移送すると必要なアプリケーション以外の画像が移送され、オーバーヘッドが大きくなる。ゆえに、アプリケーション単位で画像を移送した方がデータの転送量、オーバーヘッド共に少なくなる。しかし、いずれにせよ画像のデータ量は大きく、低速なネットワークでの利用は転送速度が遅くなり困難となる。実際、関連研究で挙げた VNC は、デスクトップ全体の画像を移送するアプローチを採っており、第 3.3 節で行なった測定により、このアプローチでは転送量が大きいことが実測されている。

次に、アプリケーションの状態を移送するアプローチについて考察する。必要なアプリケー

ションの状態のみを移送すればデータの転送量は小さくすることが可能である。そのため、画像を移送するアプローチに比べネットワークの速度によらず利用が容易になる。

画像を移送するアプローチではアプリケーションの構造を変える必要はない。しかし、アプリケーションの状態を移送するアプローチでは構造を変える必要がある。それは、現在のアプリケーションの構造が状態の移送を考慮して設計されていないためである。

しかし、インターネットを介しての作業の継続を考えると、低速なネットワークを考慮しなければならない。そのため、データ転送量の大きくなる画像を移動するアプローチは適していない。ゆえに、本論文ではアプリケーションの状態をサーバ・クライアントとして移送するアプローチを採用し、新たなシステムを設計・実装する。

第 5 章.　設計

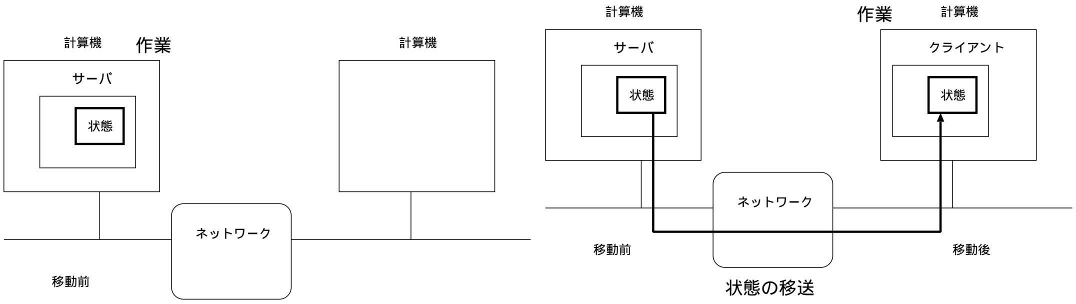
本章では、第 4 章で述べた問題点を解決するための手法を用いて、移動時の作業継続を支援するモデルの設計について述べる。

第 5.1 節.　モデルの概要

第 4 章で、計算機を持たずユーザのみが移動する移動計算機環境で、作業を継続する場合の問題点を挙げた。さらに、その問題点を解決するアプローチを挙げた。アプリケーションの状態をサーバ・クライアントとして移送するアプローチである。

本章ではこのアプローチを用いて、状態を移送することにより移動計算機環境で、作業の継続が可能となるアプリケーションのためのモデルを設計する。移動前をサーバ、移動後をクライアントとする。

始めは、サーバを利用して作業を行なう (図 5.1(a))。移動後にクライアントを起動して、クライアントをサーバに接続する。この時、クライアントはサーバから作業の状態を受け取る (図 5.1(b))。クライアントは受け取った状態に基づき、移動前に作業をしていた状態にする。その後、ユーザは移動前の作業を継続できる。



(a) 移動前の作業　(b) 移動後の作業

図 5.1: システム概要図

本モデルの特徴を以下に挙げる。

- 作業の継続

本モデルを元にアプリケーションを開発することにより、移動前の作業を移動後に継続して行なうことができる。

- 小さい転送量

  状態のみを移送するため、転送量が大きくならない。転送量が小さいため、低速なネットワークでの利用が可能となる。

第 5.2 節. 本モデルの設計

本節では状態を移送するモデルの設計について述べる。まず、本モデルの構成について述べる。次に、動作概要について述べる。

第 5.2.1 節 本モデルの構成

本モデルでは、アプリケーションの状態を移送することによって作業の継続を行なう。アプリケーションの状態を移送する仕組みを設計する際、以下の 5 つの機能が必要となる。

- 入出力機能

  インタラクティブなアプリケーションである限り、入力を受け付け、画面に表示させる必要がある。

- アプリケーションの処理機能

  アプリケーションを動作させるための処理が必要である。

- 状態管理機能

  本モデルを実現するためには、移送する状態を管理する必要がある。

- 通信機能

  状態を移送するためには、サーバとクライアントで通信を行う必要がある。

- 外部管理機能

  第 4.3 節の考察により、ソケットやファイルといったアプリケーションにとって外部的な要素は、移動先に移送しない。そのため、サーバ側で管理する必要がある。

本モデルを構築する上で、以上の 5 つの機能が必要となる。本モデルは機能別に以下の 5 つのモジュールとして形成される。

- ユーザインタフェース部 (以後、UI)

  入出力機能に相当する。キーやマウスの入力を受け付け、状態管理部にある情報を元にして画面に表示させる部分。

- 動作部

  アプリケーションの処理機能に相当する。UI から入力を受け付け、状態管理部の情報と外部情報を操作する部分

- 状態管理部

  状態管理機能に相当する。アプリケーションの再現に必要な情報を管理する部分

- メッセージ通信部 (以後、MT)

  通信機能に相当する。移動元と移動先の動作部と状態管理部のメッセージを仲介する部分

- 外部情報

  外部管理機能に相当する。ファイル、ソケットといったアプリケーション本体以外の外部的な部分

第 5.2.2 節 動作概要

サーバとクライアントの初期動作と、初期動作で移動前の作業を継続した後の動作を述べる。

初期動作

初期動作をサーバとクライアントに分けて述べる。移動後にクライアントを起動し、サーバと接続を行なうことによって作業の継続が行なわれる。

- サーバ

始めに起動した時は、アプリケーションは単独で動作する。移動後に、MT はクライアントからの通信を待つ。クライアントから接続要求を受け取ると、情報管理部が管理している情報を移送する。

- クライアント

起動した時に、サーバへ移動前の状態を移送するよう要求する。サーバと接続されると移動前の情報管理部が管理している情報を受け取る。そして、UI で画面に反映させる。

この動作により、移動前の作業の状態が移送され、移動後に作業の継続が行なわれる。

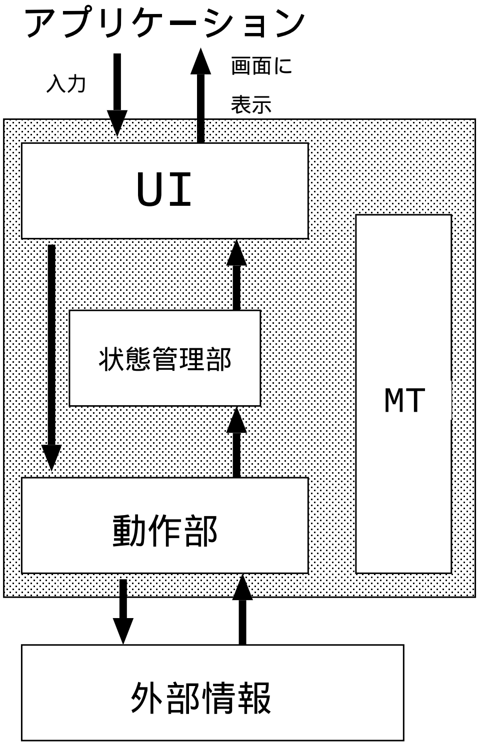


図 5.2: 移動せずに作業を行なう場合の動作概要

作業継続が完了した後の動作

移動後に作業の継続が完了した後、本モデルは以下の 3 つの場合に分かれて動作する。

- 移動せずに作業を行なう場合

UI が受けとった入力を動作部に伝える。動作部は UI から入力を受け、状態管理部の情報を操作する。また、動作部は外部情報を扱う場合は外部情報を操作する。そして、UI は操作された情報を元に画面に表示する（図 5.2）。

- 移動後に作業を継続し、外部情報を扱わない動作を行なう場合

  UI が受けとった入力を動作部に伝える。動作部は UI から入力を受け状態管理部の情報を操作し、UI は操作された情報を元に画面に表示する。動作部は状態管理部の情報操作と同時に MT を介し、移動元の動作部に移動先の UI から受け取った入力を伝える。移動先と同様に移動元の動作部が状態管理部の情報を操作する（図 5.3）。

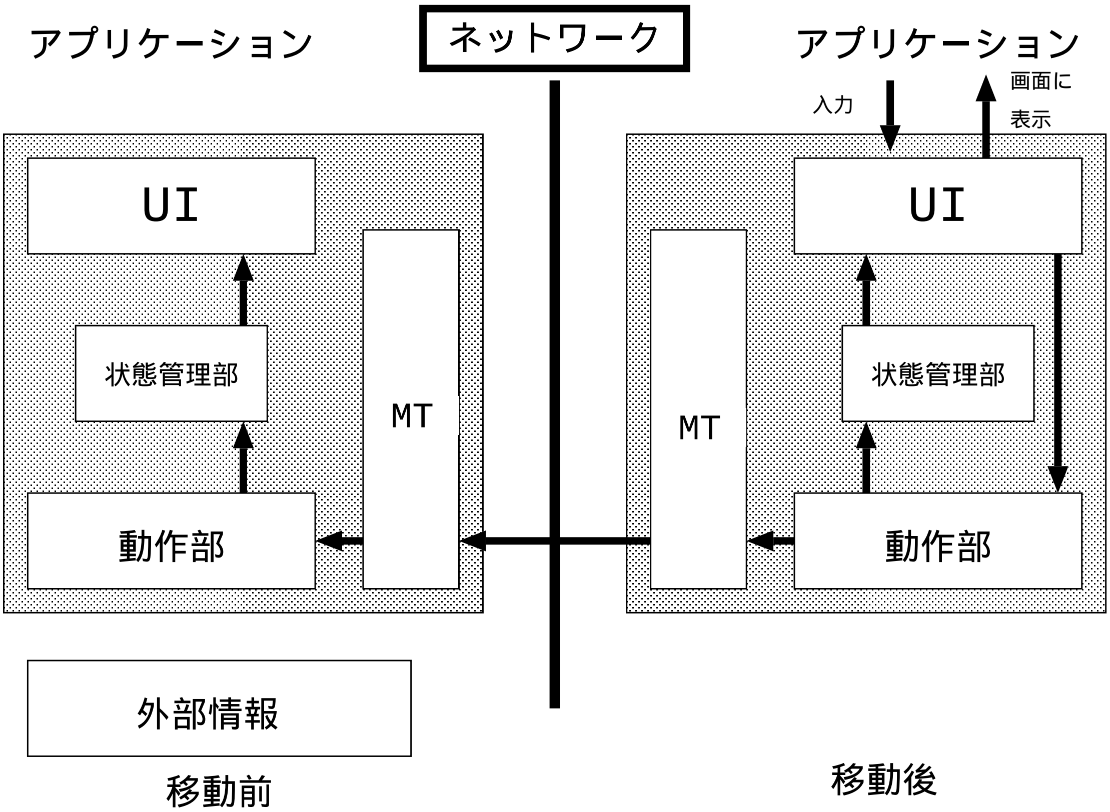


図 5.3: 外部情報を扱わない場合の動作概要

- 移動後に作業を継続し、外部情報を扱う動作を行なう場合

  UI が受けとった入力を動作部に伝える。動作部はこの入力を MT を介し、移動元の MT へ送る。移動元の MT は動作部に移動先の UI から受け取った入力を伝える。移動先の動作部は外部情報と状態管理部の情報を操作する。その後、状態管理部の情報を MT を介し、移動先の状態管理部に送り、UI は操作された情報を元に画面に表示する（図 5.4）。

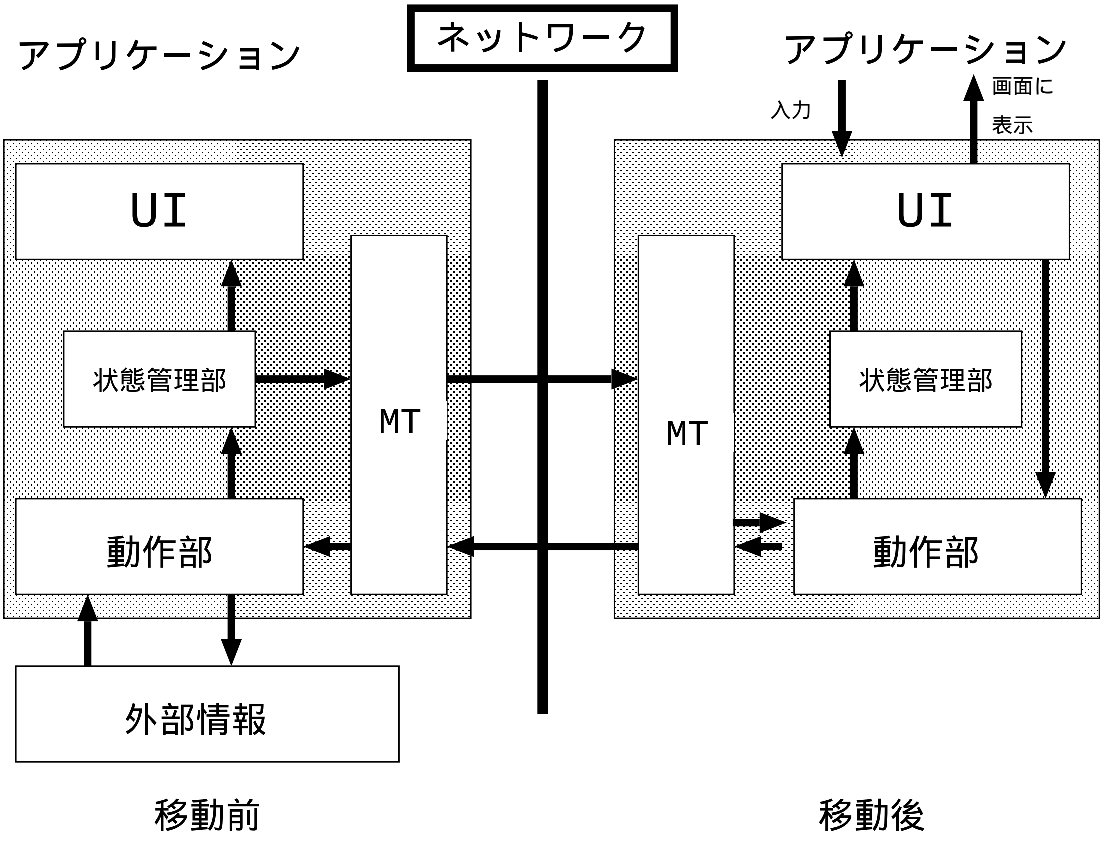


図 5.4: 外部情報を扱う場合の動作概要

第 5.3 節.　ライブラリとテンプレートの設計

第 5.2 節で設計したモデルを使用して、実際に本モデルが利用できるようにする。そのために、モデルの枠組をライブラリとテンプレートの形で提供する。UI と MT をライブラリとして、動作部と情報管理部をテンプレートとして実装する。そして、本モデルを用いたアプリケーション開発が容易にできるようにする。

本節ではライブラリとテンプレートの設計について述べる。さらに、アプリケーション開発者（以後、開発者）による利用の仕方を述べる。

第 5.3.1 節　UI

UI をライブラリとして提供する。以下に、UI のライブラリの設計を行なう。

- 機能

ユーザからのキーボードとマウスの入力を受け付ける。その受け付けた入力情報を動作部へ渡す。

- 動作手順

  1: ユーザからのキーボードとマウスの入力を受け取る

  2: 受け取った入力情報を動作部に受け渡す。

第 5.3.2 節　動作部

動作部をテンプレートとして提供する。以下に、動作部のテンプレートの設計を行なう。

- 機能

  UI から受け取った入力情報を元に、処理を行なう。

- 動作

  動作部の処理をサーバとクライアントに分けて述べる。

  - サーバ

    移動前は、UI からの入力情報によって処理を分岐する。移動後は、クライアントから送られてきた入力情報によって処理を分岐する。移動前後で処理を分岐させる部分は共通するため、1 つにまとめる。

  - クライアント

    UI からの入力情報によって処理を分岐する。外部情報を扱わない処理では、処理を行なうとともに入力情報を MT を介してサーバに送る。外部情報を扱う処理では、入力情報を MT に渡し、サーバから動作結果を MT を介して受け取る。

  処理の内容はアプリケーションによって異なる。しかし、入力情報によって処理が分岐する部分は、アプリケーションによらずサーバ・クライアント共通である。処理の内容が異なるだけである。よって、共通する処理の分岐の形式をテンプレートとして提供する。

- 利用方法

  開発者は入力情報によって必要な処理を、テンプレートの分岐部分に挿入することによって利用可能にする。

第 5.3.3 節　状態管理部

状態管理部をテンプレートとして提供する。以下に、状態管理部のテンプレートの設計を行なう。

- 機能

  アプリケーションに必要な情報を一括して管理する。

- 動作

  アプリケーションの再現に必要な情報を管理する。必要な情報はアプリケーションによって異なるため、そのため、その情報を管理する枠組をテンプレートとして提供する。

- 利用方法

  開発者は、アプリケーションを再現するのに必要な情報の抽出が必要である。そして、抽出した情報をテンプレートの中に挿入することにより利用可能にする。

第 5.3.4 節　MT

MT をライブラリとして提供する。以下に、MT のライブラリの設計を行なう。

- 機能

  MT で必要な機能は次の 6 つがある。

  1. サーバの接続要求を待つ機能
     サーバがクライアントから接続要求を待つ
  2. クライアントの接続要求する機能
     クライアントがサーバへ接続要求を送る
  3. サーバの状態を移送する機能
     サーバがクライアントから接続要求を受けた際、サーバの状態をクライアントへ移送する
  4. クライアントの状態を受け取る機能
     クライアントがサーバからサーバの状態を受け取る

5. サーバの入力情報を受け取る機能
   サーバがクライアントからの入力情報を受け取り動作部へ渡す
6. クライアントの入力情報を移送する機能
   クライアントが受けた入力情報をサーバへ移送する

- 動作手順

  サーバとクライアントに分けて動作手順を述べる。

  サーバ：

  1. クライアントからの接続要求を待ち続ける。
  2. クライアントから接続要求を受けると、サーバの状態をクライアントへ移送する。
  3. クライアントから入力情報を受け取り、動作部へ渡す。

  クライアント：

  1. サーバへ接続要求を送る。
  2. サーバからサーバの状態を受け取る。
  3. サーバへ入力情報を送る。

第6章. 実装

本章では第 5 章の設計に基づき、本モデルを用いたライブラリとテンプレートの実装について述べる。

第6.1節. 実装目標

今回の実装の目標を以下に示す。

- 本モデルを使用したアプリケーション開発が容易にできるようにする

アプリケーションの汎用性を高めるために、本モデルを利用した容易なアプリケーション開発が可能となるようにライブラリとテンプレートを実装する。そのために、アプリケーション開発者が利用しやすいような枠組を提供する。

第6.2節. ライブラリとテンプレートの実装

本論文での作業の継続を考慮したアプリケーションを開発するためにライブラリとテンプレートを実装した。X-Window System のライブラリである Xlib を利用し、Linux2.0.35 上で C 言語を使用して実装を行なった。

第6.2.1節 UI

図 6.1 にユーザから入力を受け、その入力情報を動作部に渡す関数 input() を示す。

- 引数

  なし

- 返り値

  なし

- 機能

まず、XNextEvent() で、ユーザから入力を受け付ける。この時、受け取る入力の種類を表 6.1 に示す。次に、SendInput() で、受け取った入力情報を動作部に受け渡す。

```
void input(void){

  XNextEvent( dis, &ev ); /*  入力情報を受け取る    */
  SendInput(ev);          /* 入力情報を動作部に送る */

}
```

図 6.1: UI の実装概要

表 6.1: 入力表

KeyPress	キーが押された
KeyRelease	キーが離された
ButtonPress	ボタンが押された
ButtonRelease	ボタンが離された
ButtonMotion	押されたボタンが動かされた

第 6.2.2 節　動作部

受け取った表 6.1 で示した入力情報の種類で分類し、動作部のテンプレートを図 6.2 で示すように実装する。

開発者は、入力情報によって分岐された処理の部分に、アプリケーションの必要な処理を挿入する。

```
void SendInput( XEvent ev){

  switch( ev.type){
  case KeyPress :
        キーボードが押された時の処理
    break;

  case KeyRelease:
        キーボードが離された時の処理
    break;

  case ButtonRelease :
        ボタンが離された時の処理
    break;

  case ButtonPress :
        ボタンが押された時の処理
    break;

  case MotionNotify :
        ボタンが動かされた時の処理
  }

}
```

図 6.2: 動作部のテンプレート

第6.2.3節　情報管理部

図6.3に示すように、構造体として状態を管理する。図6.3中に示すように、開発者はアプリケーションを再現するのに必要な情報を挿入する。

```
struct _Reg{

/*      アプリケーション毎に開発者が定義した状態を
        ここに挿入する                                    */



};
```

図 6.3: 情報管理部のテンプレート

第6.2.4節　MT

第5.3節の設計を元にMTのライブラリの実装を行なう。サーバ・クライアント間の通信にはTCP[3]を用いる。以下に、実装したライブラリのAPIの定義を示す。

- サーバ

  int accept_server(int)

  - 引数

    ポート番号を引数とする。

  - 返り値

    ソケット記述子を返り値とする。

  - 機能

    引数で指定したポート番号で、クライアントからの接続要求を待つ。接続要求が来ると、クライアントと接続する。

  void set_appli(int)

- 引数

  ソケット記述子を引数とする。

- 返り値

  なし

- 機能

  クライアントへアプリケーションの状態を移送する。状態を移送することにより、クライアントで作業の継続ができる。

void get_info(int)

- 引数

  ソケット記述子

- 返り値

  なし

- 機能

  クライアントから入力情報を受け取る。

- クライアント

  int setup_client(char , int)

  - 引数

    第 1 引数にホスト名、第 2 引数にポート番号をとる。

  - 返り値

    ソケット記述子を返り値とする。

  - 機能

    第 1 引数で指定したサーバへ接続要求する。その後、サーバと通信をする。

  void get_appli(int)

  - 引数

    ソケット記述子を引数とする。

- 返り値

  なし

- 機能

  サーバからアプリケーションの状態を受け取る。状態を受け取ることにより、作業が継続できる。

void send_info(XEvent)

- 引数

  入力情報を引数とする。

- 返り値

  なし

- 機能

  UI で受け取った入力情報をサーバへ送る。

第 7 章. サンプルアプリケーション

第 7.1 節. サンプルアプリケーションの設計

本研究では、本モデルの評価を行なうために、サンプルアプリケーションとしてペイントソフトの設計・実装を行なう。本節ではそのサンプルアプリケーションの設計について述べる。

第 5.2 節で述べたモデルが実際に利用可能であることを示し、他の研究と比較・評価を行なう。そのために、上記のシステムの設計に基づきライブラリとテンプレートを利用する。そして、サンプルアプリケーションとして、ペイントソフトを設計する。

本アプリケーションはマウスを使い、画面に線を引くことにより描画するアプリケーションである。任意の計算機上で本アプリケーションを起動し、作業を行なう。移動する時に書いていた状態を、移動後に本アプリケーションを起動することにより移送する。そして、移動前で行なっていた作業を継続する。移動前に書いていた状態とは、書いていた絵、線の色、太さを指す。

第 7.1.1 節 アプリケーションの構成

ペイントソフトの 5 つのモジュールの設計について述べる。第 5.2 節のモデルの設計に基づき、第 5.3 節でのライブラリとテンプレートを利用する。

- UI

  第 6.2.1 節で提供されているライブラリの input を用いてマウス入力を受け取り、受け取った入力情報を動作部に送る。

- 状態管理部

  ペイントソフトでは、アプリケーションを再現するのに必要な情報を線の色と太さとする。第 6.2.3 節で提供されたテンプレートに線の色と太さの情報を挿入し、管理する。

- 動作部

  第節で提供されているテンプレートに UI でのマウス入力情報によって分岐された処理を、図 7.6 で挿入する。

- MT

  サーバは、第 6.2.4 節で提供されているライブラリ群 accept_server、set_appli、get_info を を用いることによって、クライアントの接続要求を待ち、アプリケーションの状態をクライアントへ移送し、さらに、接続後は入力情報をクライアントから受け取る。

  クライアントは、第 6.2.4 節で提供されているライブラリ群 setup_client、get_appli、put_input を利用することによって、サーバへ接続要求をし、サーバのアプリケーションの状態を受け取り、さらに入力情報をサーバへ送る。

- 外部情報

  描画した絵を保存するファイル、保存されているファイルを外部情報とする。

第 7.1.2 節　アプリケーションの動作概要

前節のシステムの設計に基づき、本アプリケーションは以下の 3 つの場合に分かれて動作する。

- 移動せずに作業を行なう場合

  まず、マウス入力を UI が受けとる。動作部は UI から入力を受け状態管理部の情報を操作する。また、動作部はファイルに保存する場合とファイルから読み出す場合はファイルを操作する。 そして、UI は操作された情報を元に画面に表示する (図 7.1)。

- 移動後に作業を継続し、外部情報を扱わない動作を行なう場合

  マウス入力を UI が受け取り動作部に伝える。動作部は UI から入力を受け状態管理部の情報を操作し、UI は操作された情報を元に画面に表示する。動作部は状態管理部の情報操作と同時に MT を介し、移動元の動作部に移動先の UI から受けとった入力を伝える。移動先と同様に移動元の動作部が状態管理部の情報を操作する (図 7.2)。

- 移動後に作業を継続し、外部情報を扱う動作を行なう場合

  データのファイルへの保存、データのファイルからの読み出しがこの場合に当たる。

  マウス入力を UI が受け取り動作部に伝える。動作部はこの入力を MT を介し、移動元の MT へ送る。移動元の MT は動作部に移動先の UI から受けとったマウス入力を伝える。移動先の動作部はと状態管理部の情報を操作する。その後、状態管理部の情報を MT を介し、移動先の状態管理部に送り、UI は操作された情報を元に画面に表示する (図 7.3)。

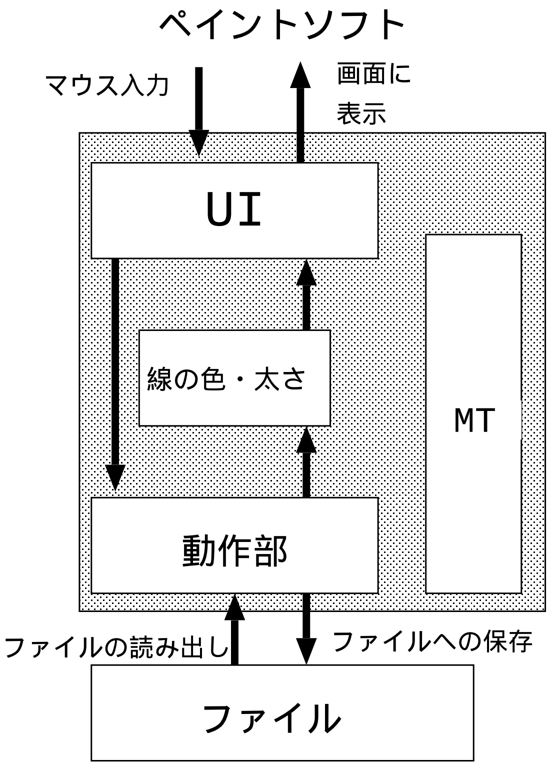


図 7.1: 移動せずに作業を行なう場合の動作図

第 7.2 節.　サンプルアプリケーションの実装

サンプルアプリケーションであるペイントソフトは Linux2.0.35 上で実装した。5 つのモジュールの実装について述べる。

第 7.2.1 節　UI

アプリケーション画面は１４種類のボタンと描画する部分となる。図 7.4 に画面の１４種類のボタンは 2 種類の色、8 種類の太さ、ファイルの保存、読み込み、すべてやり直し、終了のボタンである。

マウスの入力を受け取る。受け取る情報は、(X.Y) 座標とマウスのボタンの状態をイベントとして受け取る。また、マウスの状態は以下の 2 種類の情報である。

- ButtonPress ：ボタンが押された

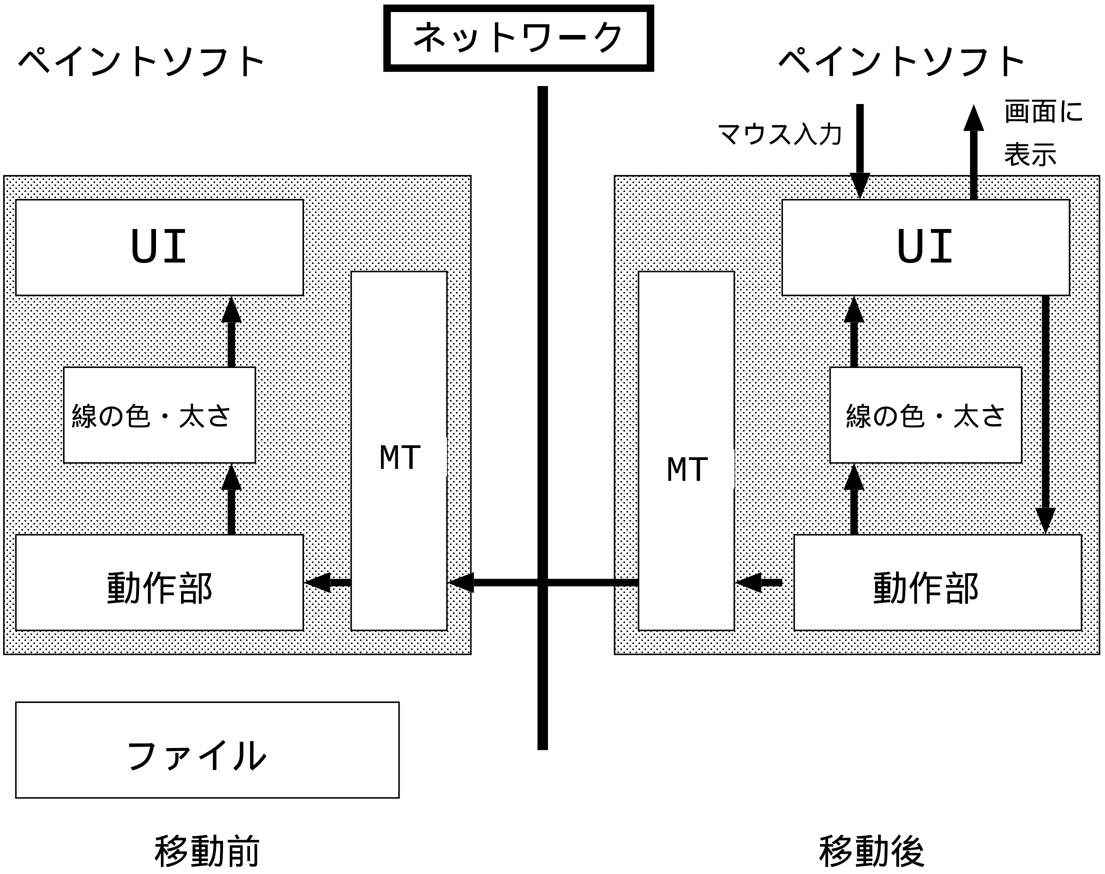


図 7.2: 外部情報を扱わない場合の動作図

- ButtonMotion　：ボタンが押されマウスが動かされている。

第 7.2.2 節　状態管理部

本アプリケーションでは、状態を線の色と太さとして管理する。図 7.5 に示すように、線の色を変数 col、線の太さを変数 bol で表す。

第 7.2.3 節　動作部

UI から受け取った入力情報を元に、図?? で示すように分岐する。

サーバ ： クライアントからの入力情報を受け取る。受け取った後は、UI とのインタフェース

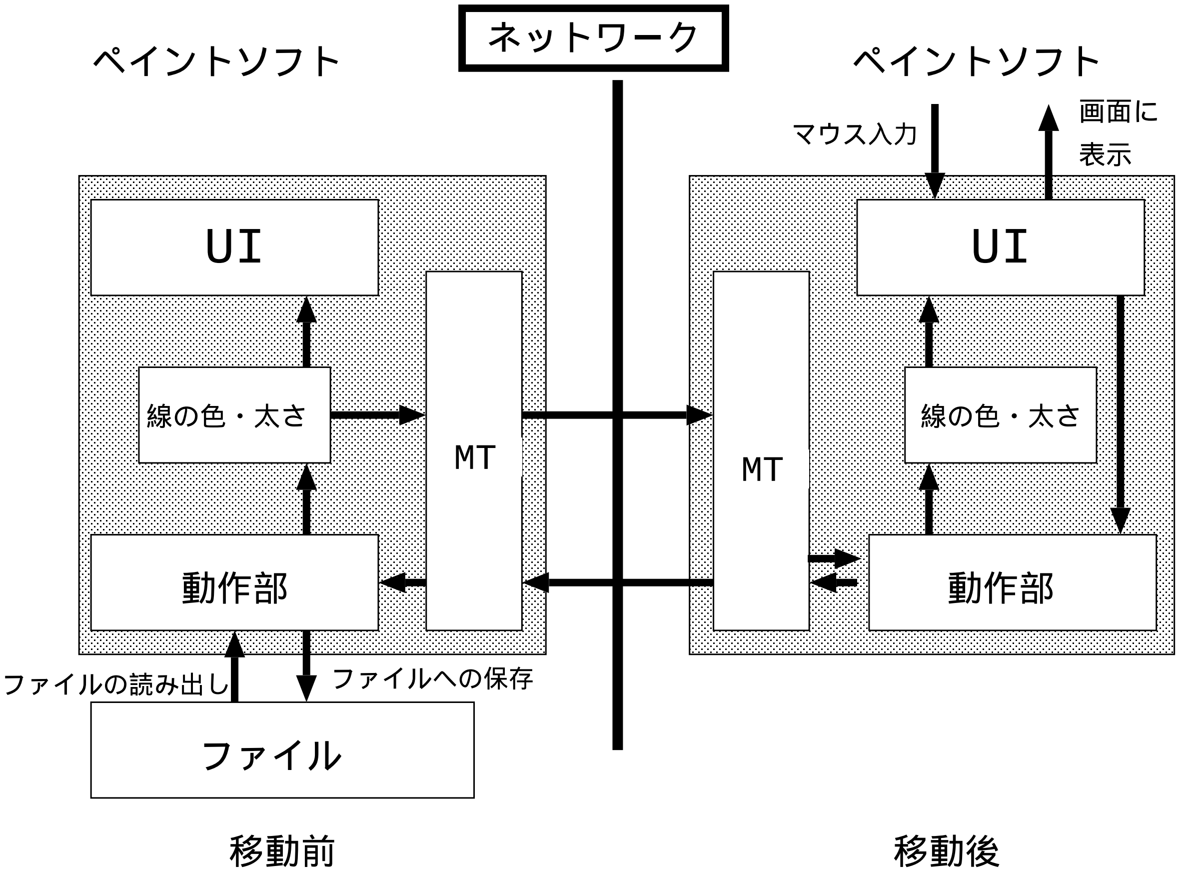


図 7.3: 外部情報を扱う場合の動作図

と同様である。

クライアント ： UI から受け取った入力情報をサーバに送る。

分岐された処理の部分は以下の処理を挿入する。

- ボタンが押された時の処理
  - 色の変更

    UI から色のボタンへの'ButtonPress' イベントを受け取ると、状態管理部の色の変数 col を押された色の数値に変更する。
  - 太さの変更

    UI から太さのボタンへの'ButtonPress' イベントを受け取ると、態管理部の太さの変数 bol を押された太さの数値に変更する。
  - データの保存

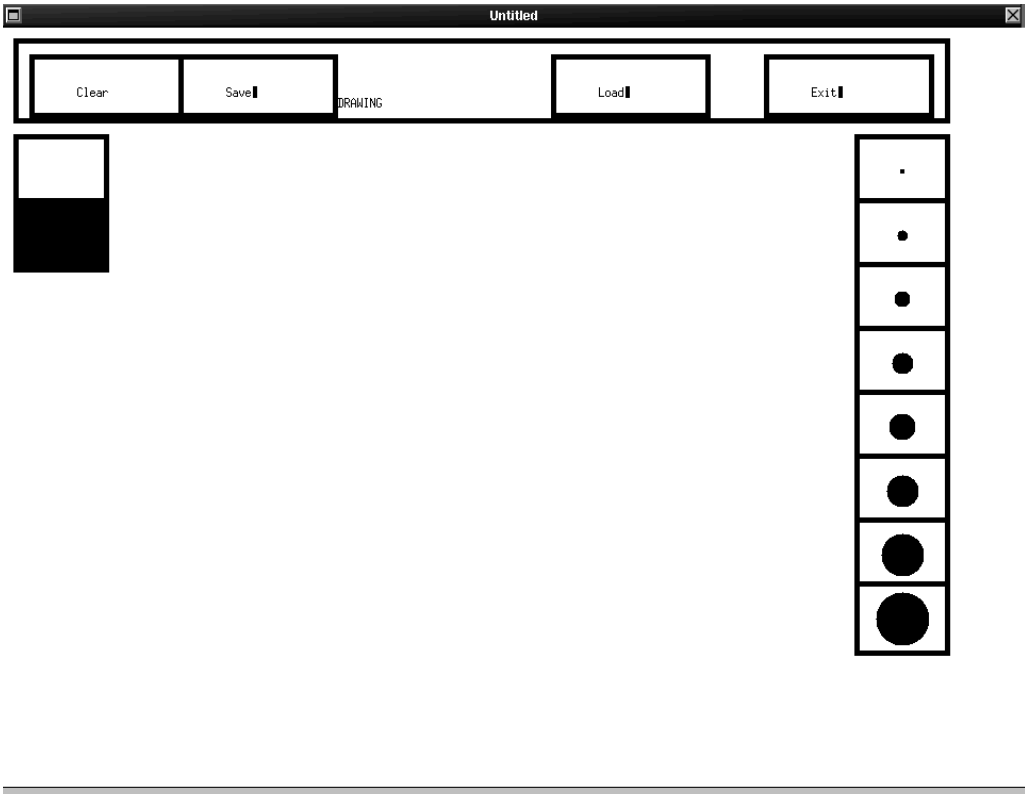


図 7.4: 実際の UI 画面

UI から Save のボタンへの’ButtonPress’ イベントを受け取ると、ファイルへデータの保存を行なう。この時、サーバとクライアントでは動作が異なる。サーバの場合、現在表示されている画像をビットマップとして取り込み、ファイルに保存する。一方クライアントの場合、サーバへ”UI から Save のボタンへの’Press’ イベントが発生した”ことを送る。

– データの読み込み

UI から Load のボタンへの’ButtonPress’ イベントを受け取ると、ファイルからデータを読み込む。この時、サーバとクライアントでは動作が異なる。サーバの場合、現在ファイルに保存されているビットマップ情報を画面に表示する。一方クライアントの場合、サーバへ”UI から Load のボタンへの’Press’ イベントが発生した”ことを送る。

– すべてやり直し

UI から Clear のボタンへの’ButtonPress’ イベントを受け取ると、いままで描画されていた画像を消去し、何の描画されていない状態にする。

```
struct _Reg{

  int col;
  int bol;

};
```

図 7.5: アプリケーションの情報管理部

```
void SendInput( Xevent ev){

  switch(入力の種類){
  case ButtonPress :
ボタンが押された時の処理
    break;

  case MotionNotify :
ボタンが動かされた時の処理
  }

}
```

図 7.6: UI の実装

  - 終了

    UI から Exit のボタンへの’ButtonPress’ イベントを受け取ると、プログラムを終了する。

- ボタンが動かされた時の処理
  - 描画

    描画はマウスのボタンを押した状態で動かし、マウスの動いた点の (X,Y) 座標を結び続けることで線を引く。

第7.2.4節　MT

提供したライブラリaccept_server、set_appli、get_info setup_client、get_appli、send_infoを利用することにより、以下の通りに動作する。

サーバ クライアントと接続されていない時は、クライアントからの接続要求を待ち続ける。クライアントとの接続後は、クライアントからのマウスの入力情報を待ち続ける。

クライアント サーバへ接続要求をする。接続後、サーバにマウスの入力情報を送る。

第8章.　評価

本章では、第 5 章の設計に基づき第 6 章、第 7 章で行なった実装を用いて、測定実験に基づく定量的な評価及び定性的な評価を行なう。

第8.1節.　本モデルの実現性の評価

本論文では、状態を移送することによって移動後に作業の継続が可能となるモデルを提案した。本節では、提案したモデルの実現性を評価を行なう。

本モデルに基づき、Linux 2.0.35 上でライブラリとテンプレートの実装を行なった。そして、それらを利用して、同じく Linux 2.0.35 上でサンプルアプリケーションの実装を行なった。その結果、移動前の作業を、移動後に継続することが可能であった。このことより、現実に本モデルを利用し、移動時の作業継続が実現可能であることが検証された。

第8.2節.　ライブラリとテンプレートの汎用性の評価

本節では本研究で実装を行なったライブラリとテンプレートを汎用性の点から評価する。

ライブラリとツールキットは、本モデルを利用したアプリケーションの開発を容易にするために、Linux 2.0.35 上で本研究で実装を行なった。本モデルで必要となる部分を用意し、アプリケーション開発の枠組を構築した。サンプルアプリケーションとしてペイントソフトを実装した結果、アプリケーションの開発が容易にできることが検証された。

本ツールキットを利用し、容易にアプリケーションが開発可能となる。そのため、汎用性のあるシステムといえる。

第8.3節.　ネットワーク負荷の評価

本研究で構築したシステムを用いて、ペイントソフトを実装した。本節では、ペイントソフトについて、移動する際に、サーバ・クライアント間で必要な転送量を測定する実験を行なった。まず、実験の動作環境と実験内容を示した後、必要な転送量と動作時間の計測結果をまとめる。

第 8.3.1 節　動作環境及び実験内容

Linux 2.0.35 上で screen,VNC, ペイントソフトのサーバを動作させた。そして、FreeBSD 2.2.5 上でクライアントを動作させた。

転送量の測定は本研究で実装したペイントソフトを用いた。具体的には、VNC の測定は VNC クライアント上でペイントソフトのサーバを動作させた。ペイントソフトの測定はクライアントを動作させた。

図 8.1 で示すように、ペイントソフト上で 3 本の線を引いた。サーバの動作している計算機で tcpdump で流れてくるデータを取り出し、そのデータ量を必要な転送量とした。

ただし、screen は文字端末上のアプリケーションしか動作せず、ペイントソフトを用いた実験は行なえなかった。そのため、文字端末上に 100 文字の文字を打ち込んだ時の転送量を計測した。この値を参考として用いる。

動作時間の測定は補研究で実装したペイントソフトを用いた。ペイントソフトの中にミリ秒まで測定する関数を入れ、線一本にかかる時間を測定した。この測定は、VNC と本システムで行なった。動作環境は転送量の測定と同一である。10 回の実験を行ない、その平均を出す。

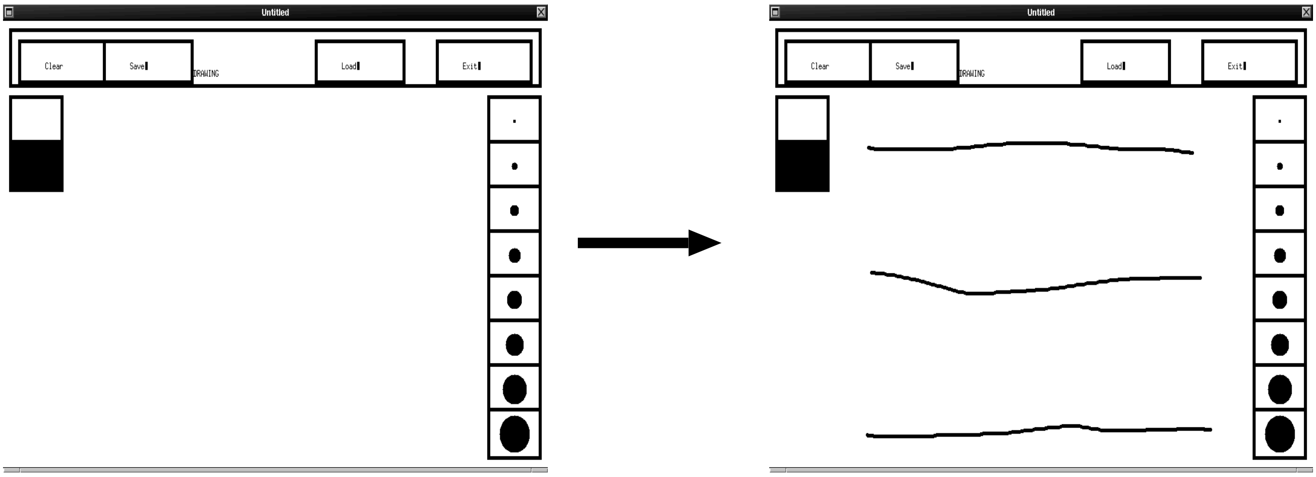


図 8.1: 実験内容

第 8.3.2 節　実験結果

ネットワークにかかる負荷を計測するために、サーバ・クライアント間での転送量を計測する。実験で測定した本システム、VNC、screen の転送量と動作時間を表 8.1 にまとめる。

表 8.1: 実験結果

	必要な転送量 (byte)	動作時間（秒）
VNC	1524572	2.38
本システム	107060	1.18
screen	12408	-

第 8.3.3 節　動作結果と既存研究との比較

- 汎用性

  screen は文字端末上で動作するアプリケーションしか利用できない。このシステムを使用して、利用できるアプリケーションが少なく、汎用性は低い。

  VNC サーバが起動している OS 上で動作するアプリケーションがすべて利用できる。このシステムは既存のアプリケーションを利用でき、汎用性は極めて高い。

  本システムでは、ツールキットを利用することにより、様々なアプリケーションが容易に開発できる。しかし、本システムを使用するためには新たにアプリケーションを開発しなければならない。その点で、本システムは VNC に劣る。しかし、様々なアプリケーションを利用できるという点で、screen より優れている。

- 必要な転送量

  サーバ・クライアント間で必要となる転送量は、第 8.3 節で行なった実験の結果より、表 8.1 のようになった。

  まず、本システムと screen を比較する。screen の実験の結果は参考値であるが、本システムの 10 分の 1 以下の転送量しか必要としなかった。

  次に、本システムと VNC を比較する。本システムは VNC の 10 分の 1 以下の転送量しか必要としなかった。

- 動作時間

  システムの動作時間は、第 8.3 節で行なった実験の結果より、表 8.1 のようになった。本システムと VNC を比較する。本システムの動作時間は VNC の約半分であった。

以上の実験の比較評価を表 8.2 にまとめる。

表 8.2: 他の研究との比較評価

	汎用性	転送量	動作時間
VNC	◎	×	×
本システム	○	○	○
screen	×	◎	-

8.4 考察

本システムは VNC のように既存のアプリケーションを利用できない。しかし、様々なアプリケーションの容易な開発が可能であるため、汎用性がある。また、状態を移送する手法によって、必要な転送量を抑えることができた。転送量を抑えることにより、動作時間が短縮できた。よって、本システムはネットワークに負荷をかけないといえる。

低速なネットワークを介した移動計算機環境で、汎用性のあるアプリケーションを利用するのには、本システムは適していることが検証できた。

第9章. 結論

本章では第 8 章での評価に基づいて結論を述べ、今後必要とされる研究課題について述べる。

第9.1節. まとめ

計算機を持たない移動計算機環境で作業を継続するためには、ファイルやソケットといった外部的な問題と移動前後でアプリケーションの状態が異なるといった問題が生じる。

本研究では、この問題を解決し移動計算機環境で作業継続の実現を支援するために、アプリケーションの状態を移送する機構の構築を行なった。第 8 節で示したように、本実装により状態を移送することにより作業が継続できた。

以上の結果から、本システムは計算機を持たずに移動するといった移動計算機環境での作業継続の問題を解決した。そして、移動計算機環境での作業継続を支援できることが確認できた。

第9.2節. 今後の課題

本研究は移動時の作業継続という視点で移動計算機環境の支援を行なった。最後に完全な移動計算機環境の実現に向けて、今後解決しなければならない問題点を示す。

- 他のプラットフォームでのツールキット作成

  今回は Linux 2.0.35 上で実装を行ない、FreeBSD 2.2.5 上での動作を確認した。他のプラットフォームで本システムを利用したアプリケーション開発が可能となるために必要となる。

- ライブラリ化

  今回のシステムのライブラリ化はまだ一部であった。より汎用性を高めるために、本システムで共通する部分のさらなるライブラリ化が有効である。本システムを利用したアプリケーションの開発がより容易になる。

以上の課題を解決することにより、より効果的な移動計算機環境が実現される。

謝辞

本論文の作成にあたり、御指導いただきました慶應義塾大学環境情報学部村井純教授並びに同学部徳田英幸教授に感謝致します。

絶えず御指導と御助言頂きました慶應義塾大学部植原啓介助手、慶應義塾大学政策・メディア研究科後期博士課程の冨永明宏氏、渡辺恭人氏、及び同修士課程の宅間啓氏、大西孝義氏、小浦大将氏、関谷勇司氏、西田視磨氏、慶應義塾大学環境情報学部川喜田佑介氏、佐藤雅明氏に感謝します。その他、多くの村井・徳田・楠本・中村研究室の諸氏に感謝します。

参考文献

[1] GNU screeen HOME PAGE http://www.gnu.org/software/screen/screen.html

[2] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, Andy Hopper. Virt ual Network Computing,IEEE INTERNET COMPUTING, JANUARY FEBRUARY 1998

[3] J.Postel. Transmission Control Protocol. RFC 793, September, 1981.

[4] N.Neigus. File Transfer Protocol. RFC542, December, 1973