２００５年度未踏ソフトウェア創造事業
（ネット公開を目的としたマルチウィンドウアプリ用フレームワーク）
成果報告書

開発代表者：永井 秀利
担 当 ＰＭ：並木 美太郎
プロジェクト管理組織：日本エンジェルズ・インベストメント（株）

２００６年８月３１日

目次

1. 要約

Ruby/Tk ベースでローカルのウィンドウシステム向けに書かれたマルチウィンドウの GUI アプリケーションを、ソースの変更はほとんど必要とせず，操作性はそのままに，セキュリティ確保のための監視・制御の下で不特定多数向けのネットワークアプリケーションとして公開可能にするためのフレームワークと，そのフレームワークに基づいてサービスを提供するサーバを構築するためのプログラム群とを開発した．

2. 背景及び目的

従来の状況においては，ローカルのウィンドウシステム向けの GUI アプリケーションと一般的なネットワークアプリケーションとを比較した場合，操作体系に関する考え方から根本的に異なっており，それゆえ，プログラミングにおける技術も大きく異なるものであった．また，ローカルシステムで非常に有用なライブラリが存在するにもかかわらず，ネットワークアプリケーションではそれが使えずに代用手法に頼らざるを得ないというケースも多く存在した．GUI アプリケーションを外部から操作する手段が全く存在しなかったわけではないが，それは絶対的に信頼の置けるサーバ・クライアント間でのみ有効と言えるものであり，必ずしも信頼できない不特定多数の他者を相手にする場合に使えるような手段ではなかった．

そこで本プロジェクトでは，特殊な外部ライブラリを用いていたり，精緻なインタラクティビティを有していたりするようなアプリケーションでもそのままにネットワークアプリケーション化できるようにすることで，GUI アプリケーションの作成技術がカバーする範囲をローカルのウィンドウシステムからネットワークアプリケーションに拡大することを目的とする．

これにより GUI アプリケーションのポータビリティを強化し，ローカルのウィンドウシステム用に作成したものと似て異なるものをネットワーク用に一から作り直さねばならないというような愚を避け，低コストでネットワークアプリケーションサービスを実現可能にする．

3. プロジェクト概要

本プロジェクトの根底には，「ローカルからネットワークまで『お気軽に』GUI プログラミングを楽しめるようにしたい」という想いがある．それゆえ，決りごとが多くてプログラミングに手間のかかる GUI ツールキットではなく，簡単なものはより簡単に，高度なものはそれなりに作ることができる GUI ツールキットをターゲットとする．そのような GUI ツールキットとして本プロジェクトで選択したのが Ruby/Tk である．Ruby/Tk は，気軽に GUI を作れることで有名な Tcl/Tk に対して Ruby によるオブジェクト指向の殻をかぶせたものになっている．Tcl/Tk は，簡単な GUI なら数行から十数行程度で書けるという入口の広さと，高度な機能を持った GUI をも構築できるという奥の深さとをあ

わせ持っている．Tcl/Tk は基本設計が古く，規模が大きくなるとプログラムの見通しが悪くなりやすいという欠点があるが，Ruby/Tk ではそうした欠点を包み込むことにより，より広範囲で「お気軽さ」を維持できるようになっている．

そこで本プロジェクトでは，Ruby/Tk で書かれた GUI アプリケーションを，操作性はそのままに，低コストで簡単かつ安全にネットワークアプリケーション化できるようにすることを目標に Ruby/TkORCA（Ruby/Tk On RFB Canvas)と名付けたフレームワークを開発した．その際，公開用アプリケーションの開発・テストから公開（サーバ構築）までの作業をスムーズに行えるように，一連のソフトウェアとその機能とを設計・作成し，「お気軽なネットワーク GUI プログラミング」が行えるものにした．

ネット公開したい Ruby/Tk アプリケーション（同時に複数個も可能）があるなら，それらをそのまま開発物に渡してやればよい．セキュリティ上の問題があるために修正が必要となる場合を除き，それだけでそれらのアプリケーションをネットワーク GUI アプリケーションにすることができる．サービス提供には RFB プロトコルを使っているので，何らかの VNC ビューアが動きさえすればサービスを利用してもらうことが可能である．そのため，Web や PC のみならず一部の携帯電話にすら GUI アプリを提供できる．

開発物は，大きく分けると Ruby/TkORCA 本体と Ruby/TkORCA を用いたサーバを稼働させるための部分とから構成される．さらに Ruby/TkORCA 本体は，アプリケーションの実行環境である daughter 部と実行監視・制御を司る mother 部とに分けられる．daughter は mother の完全な支配下にあり，アプリ上でのコマンド呼び出しが妥当なものであるかをチェックしたり，動作状況と無関係に稼働中のアプリ内部を操作したり，アプリ利用者によるウィンドウ操作までも監視対象にしたりといったことが可能である．詳しくは第 4 章を参照されたい．

開発物では，その利用者が独自に監視や制御の機能を追加したい場合のために，それを支援するための API も提供している．ただし，今回の開発結果として提供する API は非常に基本的なものであり，機能追加をより簡便にする高レベル API の整備は今後の課題とした．

本プロジェクトの開発物を用いれば，ほんの数行のトイプログラムから精細な制御を行う高度なネットワーク GUI アプリケーションまでを，同じ技術を発展的に用いて作成することが可能になる．その途中にはそうした発展を阻害するような大きな技術的障壁などは存在しないので，お気軽に作成したものをお気軽に改良していくことができる．また，最初は外部公開などする気持ちなしに作成してきたものを，思い立った時点で非常にわずかな労力でネットワーク公開できるようになる．

実例として，本プロジェクトの開発物を用いることで，いわゆる「Hello, World」プログラム程度ならほんの５分でネットワーク GUI アプリとして作成・公開できるようになり，ローカルのウィンドウシステム用に作成していた 2400 行超，外部ライブラリ使用，コンソールへの出力ありというようなプログラムですら，そのプログラムを初めて受け

取ってから１時間と経たずにネットワークGUIアプリとして利用可能にすることができた．特に後者の場合，ネット公開できれば嬉しいと思いつつも既存の枠組では新たに作り直しになるために諦めていたものがあっさりと公開可能になるという非常に喜ばしい成果を得ることができたと言える．

大学の研究室などでは，実験用として作成していたシステムをデモ用にも使いたくなるというケースがしばしば見られる．その際，機密事項になるプログラムやデータは一切外には出せないためにソース配布では公開することができず，システムの操作性を含めて多くの部分を全く新しく開発し直さなければならないということも珍しい話ではない．実例のように，最初にRuby/Tkを選択していさえすれば，本プロジェクトの開発物によって好きな時点で少ない労力でネット公開することが可能になる．

4.　開発内容

4.1　開発物の名称

開発物の構造上の特徴に基づいて，正式名称をRuby/Tk On Remote-Frame-Buffered Canvas（Ruby/Tk On RFB Cnavas）と名付けた．同時に略称をRuby/TkORCAとした．

4.2　開発物の内部構造

4.2.1　開発物中心部の構造

開発したRuby/TkORCAの基本構造を図１に示す．

基本的には，一つのクライアントに対するサービスは，Ruby/TkORCAプロセスとVNCサーバプロセスとの二つのプロセス構成される．提供するアプリケーションが特に要求する外部プロセスが存在しない限りは，サービス提供において他のプロセスは必要としない．必須となる二つのプロセスの内，VNCサーバには既存のソフトウェアを援用するため，本プロジェクトでの直接の開発対象はRuby/TkORCAプロセスの部分である．なおVNCサーバは，稼働中のローカルの画面とは独立した画面を持つ必要があるため，Xvncまたはその同等品(Unix系OSではこのタイプが標準)を用いるものとする．

一つのRuby/TkORCAプロセス上には１個のmotherと複数個のdaughterとが稼働するが，開発物ではその纏まりをfamilyと呼ぶ．一つのクライアントへのサービス提供は一つのfamilyが担当することになる．このmotherとdaughterとはマスター・スレーブの関係にあり，motherはdaughterを完全にコントロールすることが可能である．

familyはRuby/Tkで書かれた一つのアプリケーションである．mother上のウィンドウマネージャ相当機能は，mother上のRuby/Tkの一つのキャンバスウィジェットの上に

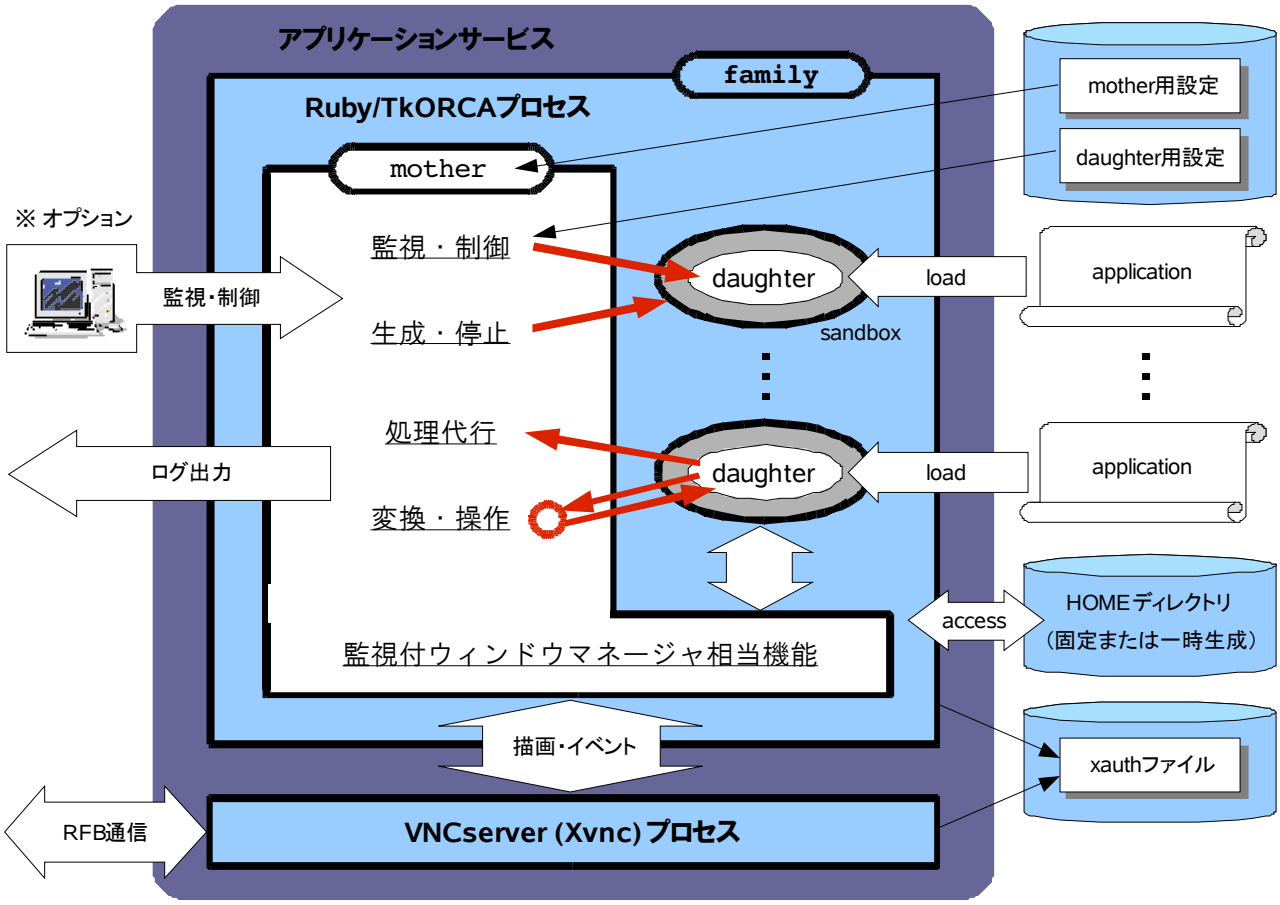

図 1: Ruby/TkORCA の構造

実装した．つまり，すべてのウィンドウフレームはキャンバスウィジェット上のアイテムとして作られており，そのアイテム上に配置されたコンテナフレームウィジェットに daughter の各トップレベルウィジェットを埋め込むことによって実現する．ウィンドウマネージャとしての機能は，ウィンドウフレームおよびその構成要素で発生したイベントに対する動作を定義することで達成している．daughter で新しいウィンドウを生成しようとした場合には，そのトップレベルウィジェットを mother のキャンバスウィジェット上に作成した新しいウィンドウフレームに埋め込む必要があるが，これは daughter におけるトップレベルウィジェット生成コマンドである toplevel コマンドを wrap して置き換え，mother に処理依頼するような形にすることで実現している．ウィンドウマネージャに関係する他のコマンドもすべて同様の手法を用いて実装している．

なお，VNC サーバを利用せずに family 部分を単独で起動することも可能である．その場合は表示がローカルのウィンドウシステム上に行われるため，そのものが公開アプリケーションのテスト環境にもなる．このことは，アプリケーションの作成から公開までの流れの中でのステップ間の段差を縮小し，スムーズにすることに寄与する．

図で「オプション」と書かれている監視・制御機構は，外部から直接に family の内部状態を確認したり操作したりするためのものであり，アプリケーションのデバッグ時にも非常に役立つ機能である．この機能の有効/無効は，設定ファイル上での定義，または family 起動時に指定するコマンドラインオプションによって切替えることができる．この機能は，Ruby をシェルのようにインタラクティブに操作することができる IRB (Interactive Ruby)を埋め込むことによって実装した．ただし，この機能を daughter から呼び出せてしまうとセキュリティホールとなるため，埋め込みの際に手を加えて mother からしか呼び出せないようにしている．mother からは daughter を完全に操作できるため，この機構により一つの family の全ての部分を監視し制御することが可能となる．

4.2.2 sandbox の機構

sandbox の機構について説明する前に触れておくべきことがあるため，まずその点について述べた後に開発した機構の説明を行う．

Ruby/TkORCA が提供している sandbox のデフォルトのセキュリティ設定は「信頼できないスクリプトを安全に実行するための設定」であり，想定される通常利用の範囲では厳し過ぎるとも言えるレベルとなっている．sandbox すなわち daughter 上で動かすアプリケーションは，通常はサービス提供者が作成または用意した信頼の置けるものであるはずである．Ruby/TkORCA の機構上，アプリケーションが利用者から受け取るのはマウスやキー入力などのイベントのみであるため，ウィジェットに入力された文字列を確認なしに評価するようなことをしていない限りは，さほど厳しいセキュリティ制約を与える必要はないと言える．必要であれば，危険と思われる部分のみに制約を与えることもできるため，アプリケーション全体を厳しく制約しなければならない必然性は乏しい．

そうしたことを承知の上でデフォルトを厳しく設定している理由は，サービス提供者に注意を促すためである．セキュリティ上の理由により，アプリケーションが daughter 上で動かなかった場合にサービス提供者が取れる手段は，(1)ソースを書き換える，(2)チェック等を mother に依頼するような wrapper を書く，(3)セキュリティ制約を緩める，の３種の内のいずれかまたはその組合せである．前述のように，信頼できるソースの場合は(3)でも構わない可能性は高いが，その判断を強いることにより，セキュリティの再確認を促すことができると考える．

続いて開発した sandbox の機構について述べる．

sandbox は Ruby の機能，Tcl/Tk の機能，Ruby/Tk の機能を組み合わせて構築した．

それぞれの sandbox (daughter) 内で動く各アプリケーションは GUI を持つものであるので，当然それぞれにその GUI を構成するウィジェット群が存在する．同時に mother の管理下に置かれてウィンドウマネージャを構成するウィジェット群も必要である．一つの faimily 上のこれらのウィジェット群は，セキュリティ上の理由により明確に分離

されている必要がある．言い替えれば，他のウィジェット群に属するウィジェットオブジェクトを偶然に入手できたとしても，そのウィジェットを操作できるようなことがあってはならない．ただし，例外的にmotherからは，各sandbox（daughter）の単方向に限っては，それぞれに属するウィジェット群を操作できる必要がある．この仕組みは，本プロジェクトの開発者の手で開発され，本プロジェクトの開発物での必要に合わせて改良を加えたMultiTkIpクラスの機能を活用することで実現させた．

MultiTkIpクラスは，一つのRubyプロセス上で複数のTcl/Tkインタープリタを利用できるようにするためのクラスである．Ruby/Tkのウィジェットオブジェクトは Tcl/Tkインタープリタに依存して存在しており，しかもオブジェクト自身はインタープリタの情報を保持していないため，ウィジェットオブジェクトを入手しただけではそのウィジェットを操作することは不可能である．また，極一部の例外を除き，Ruby/Tkにおけるメソッドは操作対象であるTcl/Tkインタープリタを指定する引数を持たない．MultiTkIpクラスのインスタンスとして生成されたものの内，どのTcl/Tkインタープリタが用いられるかは，メソッド呼び出しを行ったRubyのThreadが属しているThreadGroupに依存して決定される．MultiTkIpオブジェクトの生成時に同時に生成され関連付けられるThreadGroupは，解除不能な「enclosed」という状態に設定される．この状態に設定されたThreadGroupは，新しいThreadを生成することはできても，そのThreadを別のThreadGroupに移動させたり，逆に他のThreadGroupからのThreadを加えたりすることはできない．この機構により，各Threadのデフォルトの Tcl/Tkインタープリタは固定される．MultiTkIpオブジェクトを入手した際にそれを操作できるかどうかは，そのMultiTkIpオブジェクトに対応するTcl/Tkインタープリタがデフォルトの Tcl/Tkインタープリタから直接に生成されたスレーブインタープリタである場合に限られる．これにより，Tcl/Tkインタープリタの不正操作についても回避することができる．

開発物ではこの特性に基づき，mother部にマスターインタープリタとなるMultiTkIpオブジェクトを対応させ，各daughter部にはそれぞれにマスターインタープリタから生成されたスレーブのMultiTkIpオブジェクトを対応させることで．求められるウィジェット群の分離機構を実現した．

sandbox内のウィジェット操作に関するセキュリティ維持にはTcl/Tkのsafe Tkインタープリタ機構を活用する．このsafe Tkインタープリタは信頼できないTcl/Tkスクリプトを実行するためのものであるため，「トップレベルウィジェットは生成できない」とか「grabは実行できない」などと，非常に厳しい設定になっている．これらの制限は，「画面を無数のトップレベルウィジェットで埋めつくす」とか「grabを設定したまま解除せず，操作不能にする」などの攻撃を防ぐために理にかなったものである．しかし開発物の場合は，sandboxで動かすスクリプトは基本的には信頼できること，また，もし利用者によってそのような攻撃を喚起できたとしても，その悪影響を被るのはその利用者

自身にほぼ限られることを考えると，そのままではやや厳し過ぎる制約(例えばマルチウィンドウのアプリケーションは動かせない)と言わざるを得ない．とはいえ無条件で実行を許可するのはややリスクが大きいため，開発物ではそれらの操作を呼び出すコマンドをwrapして，motherの監視下で許可するように実装した．なお，daughter生成時の実行オプションとして，safe Tkではなく通常のスレーブインタープリタを用いることも可能であるが，その場合でも安全のためにコマンドのwrapは有効となっている．

他の処理については，Rubyの機能に基づくsandboxを構築する．sandboxはある特定の無名Moduleオブジェクトであり，実行するアプリケーション名そのモジュール内に読み込まれ，そのモジュールのコンテキストで評価する．これにより名前空間の汚染を避けてアプリケーションを実行できる．デフォルトでは現状のRubyにおいて最も厳しいセーフレベルで実行されるが，アプリケーション実行上で必要とされる処理は，前述のTcl/Tkコマンドの場合と同様に，特権レベルで動作しているmotherに処理依頼をする形にメソッド呼び出しをwrapすることができるように構築した．

Rubyの無名Moduleオブジェクトを用いたsandboxでは，モジュールのコンテキストで評価するという関係上，読み込んだスクリプトが関数型メソッドを定義していた場合にそれを関数型メソッドとしては使えなくなる．小さなスクリプトでは，わざわざクラスを定義せずに関数型メソッドでアプリケーションを作成している場合も多いため，そのままではsandboxで動かせないアプリケーションが増えてしまい，望ましくない．そこで開発物では，sandbox内でも関数型メソッドが期待どおりに機能するようにpseudo toplevelと呼ぶ機構を開発した．この機構は完璧ではなく，期待どおりには働かないケースも存在するが，ほとんどの場合でうまく機能するものとなっている．

4.2.3 開発物によるサーバの構造

Ruby/TkORCAによるサーバの概略図を図2に示す．

クライアントからの接続要求を受け付けるサービスデーモンは，接続要求ごとに4.2.1項で示したようなアプリケーションサービスを一つ生成し，サービスを提供する．これによりサーバでは，一つのサービスデーモンを中心に複数のアプリケーションサービスからなる纏まりが形成される．この纏まりを本プロジェクトではpodと呼ぶ．1台のサーバマシンで一つのpodを動かすのが一般的であるが，動作設定を変えたり提供するアプリケーションの種類を変更したりの必要により，複数のpodを動かすことも可能である．ただし現状では，援用するVNCサーバの仕様上の制限により，1台のサーバマシン(OS)で同時稼働できるアプリケーションサービスの総計は，podの数や個々のpodで稼働中のアプリケーションサービスの数とは無関係に最大で99個までとなっている．現実に1台のサーバマシンで多数のサービスを提供しようとする場合には，必要以上に高品位にしない（例えば8bitカラーでも十分であるのに24bitカラーで提供するなど）

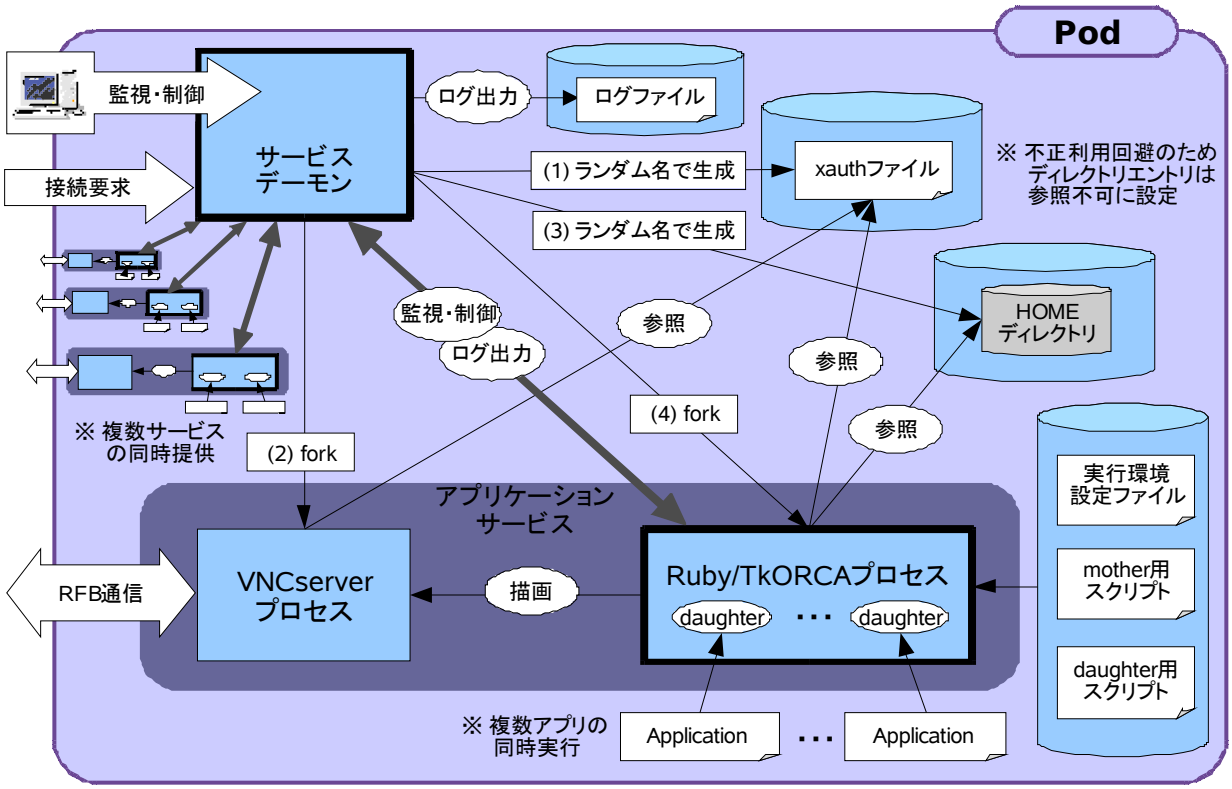

図 2: Ruby/TkORCA サーバの構成概略図

とか，QoS のために通信帯域制限を行うとかの工夫をすることが望ましいが，それらは運用上の話または OS レベルで考えるべき話であるため，開発物の範囲では特に対策などは行わない．

セキュリティ上，起動するアプリケーションサービス間での干渉を排除することは重要である．VNC は不特定多数へのサービスを想定していないため，Ruby/TkORCA サーバで対策を施さねばならない．その際，アプリケーションサービスの最大数分だけのユーザアカウントを登録し，接続ごとにその時点で未使用のアカウントを利用するという方法も考えられなくはないが，現実的な手段とは言い難い．そこで開発物では xauth ファイルによる認証機構を用い，接続ごとに新たな xauth ファイルと HOME ディレクトリとを十分な長さのランダムな名前で生成する方法を選択した．それらを生成するディレクトリのディレクトリエントリを参照不可とすることで，万一の場合でも現実的な時間ではファイルを見出せないようにした．

起動したアプリケーションサービスとサービスデーモンとの間はパイプによって接続される．このパイプを通して，アプリケーションサービスからはログメッセージが送られ，逆にサービスデーモンからはアプリケーションをコントロールするための命令を送り込むことができる．このコントロールには，4.2.1 項で述べた監視・制御機構をそのまま用いているため，アプリケーションサービスをほとんど完全に支配することが可能

である．

アプリケーションサービスのコントロールは，ログ出力の内容などの何らかの要因に基づいて行う他に，サーバ管理者がサービスデーモンに接続して直接に行うこともできる．従ってサーバ管理者は，サービスデーモンの起動や停止だけではなく，サービス全般に対して非常に強い権限を持つ．サーバ管理者としての認証は，ローカルホストからの接続に限る形で一般的なパスワード認証によって行うように実装している．

4.3 開発物の動作環境

本成果報告書作成時点での公式サポートOSはUnix系OSとする（開発時の主たるテストOSはLinux）．ただし，公式サポート外でも次の必要物が揃えられる環境（Cygwin, MacOS X等）であれば動作する可能性がある．

開発物を利用するために必要とされるソフトウェアは下記の通りである．なお，開発物はpure Ruby/Tkで作られているので，次の必要物が揃っていれば，インストールにあたりコンパイル等の必要はない（環境に合わせて若干の設定ファイルの書き換えが必要になる可能性はある）．

a) Xvnc ： inetd対応(標準入出力をRFBプロトコル通信用に用いることができる)ものであること．通信量の点からはTight VNCを推奨する．
Xvnc（またはその同等品）は何種類か存在するが，ローカルシステムで稼働中のデスクトップとは別の画面を持つことができ，Xauthority認証機構が利用可能なものであれば，いずれでも構わない．

b) Tcl/Tk ： (a)のXvncに表示できるものであること．現在のサポート状況から見て，Tcl/Tk 8.4系を推奨する．

c) Ruby ： 1.8.5以降，または1.8.5以降のRubyに付属のRuby/Tkをバックポートしたもの（1.8.5-preview3以降でも可）．Ruby/Tk（の構成要素であるtcltklib）が(b)のTcl/Tkライブラリを用いることができるように適切にコンパイルされている必要がある．

d) その他 ： 公開するアプリケーションで必要とされるライブラリ類が適切にインストールされていること．

4.4 開発物の利用方法と各部の解説

4.4.1 開発物を用いたネットワークGUIアプリ作成の流れ

Ruby/TkORCAを用いてネットワークGUIアプリを作成する場合の通常の流れは

(1) ローカルのウィンドウシステムで動くRuby/Tkスクリプトを用意する
(2) テスト環境を用いてローカルのウィンドウシステム上で実行テストを行う
(3) サーバスクリプトでネットアプリ化する

である．特別に詳細な要求をしないのであればこの流れには

tkorca.rb, tkorca_server.rb, tkorca_daemon.rb

という3個のスクリプトを用いるだけで良い．一般的な設定バリエーションについては，これらのスクリプトのオプションだけで十分に対応可能である．

4.4.2 tkorca.rb

Ruby/TkORCAの中心となるスクリプトである．図1のfamilyに相当し，ネットアプリを公開する際のRuby/TkORCAプロセスになる．

同時に，単独で動かした際にはローカルのウィンドウシステムに表示を行うため，ネットワークGUIアプリ作成の流れにおける「ローカルウィンドウシステム上でのテスト環境」としても用いられる．

tkorca.rbスクリプトのコマンドラインは次の形式である．

tkorca.rb [...*options*...] [--] [...*applications*...]

“--”はアプリケーションが“-”で始まらない場合は省略できる．
オプションの一部を次に示す(他のオプションについてはtkorca/opt_parse.rbを参照)．

-D, --trust_mode ： safe Tkインタープリタを用いない
-B, --broadband ： 通信帯域が広い場合の動作モードを用いる
-I, --interactive ： IRB相当のインタラクティブ操作を有効にする
-x, --disable_xim ： X input methodによる入力を無効にする
-w *size*, --width *size* ： 画面幅をsizeに設定(デフォルトは800)
-h *size*, --height *size* ： 画面高さをsizeに設定(デフォルトは600)
-W *size*, --vroot_width *size* ： 仮想画面幅をsizeに設定(デフォルトは1600)
-H *size*, --vroot_height *size* ： 仮想画面高さをsizeに設定(デフォルトは1200)
-T *sec*, --timeout *sec* ： 連続利用可能時間をsec秒に制限
-m *file*, --mother_rcfile *file* ： 起動時にmotherで実行するファイルを指定

これらのオプションの内のいくつかのデフォルト値は，tkorca.rb スクリプトと同じディレクトリに置かれた tkorca_base_defs ファイルの中で定数定義として指定することができる．インストールしたマシンの環境でのデフォルト値を特に設定したければ，そのファイルを編集すればよい．
アプリケーション引数は次の形式を取り，指定ごとに一つの daughter が起動する．

app_path[,[*trusted_ip*][,[*safe_level*][,[*use_console*][,[*init_script*]]]]]

各要素の意味は次の通りである．

app_path　：起動するアプリケーションのパス．もしフルパスでなければ，インストールされた Ruby/TkORCA の scripts ディレクトリからの相対パスとみなされる．scripts ディレクトリの場所は，設定ファイルでの定義によって変更することもできる．

trusted_ip　：“true”または“false”．true の場合は safe Tk インタープリタを使わない．デフォルトは false.

safe_level　：アプリケーションを実行する sandbox 内での Ruby のセーフレベル値．デフォルトは 4（最も制約が厳しい）.

use_console：“true”または“false”．標準入出力を繋ぐ疑似コンソールを利用するかどうかを指定する．ただし現在のバージョンでは，出力は可能だが入力としてはまだ不完全である．デフォルトは false.

init_script：アプリケーション起動時に，初期化処理のように最初に実行されるスクリプトの指定．mother の権限で評価される．このオプションが“{ ... }”というように波括弧で囲まれている場合は，それがスクリプトを直接記述したものとみなし，そうでない場合はスクリプトファイルのパスを指定したものとみなす．

ただし，引数として与えたアプリケーションのそれぞれが実際に起動されるかどうかは，scripts ディレクトリの application_selector.rb によって左右される．引数よりもこのスクリプトの方に優先権を与えているのは，アプリケーション引数の解釈の仕方に柔軟性を与えるための仕様である．サンプルとして含まれているものは，アプリケーション引数が与えられればそれを優先し，与えられなければデモスクリプト候補を提示

して選択してもらうという動作になっている．もしapplication_selector.rbが存在しなくてもエラーにはならず，アプリケーション引数をそのまま優先して用いる．

motherは稼働中のdaughterのリストを管理しており，すべてのdaughterが終了して管理下のdaughterの数が0になったときにはmotherも終了する．

tkorca.rbの起動時の初期化ステップはtkorca/initialize.rbで規定されている．初期化は，処理順序の重要性からいくつかのステージに分かれる．各ステージごとに以下のように決められたスクリプトを読み込むようになっているので，環境に合わせた処理の追加や変更を行いたい場合は指定されたスクリプトを書き換えるようにする．

(1)STAGE-1 ： tkorca/setup/stage1.rb
　Ruby/Tkを読み込む前に設定しておくパラメータを定義する．
(2)Input Methodの設定
　定数TkORCA::USE_INPUT_METHODに値に基づいてtkorca/input_methodディレクトリのスクリプトを読み込んでInput Methodを起動する．
(3)Ruby/Tkの読み込み
(4)STAGE-2 ： tkorca/setup/stage2.rb
　イベントループを起動する前に済ませておかねばならない処理を実行する．
(5)イベントループの開始
(6)STAGE-3 ： tkorca/setup/stage3.rb
　イベントループ開始後に最初に済ませておくべき処理を実行する．
(7)Ruby/TkORCAに必要なライブラリの読み込み ： tkorca/setup/requrie_libs.rb
(8)一部のグローバル変数の排除 ： tkorca/setup/rm_global_consts.rb

sandboxの構築に必要な置き換え処理は，上記(7)で読み込むライブラリの中で定義される．ライブラリ読み込みの際に行うのは置き換え処理の登録だけであり，daughterを生成する際に順次呼び出してdaughterのsandbox化を完了させる．

初期化ステップを完了したならば，motherオブジェクトとそのmotherが管理するwall（ウィンドウ配置の土台．画面に相当する）を生成する．wallはRuby/Tkのキャンバスウィジェットで作られているため，motherはその上に自由に描画したりウィジェットを置いたりすることができる（当然，daughterにはその権限はない）．コマンドラインオプションで与えることができるmother_rcfileが読み込まれ評価されるのは，このwallの生成が完了した直後である．これにより，アプリケーション起動の前にwallその他に対して思うままに初期化処理を施すことができる．

4.4.3　tkorca_server.rb

外部からの接続要求に対し，tkorca.rbを中心としたアプリケーションサービスを起

動してサービスを提供するためのサーバスクリプトである．図2において「サービスデーモン」と書かれた部分に対応する．tkorca_server.rbのコマンドラインは次の形式を取る．

tkorca_server.rb [...*options*...] [--] [...*tkorca_options*...]

“*tkorca options*”には，起動するtkorca.rbに渡すコマンドラインオプションをそのまま書く．もし“*tkorca options*”がアプリケーション引数のみでtkorca.rbのオプション(“-”で始まるもの)を全く含まないのであれば“--”は省略することができる．
オプションの一部を次に示す(他のオプションについてはserver/opt_parse.rbを参照)．

-L, --reverse_lookup ： 接続相手のIPアドレスからの逆引きを行なう
(接続相手によっては起動が遅くなる可能性がある)．
-p *port*, --port *port* ： サーバの接続受け付けポートをportに設定する．
(デフォルトは5940)
-c *port*, --control_port *port* ： 制御用ポートをportに設定する．
(デフォルトは5941)
-C *path*, --control_passwd *path* ： 制御用ポートへの接続認証用の
パスワードファイルの指定．
-s *file*, --serverrc *file* ： 起動時にサーバで実行するファイルの指定．
-r *dir*, --app_root_dir *dir* ： サービス起動時に*dir*へのchrootを試みる．
-u *user*, --user *user* ： サービスプロセスのオーナーを*user*に
設定するように試みる．
-g *group*, --group *group* ： サービスプロセスのグループを*group*に
設定するように試みる．
-t *dir*, --tmp_dir *dir* ： xauthファイルや一時的なHOMEディレクトリを
生成するためのディレクトリを*dir*の下に作る．
-b *path*, --ruby_bin *path* ： Rubyバイナリのフルパスの指定．
-a *path*, --xauth_path *path* ： xauthコマンドのフルパスの指定．
-x *path*, --xvnc_path *path* ： Xvncコマンドのフルパスの指定．

これらのオプションの内のいくつかのデフォルト値は，tkorca_server.rbと同じディレクトリに置かれたserver_base_defsファイルの中で定数定義として指定することができる．サーバとなるマシンの環境でのデフォルト値を特に指定したければ，このファイルを編集すればよい．

起動したtkorca_server.rbは，コマンドラインオプション等のサーバオプションの

設定を完了し，ログファイルを開いた後，TkORCA_Accepter と TkORCA_Controller という二つのモジュールに start メッセージを送る（それぞれ server/tkorca_accepter.rb, server/tkorca_controller.rb で定義）.

TkORCA_Accepter は，外部からの接続要求を待ち受けるモジュールである.

接続要求を受け付けたならば，まず xauth ファイルを置くディレクトリでロックファイルを排他ロックし，複数のサービス起動が同時に進行しないようにする．これは，Xvnc のディスプレイ番号を決定して立ち上げるまでの間に別のプロセスが同じ番号を使用してしまわないようにするために必要な措置である．したがって，もし 1 台のマシンで複数の Ruby/TkORCA サーバを起動したいのであれば，xauth ファイルを置くディレクトリは共通にしておかねばならない．同様に，新しい X のディスプレイを開く(ディスプレイ番号を消費する)ようなプログラムを動かすことも避ける必要がある．でなければディスプレイ番号の整合が取れず，アプリケーションサービスの起動に失敗する可能性がある.

ロックを獲得した後，Xvnc サーバが行うのと同じ手順で未使用のディスプレイ番号を検索し，そのディスプレイ番号用にランダムなファイル名(現状はプロセス番号と 16 進数 16 桁の乱数に基づいて作成)で xauth ファイルを作成する．xauth ファイルを生成するディレクトリのエントリを読み出し不可とすることで xauth ファイルの不正利用を難しくして，稼働中のアプリケーション間の不正な干渉を避ける．xauth ファイルの生成に成功したならば，そのファイルを認証用に指定して Xvnc プロセスを起動する.

ソケット利用状況のチェックにより Xvnc の起動を確認したならば，TkORCA_Accepter はロックを開放した後，Ruby/TkORCA プロセス(tkorca.rb)の起動を試みる．xauth ファイルの場合と同様に不正な干渉を避けるため，ディレクトリエントリが読み出し不可のディレクトリにランダムな名前でディレクトリを作成し，それを HOME ディレクトリとして Ruby/TkORCA プロセスを起動する．このとき，サービスデーモンと Ruby/TkORCA プロセスとの間には複数のパイプを確立しておき，ログ出力および監視・制御のために活用する．こうしてアプリケーションサービスの起動を完了したならば，TkORCA_Acceper は次の接続待ち受けに入る.

TkORCA_Controller は，管理者による監視・制御要求を受け付けるモジュールである.

TkORCA_Controller の処理は単純であり，telnet などでコントロールポートに接続があると，ログイン認証を行った後にコマンド受け付けに入る．TkORCA_Controller 自身が処理するコマンドの数は多くない．connect コマンドが入力されてアプリケーションサービスへの介入が求められたときには，サービスデーモンと Ruby/TkORCA プロセスとの間のパイプを用いて命令や結果を中継しているだけである.

4.4.4　tkorca_daemon.rb

このスクリプトは tkorca_server.rb をデーモン化するための処置を施しているスク

リプトであり，それ自体は大したことはしていない．ただ，このスクリプト内で Ruby のバイナリを検索し決定するようにすることで，tkorca_server.rb と tkorca.rb とが異なる Ruby バイナリの下で動いてしまわないようにしている．

4.4.5 管理ツール

サーバの管理を手助けするツール類をいくつか提供している．

(1) tkorca_logrotate.rb
Ruby/TkORCA サーバが出力するログファイルの名前を変更した後，サーバに対して Hang-Up シグナルを送ることで，新しいログファイルに切替える．

(2) tkorca_shutdown_server.rb
稼働中の Ruby/TkORCA サーバを停止させる．

(3) server/controller_passwd.rb
Ruby/TkORCA サーバのコントロールポートに接続する際の認証用パスワードを生成・管理する．

(4) 外部からの接続受け付け
「ツール」とは違うが，Ruby/TkORCA サーバはサーバ管理のための外部からの接続を受け付ける機能を持つ．この機能は telnet でサーバのコントロールポート（デフォルトではサーバの接続受け付けポート＋１番のポート）で接続することで利用できる．接続の際の認証は登録された ID とパスワードとで行われる．パスワード管理には上記の controller_passwd.rb を用いる．
利用可能なコマンドは“?”を入力することで表示できる．例えば list コマンドを入力すれば現在稼働中のサービスがどの IP アドレスからの接続に対して提供されているかの一覧を見ることができるし，connect コマンドを用いれば特定の稼働中のサービスに接続して（tkorca.rb のインタラクティブ操作と同じに）そのサービスやアプリケーションをコントロールすることができる．

4.5 監視・制御用 API

以下は，Ruby/TkORCA の利用者が独自にメソッド等の wrapper を書きたい場合のために，Ruby/TkORCA のフレームワークの一部として用意したメソッド群である．

4.5.1 判別用ユーティリティメソッド

- TkORCA.is_mother?

現在の実行が mother 上であるかを調べる.
mother なら true, daughter なら false を返す.

- TkORCA.security_check(*safe_level* = 4)
  セキュリティチェックを行う
  現在の実行が daughter であるか, mother であっても現在のセーフレベルが
  引数で与えられた値以上であれば例外を発生する.

4.5.2 ログ処理メソッド

- TkORCA::LOG.log(**string*)
  引数として渡された文字列を１行でログとして出力する.

- TkORCA_LOGGER.logfilter=(*proc*)
  Ruby/TkORCA サーバ専用のメソッド.
  ログをファイルに書き出すかどうかのフィルタリングを行う手続きを登録する.
  ログが送られてきたときに, そのログ文字列を引数として *proc* が呼ばれる.
  proc が true を返せばログファイルへ書き出すが, false を返した場合は出力
  せずに破棄する.
  proc で処理する内容に特に制限があるわけではないため, ログメッセージをト
  リガーとして起動される hook の登録としても用いることができる.

4.5.3 関数型メソッドの wrap

- TkORCA::MethodWrapper.wrap_function(*method*, *body*)
  正当性評価等を行った上で呼び出すようにメソッドを置き換える.
  - 指定したメソッドが呼ばれたとき, 代りに *body* を実行する.
  - *body* には, 本来のメソッドオブジェクト(*obj_m*), 呼ばれたときの引数並
    び(**args*)が渡される(*obj_m*.call(**args*)で本来のメソッドを呼べる).
  - *orig_m*.call は, 置き換え時の mother の権限で実行される. したがって,
    制限付与と特別許可とのいずれのパターンも実装できる.

- TkORCA::MethodWrapper.deny_daughter_to_function(**methods*)
  指定したメソッドを daughter が呼ぶことを単純に禁止する.
  違反して呼ぼうとした場合はセキュリティエラーの例外を発生するようになる.

4.5.4 クラス/モジュール/特異メソッドの wrap

- TkORCA::MethodWrapper.wrap_singleton_method(*obj*, *m*, *body*)

- TkORCA::MethodWrapper.wrap_private_singleton_method(*obj*, *m*, *body*)
- TkORCA::MethodWrapper.wrap_protected_singleton_method(*obj*, *m*, *body*)
  正当性評価を行った上で呼び出しを行うようにメソッドを置き換える．
  それぞれpublic, private, protectedメソッド用．

- TkORCA::MethodWrapper.deny_daughter_to_singleton_method(*obj*, **methods*)
- TkORCA::MethodWrapper.deny_daughter_to_private_singleton_method(*o*, **m*)
- TkORCA::MethodWrapper.deny_daughter_to_protected_singleton_method(*o*, **m*)
  指定したメソッドをdaughterが呼ぶことを単純に禁止する．

4.5.5 インスタンスメソッドのwrap

- TkORCA::MethodWrapper.wrap_instance_method(*class*, *m*, *body*)
- TkORCA::MethodWrapper.wrap_private_instance_method(*class*, *m*, *body*)
- TkORCA::MethodWrapper.wrap_protected_instance_method(*class*, *m*, *body*)
  正当性評価を行った上で呼び出しを行うようにメソッドを置き換える．
  それぞれpublic, private, protectedメソッド用．

- TkORCA::MethodWrapper.deny_daughter_to_instance_method(*class*, **methods*)
- TkORCA::MethodWrapper.deny_daughter_to_private_instance_method(*c*, **m*)
- TkORCA::MethodWrapper.deny_daughter_to_protected_instance_method(*c*, **m*)
  指定したメソッドをdaughterが呼ぶことを単純に禁止する．

4.5.6 Tcl/Tk上のコマンドのwrap

- TkORCA::TclCmdWrapper.wrap_tcl_command(*ip*, *cmd*, *body*)
  正当性評価を行った上で呼び出しを行うようにメソッドを置き換える．
  - Tkインタープリタ *ip* 上で指定したコマンド *cmd* が呼ばれたとき，代りに *body* を実行する．
  - *body* には，コマンドを呼び出したTkインタープリタ，コマンド名文字列，呼ばれたときの引数並びが渡される．
  - *body* は，定義時のmotherの権限でmother上で実行される．つまり，motherがdaughterからの依頼を受けて実行する形となる．
  - *ip* の本来のRubyのセーフレベルは *ip*.safe_levelで得ることができる．

- TkORCA::TclCmdWrapper.wrap_tcl_command_transaction(*ip*, *cmd*, *script*)
  一部の特別なコマンドにおいて，置き換えをどうしてもTcl/Tk上だけで行わねばならない場合に用いるメソッド．

例えば，置き換えたコマンドを呼び出すことによってイベント処理のシーケンスが途切れてしまい，正しく動作しなくなってしまう場合に用いる．

- TkORCA::TclCmdWrapper.convert_tcl_command_args(*ip*, *cmd*, *body*)
  TkORCA::TclCmdWrapper.wrap_tcl_command_transaction が必要なケースと同様の状況ではあるが，コマンドの引数をチェックして置き換えるだけでよい場合用いるメソッド．

5. 開発成果の特徴

一言で言えば「どこでも GUI！（GUI, Anyware!)」がコンセプトである．

開発物では，Flash 等の独自フレームワークや Ajax などの他手法と異なり，ローカルのウィンドウシステムでの GUI アプリケーションをそのままにネットワークアプリケーション化することができる．また，クライアントへの要求性能が低いため，PDA や携帯電話クラスの性能でもクライアントになることが可能である．また，他手法と異なり，プログラム片もデータもクライアントには送らないため，情報漏洩リスクを低く抑えることができる．

GUI アプリケーションを他のマシンから利用する手段としては，X window system で使われる X プロトコルも存在するが，X プロトコルを用いるようにした場合，サービスを受けるクライアントの制約が大きくなってしまうこと，初期化にかなり多くの通信量を必要とするために LAN クラスの速度でなければ利用開始までの待ち時間が長くなり過ぎること，悪意あるアプリケーションサーバからの攻撃に対して脆弱であること，といった点で難があると考える．

開発したフレームワークに基づく場合，他手法のように，開発者に同手法専用の特殊な技能を要求したりはしない．ソース変更はほとんど必要とせず，ローカルのウィンドウシステムでの GUI アプリケーションの開発とネットワークアプリケーションの開発とをシームレスにすることができる．これにより，開発から公開までに要する作業コストも非常に低いものとなる．

これらの点について，キーワード的に開発物の特長を示すと次のようになる．

- インタラクティビティ
  - ローカル GUI と同等の操作性を獲得可能

- ポータビリティ
  - Ruby/Tk ベースであり，多くの OS でソース互換
  - 単一のアプリケーションソースでローカルからネットワークまで対応可能

- クライアントはPCからPDAや携帯クラスまで(何らかのVNCviewer相当品の稼働が条件)

- スピーディ
  - 開発や公開における作業コストが低い
  - 特殊なクライアントを必要としないことなどにより，改良要求等への即応性が高い

- スケーラビリティ
  - トイプログラムから大規模アプリまで
  - １台のみへの提供から，巨大バックボーンを持つ大規模サービスまで

本プロジェクトの開発物によってアプリケーションを提供する場合，他の手法よりもはるかに高度なレベルでの公開アプリケーションの監視・制御を行うことができる．起動・停止やログ監視のような単純な操作だけではなく，稼働中の公開アプリケーション内部に立ち入っての操作すら可能である．公開アプリケーションの動作状況とは無関係にサーバ側で能動的に行った操作の効果を即座に反映させることもできる．

制御機構についての特長をキーワード的に並べると次のようになる．

- マスター・スレーブコントロール
  - mother（マスター)からdaughter（スレーブ)への単方向の支配関係に基づき，daughter上で稼働しているアプリケーションをmotherによって支配的に制御することができる

- パッシブコントロール
  - daughterから出力されるログ内容やwrapがなされている特定のコマンドの呼び出しに対して受動的に制御機能を起動できる

- アクティブコントロール
  - アプリケーションあるいはdaughterの動作状況とは無関係に，サーバ側からサービスプロセスに，さらにはアプリケーション内部に直接に干渉することができる

- イミディエイトコントロール
  - 利用者によるクリックなどの操作を待つ必要なく，制御内容の反映やメッセージ表示などを即座に適用させることができる

6. 今後の課題，展望

機能面では，開発物は現状でも十分に実用可能なものになっていると自負する．

しかしながら，現時点では負荷耐性についての評価が不十分であり，大規模なサービスにどこまで耐えられるかの判断材料としての数値的な情報が不足しているため，そうしたデータの提供は一つの課題である．

利用者が新たな監視・制御の機能追加を望む際の作業をより楽にできるように，本開発ではあまり提供できなかった高レベル API の提供が求められる可能性もある．ただし高レベル API はそれを必要とするアプリケーションの性質や構造に大きく依存するため，実際の利用者からの要望が十分に集まった後にそれを集約・整理して個別に実装に取り組む方針とする．

また，現時点では非公式サポートに留まっている環境(第 7 章を参照)についても，できるだけ早期に公式サポートに格上げしたいと考えている．

今後の普及活動については，まずは国内外の Ruby ユーザへの宣伝活動を行い利用を促すと同時に公式 Web サイトを作成することを目指す．Ruby ユーザの間で実用性が十分に認められ開発物のユーザコミュニティを作りだすことができれば，非 Ruby ユーザにも利用を促し普及させる原動力となるはずである．

なお，開発物はプラグイン的に監視・制御等の機能の強化や追加が簡単にできるように設計している．現時点ではプラグインとして組み込むものを具体的に用意してはいないが，利用者が増えればこの機能が活きて有用なプラグインが開発され，普及にも貢献するものと予測する．

7. 実施計画書内容との相違点

開発物の機能実装を優先したことにより，ドキュメントの整備と性能評価とについては計画時の想定と比較して不十分と言える．また仮想コンソール機能については，出力に関しては完成しているが，入力については不完全な部分を残している．さらに，計画時には Windows および MacOS X も公式サポート環境に含める予定であったが，両 OS 環境での十分な検証が完了しなかったために現時点では非公式サポートの扱いに留まっている(現時点での公式サポートは Unix 系 OS)．

しかしながら，優先度を高くした種々の機能面では計画時の想定以上のものとして仕上っている．

実行状況監視機能については，計画時に想定していたような単なる監視・制御に止まらず，サーバ管理者が能動的に働きかけ，直接操作する機能まで導入できた．

サーバ設定ツールについても，計画時に想定していたような単純なばらばらのツール

群ではなく，公開用アプリケーション作成の流れにスムーズに結合するように開発がなされ，アプリケーション公開までの作業コスト低減に寄与するものとなった．

8. 付録（用語説明，関連 Web サイト等）

(1)用語説明

・VNC

Virturl Network Computing の略．AT&T ケンブリッジ研究所で開発された，ネットワークに接続されたコンピュータの画面を遠隔操作できるようにするためのソフトウェア．現在ではいくつかの改良版も開発され配布されている．
なお，本開発物ではデータ転送量を効率的に抑制することができる改良版の Tight VNC の利用を推奨する．

・RFB プロトコル

Remote Frame Buffer プロトコルの略．VNC で用いられている通信プロトコル．GUI にリモートアクセスするための単純なプロトコルであり，クライアント(VNC ビューア)にほとんど制約を掛けないことを重視して設計されている．そのため，非常に多様なハードウェアでクライアントが利用可能であり，現実に携帯電話の一部機種で動く VNC ビューアが配布あるいは販売されている．

・grab

ここでは，GUI アプリケーションにおいて特定のウィンドウまたはウィジェット下の範囲にマウスイベントやキーボードイベントを限定すること，またはそのための命令を指す．通常，GUI 操作の流れを適切にコントロールするために用いられる．local grab と global grab とが存在し，local は特定の GUI アプリケーションの範囲で，global はウィンドウシステム全体に対して制約を課す．そのため，ある GUI アプリケーションが global grab を設定したまま開放しなかったとするとウィンドウシステム全体が操作不能に陥る危険がある．

・xauth ファイルによる認証機構

ここでは X サーバへのアクセス許可制御をサーバとクライアントと間で共通の認証データを用いて行う機構(Xauthority)のことを指している．Xvnc も同じ認証機構を持つ．共通データは xauth コマンドによって管理できるファイルに保管され利用されるので，開発物ではこのファイルを Xvnc サーバごとに個別に生成して指定することで Xvnc 間の不正アクセスを防いでいる．

・wrap する

コマンドやメソッドを直接実行する代わりに，実行前のチェックを加えたり，より高い権限をもつ対象に処理依頼をしたりするように，そのコマンド等を置き換えてしまうことを指す．コマンドの引数等はそのままであるため，コマンド等を呼び出す側にはプログラム変更の必要はない．開発物では，処理の正当性検査や処理依頼だけではなく，単に特定のコマンド呼び出しのログを取りたいという場合にも用いている．

・関数型メソッド

Ruby において，プログラムの任意の位置でレシーバを指定することなく呼び出すことができるタイプのメソッド．見掛け上，関数のように見え，そのように用いることができる点からこう呼ばれる．わざわざクラスを作らなくとも気軽に定義して使えるため，小物のプログラムをお手軽に作成する際などによく用いられる．

・pseudo toplevel

Ruby において，あるモジュールのコンテキストでスクリプトを評価した場合，そのスクリプト内で定義していた関数型メソッドはそのモジュールの特異メソッドとなり，関数型メソッドではなくなってしまう．つまり，モジュールを sandbox として使った場合，sandbox 内で実行するスクリプトの一部でメソッド呼び出しができなくなる可能性がある．この点は sandbox の性質としてはやや都合が悪いため，メソッド検索階層のトップレベルまで検索してもメソッドが見付からなかった場合に sandbox となっているモジュールを擬似的なトップレベルとして検索するようにしたのが pseudo toplevel である．開発物では，sandbox ごとに擬似的なトップレベルが適切に選択されるように設計している．pseudo toplevel の仕組み上，スクリプトにおいて既存の関数型メソッドを置き換えるように定義していた場合には，置き換えた方が呼ばれないケースが存在する点は注意を要する．

(2)関連 Web サイト

公式 Web サイトの整備は今後の課題であり，本成果報告時点のものに関するサイトは存在しないが，開発初期のコンセプトテストバージョンについては，本報告書執筆時点において次のサイトで触れることができる．

http://131.206.154.81/

(3)外部紹介実績

・発表等による紹介

- ✗ LinuxUsers 九州　2005/11/19（未踏採択決定直前）

- ✗ Rubyist 九州　　2005/12/10 および 2006/01/21
- ✗ Ruby 関西　　2006/01/28
- ✗ OSC2006　　2006/03/17ー18
- ✗ ESPer2006　　2006/05/20
- ✗ 日本 Ruby カンファレンス 2006（2006/06/10ー11）の懇親会の会場にて宣伝

・メーリングリスト上での紹介

- ✗ 未踏応募前　2005/06 頃（ruby-talk ML, ruby-list ML）
  最初のコンセプト紹介
- ✗ 未踏採択前　2005/10 頃（ruby-talk ML, ruby-list ML）
  公開デモを改良して紹介
- ✗ “is GUI a weak point?”という話題に絡めて紹介（ruby-talk ML:2006/04/06）
- ✗ エイプリルフールのジョークに絡めて紹介（2006/04/01）
  ジョークのタイトル：“[ANN] Project: Ruby/TkORCA on Rails”
- ✗ 成果報告会の案内として紹介（ruby-list ML:2006/08/08）