B-015

携帯機器で多言語リソースを利用可能にするクライアント・サーバモデル

Client-Server Model for Mobile Computers with Multilingal Resources

小林 さとみ †
Satomi Kobayashi

1. はじめに

情報インフラの整備にともない，いつでもどこでも利用出来る携帯機器の質の向上は緊急に行うべき課題である．現状の携帯機器では，情報インフラや研究機関の情報公開サービスを十分活用しきれているとはいえない．例えば母国語とは異なる言語圏のサーバのサービスを利用する時に，適宜必要な言語に変換して入力を行うことが出来ないなどの問題がある．その理由の一つに携帯機器には，厳しいハードウエア上の制約があることがあげられる．利用者からは常に高度な処理を必要とされているが，携帯性と機能の充実にはトレードオフの関係があるためこの様な問題が発生してしまっている．

本研究では、アジア圏各国で使われる言語を携帯機器で柔軟に利用するという課題を中心として，言語リソースの支援を行うサーバ・クライアントモデルの設計について述べる．利用者の携帯機器のシステムリソースとコンテクストを検知するリソースコントロールサーバを設定し，携帯機器の言語環境を柔軟に変えうるシステムのモデルとその実現方法を提案する．

2. 関連研究

従来から X Server を使った端末など，軽いクライアントの実現については様々な研究が行われているが，携帯機器のクライアント・サーバモデルに関しては，近年盛んに研究がされている．例えば，Jing らが文献 [1] で，中間サーバを設け携帯端末に適した変換処理を行うパラダイムの研究を発表している．

またインターネット通信で動的に QoS を維持する研究として，以下のように資源管理を行う研究がなされている．2000 年に Luca Abeni らが，ハードリアルタイムの制約の下で動的な QoS をサポートできる Hertik OS のカーネルの構築を行った [2]．また，1997 年に Scott Brandt らが、分散的計算機の環境下でアプリケーションレベルとオペレーティングシステムレベルで、資源管理が行える DQM というライブラリを提案し、MPEG 動画再生時における最適化を図っている [3].

その他，コンテキストという概念を用いて，ユーザの状況に適応した携帯機器の開発が進められている．文献 [4] では，MIT の Stalker らが，コンテクスト適応型携帯用コンピュータの具体像を挙げている．本稿では，携帯機器の資源の有効活用に対して携帯機器の環境や利用状況に応じてシステムが可変に出来るコンテクスト適応型のサービスを考える．

†京都大学 人文科学研究所
†Institute for Research in Humanities, Kyoto University

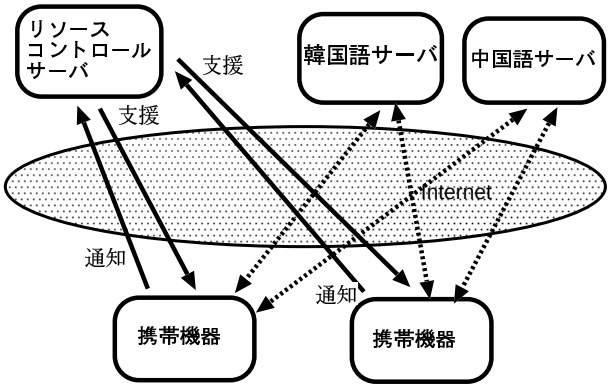

図 1: 想定しているシステム構成

3. リソースコントロールサーバ

3.1 概要

リソースコントロールサーバ (RCS) は携帯機器で装備されていない機能が必要なリクエストが発生した時に，ユーザのリクエストに応じたシステムの変更可能にできるように端末をサポートするサーバである．本研究ではサーバには据置型でハードウエア上の制限はない機器，クライアントには，無線用の通信デバイスがあり，メモリの容量やプロセッサの処理速度などの性能が低い機器を想定している．

RCS を利用すると，クライアント機はユーザから利用できるサービスのインターフェースが供給される．クライアント機にはインストールされていないプログラムが適宜利用することが出来るようになる．

ユーザがある言語変換プログラムを利用する時には，クライアント機の性能や通信状態などのコンテクスト情報と供給すべきプログラムの言語情報のイベントをサーバ機が受信して，最適なサービスを供給する．またその際に，リソース不足で不要と判断されるクライアント機のプログラムは本リソースコントロールサービスによって自動的に削除される．

3.2 課題

携帯機器のソフトウエア的リソース制御を行うことで，ソフトウエア構成の最適化を図る．利用可能な TCP/IP 通信では，サービスの一つにファイル共有を行う NFS が挙げられる [5]．NFS は，/etc/exports や autofs でサーバ側で共有するディレクトリを設定し，クライアント側のホストや利用パーミッション，ユーザを登録することでセキュリティを確保し，リモートプロシージャコールに

よってディスクリソースの制御ができるシステムである．

ところが NFS はセキュリティを考えた設計をしているため，スタティックに宣言するパラメータが多く柔軟性に欠ける．RCS では，クライアント機のコンテクストを知り，クライアント機の RC サービスの設定変更を適宜行って最適な携帯環境を支援する．

この際に発生する課題には次のようなものがある．

(1) 通信遅延への対応

無線 LAN での NFS 環境下でのプログラム実行に関しては，通信コストが非常に高いため，言語変換処理などの実行ファイルやフォントがメモリに入っていない状態の時にアクセスする時の遅延が問題になる．

通信状態は変化するので，クライアント機のコンテクストとしてサーバ側に通知する機能が必要となる．そこで本稿では，コンテクストをクライアントが最近利用したプログラム，CPU の利用状況，デバイスの使用状況，およびクライアントが利用可能な通信状況と定義する．

サーバはユーザが常に利用する信頼のおけるサーバを想定していることと，プログラムを利用した場合の通信経路からの情報漏洩に対しての安全性はある程度犠牲にできることから，通信速度を向上させることが望める．さらに，クライアント機のコンテクスト情報と，ハードウエアリソース情報から通信処理の最適な方法を求め，オーバーヘッドの少ない処理の実現を行う．

(2) リモートプロシージャコール実行時の安全性

クライアント機のコンテクストを反映させるには，カーネルのリソースパラメータなどの変更を行う場合がある．RCS は，サーバ側からクライアント側の設定変更を行うため，ルート権限の必要な機能をリモートマシンからの実行を許可する場合に悪意のある操作が行えないよう防御しないといけない．

この場合，例えば利用可能エリアや実行ユーザの設定を考える対策があるが，任意の操作に対して強力な防御が出来る対策が必要になる．

(3) ソフトウエアリソースの取得と解放の実現

取得に関しては，RCS 側でクライアント機に支援するリソースの決定後に，RCS 側の実行ファイルが携帯機器でも実行できる設定を自動的に行う必要がある．例えば，/usr/lib などの共通ライブラリ (shared library) 領域とフォントなどのデータリソースを mount する必要が発生する．ライブラリを検索するための環境変数や，リンケージローダの conf ファイルの設定に関する処理である．

また，クライアント側にディスクキャッシュを設けた場合，コンテクストに応じて最大実行可能プロセス数，ファイルシステム，共有メモリなどのリソースを短時間に取得/変更できる方法が必要である．

解放は，しばらく利用した形跡のない言語リソースに関して mount 実行を停止しアンインストールすることにより行う．解放するプログラムとその領域の選定，および実行のタイミングが課題になる．

(4) 利用するハードウエアリソースの最適解の策定

NFS は，サーバ側のファイルシステムをクライアント側でマウントしてクライアント側にあるように見せかけるものであるが，クライアント機側にはディスクキャッシュが用意されており，キャッシュを経由してアクセスする．本システムは複数の言語を併用し，マウントポイントが複数あることを想定しているので，クライアント機のディスクキャッシュ領域の検出と制御が必要になる．ただし書き込みは発生しないので，キャッシュコンシステンシーについては考える必要がない．

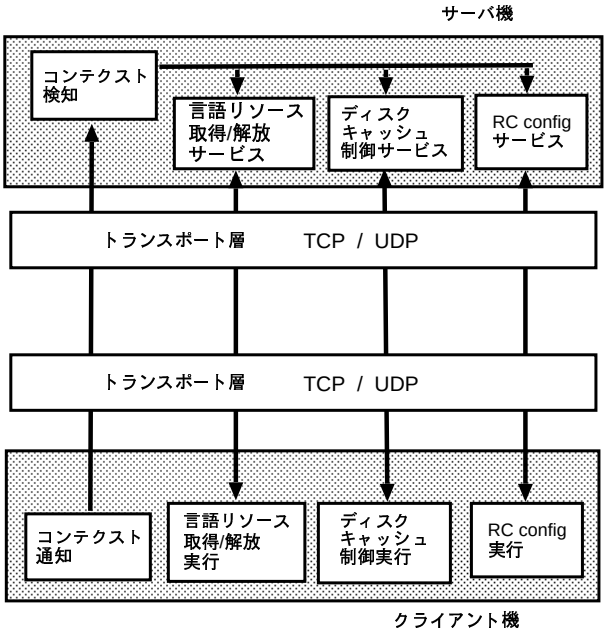

図 2: 装備する機能の概要

クライアント機の機器には制約があるので，利用の出来るデバイスや CPU などのハードウエアリソースを検知し，有効利用しないといけない．言語サービスを提供するかどうかの判断で必要である．提供しているプログラム単位で，どのハードウエアリソースを利用するかという使用状況の割合を決めることが必要になる．

4. 実現方法

本研究で実現するプログラムの機能について述べる (図 2 参照)．

4.1 通信方式

プログラムはサーバ側の rcd とクライアント側の rc で構成され，相互に操作に関するコマンドとデータリソースの送受信を行うことで通信がなされる．インターネット上での利用を想定しているため，TCP と UDP のトランスポートを用いる．通信が確立して RC 通信を行う通知を受けると通信ポート番号の設定，排他制御を行う．

通信方式は通常のソケット経由で，設定した所定のサイズでの表 4.1 に挙げるコマンド，サブコマンド，及びそれらに追加されるデータ情報の転送で行われる．

またディスクキャッシュなど大量のデータを転送する際には，圧縮して送信を行い通信時間の短縮化を図る．一度に送信する通信時のデータのサイズは，コンテクスト情報次第で変更可能とする．

表 1: リソースコントロール通信で実現される処理名と機能の例.

処理名	機能
SEND_CONTEXT	コンテクストを通知する
GET_LANGINFO	言語サービス情報を受信する
RC_CONFIG	リソースコントローラの設定を行う
LANG_INST	言語リソースを取得する
CACHEDATA	ディスクキャッシュデータ転送
LANG_UNINST	言語リソースを解放する

4.2 コンテクスト検知処理

クライアント機が，プログラムを実行するための言語リソースのリクエストを，サーバ機側に発行する際に，クライアント機の機器仕様，実行されているユーザプロセス，通信状況，デバイスの使用状況，CPU の利用状況を通知するものである．必要な言語リソース情報以外の状況を検知して，クライアント機の端末の状況を知ることで，言語リソースサービスを利用する時の，障害となりやすい設定のコンフィギュレーションを行う情報を受信する．

コンテクスト構造体は言語リソース取得・解放処理に渡され，インポートすべき言語処理機能を選択し，RC クライアント設定変更処理を経由して，ディスクキャッシュ制御処理に渡される．以下にコンテクスト構造体の一部を記す.

```
struct invoke_process{
   time_t start_time;
   uint size;
   unsigned char priority;
   char language;
   p_table *process_table;
}
struct user_network{
   unsigned char *dev_name;
   uint speed;
}
struct cpu{
   time_t now_time;
   uint process_number;
   unsigned char status;
}
struct terminal{
   unsigned char term_type;
   unsigned char *operating_system;
   unsigned short version;
}
```

4.3 RC クライアント設定処理

検知したクライアント機の通信状況とデバイス状況などのコンテクストによって，CPU の負荷を下げ通信速度を向上させるための，クライアント機側の設定変更処理を行う．本処理実行のためのユーザ rc_user を設定し，

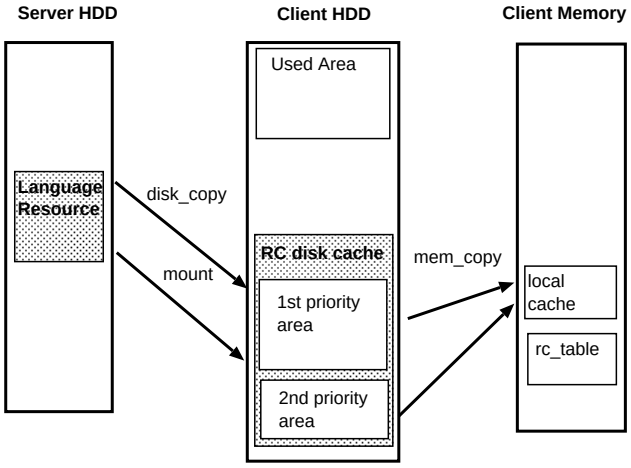

図 3: 言語リソースの取得とディスクキャッシュ.

この rc_user には quota などの制限を設定して実行が許可されるようにする．

コンテクストを検知して通信処理が遅かった場合には，通信方式のパラメータの変更を行う．またディスクキャッシュ制御処理で，決定されたキャッシュ領域の処理取得する言語リソースの分類に基づいて，ディスクキャッシュのサイズを決め領域をセットする．

4.4 言語リソース取得・解放処理

(1) 取得処理

クライアント機のコンテクストから得る言語情報によって，RCS から利用可能な言語と変換処理を選択し，リソース取得情報をクライアント機に返す．言語リソースは，

{実行ファイル，ライブラリ，辞書，フォント}

に分かれている．

ディスクキャッシュ制御処理により，上記の言語リソースを割り当ての領域を決める．また，言語リソースを複数取得した場合のため，ディスクキャッシュの管理テーブル (Cache 管理テーブル) にアクセスする．

サーバ機はクライアント機の転送待ち状態の返事を受けて，所定の言語リソースをユーザ領域のディスクキャッシュ領域に転送し，転送出来なかった領域分に対して mount 実行を行い必要な言語リソースを参照可能にする (図 3 参照)．クライアント機はサーバから受信したシェルプログラムによって実行可能状態にする．(インストール処理)

サーバ側では，キャッシュ言語リソースの export について実行される．転送すべきキャッシュデータの情報を受け，キャッシュ転送されない領域をクライアント機に export する．

新しい言語リソースが必要なプログラムが起動された場合，既存の言語リソース領域のディスクキャッシュの領域が変更され，次の言語リソースの領域を設ける．Cache 管理テーブルにはその情報が付加される．

(2) 解放処理

クライアント機がコンテクスト情報を送信するタイミングで，クライアント機はサーバから取得しているプログラムを維持するかチェックする．

クライアント機に RCS から受けているサービスの情報テーブル (RC 管理テーブル) を設けてある．解放言語リソースの選択には，RC 管理テーブルを検索して，長い時間使われていない LRU 等のポリシーに従って行われる．

解放処理は言語リソースを取得した時に，RCS 側から解放スクリプトを保存してそれを実行することで行う，具体的には，クライアント機に保存されている当該言語リソースのマウントポイントや, キャッシュデータ領域の解除などを行い，RC 管理テーブルから当該言語リソース情報のリンクが削除される．

4.5 ディスクキャッシュ制御処理

(1) 領域の確保

リモートディスクに対するアクセスコストが大きいので，クライアント機のハードディスクにディスクキャッシュ領域を設ける．そのためメモリ上のキャッシュとリモートディスクに対するキャッシュの二段構成をとる．

クライアント機には，RC 用の領域をスタティックに設定する．これを RC 領域と呼ぶが，リソースコントロールクライアントは，利用可能な最大のキャッシュ領域をとるように RC 領域の割り当てを決める．

言語リソースの利用性質 (パーミッション，参照頻度など) があるため，キャッシュの領域は 4.4 (1) で述べた要素で分類し，それぞれに個別に一定の領域を割り当てる．そのサイズは，参照の頻度が高いものに優先されるように設定する．例えばコンテクストスイッチの可能性の高い文字変換処理などの実行ファイルの領域は優先度を高く設定する．本機能で使われる主な関数を記す．

set_cache_parameters: 取得する言語リソースの分類に基づいて，ディスクキャッシュのサイズを決め領域をセットする．

make_resource_table: RC 領域を管理するためのテーブルを構築する．

resize_cache_area: 新しいプロセスが起動して言語リソースが取得された場合に，これまでに設定した各言語リソースの要素ごとのキャッシュサイズの変更を設定する．

(2) 領域の解放

新しいプロセスが発生し，第二の言語リソースが RCS から供給される時に，4.4(2) の解放処理で解放の対象にされたリソースは，mount 領域とキャッシュの領域が削除される．RC 領域には空きが発生するが，その領域は，現在使われている言語リソースのために利用される．以下にディスクキャッシュ解放アルゴリズムを示す．

```
キャッシュ解放 (言語リソース){
    Cache 管理テーブルの参照数を減らす;
    if(参照数が 1 なら) return;
    ディスクキャッシュを削除;
    マウントポイントを解除;
    for(残った言語リソース){
        キャッシュ領域判定機能;
        各キャッシュ領域を拡大する;
    }
}
```

5. おわりに

本稿ではクライアント機のコンテクストを通信で検知して，多言語リソースの利用を支援し，ハードウエア性能を効果的に利用することが出来る RCS とクライアントのモデルについて述べた．近年の携帯機器はハードウエア性能が限界に達しているが，ソフトウエアについては機能が拡大する傾向にある．また，汎用性のある携帯機器には，不要なプログラムや共有ライブラリがインストールされており，軽いソフトウエアシステムを再構築する手法を実現させる必要性が浮上している．

今後の課題は，クライアント機の利用状況をサーバ機に転送しているので，セキュリティ上や通信経路からの漏洩を守る方法が挙げられる．また，それに加えてクライアントの CPU の負荷の軽減については，十分な検討が行われていない．今後は，セキュリティの考察と CPU の負荷の軽減を図り，定量的に調査することで，利便性をさらに向上させた携帯機器の実現を課題にしたい．

参考文献

[1] Jin Jing, Abdelsalam (Sumi) Helal, Ahmed Elmagarmid “Client-Server Computing in Mobile Environments,” *ACM Computing Surveys*, 1999

[2] Luca Abeni and Giorgio Buttazzo, “Support for Dynamic QoS in the HARTIK Kernel,” *Proceedings of the 7th IEEE Real-Time Computing Systems and Applications Conference*, December 2000.

[3] Marty Humphrey, Toby Berk, Scott Brandt, and Gary Nutt, “The DQM Architecture: Middleware for Application-centered QoS Resource Management,” *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.

[4] T. Selker and W. Burleson: ”Context-aware design and interaction in computer systems,” *IBM Systems Journal*,Volume 39, Numbers 3 & 4, 2000.

[5] S. Shepler, et.al.:
”NFS version 4 Protocol,” *Request for Comments*, http://www.ietf.org/rfc.html, 2000.