2005－OS －98（13）
2005／2／23

異種計算機間でのモバイルコードのための資源予約機構の構築

守分　滋[†]　滝沢　允[†]
間　博人[†]　徳田英幸[†,††]

近年の情報科学の進歩により，様々な計算機やセンサが遍在するユビキタスコンピューティング環境が実現しつつある．このような環境では，ユーザに最も近い計算機へサービスをローミングしたり，空間の状態に応じてサービスをローミングできる．連続メディアデータを扱うサービスがローミングする際，CPU やメモリ，ネットワーク帯域などの計算機資源がローミング先で確保する必要がある．そこで本論文ではサービスに対し計算機資源の保証を行うサービスローミングフレームワーク AROME(A Realtime OS-based Mobile Environment) を設計・実装した．

AROME は，サービスローミングフレームワーク「Sarari」を拡張し資源管理機能を追加した TaRaRi と資源保証を行うリアルタイム OS である Linux/RK，そして多様な情報機器が提供する資源とサービスが消費する資源を動的にモデル化する Mona によって構成される．

プロトタイプの評価の結果，資源管理を伴ったサービスローミングでも，オブジェクトの送受信時間に対して充分小さいオーバーヘッドに収まった．

Resource Management Mechanism for Mobile Code on Heterogenious Hosts

Shigeru Moriwake [†], Makoto Takizawa [†], Hiroto Aida [†]
and Hideyuki Tokuda [†,††]

The continuing progress in computing power and network bandwidth is leading to the realization of the Ubiquitous Computing Environment. In this environment, the services can move to the nearest computer to a user or rearrange itself among computers. Roaming servicies handling continuous media need guarantees for resource such as CPU, memory and network bandwidth for real time operation. In this research, We propose the service-roaming framework with resource management mechanism. The *AROME* framework consists of three mechanisms. One is application migration mechanism named *TaRaRi*, which is based on extention of existing service roaming framework "*Sarari*". Another is a Linux-based realtime OS *Linux/RK*, which provides resource guarantee. Third is resource monitoring and modelling daemon *Mona*.

The evaluation of the prototype implementation shows overhead caused by addition of resource management mechanism is negligible compared to time taken to migrate services.

1. はじめに

情報技術が発達したユビキタスコンピューティング環境では，高度な計算能力を持つ様々な機器やセンサがユーザの身の回りに遍在し，相互にネットワークで接続される．この環境を用い，アプリケーションからユーザへ提供される機能をサービスと捉え，複数の計算機を協調させてユーザが利用するサービスを別の計算機へ移送するサービスローミングの研究が行われている [1)~3)]．

サービスローミングが実現されている環境では，サービスがユーザの身の回りの計算機に移送され，ユーザは計算機の場所を意識することなく移送されたサービスを継続して受けられる．またこの環境では，ユーザは計算機を持ち歩く必要がなく，移動先の計算機にサービスが設置されているかということを気にする必要もない．

サービスによっては，CPU の処理時間やネットワークの帯域を大量に消費したり，マイクやスピーカ等の専用のデバイスを必要とする．そのようなサービスを計算機間で移送する際，移送先の計算機の CPU やデバイス等の資源が不足すると，ユーザは満足にサービスを受けられない．そのため，こういったサービスを移送する際には資源の確保を事前に行うべきである..

本論文の構成として，まず研究背景であるサービスローミングの概要について述べ，資源管理の問題を指摘し，それを解決する手法を提案する．次に，提案した手法を用いる関連研究を述べ，資源管理を行うサービスローミングフレームワーク AROME の設計，実装について説明する．最後に，AROME の評価を行う．

2. サービスローミングにおける計算資源保証

本節ではまず，サービスローミングにおける資源保証の必要性について述べる．次に，本研究で実現する資源保証を伴ったサービスローミングフレームワークの機能要件について述べる．

ユビキタスコンピューティング環境では，ユーザが環境内を移動することにより，ユーザの周辺に存在する機器は変化すると考えられる．このような環境では，ユーザの周辺に存在する機器へサービスを移送するサービスローミングにより，ユーザへ計算機透過的なサービス提供が可能となる．

† 慶應義塾大学大学院 政策・メディア研究科
Graduate School of Media and Governance, Keio University
†† 慶應義塾大学 環境情報学部
Faculty of Environmental Information, Keio University
本研究は文部科学省科学技術振興調整費「人間支援のための分散リアルタイムネットワーク基盤技術の研究」の下に行われています

2.1 サービスが要求する計算機資源

サービスを実現するためにはそのサービスを実現できるだけの計算機資源が必要となる．ここでの計算機資源とは，計算処理を進めるための CPU 時間や，データを保持するためのメモリ，そしてネットワークを含めた入出力機器へのデータ帯域である．サービスには様々な種類が存在し，それらのサービスが必要とする計算機資源も異なる．単純な計算処理だけのサービスに加え，近年では大量のデータを処理する音声・映像ストリーミング等の連続メディアサービスが増大し，ローミングの対象となる．

必要とする資源を満足に得ることができなければ，動画再生中にフレーム落ちが頻繁に生じたり，再生される音楽が途切れ途切れになる可能性がある．移送後，このようにサービスの品質が著しく低下する場合，本質的にサービスを移送できたとは言えない．このサービスの品質について述べるために，次項でリアルタイム性の概念について述べる．

2.1.1 リアルタイム性

リアルタイム性とは，一定時間内にある処理を完了しなければならないという制約を満たすことを保証することである．この制約を，時間制約と呼ぶ．リアルタイム性をもったサービスでは処理の時間的制約により，ハードリアルタイムタスクとソフトリアルタイムタスクに分けられる．ここでは図 1 の value function[4] を用いて定義する．あるタスクの時間的制約が満たされな場合に，システムにとってその効用がマイナス無限大になる時，そのタスクをハードリアルタイムタスクという．また時間的制約が満たされない場合に，その効用が穏やかに減少していく時，そのタスクをソフトリアルタイムタスクという．

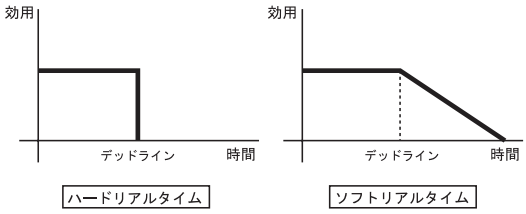


図 1 ハードリアルタイムとソフトリアルタイム

最低限の品質のサービスを保証したサービスローミングを実現するためには，このソフトリアルタイム性を考慮したサービスローミングを行う必要性がある．

2.2 機 能 要 件

ソフトリアルタイム性を実現するために，サービスに対して計算資源の割り当てが保証されたサービスローミングフレームワーク AROME(A Realtime OS-based Mobile Environment) を実現する．そのためには，以下の要件が必要となる．

- 資源の予約，管理を行うローミング基盤
- 資源管理 を行う OS
- サービスの消費する資源と情報機器の提供する資源の動的なモデル化

以下の項でそれぞれの要件について詳しく説明する．

2.2.1 資源管理機構対応ローミング基盤

ローミング基盤はユーザの携帯端末から周辺のユビキタスコンピューティング環境に存在する機器にサービスを移送する．この際に必要となる機能として次のようなものが挙げられる．

サービス移送機能 サービスを複数情報機器間で移送する機能が必要である．このことにより，ユーザは様々な特性を持つ情報機器上で多様なレベルのサービスを受けられる．

サービス使用資源量把握 移送するサービスがどの程度の資源を利用しているかを把握し，さらに移送先ホスト上でどの程度の資源が利用可能かどうかについて把握する必要がある．サービスが必要とする資源が移送先ホストで確保できる時のみ移送することで，ユーザに対して確実にサービスを提供できる．

サービスを断絶しないで移送する機能 移送時にユーザに対するサービスを断絶せず，移送先ホスト上で実行を再開する必要がある．このことによりユーザは移送を意識せず高度なサービスを受けられる．
また，移送先ホスト上で確保した資源が利用できなくなる場合も考えられるが，その場合にもユーザに対するサービスを断絶しないための機構が必要である．

2.2.2 資源保証対応 OS

第 2.1 節で述べた資源を管理するのは通常 OS である．そのため資源保証機構は OS において実装する．このように資源管理を行なう OS をリアルタイム OS と呼ぶ．AROME では，資源を確保するためにリアルタイム OS を利用する必要がある．また，このリアルタイム OS はユビキタスコンピューティング環境に存在する多様な情報機器で動作する必要がある．本研究では情報機器をサービスを提供できる計算機のことを指す．例えばディスプレイやスピーカなどである．

2.2.3 資源のモデル化

ユビキタスコンピューティング環境では様々な情報機器が遍在し，それぞれ保持する計算機資源の量と種類が違う.. また，サービスの計算機資源要求はそれぞれ異なることから，資源をモデル化する必要がある．必要となる項目としては以下が挙げられる．

I/O 資源 サービスが処理するデータの入力先と出力先のデバイスが，そのデータ量を処理可能かどうかの指標が必要となる．例えばホスト上のネットワークインタフェースの利用可能帯域以上のストリーミングデータの受信は不可能である．

計算資源/記憶資源 計算機上に搭載されている CPU の特性により，同じデータに対する処理量が変化するため，計算資源に関する統一的な指標が必要となる．同時に記憶資源に関しても考慮する必要がある．例えば，FPU が存在しない場合，必要な計算量が増大し，同時に必要記憶資源も増大する．

3. 関 連 研 究

本節では，資源保証機能を持ったサービスローミングを行なっている既存研究を取り上げ，本研究を位置付ける．

3.1 Mogul

情報機器の資源を考慮した移動コードの研究として，中澤仁らの Mogul[5] があげられる．Mogul は，アプリケーションのユーザインタフェースを移送させ，ネットワークの帯域や CPU パワーにあわせて，そのサイズを変化させる．

Mogul の特徴は，複数のホスト上を連続移送することが可能で，移動可能オブジェクトと移動不可能オブジェクトを動的に分離する点にある．これは，移動不可能な変数をトレースすることで実現する．これにより，プログラマは移動可能か不可能かを意識することなく，オブジェクトを移送できる．また，Java 言語によって記述されているため，マルチプラットフォームを実現する．

Mogul ではサービスを変更することなく，サービス実行を保証したサービス移送が可能なため，ユーザに統一的なサービスを提供することが可能である．

3.2 m-P@gent

電機通信大学の高汐一紀氏は，PDA や携帯電話のような資源の貧弱な環境でも動作するエージェントシステムとして，m-P@gent[6] を設計，実装した．

m-P@gent では，エージェントはコアモジュールと追加モジュールとで構成される．エージェント移動時，移動先の環境が持つプロファイルに応じて追加モジュールを変化させることで，エージェントを PDA や携帯電話でも動作可能にする．また，移送先のプロファイルに応じ，エージェントのモジュールをすべて移送する全移送

と，ある部分のみを移送する部分移送を使い分けられる．そのため，資源に応じて移送するモジュールを選択し，移送時のコストを最小限に抑えられる．

m-P@gent は周辺に様々な計算機が遍在する環境を想定しており，移動先の計算機上でサービスが変更されることなく完全に動作することを保証するため，モジュール変化の処理を行う必要がない．よって様々なサービスに対して適応可能となる．

3.3 NOMADS

ホスト上の資源状態を把握しつつ実行コードを移動するシステムとして，Niranjan Suri らの NOMADS システム [7] が挙げられる．

NOMADS システムは Java 互換の Virtual Machine (VM)，Aroma と移動エージェント環境の Oasis の 2 つのコンポーネントから構成される．Aroma では Java で実現されている弱いモビリティ (Weak Mobility) よりも高い移動透過性をもつ強いモビリティ (Strong Mobility) を実現し，Oasis では移動に際してやりとりされるメッセージを扱う機構や，移動のポリシーを設定する機構などの移動エージェントに必要な機構を実現している．NOMADS ではホスト上で実行している他のエージェントが利用する CPU やネットワーク資源を使いきり動作を阻害することを防ぐため，Aroma VM においてホスト上の資源利用率を監視している．この機構により，NOMADS では安全に移動コードを実行できる，複数ホスト間における資源予約は bytecodes/ms といった統一基準を設けている．

NOMADS は Java VM を対象として資源管理を行っているのに対して，本研究は Java VM を含めた OS レベルで資源管理を行っているため，他のプロセスとの関係性を考慮することができる．

3.4 RCANE

移動コードの利用可能資源を管理しているシステムとして，Paul Menage の RCANE [8] が挙げられる．

RCANE は Active Network[9] 環境において各ノード上で実行されるコードに対して，まずコンパイル時 / プログラムロード時に利用可能資源の制約を行なう．さらに，ノード上のリアルタイム OS を用いて計算機資源の QoS を行なうシステムである．RCANE は実行コードをリアルタイム性の高い処理を要求するコードとベストエフォートで処理されるコードの 2 つに分類し，リアルタイム性の高いコードを 1 つの CPU にスケジュールすることで，高いリアルタイム性を確保している．また，ベストエフォートで実行されるコードも，各コード生成時に CPU 利用率を指定することで，QoS を実現している．

RCANE が提案する計算資源の制約はコンパイル時ではなく移送時に決定されるため，動的なサービスの変化に対応した計算資源管理が可能となる．

4. AROME の設計

本節では，2 節で挙げた要件をもとに，サービスローミングにおける資源保証の基本的な機能を実現する AROME システムの設計について述べる．

4.1 AROME の全体構成

本項では，AROME の全体構成について解説する．AROME は，サービスローミングを実現するミドルウェアである TaRaRi，資源保証を行うリアルタイムオペレーティングシステム，そして情報機器とアプリケーションをモデル化するためのベンチマーク機構である Mona で構成される．本研究では，資源管理機能を持ったリアルタイムオペレーティングシステムとして Linux/RK[10] を利用する．Linux/RK は，Linux へ RK(Resource Kernel) モジュールを組み込むことで実現される．また，TaRaRi の基盤として Sarari[11] を利用し，資源予約のための拡張を行う．AROME の全体構成を図 2 に示す．

Mona は，情報機器の提供資源とサービスの消費資源を OS を通して監視し，TaRaRi からの要求にしたがって動的にモデル化する．TaRaRi はサービス移送時，Mona によって取得された資源利用状況を参考にし，サービス移送を行うかどうかを決定する．また，TaRaRi は，サービスへの資源を確保するため，OS の RK モジュールとの連携を行う．

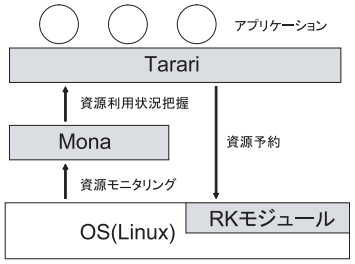


図 2 AROME の全体構成

4.2 AROME 動作概観

本項ではまず，資源確保を伴ったサービスローミング機構 AROME の動作概要を示した後，移送先ホスト上で資源不足が発生した場合など，様々な状況への対処について述べる．

4.2.1 動 作 概 要

図 3 に AROME の動作概要を示す．AROME では，Mona より資源の利用状況を取得し，その情報からアプリケーションの移送先ホストで動作している AROME へ資源保証の問い合わせを行う．ここでは移送元の計算機をホスト A，移送先の計算機をホスト B とし，図中の番号に沿って AROME の動作概要を以下に述べる．

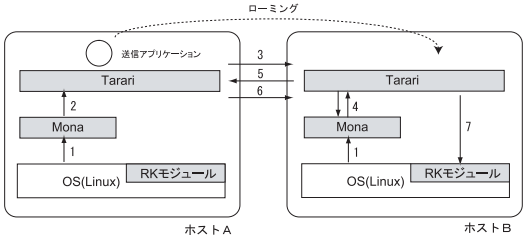


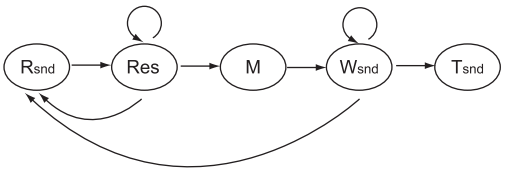
図 3 AROME の動作概要

(1) 各ホストにひとつ存在する Mona のモニタリングにより，OS の資源利用状況が把握される．
(2) サービス移送時，ホスト A の TaRaRi は Mona から資源利用状況を取得する．
(3) 移送するサービスに必要な資源が確保できるかどうかをホスト B の TaRaRi に問い合わせる．
(4) ホスト B の TaRaRi は，Mona より情報機器の資源状況を把握し，要求された資源確保が可能かどうかを把握する．
(5) ホスト B の TaRaRi は，資源確保が可能であるかどうかをホスト A の TaRaRi へ返信する．
(6) 資源確保が可能であった場合，ホスト A の TaRaRi はサービスをホスト B の TaRaRi へ移送する．
(7) ホスト B の TaRaRi は Linux/RK によって資源を確保し，受信したサービスを動作させる

4.2.2 様々な状況への対応

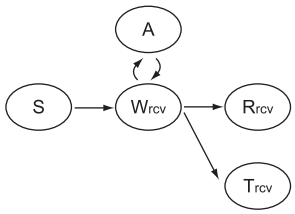
図 4 と図 5 に，それぞれ図 3 のホスト A とホスト B の TaRaRi の状態遷移を示す．これらの図を用いてまず，正常にサービスを移送する通常の動作概要を解説した後，資源不足が起きた場合による動作について示す．

通常動作 ホスト A において実行中 (R_{snd}) のサービスがホスト B に対しローミングする必要が発生した場合，ホスト A の TaRaRi はホスト B 上において必要資源が確保できるかを問い合わせる (Res)．ホスト B では，資源確保ができるかを Mona に問い合わせる．資源確保ができるのが確認できたのでホスト A に対しローミング可能と通知する．ホスト A で



Rsnd: 実行中　Res: 資源問い合わせ　M: ローミング
Wsnd: ローミング　Tsnd: 終了

図 4　ホスト A における TaRaRi の状態遷移図



S: 開始　Wrcv: 監視待ち　A: 適応
Rrcv: 実行中　Trcv: 終了

図 5　ホスト B における TaRaRi の状態遷移図

は，実行中のサービスのコピーを作り，ネットワークを利用しローミングを開始する (M)．ホスト B では受けとったサービスの実行を開始 (S) し，サービスが必要な資源が確保できているか一定期間監視 (W) する．この期間中，ホスト A は送信したサービスを監視待ち状態 (W_{snd}) のまま実行を続ける．ホスト B 上でサービスが問題なく動作していることを確認し実行中状態 (R_{rcv}) に移行した後，ホスト A に対し監視終了の通知を行ない，ホスト A 上のサービスは終了する (T_{snd})．

資源問い合わせの結果資源不足の場合　ホスト A からホスト B に対し資源の問い合わせをした (Res) 時，ホスト B 上において必要な資源が確保できなければ，ホスト B は確保不可能という通知をホスト A に対して行なう．確保不可能の通知を受けとったホスト A の TaRaRi はローミングを取り止め実行中状態 (R_{snd}) に遷移する．

ホスト B 上での監視待ち状態 (W_{rcv}) に資源不足の場合　ホスト B に対しサービスローミングが終了しても，ホスト A では瞬時にサービスを終了 (T_{snd}) させず，ホスト B から監視終了の通知が到着するまで待つ (W_{snd})．この状態中に，ホスト B 上においてサービスの必要とする資源が確保できなかった場合，ローミング先のサービスを終了させ (T_{rcv})，ホスト A 上でサービスを実行し (R_{snd})，ユーザにサービスを提供する．

ホスト A におけるローミング前適応動作 (A)　TranService[12] などのフレームレートや解像度を状況に応じて適応動作できる機構が利用可能な場合，資源の問い合わせ (Res) の結果，ホスト B より資源確保不可能の通知を受信した場合でも，資源確保要求を適応動作できる機構を用いて動的に低下させ，複数回問い合わせる (Res) ことで，適応しつつローミングできる．

ホスト B におけるローミング後適応動作 (W_{rcv})　ホスト B に対してローミング後の監視期間に資源の不足が明らかになったとしても，適応動作できる機構などを利用し必要資源を低下させ (A)，それを再度監視・確認 (W_{rcv}) することで，適応的なローミングができる．

4.3 Linux/RK

AROME にて資源保証 OS として利用する Linux/RK は Linux に対するリアルタイム拡張として実装されている．追川ら [10] によって，CMU(Carnegie Mellon University) の Linux/RK を元に，リソースセットと周期スレッドの実装が行われている．さらに，ユビキタスコンピューティング環境に存在する多様な情報機器上でソフトリアルタイム性の保証を実現するため，組み込み機器向け CPU である Strong ARM および Xscale でも動作するようにポートしたものを利用する．

4.3.1 予約 (Reserve)

予約とはプログラムに対する計算資源の割り当てである．CPU 時間予約は C, D, T の 3 つのパラメータで表現され，T はプロセスの周期を表し，C は周期 T 内における実行時間を表す．D はデッドラインを表し周期 T 以下である．実行時間 C は D 以前に完了する必要がある．

周期 T 内で C を使い果たした時点を枯渇状態，C 時間を実行していない時点を非枯渇状態と呼ぶ．また次の周期で新たに実行時間 C を得ることを補充という．枯渇時点から周期 T の終端までの予約機構の動作から，リソース管理モデルは以下 3 種類に分かれる．

- Hard reserves: 周期 T 内において，C を使い果たせば次の周期までスケジューリングされない．
- Firm reserves: 他の予約を使い果たしていないプロセス，または通常のプロセスが実行可能状態になければスケジューリングされる．
- Soft reserves: C をすべて消費したとしても，スケジューリングされる．

4.3.2 リソースセット (Resource Set)

リソースセットは予約の集合であり，1 つ以上の予約をグルーピングして，リソースセット単位で 1 つ以上のプロセスに関連付けされる．リソースセットは各資源の予約を抽象化したものであり，ユーザアプリケーションへのインタフェースとなる．

4.3.3 周期スレッド (Periodic thread)

RK は周期スレッドをサポートする．周期スレッドは周期 t でプロセスを実行可能状態 (ready) へ遷移する．通常 OS のプロセススケジューラである，タイムシェアリングスケジューラでは困難な周期性を実現し，動画などの連続メディアの再生に必要な周期性を保証する．

4.4 TaRaRi の設計

本研究では，資源保証機能を備えたサービスローミングミドルウェアを作成する．このサービスローミングミドルウェアを TaRaRi と呼ぶ．本研究では TaRaRi を，サービスローミングミドルウェア SaRaRi を拡張し，資源保証の能力を付け加えることで実現する．本項ではまず TaRaRi の基盤となる SaRaRi の概要を述べ，次に本研究で拡張する部分の設計について解説する．

4.4.1 SaRaRi 概要

SaRaRi[11] は小泉らによるサービスローミングミドルウェアである．SaRaRi の特徴として，移送オブジェクトをコピーして，移送先ホストにて実行を開始したのを確認してから，移送元ホストのオブジェクトを終了・削除する Copy and Move モデルを用いている点が挙げられる．この特徴によって，SaRaRi はサービス断絶時間のないサービス移送を提供する．SaRaRi では，サービスをオブジェクトとして扱う．構成は**管理部**，**送受信部**，**状態監視部**の三つに分けられる．また，実行中のオブジェクトを保管する ObjectStack も存在する．なお，SaRaRi は Java 言語を用いて開発している．これは，Java を用いることによって，バイト列化されたオブジェクトのネットワーク間移送が容易に行なえるためである．

4.4.2 SaRaRi への拡張

本研究で作成する TaRaRi は，SaRaRi へ資源保証機能を追加実装することで実現する．SaRaRi へ追加実装する資源保証機能は次の 3 点である．

資源保証問い合わせ機能
移送前に，サービスの移送先ホストに対し，資源が確保できるかを問い合わせる．この機能により，サービスを移送できるかどうかの確認を行うことができる．

移送後の状態監視機能
移送後に，サービスが問題なく動作しているかを監

視する．この機能により，移送前の資源保証では予期しなかった資源不足が生じた場合，サービスを移送前のホストへ戻して動作させられる．

Linux/RK との連携機能

Linux/RK との連携を行い，TaRaRi から RK モジュールを利用可能にする．この機能により，TaRaRi で動作するサービスに対し，リアルタイム性を保証できる．

これらの機能は，Mona 及び Linux/RK と連携し動作する．次に，Mona について説明する．

4.5 Mona の設計

本項では，Mona について述べる．まず概要を述べ，次に計算機資源のモデル化手法について述べる．

4.5.1 Mona の概要

Mona(MONitorintg Agent) はサービスの移送元ホストと移送先ホストで異なる動作をする．移送元ホストでは，Mona は現在実行しているサービスが必要とする計算機資源のモデルを Linux/RK を監視して作成し，TaRaRi がそのモデルを利用して，移送先で必要資源を確保できるか移送先に確認する．また，移送先ホストでは Mona は移送元の TaRaRi から送られてきたサービスが利用するモデルを解釈し，現在の計算機資源の状況を Linux/RK を参照して把握した上で，サービスに必要な資源が確保できるかどうか判定する．Mona の構成図を図 6 に示す．

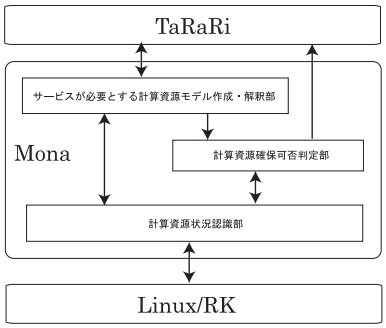


図 6 Mona の構成図

4.5.2 計算機資源のモデル化

Mona はサービスを実行するために必要な計算機資源を確保できるかどうか判定するために，サービスが必要とする計算機資源のモデルと，機器が提供できる計算機資源のモデルを作成する．Mona はこの 2 つのモデルを照らし合わせて，サービス移送の可否を TaRaRi に通知する．モデルは XML による構造化されたドキュメントとして表現される． XML を用いる利点は，構造化言語処理系の実装の容易性である．汎用 XML パーサを用いることで，字句解析，構文解析，意味解析を容易に行える．また， XML パーサは，Java，C/C++，Perl をはじめ様々な言語に実装されているため，処理系の実装も多様な言語による実行環境上で可能となる．このような理由で，XML による記述が適している．以下に，情報機器が提供可能な計算機資源のモデル記述方法と，サービスが必要とする計算機資源のモデル記述方法について述べる．

4.5.2.1 サービスが必要とする計算機資源のモデル化

Mona はサービスの実行中に資源消費を監視し，ローミング時に TaRaRi から問い合わせがあった時に計算機資源状況認識部を用いて，そのサービスを実行するために必要な計算機資源をモデル化する．作成されたモデルは図 7 のように記述される．このモデルはそのサービスが必要とする CPU やメモリ，IO デバイス等の計算機資源が全て記述されている．

それぞれのタグには計算機資源のモデル化手法がタグの属性として記述される．Mona は様々なモデル化手法を有しており，例えば CPU の処理能力の指標として浮動小数点演算能力を使用する場合，`<processor model="FP">` と記述される．

```
<service>
    <processing>処理能力</processing>
    <memory>記憶容量</memory>
    <io>
        <network>ネットワーク帯域</network>
        <disk>ディスク帯域</disk>
    </io>
</service>
```

図 7 サービスが要求する計算機資源モデル

4.5.2.2 情報機器が提供可能な計算機資源のモデル化

サービスを実行するために必要な計算機資源のモデルに即して，情報機器が提供可能な資源をモデル化する．サービスに必要な計算機資源のモデル化手法と同じ方法を用いて提供可能な性能のモデル化を行う．情報機器の資源モデルは図 8 のような XML で記述する．

```
<architecture>
    <processing>処理能力</processing>
    <memory>記憶容量</memory>
    <io>
        <network>ネットワーク帯域</network>
        <disk>ディスク帯域</disk>
    </io>
</architecture>
```

図 8 情報機器の計算機資源モデル

4.5.3 計算機資源確保可能可否判定部

Mona はサービスが必要とする計算機資源と計算機が提供できる計算機資源を照らし合わせて，その計算機がサービスを実行できる環境であるかどうかを判定する．4.5.2 で述べた 2 つのモデルを用い，比較する．比較方法として，XML で記述されたモデルの，CPU やメモリのそれぞれの計算機資源ごとに比較し，

の場合にサービスが実行可能であると判断する．この場合にサービス移送の可否を TaRaRi に通知する．

5. AROME の実装

本節では，4 節で行った設計をもとに実装を行った AROME のプロトタイプについて述べる．本論文の AROME では，計算資源として CPU 処理能力を扱った．

5.1 実 装 概 要

AROME のプロトタイプ実装では計算資源として CPU 処理能力を扱う．本論文では，この計算機資源を保証した，サービスローミングフレームワークを実装した．

5.2 TaRaRi の実装

AROME では，サービスローミングミドルウェアに資源保証能力を付加し，サービス移送時の資源保証を行う．そのため，SaRaRi を拡張し，資源保証の能力を持つサービスローミングミドルウェア TaRaRi を作成した．次項に TaRaRi に含まれる SaRaRi からの拡張部分について解説する．

5.2.1 資源保証の問い合わせ

AROME では，資源確保を考慮したローミングモデルを実現するために SaRaRi の Copy and Move モデルを拡張し，資源保証の問い合わせ機能を付加した．問い合わせ機能では，計算機の持つ CPU のクロック数および予め計測したサービスの CPU 使用率を利用し，移送先のホストでもサービスの動作を保証できるかどうかを判断する．この機能を付加するため，SaRaRi の送受信部に

対し追加実装を行った．本項では計算機の持つ CPU のクロック数およびサービスの CPU 使用率を要求計算能力と呼ぶ．以下に問い合わせ時に移送元ホストと移送先ホストで交わされるトランザクションを示す．

(1) 移送元ホストから移送先ホストへ要求計算能力を送信する．
(2) 移送先ホストにて，要求計算能力を満たせるかどうかを判断し，移送元ホストへ送信する．
(3) 要求計算能力を満たせる場合，複製したサービスを移送する．要求計算能力を満たせない場合，移送を取りやめる．

TaRaRi の送信部では，要求計算能力を移送先ホストへ問い合わせ，その結果に応じて移送を行うかを判断する．受信部では，CPU のクロック数が要求された計算能力を満たすかどうかを判断し，満たせれば true を，満たせないならば false を返信する．

5.2.2 移送後の状態監視

AROME では，移送後のエラーに対応するため，移送したオブジェクトの状態監視を行う．この状態監視機能を実現するため，SaRaRi の状態監視部を拡張した．具体的には，オブジェクトが実行状態に遷移した後も一定期間監視し，サービスが期待通り動いていることを確認した後に移送元のオブジェクトを終了するようにした．本プロトタイプではオブジェクトを 2 秒間監視し，サービスよりエラーが返されなければ SaRaRi の ObjectStack へオブジェクトを渡し，移送元ホストの TaRaRi に ack を送信する．

AROME では，SaRaRi の状態監視クラスである StateObserver クラスを拡張し，オブジェクトが実行状態になった後も 2000 ミリ秒間の監視を行うようにした．2000 ミリ秒の間にエラーが返されると，オブジェクトを ObjectStack へは格納せずに移送元 TaRaRi へエラーの発生を知らせる．

5.2.3 javax.realtime.RealtimeThread クラス

Linux/RK にて作成したリソースセットを割り当てられたスレッドを，Java サービスから利用するために javax.realtime.RealtimeThread クラスを実装した．RealtimeThread クラスは，start() 時に JNI(Java Native Interface) にてリソースセットを作成する．そして，リソースセットを関連付けたスレッドを Java 仮想マシンへアタッチして実行を開始する．スレッドアタッチとは，JNI で定義されている新たに生成した OS のスレッドに，Java のスレッドの実行を担当させる機能である．

RealtimeThread は Realtime Java 仕様 [13] にて定義されているクラスである．本来は，スレッドの変数もスワップアウトされないよう物理メモリに固定する必要があるが，今回の実装では計算機資源のうち CPU のみ扱っているため，CPU 時間のみが保障された周期的 RealtimeThread を実装した．

周期スレッドは，処理が終わると次回の周期までブロックする．Linux/RK では API の rt_wait_for_next_period() 関数を呼ぶことでこれが実現される．RealtimeThread クラスには，この関数を Java から利用するラッパ関数である waitForNextPeriod を実装した．リソースセットの作成と同様に，JNI にて Linux/RK の API にアクセスする．

5.3 Mona の実装

Mona の機能のうち，CPU 処理能力について BogoMips および CPU 使用率ベンチマークを使用したモデル化を行い，計算機資源確保可否判定機能を実装した．

5.3.1 サービスの必要とする計算機資源のモデル化

プロトタイプ実装では，CPU 処理能力の指標として，Linux の/proc/cpuinfo から取得できる BogoMips 値を使用した．BogoMips は Linux の起動時に，特定のビジーループが 1 秒回に何回実行できるかを示す．Linux があるマシンでは汎用的に利用できるため，Linux を拡張した Linux/RK でも利用できる．また，BogoMips は起動時に測定するため，並列プログラミングによる影響を受けにくい．

サービス実行時の CPU 使用率を監視・記録する．記録した使用率の中で最大の値を Bogomips で割った値にてモデル化する．

$$Usage = Bogomips \times MAX(CPU\,Usage)$$

5.3.2 情報機器が提供可能な計算機資源のモデル化

サービスが必要とする計算機資源のモデルを受けとったら，機器がその計算機資源を提供できるか判定するために，計算機資源のモデル化を行う．監視の結果から，リアルタイム性を持つプロセスが CPU を何%保持しているか把握し，モデル化を行う．

$$Resource = Bogomips - (Bogomips \times CPUamount)$$

$CPUamount$:リアルタイム性を持つプロセスに割り当てられた CPU 使用率の総和

5.3.3 計算機資源確保可能可否判定部

図 ??のサービスが必要とする計算機資源のモデルと，図 ??の計算機が提供可能な計算機資源のモデルを元に計算機資源の確保が可能か判定する．この場合，計算機が提供可能な CPU 処理能力 (P) が 3500，サービスが必要とする CPU 処理能力 (U) が 1667.0 となり，

$$P \geq U$$

を満たすため，計算機資源の確保が可能となり，TaRaRi にその結果を通知する．

6. 評　　価

本節では，AROME システムの評価を行なう．まず，実装したシステムの動作時のパフォーマンスを，Linux/RK, TaRaRi，RealtimeThread クラス，Mona の各構成要素毎に測定し，本システムが目指す資源保証を伴ったサービスローミングが実現されたか確認する．また，今後のシステム改良点を探すためのデータとして利用する．最後にシステム全体が，本研究の目的を達成したことを関連研究と比較しつつ述べる．

6.1 TaRaRi の評価

本項では SaRaRi に対し資源保証問い合わせ機能を付与したことによるオーバヘッドの測定と，Java 付属の Thread クラスと資源予約を行なうための RealtimeThread クラスの性能比較を行なう．

6.1.1 資源保証問い合わせのオーバヘッド

TaRaRi の資源保証問い合わせ機能によるオーバヘッドを測定するため，SaRaRi と TaRaRi を用いて実験用サービスのローミングを 2 つのホスト間で 10 回行なった．実験環境は，IBM Thinkpad X31 を 2 台用い，無線接続には IEEE 802.11b を利用した．測定結果を図 9 に示し，縦軸は SaRaRi と TaRaRi を項目とし，横軸は時間をミリ秒で表す．

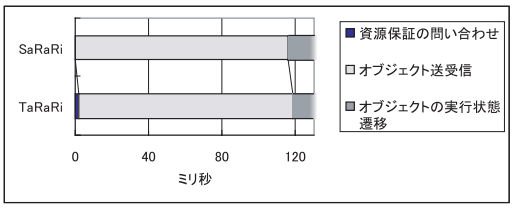


図 9　TaRaRi のオーバヘッド

測定結果より，TaRaRi の資源保証問い合わせに要する時間は，オブジェクト送受信に要する時間と比べて 1.87%となることが分かり，誤差範囲内であるといえる．また，ここで要している時間は，移動元から移動先に対する資源問い合わせにかかる時間であり，図 3 より 1 RTT (Round Trip Time) 分の時間であることが分かる．送受信されるデータは現在 CPU の Bogomips 値とローミングの可否

であることから，データ量は一定である．よって，ネットワークの途中経路の遅延のみによって左右される値である．従って，資源保証問い合わせ機能によるオーバヘッドがシステムにあたえる影響が低さが確認できた．

6.1.2 RealtimeThread クラス

RealtimeThread を通常の Thread クラスと比較し，生成，実行，及び終了に発生するオーバヘッドを測定した．

測定は IBM Thinkpad T42p にて，Thread と RealtimeThread クラスのそれぞれに対し，オブジェクト生成 (new) 時，スレッド開始 (start) にかかる時間，スレッド終了 (join) から呼びだしスレッドに戻るまでにかかる時間をそれぞれ測定した．OS には Linux/RK，Java は JDK 1.4.2 を利用した．時間はペンティアムカウンターを JNI 経由で呼びだし計測し，実験は 10 回スレッドを作成し，時間の値はその平均をとった．図 10 に表す．縦軸はそれぞれの操作にかかった時間を表し，3 つの操作を横軸に並べる．

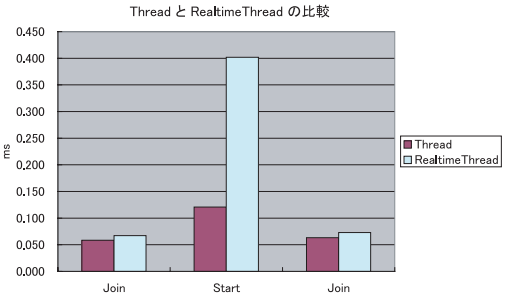


図 10 Thread クラスと RealtimeThread クラスの比較

結果より，RealtimeThread は Thread に比べて，各処理のオーバヘッドの増加が確認できる．RealtimeThread クラスが new と join を実行するのにかかる時間は RealtimeThread クラスと比べて両方とも 1.15 倍の増加に留まっているのに対し，start にかかる時間は 3.33 倍となっていることが分かる．これは，Linux/RK が提供する API のうち，リソースセットを作成する rk_resource_set_create() と rk_cpu_reserve_create() と周期スレッドを作成する rt_make_periodic() を呼びだしてリソースを確保し，そのリソースを RealtimeThread スレッドへアタッチすることによるオーバヘッドだと考えられる．

しかし，いずれのオーバヘッドも ms 以下のオーバヘッドであり，オブジェクトがネットワークを介して移動する時間が図 9 より 120ms 程度要するのに比べると，充分小さい値であり，実際の使用に耐えることが分かった．

6.2 Mona の評価

本項では，Mona で利用している Bogomips 値が異なる計算機間でやりとりされる統一的な計算量の指標として妥当かについて評価を行なう．

評価は表 1 に示した環境を利用し，各環境上で同じ mp3 ファイルを madplay を用いて演奏し行なった．周波数と Bogomips 値は/proc/cpuinfo の値を用いた．プロセスに対し割り当てられた CPU 時間に Bogomips 値をかけた値を Bogomips 量とし，各環境上で madplay を用いて同じ mp3 ファイルを演奏した時の Bogomips 量を取得した結果を図 11 に示す．図の縦軸は Bogomips 量を表し，横軸には各環境を配置した．また，Bogomips 量と比較するため，図 12 に各環境における CPU 量を示す．縦軸は CPU 量を表し，横軸には各環境を配置した．CPU 量は Bogomips 量と同様に，madplay が割り当てられた CPU 時間を CPU の周波数にかけた値とする．

図 11 より，Bogomips を指標とした場合の，各環境間から得られた Bogomips 量の標準偏差は 8.47 となった．

表 1 Mona 評価環境

機種	CPU の種類	周波数 (MHz)	Bogomips
Thinkpad T42p	PentiumM	2093.244	4181.19
Thinkpad 240X	Pentium III	497.798	990.06
Zaurus SL-C3000	Xscale-PXA270	520	415.33

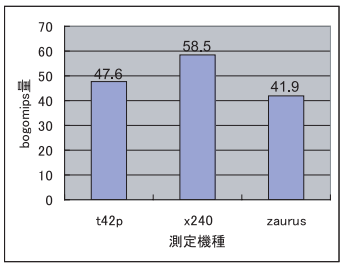


図 11 各環境における Bogomips 量

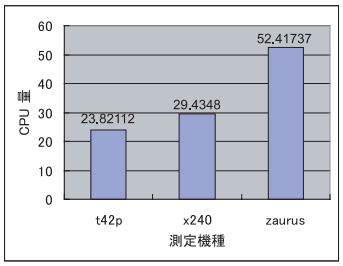


図 12 各環境における CPU 量

図 12 より，CPU を指標とした場合の標準偏差は 15.15 となった．このことから，異なる環境の CPU 周波数を環境間で交換するよりも，Bogomips を利用した方が妥当であることが分かる．

6.3 AROME の評価

本項では，2.2 項で述べた要件を軸に AROME と 3 章で紹介した関連研究を比較し，AROME の特徴を分析した結果を表 2 に示す．

	資源保証対応 OS	資源のモデル化	資源管理対応ローミング基盤
Mogul	×	×	CPU 制限
NOMADS	Aroma VM	CPU bytecode/ms	CPU 予約
m-P@gent	×	情報機器プロファイル	モジュール選択
RCANE	NEMESIS OS	CPU 使用率	○
AROME	Linux/RK	動的に監視	○

表 2 関連研究との相違

資源保証対応 OS については，Mogul と m-P@gent は利用していない．NOMADS は Aroma Java VM にて資源管理に対応しているが，OS の資源管理を考慮していないため，二重の管理構造となる．また，VM を利用しない既存のアプリケーションと共存できない．RCANE は NEMESIS OS を，AROME は Linux/RK を資源管理 OS として利用している．

Mogul は特に資源のモデル化は行っていない．NOMADS は CPU 資源を bytecodes/ms にてモデル化しており，Java VM を想定して情報機器をモデル化している．m-P@gent は情報機器のプロファイルに基づいて，移送するモジュールを選択し，消費する資源を調整している．しかし，情報機器のプロファイルは固定的なため，他アプリケーションのローミングやネットワークの変化などまた動的に変動する資源に対応できていない．RCANE は CPU 使用率にて資源消費をモデル化をしているが，アクティブネットワークにて単一種類のルータを想定しているため，情報機器の提供資源のモデル化をしていない．

AROME は Mona にて動的にアプリケーションの消費資源と情報機器の提供資源をモデル化している．

Mogul はローミング先にて CPU パワーの制約を管理者が明示的に指定し，DoS アタックを防止できる．NOMADS は Java VM にて bytecodes/ms の保証を行ったローミングが可能である．m-P@gent は資源が少ない情報機器にはコアモジュールのみ移送し，多様な資源に対応している．RCANE はローミングの際に CPU を確保できる．AROME は資源管理対応ローミング基盤として SaRaRi を拡張した TaRaRi を実装し，移動前の資源の問い合わせと，移動後の監視をサポートしている．

AROME は資源管理をともなったサービスローミングの要件を満たし，有意性を示した．

7. まとめと今後の課題

本研究では，資源管理機構を備えたサービスローミングフレームワークを提案した．リアルタイム OS として Linux/RK，資源の動的モデル化のための監視機構 Mona，資源をサービス移送機構を TaRaRi として，それらの要素から構成される AROME の設計およびプロトタイプ実装を行った．Mona によってモニタリングされた計算資源に基づき，TaRaRi はサービス移送時に移送先 AROME の TaRaRi に対し，資源の予約を行う．資源の予約が可能である場合にサービスは移送される．実際に資源を予約する Linux/RK との連携は，TaRaRi から生成される RealtimeThread を用いて行われる．

AROME の評価として，まず AROME の構成要素である TaRaRi と Linux/RK のオーバヘッドを測定し許容範囲内であることを示した．次に Mona の資源のモデル化手法を実際のサービスに適応し，実用に耐える制度であることを示した．最後に関連研究と比較し，資源管理を伴ったサービスローミングの要件を満たしているのは AROME のみであり，ここに本研究の意義と新規性があることを述べた．

7.1 今後の課題

本研究の今後の課題を，CPU 以外の計算機資源への対応実装，より詳細な資源モデルの構築，そして複数サービスへの対応の三つ述べる．

7.1.1 CPU 以外の計算機資源への対応実装

本論文で作成した AROME へは，CPU の資源管理を実現する実装を行った．今後は，メモリ領域や I/O 帯域などの資源も同様に管理するための実装を行う必要がある．

7.1.2 より詳細な資源モデルの作成

今回のプロトタイプ実装では，資源モデルとしてベンチマークによる数値化を利用したが，情報機器やサービスを Kahn Process Network[14] などを利用してグラフでモデル化することによって，より正確な動作の予測ができる．

また，動的に変動する資源への対応をする必要がある．現在，資源の記述は固定値でなされ，動的な変更へは監視機構で対応している．しかし，一日のある特定の時間は利用者が多く資源が不足する，など長期的なベンチマークに基づくモデルを作成する．

7.1.3 複数サービスへの対応

複数のサービスが同一ホスト上に対しローミングした場合，資源の競合が起こる．この問題を解決するために，以下の 2 つの手法が考えられる．

- 現在固定的に 1 つのサービスに対して割り当てている資源を，動的ぶ割り当てる．
- サービスに対し優先度を設け，優先度の高いサービスに対して優先的に資源を割り当てる．

今後この 2 つの手法について検討し，実装・評価を行なう必要がある．

参 考 文 献

1) Murase, M., Iwamoto, T., Nagata, T., Nishio, N. and Tokuda, H.: Implementation and Evaluation of Wapplet Framework, *IEEE International Workshop on Networked Appliances(IWNA)*, pp. 275–284 (2002).
2) Fuggetta, A., Picco, G. P. and Vigna, G.: Understanding Code Mobility, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342–361 (1998).
3) 永田智大, 西尾信彦, 徳田英幸: サービス利用状況の変化に対する適応支援機構, 情報処理学会論文誌, Vol. 44, No. 3, pp. 835–847 (2003).
4) Jensen, E. D., Locke, C. D. and Tokuda, H.: A Time-Driven Scheduling Model for Real-Time Operating Systems, *Proceedings IEEE Real-Time Sytems Symposium*, pp. 112–122 (1985).
5) 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, No. 6, pp. 2573–2584 (1999).
6) Takashio, K., Mori, M., Funayama, M. and Tokuda, H.: Constructing Environment-Aware Mobile Applications Adaptive to Small, Networked Appliances in Ubiquitous Computing Environment, *Proceedings of 4th International Mobile Data Management Conference*, Lecture Notes in Computer Science, Vol. 2574, Melbourne, Australia, Springer Verlag, pp. 230–246 (2003).
7) Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., Jeffers, R. and Mitrovich, T. S.: An Overview of the NOMADS Mobile Agent System, *Proceedings of ECOOP'2000, Nice, France, 2000* (2000).
8) Menage, P.: RCANE: A Resource Controlled Framework for Active Network Services, *First International Working Conference on Active Networks (IWAN)*, Vol.1653, Springer-Verlag, pp.25–36 (1999).
9) Alexander, D.S.: *ALIEN: A Generalized Computing Model of Active Networks*, PhD Thesis, University of Pennsylvania (1998).
10) Oikawa, S. and Rajkumar, R.: Linux/RK: A Portable Resource Kernel in Linux, *In Proceedings of the IEEE Real-Time Systems Symposium* (1998).
11) 米澤拓郎, 小泉健吾, 守分滋, 永田智大, 徳田英幸: Smart Furniture 間の柔軟なサービスローミングを実現するミドルウェアの構築, 情報処理学会 第三回ユビキタスコンピューティングシステム研究会, Vol. 2004(4), pp. 39–46 (2004).
12) 由良淳一, 中澤仁, 徳田英幸: 知覚およびメディア主導なコンテンツ変換システムの設計と実装, 情報処理学会ユビキタスコンピューティングシステム研究会論文集, Vol. 2003, No. 115, pp. 171–176 (2003).
13) Greg Bollella, Ben Brosgol, Steve Furr, David Hardin, Peter Dibble, James Gosling, Mark Turnbull and Rudy Melliardi: *The Real Time Specification for Java*, Addison-Wesley.
14) Kahn, G.: The semantics of a simple language for parallel programming, *IFIP Congress 74*, North-Holland Publishing Co. (1974).