システムソフトウェアと
オペレーティング・システム 78－18
（1998．5．8）

AQUARIUS: 動的適応可能なQOSアーキテクチャ

大越 匡[1]　望月 祐洋[1]　徳田 英幸[1,2]
[1]慶應義塾大学大学院 政策・メディア研究科　[2]慶應義塾大学 環境情報学部

計算機外部環境の動的な変化と，それにともなう計算機資源の量・性質面の動的変化がおきるモバイルコンピューティング環境においては，計算機利用者への作業品質の保証と，作業の継続性提供が課題とされる．本研究において開発されたAQUARIUSは，アプリケーションに，サービス品質の保証と，作業継続性をする移動透過性を提供する．アプリケーションプログラマは，本アーキテクチャを利用することで，アプリケーションにサービス品質調整機能と移動透過性を容易に付加することができる．本稿では，AQUARIUSの設計，実装，および評価について述べる．

AQUARIUS: An Adaptive QOS Architecture for Dynamically Reconfigurable System

Tadashi Okoshi[1]　Masahiro Mochizuki[2]　Hideyuki Tokuda[1,2]
[1]Graduate School of Media and Governance, Keio University
[2]Faculty of Environmental Information, Keio University

In mobile computing environment, where the dynamic changes of the computing environment and the computing resources in quality and quantity occurs, the provision of "Continuous Operation" and "Quality Of Service" for the users is important. We developed "AQUARIUS", the adaptive QOS architecture for dynamically reconfigurable systems. AQUARIUS provides the QOS functions and the mobility which supports Continuous Operation for the applications. Using AQUARIUS, application programmers can add the QOS adaptation and the mobility to their applications easily. In this paper, we describe the design, implementation and the evaluation of AQUARIUS.

1　はじめに

本研究は，モバイルコンピューティング環境における計算機利用者への(1)作業品質の保証と(2)作業の継続性の提供を目標とし，それを実現するソフトウェアアーキテクチャについて述べる．動的適応可能なQOSアーキテクチャ"AQUARIUS (Adaptive QUality of service ARchitecture for dynamIcally reconfigUrable System)" を開発した．

新しい計算機利用形態であるモバイルコンピューティング環境においては，計算機外部環境の動的変化にともなって，計算機資源もまたその量・性質が動的に変化する．このような状況の中で，連続メディアアプリケーションや非連続メディアアプリケーションなど，異なる性質を持つ複数のアプリケーションを同時に動作させる場合，計算機資源の効率的かつ動的な配分や，計算機資源への透過的なアクセスの実現が求められる．そして，計算機利用者への作業品質の保証と，作業の継続性の提供が望まれる．

AQUARIUSの目的は，本研究の目標実現のため，計算機利用者のアプリケーションに対して(1)サービス品質の保証と，(2)作業の継続性提供を実現することである．モバイルコンピューティング環境における計算機機器構成の動的変更とそれにともなう計算機資源の変更に動的に適応し，計算機利用者のアプリケーションへ計算機資源の包括的な配分を行なう，動的適応可能なQOSアーキテクチャである．

AQUARIUSは主に，計算機資源管理サーバ，QOSサーバ，そしてユーザアプリケーションとのインターフェース部分であるライブラリから構成される．アプリケーションプログラマはこのライブラリを利用することで，アプリケーションに作業の継続性提供のための移動透過性と，サービス品質の調整機能を容易に付加することができる．

本稿では，第2節で本研究の背景となるモバイルコンピューティング環境について述べる．第3節で作業の品質と継続性について述べる．第4節では既存の関連研究を挙げその問題点を検討する．第5節では本研究における提案であるAQUARIUSの特徴と機能について述べ，第6節でその設計，第7節でその実装を詳解する．第8節では，基本性能評価および既存関連研究との機能比較を行なう．最後に第9節でまとめと今後の課題を述べる．

2　モバイルコンピューティング環境

本節では本研究の背景となる，モバイルコンピューティング環境における計算機外部環境および計算機資源の動的な変化について述べる．

2.1　計算機外部環境と計算機資源の動的変化

計算機処理能力の向上や小型化，無線ネットワークなどの普及により，モバイルコンピューティング環境が整備されてきている．このモバイルコンピューティング環境においては，計算機外部環境が動的に変化し，それにともない計算機資源もその量・質・構成といった面で動的に変化する．図1に事例を示す．

計算機外部環境の変化

モバイルコンピューティング環境における計算機の物理的な移動は，それにともなう計算機外部環境の変化をもたらす．ここでの「計算機外部環境」とは，計算機のネットワーク接続状態，電源供給状態，周辺機器接続状態などをはじめ，広義の外部環境という意味

では，計算機の現在位置，現在位置の気温，湿度，明るさといった要素も含む．

例えば事例の様に，利用者が計算機を利用しながら移動する場合，「位置」という外部環境が職場→電車内→自宅へと変化し，また，「ネットワークインターフェース機器」という環境がWaveLAN→携帯電話→ISDNへと変化する．

計算機資源の変化

このような計算機外部環境の動的な変化は，同時に計算機資源の動的な変化をもたらす．環境a→環境b→環境cと外部環境が変化するにともない，計算機資源であるネットワーク資源は，その量・質の両面において動的に変化する．資源量としてのネットワーク帯域は，2Mbpsから9.6kbps，128kbpsへと変化する．資源性質の一つであるリンク層の接続方式は，環境aと環境bにおいては“無線接続”であり，環境cにおいては“有線接続”へと変化する．

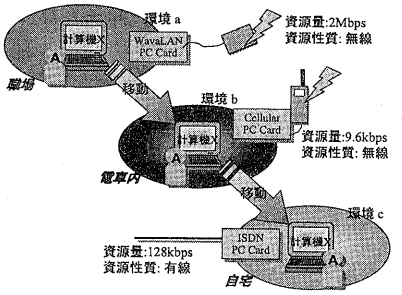

図1: 計算機外部環境および計算機資源の動的変化

3 作業の品質と継続性

前節で述べたような，計算機外部環境と計算機資源が動的に変化する状況においては，計算機利用者とそのアプリケーションに対する作業品質の保証と，作業の継続性の提供が重要となる．図2にその概念を示す．

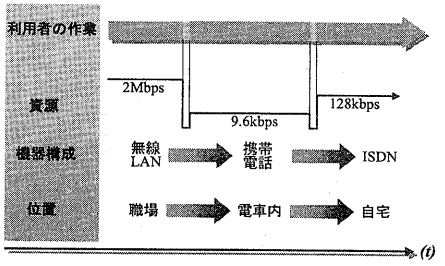

図2: 作業継続性の提供

3.1 作業品質の保証

テキストや静止画といった非連続メディア情報をあつかう既存のアプリケーションに加えて，動画や音声といった連続メディア情報をあつかう連続メディアアプリケーションが普及してきている．連続メディア情報は，その再生に一定の時間を要し(時間的制約)，またその再生時にサービス品質の調整が可能である(サービス品質の制約)という面で非連続メディア情報と異なる．

単一オペレーティングシステム上で，連続メディアアプリケーションや非連続メディアアプリケーション，またシステムの各種サーバやデーモンが同時にプロセスとして動作する状況においては，各プロセスに公平に資源を配分するのでなく，各プロセスが必要とする資源量に応じた効率的な配分を行なう必要がある．

特にモバイルコンピューティング環境では，提供される計算機資源の量および性質が一定でないため，計算機利用者へ提供される作業の品質も一定ではない．この様な状況下では，オペレーティングシステムは，供給資源量の変化に適応して，各アプリケーションに対して動的な資源配分調整を行なう必要がある．また，各アプリケーションは，配分資源量の変更に動的に適応して，その品質を調整することが必要である．オペレーティングシステムとアプリケーションの双方による適応的な動作によって，計算機利用者への作業品質の保証が実現する．

3.2 作業継続性の提供

計算機外部環境や計算機資源が動的に変化する状況においては，計算機利用者と使用中のアプリケーションに対して，いかに作業の継続性を提供するかが問題となる．作業の継続性は，計算機外部環境および計算機資源の動的変化の中で，計算機利用者のアプリケーションおよび行なっている作業が継続性をもって行なわれる時，実現する．

図2の事例では，ネットワークインターフェースの変更という計算機外部環境の動的変化により，ネットワーク資源としての，インターフェース識別子，帯域，リンク性質，IPアドレスなどがそれぞれ動的に変化する．

現状では，計算機利用者自身がアプリケーションの再起動や，ネットワーク設定変更といった作業を行なって，作業の継続性を維持する必要がある．システムが何らかの方法で計算機資源の変化に動的に適応し，既存の通信を保持してアプリケーションによる通信に継続性を持たせ，加えてアプリケーション側でも資源変化に動的に適応することで，利用者に作業の継続性を提供する包括的な機構が求められている．

4 関連研究

本節では，作業品質の保証および作業の継続性提供の各分野における関連研究について述べ，それぞれの問題点を論じる．

4.1 QOSアーキテクチャ

サービス品質の保証を目標とした関連研究としては，QOSアーキテクチャ[1]と分類される種々の研究があげられる．QOSアーキテクチャは，主に連続メディア情報の効率的な処理を目標として設計され，計算機資源予約機構，アドミッションコントロール機構，動的QOS制御機構などの機構を持つ．

QoS-A (Quality of Service - Architecture)

QoS-A[2]は，多様なネットワーク環境における連続メディアフローの制御と品質管理のための，階層化されたアーキテクチャである．QoS-Aにおいては，“flow”，“Service Contract”，“flow management”の3つがキーワードとなる．flowは，それぞれの通信の性格づけを行なう．また，Service Contractは利用者

とオペレーティングシステムのQOSレベルに関する交渉の手段である．またflow managementは，設定されたQOSレベルでの資源消費状況のモニタリングとそれに対するメンテナンスを行なう．QoS-Aは，5つの“Layer”と3つの“Plane”からなり，垂直および水平方向にそれぞれ協調してQOSの管理を行なう．

4.2 移動透過性実現に関する研究

作業の継続性提供に関する関連研究としては，移動ホストでの通信における移動透過性実現を目的とした各研究があげられる．

Mobile-IP

Mobile-IP[3]は，携帯型計算機が，あるネットワークから他のネットワークに移動した場合にも，計算機のネットワーク上の識別子であるIPアドレスを変更せずに，同一IPアドレスで通信が行なえるしくみである．Mobile-IPを利用することで，計算機は，ネットワーク間の移動をIP層において透過的に行なうことができ，計算機の利用者に，ネットワーク上の移動透過性を提供する．

実際に移動を行なう計算機であるMobile Host (以下MH)は，通常Correspondent Host (以下CH)と直接通信を行なう．しかし，MHがIPアドレスを変更せずに異なるネットワーク上へと移動した場合，CHから送られてくるIPパケットはMHに届かないため，CHとMHが直接通信を行なえなくなってしまう．そこで，MHのホームネットワーク上のHome Agent (以下HA)と，MHの移動先のネットワーク上のForeign Agent (以下FA)を利用し，ルーティングを動的に変更して通信を行なう．これによって，MHがホームネットワークから他のネットワークへ移動しても，CHとMHはその移動を意識することなく通信することが可能となる．

TCP-R

TCP-R[4]は，TCP層での透過性を実現するしくみである．TCP-Rを利用することで計算機は，ネットワーク間の移動にともなうIPアドレスの変更があった場合にも，TCPコネクションを維持することが可能になり，計算機の利用者に作業の継続性を提供する．

MHは，通常CHと直接通信を行ないその間にTCPコネクションを確立するが，この状態でMHが他のネットワーク上へと移動した場合，CHから送られるIPパケットはMHに届かないため，必然的にTCPコネクションの維持も不可能となる．そこでTCP-Rでは，MHが他のネットワーク上へ移動した場合に，MHがその新しいIPアドレスをCHに通知する．CH内のTCP-Rプロトコルスタックでは，既存のTCPコネクション上で通信が発生すると，その宛先IPアドレスをMHの新しいIPアドレスに置き換え，現在MHの持つIPアドレスへとIPパケットを送信する．このしくみによってMHの移動後も，TCP層では既存のTCPコネクションを利用し続けることが可能となる．

PMI

PMI[5]は，動的に多様なネットワークインターフェースの管理を行なうアーキテクチャである．イーサネットカード，無線LANカード，モデムカードといった，複数の異なる種類のネットワークインターフェースの構成変更に対応して，IPパケットのルーティングの変更，アプリケーションへの通知などの機能により，計算機利用者に作業の継続性を提供する．

PMIアーキテクチャはそれ自身ではIP層での透過性を実現しない．移動透過性実現にはMobile-IPの併用が条件となる．ネットワークインターフェース構成の変更時に，アプリケーションに新しいネットワークインターフェースに関する情報を通知することで，アプリケーション側に動的な適応を求める方式をとっている．アプリケーションはこの通知を取得し適応することで，機器構成の動的変更にアプリケーションレベルで動的適応することが可能となる．

4.3 各研究における問題点

QOSアーキテクチャとして挙げた関連研究の問題点は，配分される資源は動的には変化しないものと前提されている点である．動的に計算機資源の構成が変化する環境において，適応する仕組みが提供されていない．

一方，移動透過性を実現する各研究における問題点は，移動透過性実現のためにネットワーク資源の変化を隠蔽することのみに着目している点である．

ネットワークインターフェースや，接続するネットワーク変更などにともなって，ネットワーク資源は，資源量としての帯域・遅延・ジッターや，性質としての接続形態・帯域保証や遅延保証をはじめとしたリンク層の機能などが動的に変化する．アプリケーションの適応的な動作のためには，これらの変化をアプリケーションに通知し，アプリケーション側で動的に適応可能な仕組みを提供する必要がある．

計算機資源の構成，量および性質が動的に変化する状況において，アプリケーションにそれらの変化を通知し，システム，アプリケーションの双方における動的な適応によって，サービス品質の保証と作業の継続性を提供する枠組みが必要とされている．

5 動的適応型QOSアーキテクチャ

本節では，本研究が提案する動的適応可能なQOSアーキテクチャ“AQUARIUS”について，その目的，機能，特徴を述べる．

AQUARIUSの目的として次の二点が挙げられる．AQUARIUSは，計算機機器構成をはじめとする計算機外部環境の動的変化と，それにともなう計算機資源の動的変化に対して，システムおよび利用者のアプリケーション側の双方で協調的な動的適応を行なうことで，アプリケーションに対して，(1) サービス品質の保証と，(2) 作業の継続性を提供することを目的としている．

5.1 機能

AQUARIUSはアーキテクチャ全体として，ユーザのアプリケーションに対し以下の5つの機能および抽象を提供する．

(1) 計算機機器と独立に抽象化された計算機資源

AQUARIUSは，物理的な計算機機器構成とは独立して抽象化された計算機資源を提供する．計算機機器の構成が動的に変化する環境では，既存のネットワークデバイスをはじめとした計算機資源もその構成を動

的に変える．このような変更に依存しない資源への透過的なアクセスをアプリケーションが実現するためには，物理的な計算機機器構成とは独立して抽象化された，計算機資源が必要となる．アプリケーションがその資源に対してアクセスを行なうことで，アプリケーションと物理的な計算機機器構成を，完全に分離することが可能である．

(2) 計算機資源予約機能

CPU，メモリ，ネットワークといった計算機資源の予約機能を提供する．連続メディア処理など，サービス品質保証の必要性のあるアプリケーションをはじめ，端末サービスやファイル転送といった従来のアプリケーションに対しても，資源予約機能を提供することは，計算機資源の有効利用を促す意味で重要である．

(3) 計算機機器構成変更時の通知機能

CPU，ネットワークインターフェース，入出力機器などの計算機機器の，動的な構成変更を通知する．ノートブック型計算機やPDAなどの携帯型計算機ばかりでなく，デスクトップ型計算機なども含め，計算機機器の構成が動的に変化する環境では，アプリケーションが計算機機器の構成変更に適応して動作を変更する必要がある．そのためのアプリケーションプログラミングインターフェース(API)を提供する．

(4) 資源量変更の通知機能

利用可能なネットワーク帯域やメモリ量の変化など，「利用可能資源量」の変更を通知する．オペレーティングシステム自身や複数のアプリケーションが，有限な計算機資源を共有する状況においては，利用可能な資源の量は動的に変化し，各アプリケーションも随時，この変化に適応して動作する必要がある．そのためのAPIを提供する．

(5) 資源性質変更の通知機能

ネットワーク資源における帯域保証機能や遅延保証機能有無の変化など，計算機機器構成の変更等に起因する「利用可能資源の性質の変化」を通知する．計算機資源を提供する計算機機器の構成が動的に変更される環境では，各アプリケーションは，計算機資源性質の変化に適応して動作することが必要となる．そのためのAPIを提供する．

5.2 特徴

アプリケーションプログラマは，AQUARIUSが提供する各種のAPIを利用して，アプリケーションを記述もしくは変更することにより，アプリケーションにサービス品質の保証と作業の継続性提供を付加することができる．

サービス品質の保証

アプリケーションは処理に必要な計算機資源スペックをAQUARIUSに通知し，計算機資源の予約を行なう．AQUARIUSは，現在利用可能な計算機資源の情報と配分ポリシに基づき，資源をアプリケーションに配分する．計算機機器変更などの外部環境の変化，同時に動作する他のアプリケーションとの配分調整などにより，配分される資源量およびその性質が変化した場合は，これをアプリケーションに通知する．アプリケーションは動的に適応し，サービス品質を調整する．

作業の継続性

アプリケーションは，AQUARIUSが提供する，物理的な計算機機器構成とは独立して抽象化された計算機資源に対してアクセスを行なうことで，機器構成の変更などに依存しない，資源への透過的なアクセスを実現する．

ネットワーク資源を例に挙げれば，アプリケーションは，計算機上で動作しているネットワークインターフェースの種類や性質，またその変更などに依存せず，透過的なネットワークへのアクセスが可能となり，通信における作業の継続性を実現する．

6 設計

本節では，AQUARIUSの内部設計を詳解する．AQUARIUSを構成する各ソフトウェアモジュール，モジュール間で交換される情報の構造，およびアーキテクチャ全体としての動作について述べる．

6.1 構造

AQUARIUSの構造は，Messengerモジュール群，Resource Center，Aquarius Server，libAQUAの4つのソフトウェアモジュールから構成される．図3にその概念を示す．

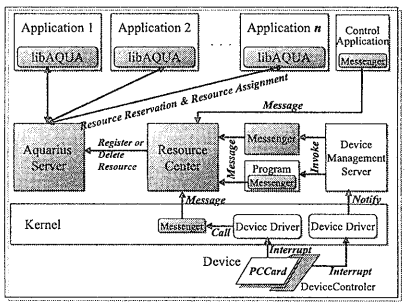

図3: AQUARIUSの構造

Messengerモジュール群

Messengerモジュール群は，オペレーティングシステムのカーネル内，各計算機機器の管理サーバ，ネットワークプロトコルスタック，アプリケーションなど様々なソフトウェアに組み込まれる，もしくはそれらから起動されるソフトウェアモジュールである．各Messengerは，各計算機機器の接続状況の変化およびそれらが提供する計算機資源関連情報を，Resource Centerへ通知する．

Resource Center

Resource Centerは，計算機機器・資源管理サーバとして，現在の各計算機機器の接続状況を一元的に管理する．また，各計算機機器が提供する計算機資源を，Aquarius Serverへ登録する．

Aquarius Server

Aquarius Serverは，QOSサーバとして計算機内の利用可能資源を一元的に管理する．また，Resource Centerから提供された計算機資源を，アプリケーションからの予約要求と資源配分ポリシに従って，複数のアプリケーションに分配する．計算機資源に何らかの変化が生じた場合は，その変化に応じて各アプリケー

ションへの資源配分量を調節し，アプリケーションにそれを通知する．

libAQUA

libAQUA は利用者のアプリケーションに，AQUARIUS アーキテクチャのインターフェースを API として提供する，ソフトウェアライブラリである．各アプリケーションにリンクされ，アプリケーションに QOS 制御 API を提供する．

6.2 情報構造

AQUARIUS ではアーキテクチャ全体としての動作のために，計算機機器の状態情報，計算機資源情報および資源予約情報を，ソフトウェアモジュール内にデータまたはオブジェクトとして保持し，必要に応じて各ソフトウェアモジュール間で交換する．

計算機機器の状態情報

現在接続されている計算機機器の状態に関する情報である．その断片的な情報が，一つまたは複数の Messenger モジュールから Resource Center に通知され，Resource Center 内で一つの完全な情報として保持される．情報は以下の 4 つの要素から構成される．これを表 1 に示す．これは PMI にて採用されている Device Availability Model[5] の一部を一般的な計算機機器へと適応したものである．各構成要素は原則として "true"/ "false" の 2 値で定義される．

表 1: 計算機機器の状態情報

構成要素	内容
Present	機器の物理的な接続状況
Powered	機器の電源状態
Connected	機器の Link レベルの接続状況
Named	計算機機器が提供する 計算機資源への名前づけ

計算機機器は状態を持ち，遷移する．状態情報の 4 要素のすべてが "true" になると，機器は「利用可能接続状態」に遷移する．すべて "false" の状態を「非接続状態」，いずれかが "true" になると「利用不可能接続状態」に遷移する．

計算機資源情報

現在計算機に接続されている各計算機機器が提供する，計算機資源に関する情報である．その断片的な情報が，一つまたは複数の Messenger モジュールから Resource Center に通知され，Resource Center 内で一つの完全な情報として保持される．そして，機器が「利用可能接続状態」に遷移すると資源情報は，Resource Center から Aquarius Server へと登録される．

資源予約情報

利用者のアプリケーションが，ライブラリを通して計算機資源の予約を行なう際の QOS スペック，および実際にアプリケーションに配分される計算機資源の情報である．

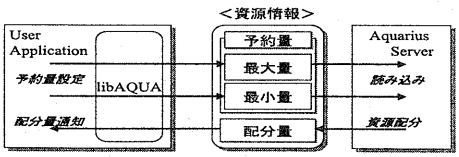

図 4: 資源予約情報

利用者のアプリケーションは資源の予約時に「希望する最大値」および「最低限度保証を要求する最小値」の組み合わせを「予約量」として定義し，libAQUA を通して予約を行なう．一方予約量をもとに，QOS サーバである Aquarius Server は，資源の配分を行なう．実際に配分される資源量は「配分量」として設定され，ライブラリを通してアプリケーションに通知される．また，配分量が予約量の最小値を満たせない場合には，アドミッション・コントロールにより，ライブラリを通じてアプリケーションに通知される．

6.3 アーキテクチャとしての動作

計算機資源の予約と配分動作

AQUARIUS を利用するユーザのアプリケーションからの，直接的な動作が計算機資源の予約である．ユーザは libAQUA を用いて計算機資源の予約を行なう．Aquarius Server は，その予約要求を処理し，現在配分可能な資源を配分する．各ソフトウェアモジュール間の協調動作の様子を図 5 に示す．

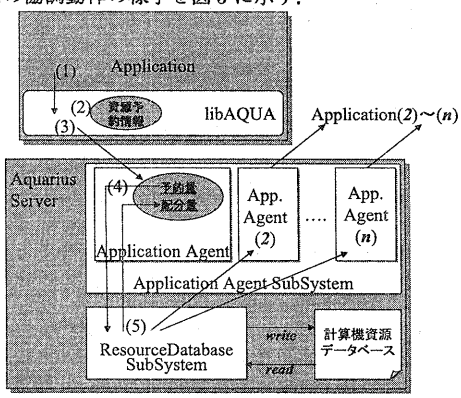

図 5: 計算機資源の予約と配分動作

(1) アプリケーションがそのプログラム中で libAQUA の関数を呼び出す．資源予約要求の最大値，最小値を設定する．

(2) libAQUA は内部に予約資源情報を作成し，予約量の最大値，最小値を設定する．

(3) libAQUA は Aquarius Server へ接続し，資源予約情報を送信する．

(4) Aquarius Server 内 Application Agent Subsystem に新規に作成された Application Agent は，libAQUA から資源予約情報を取得．そのうち予約量情報のみを Resource Database Subsystem に通知する．

(5) Resource Database SubSystem は一つまたは複数の Application Agent から取得した予約量情報と，自ら保持している計算機資源データベースから，各 Application Agent への資源配分量を算出し，配分量情報として各 Agent に通知する．

(6) Application Agent は，配分された資源量を libAQUA へと送信する．

(7) libAQUA は，配分された資源量をアプリケーションプログラムへと通知する．

計算機機器構成にともなう動作

本アーキテクチャが想定するのは計算機機器の構成が動的に変更される計算機環境である．ここでは，計算機機器の追加，削除および性質の変更などの動的変更にともなう，各ソフトウェアモジュール間の協調動作を図6に示し，計算機機器の追加，削除，性質変更時の動作についてそれぞれ解説する．

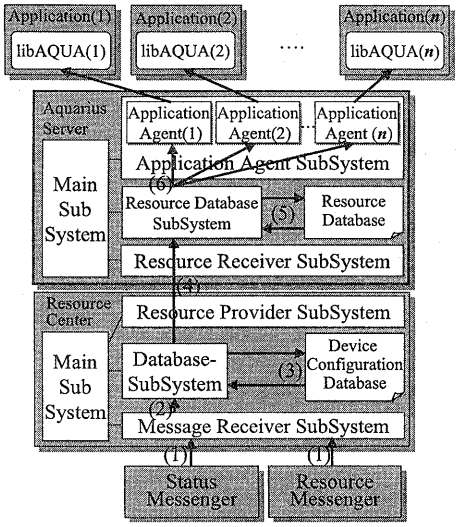

図6: 計算機機器構成にともなう動作

(1) 計算機機器の追加，削除，性質変更がおきると，オペレーティングシステムのカーネル内，各計算機機器の管理サーバ，各種プロトコルスタックなどから起動された各 Messenger が，計算機機器の状態情報，計算機資源情報を Resource Center に通知する．

(2) Resource Center 内 Messenger Receiver SubSystem はその情報を Database SubSystem に通知する．

(3) Database SubSystem は「計算機機器構成データベース」を更新し，計算機機器の状態を遷移させる．

(4) Resource Provider SubSystem は，「計算機資源の登録」，「計算機資源の削除」「計算機資源変更の登録」を Aquarius Server へと通知する．

(5) Resource Center からの通知を取得した Aquarius Server は，通知に従って「計算機資源データベース」を更新する．

(6) Resource Database SubSystem は，更新されたデータベースと，各 Application Agent から通知されている各 Application からの資源予約量から，各 Application への資源配分量を計算する．そして，各 Application Agent に通知する．

(7) 各 Application Agent は「資源量の変更」，「資源性質の変更」および「機器構成変更」を libAQUA を通じて，各 Application に通知する．

7 実装

本研究では，AQUARIUS アーキテクチャ全体のうち，特にネットワーク部分に関する計算機機器管理，計算機資源管理，計算機資源予約機構を実装した．実装に際しては，表2の実装環境を使用した．

表2: 実装環境

Hardware	IBM ThinkPad 560
NetworkIF	3Com PCMCIA-EtherLinkIII(3C589C) AT&T WaveLAN PCMCIA2.4JP
OS	FreeBSD 2.2.1R
Packages	PAO-970331[6] WIDE-dhcp-1.3b MIT pthreads-1.60beta6

7.1 実装概観

実装の概観を図7に示す．Messenger モジュール群，Resource Center サーバ，Aquarius Server サーバの3モジュールは C 言語で実装した．また，本アーキテクチャの API をアプリケーションに提供するライブラリである libAQUA は Java 言語 [7] で実装した．各ソフトウェア間の通信には Socket インターフェースを用いた．

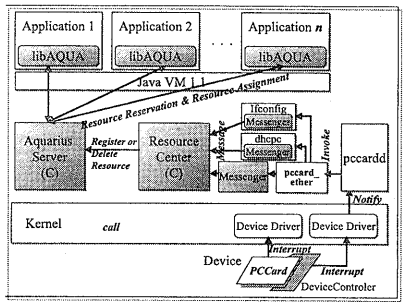

図7: 実装概観

7.2 AQUASocket クラス

今回の実装では，libAQUA におけるネットワーク資源として，AQUASocket クラスを Java 言語で実装した．AQUASocket クラスは，従来の Socket インターフェースの上位互換性を持つ，ネットワーク資源のインターフェースである．Java 言語の Java Core API1.1 においては，ネットワーク資源へのインターフェースとして，既に java.net.Socket クラスおよび java.net.ServerSocket クラスが提供されている．libAQUA は，Socket の機能に加え，クラス内部における動的な Socket 張り替えによる通信の移動透過性，資源予約・解放インターフェース，資源量・性質変更の通知インターフェース，実際の資源使用量の通知インターフェース等の機能を実現している．アプリケーションプログラマは，Socket クラスの代替として AQUASocket を利用することで，これらの機能を享受し，アプリケーションに，サービス品質の調整機能および移動透過性を追加できる．

図8に Socket クラスの利用例，図9に AQUASocket クラスの利用例を示す．AQUASocket クラスでは，資源予約を行なう場合あらかじめ NetworkQOS クラスのオブジェクトを生成，予約値を設定し，それを用いて AQUASocket オブジェクトを生成する．通信の方

法は，InputStream および OutputStream クラスを用いる点で Socket クラスと全く同じである．資源配分の変更は，Java イベントによってアプリケーションに通知される．アプリケーションで，通常の Java におけるイベント処理方法と同様にイベントを処理することで，容易に動的 QOS 制御が可能となる．

```
//Socket 生成
Socket sock = new Socket("host",14000);
sock.connect();
InputStream in = sock.getInputStream();
OutputStream in = sock.getOutputStream();
//通信開始
```

図 8: Socket クラス利用例

```
{
  //NetworkQOS を指定
  nqos = new NetworkQOS();
  nqos.setMaxBandwidth(max);
  nqos.setMinBandwidth(min);
  nqos.setImportance(importance);
  //NetworkQOSEvent のリスナーに登録
  nqos.addNetworkQOSListener(this);

  //AQUASocket 生成
  asock = new AQUASocket(
        hostname, port, nqos, true);
  out = asock.getOutputStream();
  in =  asock.getInputStream();
  //通信開始
}

//NetworkQOS 変更時の対応
public void NetworkQOSChanged(
                    NetworkQOSEvent e){
    DynamicQOSControl(e);
}
```

図 9: AQUASocket クラス利用例

7.3 利用アプリケーション例

AQUARIUS を利用したアプリケーション例として，Mobile Telnet および Mobile Video Conference を挙げる．

7.3.1 Mobile Telnet

telnet は端末サービスのソフトウェアであり，TCP/IP プロトコルの Socket インターフェースを利用する典型的なアプリケーションである．telnet をはじめとして，従来 Socket クラスを用いて記述されていたすべての Java アプリケーションは，AQUASocket を代替に利用することで，容易に移動透過性をもったアプリケーションとなり，利用者に作業の継続性を提供する．

7.3.2 Mobile Video Conference

ビデオ会議ソフトウェアや，ネット電話ソフトウェアなど，連続メディア情報を扱うソフトウェアは，AQUARIUS を利用することにより，容易に QOS 制御機能および移動透過性を付加することが可能でありアプリケーション利用者に，作業品質の保証と，作業の継続性を提供する．

8 評価

本節では，実装した AQUARIUS の基本性能評価と，関連研究との機能比較を行い，評価結果の考察による本システムの有効性や問題点の検討を目的とする．

8.1 基本性能評価

8.1.1 測定環境

測定環境として閉鎖された LAN 環境を構築した．図 10 に測定環境を示す．また，表 3 に各ホストの性能を示す．

測定環境は Ethernet の 1 セグメントで構成されている．manoa は Correspondent Host として 10Mbps の有線 Ethernet で常時接続されている．nofear は Mobile Host として，10Mbps の有線 Ethernet および 2Mbps の無線 LAN(Ethernet) の接続を選択可能である．両ホストには，本プロトタイプシステムが導入されている．

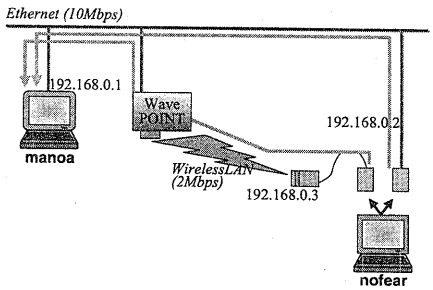

図 10: 測定環境

表 3: 測定環境のホスト

項目	nofear	manoa
CPU	Pentium 133MHz	Pentium 90MHz
主記憶	40MB	16MB
有線 Net.IF	3Com EtherLink	IBM ETHERNET-II
無線 Net.IF	N/A	AT&T WaveLAN
OS	FreeBSD 2.2.1R	FreeBSD 2.2.1R
JavaVM	JDK 1.1.5	JDK 1.1.5

8.1.2 ネットワークインターフェースの切り替えと動的 QOS 制御

AQUASocket の特徴は，ネットワークインターフェースの差し換え時の透過性と，動的 QOS 制御にある．通信中のネットワークインターフェースの差し換えと，資源予約を行なった通信における動的 QOS 制御の性能について定量評価を行なった．

AQUASocket における資源予約インターフェースを用いて，上限 3000kbps，下限 100kbps の接続を Mobile Host から Correspondent Host に行なった．そして Mobile Host から Correspondent Host に向けて，512 バイトずつのデータを繰り返し送信した．通信開始時は Mobile Host はイーサネットカードによって 10Mbps で接続されているが，次に 2Mbps の無線 LAN である WaveLAN カードに差し換える．経過時間と転送量を測定し，それぞれのネットワークインターフェース使用時における転送速度を算出した．

8.1.3 測定結果

図11に経過時間と転送量の関係を示す．また，イーサネットカード，WaveLAN カードそれぞれの使用時における資源配分量と実際の転送速度の測定結果を表4に示す．

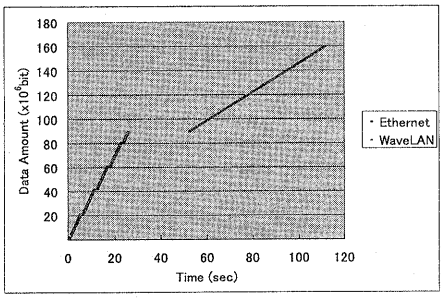

図11: ネットワークインターフェースの切り替えと動的QOS制御

表4: 資源配分と転送速度

IF	資源予約量	資源配分量	転送速度
E	100 ～ 3000kbps	3000kbps	3.32×10^3kbps
W	100 ～ 3000kbps	2000kbps	1.25×10^3kbps

8.1.4 考察

イーサネットカード(表中E)使用時は，100kbps ～ 3000kbps の予約量に対して3000kbps が配分されている．AQUASocketMoniterThread がモニタする実際の転送量に適応してアプリケーション側が動的に転送量を調節するため，転送速度 3.32×10^3kbps が実現されている．一方 WaveLAN カード使用時(表中W)には，2000kbps が配分される．WaveLAN の最大転送速度の理論値は2Mbps であるが，今回の測定では実測値として 1.25×10^3kbps が測定されている．

図4中のイーサネットカード使用時において，データの転送に一定の周期の揺らぎが観測された．これは，測定に使用したアプリケーション中の動的転送速度制御の仕組みに起因する．今回のアプリケーションにおいては，転送速度制御のため，1ms 単位での wait (Thread.sleep(long interval)) を行なう．しかし，JavaVM におけるスレッド制御のタイミングに精度上の問題があるため，実際には1ms 単位での wait は実現しない．この問題によって，一定の周期の揺らぎが生まれたものと思われる．

AQUASocket は，帯域保証機構のない Ethernet において，定期的な転送速度のモニタリングと，その通知に対するアプリケーションでの動的適応によって，転送速度の制御，および動的 QOS 制御を実現することが明らかになった．

8.2 機能比較

AQUARIUS(表中A)と，関連研究 Mobile-IP(表中M)，TCP-R(表中T)，PMI(表中P)，QoS-A(表中Q)を比較した．比較結果を表5に示す．

AQUARIUS は，Mobile-IP，TCP-R，PMI と同様，移動透過性を実現する．Mobile Host 移動後の三角ルーティングも起こらない．AQUARIUS は，QoS-A と同様，資源予約機構，動的QOS制御機構，資源量・性質の変化通知機構を有する．また加えて資源構成変化通知機構も有する．AQUARIUS では，アプリケーションに変更が必要なものの，アプリケーション側での動的適応を可能としており，変更も最小限なものにしている．実装面では，AQUARIUS はカーネル実装を必要としない．また直接通信に関係のないホストへの実装を必要としない．最終的に AQUARIUS は，(1) 作業の継続性と (2) サービス品質の保証という，本研究の2つの目標を達成していることが明らかとなった．

表5: 他システムとの機能比較

項目	A	M	T	P	Q
移動透過性	7層	3層	4層	3層	×
MH 移動後の通信	直接	三角	直接	三角	×
計算機資源予約機構	○	×	×	×	○
動的 QOS 制御	○	×	×	×	○
資源量変化通知	○	×	×	○	○
資源性質変化通知	○	×	×	○	○
機器構成変化通知	○	×	×	○	×
動的適応	可	不可	不可	可	可
実装の変更	要	不要	不要	必要	必要
カネール内実装	不要	要	要	要	要
FA,HA	不要	要	不要	要	不要
作業の継続性	○	○	○	○	×
サービス品質の保証	○	×	×	△	○

9 まとめと今後の課題

本研究では，計算機外部環境および計算機資源が動的に変化するモバイルコンピューティング環境において，利用者に作業の継続性とサービス品質の保証を行なうことを目標とし，動的適応可能な QOS アーキテクチャ "AQUARIUS" を設計し，実装，評価した．

定量的性能評価によって，実装された AQUARIUS が，帯域保証機構のない Ethernet において，システム・アプリケーション双方による動的適応によって，転送速度の制御，および動的 QOS 制御を実現することが明らかになった．また関連研究との機能比較によって，AQUARIUS が，本研究の目標である，作業の継続性の提供とサービス品質の保証の双方を実現していることが明らかとなった．

今後の課題としては，ネットワーク資源以外の実装，QOS 変換機構の実現などがあげられる．

参考文献

[1] Aurrecoechea, C., Campbell, A. and Hauw, L.: A Survey of QoS Architectures, *Multimedia Systems Journal,Special Issue on QoS Architecture* (1997).

[2] Campbell, A., Coulson, G. and Hutchison, D.: A Quality of Service Architecture, *ACM Computer Communications Review* (1994).

[3] Perkins, C.: IP mobility support (1996). RFC 2002, Internet Request For Comments.

[4] Funato, D., Yasuda, K. and Tokuda, H.: TCP-R: TCP Mobility Support for Continuous Operation, *Proceedings of IEEE International Conference on Network Protocols 97*, pp. 229–236 (1997).

[5] Inouye, J., Cen, S., Pu, C. and Walpole, J.: System Support for Mobile Multimedia Applications, *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 143–154 (1997).

[6] Hosokawa, T.: PAO: FreeBSD Mobile Computing Package (1997). http://www.jp.freebsd.org/PAO/.

[7] Gosling, J., Joy, B. and Steele, G.: *The Java Language Specification*, Addison Wesley, Reading, Massachusetts (1996).