修士論文 1995 年度 (平成 7 年度)

インターネット上での情報表現形式自動変換機構の実現

慶應義塾大学 大学院 政策・メディア研究科

林 周志

修士論文要旨　　1995 年度 (平成 7 年度)

インターネット上での情報表現形式自動変換機構の実現

本研究ではさまざまな情報表現形式間の変換をインターネット上で柔軟に行なうシステムを構築した。インターネット上では、さまざまな情報表現形式 (テキスト、画像、映像、音声といったさまざまなメディアの、さまざまなな符合化形式) を用いて情報を公開できる。また、自由な情報の表現を可能にするために、情報表現形式の種類がさらに増えていくことは避けられない、またそれは許容されるべきである。しかし、これらの情報を取得して利用するユーザは、それぞれの情報表現形式に対応した表示ソフトウェア、もしくは変換ソフトウェアを多数用意しなくてはならない。また、限られた機能しか持たないクライアントではその情報を解読不可能な場合も少なくない。

本研究では、情報サーバから転送される情報の表現形式をユーザの環境で解読可能な形式へ柔軟に変換する中間システムを設計し、実装することによってこの問題を解決した。本システムは、代理サーバと変換サーバ (トランスレータ) によって構成される。代理サーバは、サーバとクライアントの間で情報転送の仲介を行ない、サーバから送られてくる情報の表現形式とクライアントで解読できる情報表現形式の判断を行なう。トランスレータは、代理サーバと通信して情報表現形式の変換を行なう。それぞれ固有の変換機能を持つトランスレータが多数存在し、代理サーバはそれらの中から目的の変換を行なうトランスレータを自動的に呼び出す。

本システムを利用することによって、ユーザは自分の現在の環境のままで、より多くの表現形式の情報を解読することが可能になり、情報提供をする側も安心して自由な情報表現形式で情報を公開することが可能になる。

キーワード

1　インターネット	2　情報表現形式	3　代理サーバ
4　変換	5　情報システム	

慶應義塾大学 大学院 政策・メディア研究科
林 周志

Abstract of Master's Thesis
Academic Year 1995

The Implementation of an Automatic Data Type Conversion System for Internet Applications

In this thesis, a flexible data type conversion system for Internet applications is described. On the Internet, it is possible to publicly present information using many different data types – many types of media(text, image, movie, audio, etc.) and many types of encoding schemes. The increase of data types is unavoidable and should be permitted in order to express information freely. However, users who wish to obtain these types of information must prepare many viewer applications or conversion utilities to decode. limited-function clients which often cannot decode a certain type of information.

In this research, this problem was solved by the design and implementation of a new medium system that can flexibly convert data received from a server into a data type that the client can handle. This system is composed a proxy server and conversion server (translator). The proxy server examines the data type of the information received from the server. The translator communicates with the proxy server, and performs the actual data type conversion. Many translators can exist on the Internet, with each having its own conversion functions. The proxy server can select an appropriate translator automatically.

By using this system, users can decode more data types on their current environment, while information providers can freely choose the appropriate data type which best expresses the information that they would like to open to the public.

Key Words

1 Internet
2 Data Type
3 Proxy
4 Conversion
5 Information System

Keio University Graduate School of Media and Governance
Hiroshi Hayashi

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 研究の動機

今日、インターネットは世界的な情報基盤として重要な地位を占めるようになってきており、この上で非常に多くの情報が公開されている。インターネット上での情報提供は、インターネットに接続されたコンピュータさえあればだれでも行なうことができ、それは世界中のインターネットのユーザからアクセス可能になる。さらに、情報の表現のためにテキスト、画像、音声、動画といったさまざまなメディアを自由に用いることができる。

しかし、逆にこれらの情報を取得し利用するときには、この情報の表現形式の多様さは障害となる場合が少なくない。情報は、さまざまなメディアのさまざまな形式で符合化されており、さまざまな言語で表現されている。また、ユーザがインターネットにアクセスするために用いている環境もさまざまで、UNIX ワークステーション、パーソナルコンピュータ (以下 PC と略す)、さらにはより小型で機能の限定された情報機器まで用いられる。そしてそれぞれで標準的に用いられているテキストや画像、動画、音声、ファイルアーカイブなどの符合化形式に違いがあり、このことがインターネット上で公開される情報の符合化形式の多様化に拍車をかけている。これらはみな、インターネット上で取得した情報を理解し、自分の知識として活用するための障害となっている。

たとえば、MS-Windows が稼働している携帯型 PC で旅先からインターネットにアクセスし、あるところから UNIX の標準的な圧縮アーカイブツールである、tar と gzip でまとめられ、圧縮されたアーカイブファイルを持ってきたとする。もし MS-Windows 用の tar と gzip がインストールされていなければ、まずそれらを探してインストールすることからはじめなくてはならない。または、中国語の文字コードで記述された WWW(World Wide Web) のページを見たいが、ユーザの環境、または WWW ブラウザでは、中国語の表示ができないという場合、まずは中国語が表示できる環境を整えることからはじめなくてはならない。

このように、ユーザはそれぞれの符合化形式に対応した表示ソフトウェアやユーザの持っている表示ソフトウェアで見ることができるようにするための符合化形式変換ソフトウェア、言語翻訳ソフトウェアを多数用意し、取得した情報に応じて適切なソフトウェアを選

択して使用しなくてはならない。これはユーザにとってかなりの負担となる。

1.2 本研究の意義・目的

本研究では上にあげたような問題を情報表現形式の変換を柔軟に行なうシステムを開発することによって解決する。つまり、ユーザの環境ではそのままでは解読することが難しい表現形式の情報も変換することによって解読可能な形式にする。さらに、情報表現形式の変換を情報の転送経路中で行なうことによって、ユーザが情報を取得した後で改めて変換を行なう必要がなくなる。また、ごく限られた機能しか持っていない端末でインターネット上で公開されている情報にアクセスする場合でも情報の表現形式をその端末が対応している形式に変換することによって表示することができるようになる。

ここでいう「情報表現形式」とは、

- メディア形式...テキスト、画像、音声、動画など
- 符合化形式...テキストの文字コード (JIS コード、EUC コードなど)、画像のの GIF 形式、TIFF 形式、JPEG 形式など
- 自然言語...日本語、英語など
- アプリケーション固有形式...tar によるアーカイブ形式、Postscript 形式など

を表わし、「情報表現形式変換」とは

- 同じメディア形式の異なる符合化形式への変換 (例: 画像情報の符合化形式を GIF 形式から JPEG 形式に変換する)
- 異なるメディア形式への変換 (例: 中国語 Big5 で符合化されたテキスト情報を GIF 形式で符合化された画像情報に変換する)
- 言語の翻訳 (例: ASCII 形式で符合化された英語のテキスト情報を JIS 形式で符合化された日本語のテキスト情報へ翻訳する)
- 異なるアプリケーション固有形式間の変換 (例:tar によるアーカイブを lha によるアーカイブに変換する)

を表わす。

1.3 本研究の概要

まず、情報を提供するサーバと情報を受信するクライアントとは独立したトランスレータによって変換を行なうモデルを提案する。トランスレータは、サーバからクライアントへの情報の転送経路の途中に介在する。しかし、1 つのトランスレータにすべての変換機

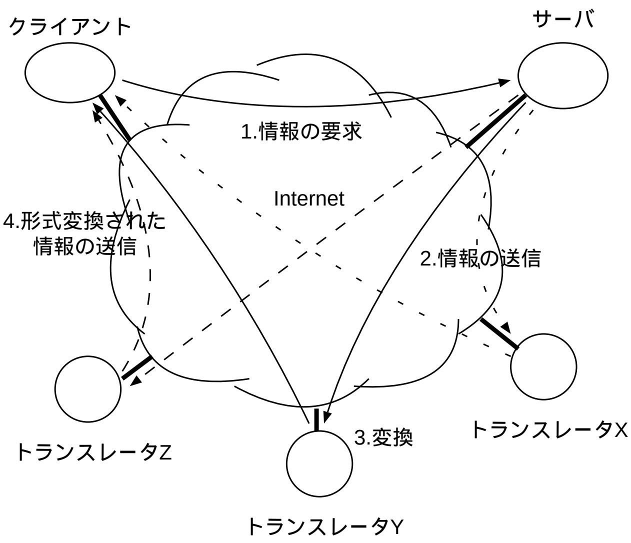


図 1.1: 本システムの基本モデル

能をもたせるのは現実的ではないので、トランスレータはインターネット上に散在しており、それぞれ固有の変換機能を持つモデルを考案した。(図 1.1)

サーバとクライアントは、インターネット上での情報提供に広く用いられている WWW、FTP のサーバとクライアントをそのまま利用することを前提とする。しかし既存のサーバ、クライアントにはトランスレータ呼び出し機能はない。そこで代理サーバ (proxy サーバ) を用い、これにトランスレータ呼び出し機能を付加することにした。代理サーバは、クライアントによって明示的に指定される。クライアントは、代理サーバを介して目的の情報の置かれているサーバにアクセスする。代理サーバは変換が必要であると判断した時にトランスレータを呼び出し、サーバから受け取った情報をトランスレータに送り、変換結果を受け取って、クライアントへ転送する。これを実現するためにはさらに解決しなければならない以下のような問題がある。

1. サーバから送られてくる情報の表現形式をどのように判断するか。また、クライアントで表示可能な、もしくはユーザが理解できる情報の表現形式をどのように判断するか。

2. インターネット上に散在するトランスレータの中から目的の変換機能を持つトランスレータをどのように発見するか。

1の問題は、WWW の場合はプロトコルヘッダ情報を用いることによって解決し、FTP の場合はコマンドの記述方法を拡張することによって解決した。2の問題は、IP マルチキャストを用いることによって解決した。トランスレータはすべてひとつのマルチキャストグループに属しており、代理サーバは変換が必要になった時にこのマルチキャストグループ

へ変換要求をアナウンスする。その変換機能を持つトランスレータは自分が変換できることを代理サーバに知らせる。代理サーバは一番早く応答したトランスレータに情報を転送し、変換を依頼する (図 1.2)。

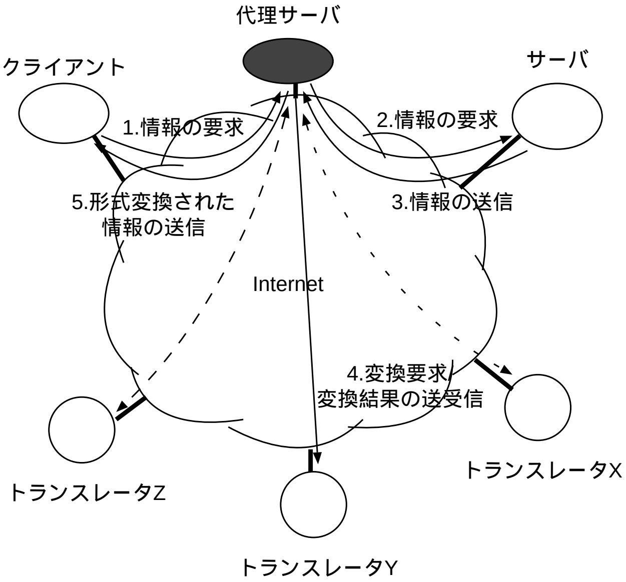


図 1.2: 代理サーバを用いたシステムのモデル

第 2 章

情報表現形式変換について

2.1　情報表現形式変換のモデル

インターネット上での情報提供システムの多くは、クライアント・サーバモデルで構築されている。情報はサーバ上に置かれており、ユーザはクライアントを利用してサーバ上の情報を取得する。このシステムの中のどこで情報表現形式の変換を行なうかによっていくつかの方式に分類できる。

2.1.1　ユーザ側で変換する方式

もっとも一般的な方法は、情報を取得したユーザが自分の環境で再生できる形式に変換する方法である。これはユーザがその変換を行なうことのできる変換ソフトウェアを持っており、かつその使い方を知っている場合はもっとも効率がよい方法であるといえる。しかし適切な変換ソフトウェアを持っていない、もしくは使い方が分からない場合、まずその変換ができるソフトウェアを探して入手することから始めなくてはならない。これはユーザにとってかなりの負担になる (図 2.1)。

2.1.2　ネットワーク標準形式を規定する方式

つぎによく用いられる方法は、ネットワーク標準の符合化形式を定める方法である (図 2.2)。ネットワークバイトオーダや XDR(eXternal Data Representation)[1] もネットワーク標準形式の一例といえる。これが定められていれば、ユーザは各メディアに関してひとつの符合化形式に対応した解読ソフトウェアを持っていればよいことになる。しかし、この方法では情報のサーバ側とクライアント側の両方で表現形式の変換が必要になる場合もある。また、ネットワーク標準の符合化形式が十分な表現能力を持っていないと、変換する際に元の情報の一部が欠落する可能性もある。

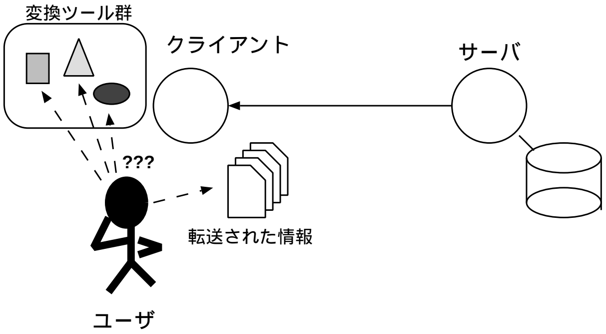


図 2.1: ユーザ側で変換する方法

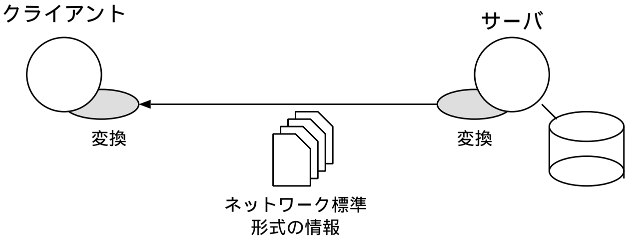


図 2.2: ネットワーク標準形式

2.1.3 サーバ側で変換する方式

表現形式の変換をサーバ側で行なうという方法も考えられるが、これはほとんど用いられていない。さまざまなクライアントを想定して、多数の変換機構をサーバに持たせるのは現実的ではない。しかし、応用によっては有効に働く機構である。たとえば、wuftpd という FTP サーバソフトウェアは、サーバ上に ‘foo’ というファイルがあったとき、クライアントから ‘foo.gz’ の送信要求を受け取ると、ファイル foo を gzip で圧縮してからクライアントに送信する。また、‘bar’ というディレクトリがあったとすると、クライアントから ‘bar.tar.gz’ の送信要求があるとディレクトリ ‘bar’ 以下を tar コマンドでひとつにまとめ、gzip で圧縮してからクライアントに送信する。このようにして実際にネットワーク上を流れる情報量を減らすことができる。

2.1.4 サーバ、クライアントとは独立したエンティティで変換する方式

サーバ–クライアントの情報転送経路中に、サーバ、クライアントとは独立した情報表現形式変換用のエンティティを介在させる方法 (図 2.3)。このエンティティのことをここでは「トランスレータ」もしくは「変換サーバ」と呼ぶことにする。この方法は他の方法に

比べ、情報の転送経路が冗長になるが、情報の転送中に変換が行なわれるので、ユーザが情報を取得してからさらに何らかの変換ツールを利用して変換するという手間がなくなる。また、トランスレータはネットワーク上にあり、どのユーザからも利用できるという利点がある。この方式は、一部の WWW の代理サーバ (Proxy server) によって実現されている。クライアントは直接目的のサーバにアクセスせずに、代理サーバにアクセスすると、代理サーバが代わりに目的のサーバにアクセスして必要な情報を取得する。本来は、セキュリティなどの問題でクライアントが直接目的のサーバにアクセスできない環境下でクライアントに代わってサーバにアクセスしたり、サーバの情報をキャッシュして、ネットワークへの負荷を小さくしたり、レスポンスを向上させたりするのが主な目的であった。後述する「DeleGate」や一部の英日翻訳ソフトウェアは代理サーバとして稼働し、文字コードの変換、ホームページの和訳などの表現形式変換機能を併せ持っている。

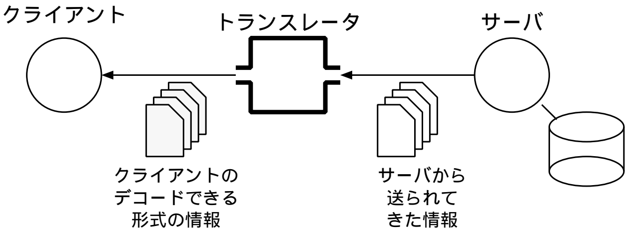


図 2.3: トランスレータによる変換

2.2 関連研究

情報表現形式変換を意識したシステムの例を紹介する。

2.2.1 アプリケーションによる対応

多くの人気のあるアプリケーションソフトウェアは、多くの情報表現形式の情報に対応していることを売り物にしている。例えば、画像表示ツールの xv は、GIF、TIFF、JPEG、Postscript、XPM、X11 bitmap、BMP など多くの形式の画像を表示可能である。WWW クライアントの netscape は、画像は、GIF 形式、JPEG 形式に対応し、テキストは、英語の ASCII コードだけでなく、日本語の JIS、EUC、シフト JIS コード、中国語、韓国語などの文字コードにも対応している。しかし、新たな情報表現形式に対応するためにはソフトウェア自身のバージョンアップを待たなければならない。それで、外部プログラムを登録することによって新しい情報表現形式に対応できるアプリケーションもある (netscape、xv にもそのような機能がある)。もちろんそのような外部プログラムは、ユーザが入手してこなければならない。

2.2.2 情報表現形式変換ツール

ある形式から別の形式への変換を行なうツールは、非常に多く存在している。ユーザはこのようなツールを用いて、インターネット上で取得した情報を、自分の利用している環境で正しく解読できる形式に変換してから利用する。有名なものとしては、以下のようなものがある。

- **nkf...**日本語の文字コード (JIS、EUC、シフト JIS) 間の変換ツール
- **pbmplus...**画像のさまざまな符合化形式に対応したツール群。組み合わせて使うことによって、さまざまな画像の符合化形式間の変換ができる
- **sox...**音声の符合化形式間の変換ツール

これらのツールは有名であるとはいえ、利用している OS には標準で利用できることは少なく、まず何らかの方法で入手して (FTP、CDROM による配布など)、インストールして初めて使うことができる。また、ある形式から別のある形式への変換ツールとして、何があり、どこから入手できるかを調べ、実際に入手してインストールするのは、ユーザにとってかなり大きな負担である。

2.2.3 DeleGate

電総研の佐藤豊氏が開発した「DeleGate」[2] は本研究で採用したのと同じトランスレータによる変換モデルをとるシステムである。DeleGate はさまざまなプロトコルの中継し、ファイアウォールを越えてサーバにアクセスする、サーバにあるデータのキャッシュを行なう、文字コードの変換をするといった機能を持つ。文字コード変換機能としては以下のようなものがある。この機能は、HTTP[3] と Gopher[4] プロトコルについてのみ有効である。

- 任意の日本語文字コード (JIS、シフト JIS、EUC) から指定された文字コード (JIS、シフト JIS、EUC のいずれか) への変換する機能
- テキストの 1 文字 1 文字を画像にして HTML に埋め込む CII(Character by Inline Image) 機能。これは HTTP に対してのみ有効である。

これらの機能によって特定の日本語文字コードにしか対応していないクライアント、さらには日本語、その他の英語以外の言語の文字コードに対応していないクライアントでも対応していない文字コードでエンコードされたテキストを表示させることができる。

クライアントから利用する場合は、DeleGate の稼働しているホスト名と目的の変換を行なうポート番号を指定する。例えば、UNIX 用の NCSA Mosaic で利用する場合は、環境変数 `http_proxy` に以下のように指定する (ポート番号と変換機能の対応は設定によって異なる)。

`http://delegatehostname:10080/` (JIS コードへ変換する場合)
`http://delegatehostname:10081/` (EUC コードへ変換する場合)
`http://delegatehostname:10082/` (シフト JIS コードへ変換する場合)
`http://delegatehostname:10083/` (ビットマップへ変換する場合)

2.3 本研究の特徴

代理サーバ方式をとるという点では、DeleGate とよく似ている。しかし、DeleGate は文字コード変換機構を代理サーバに内包しているのに対し、本システムでは代理サーバ自身は変換機能を持たず、代理サーバはトランスレータを呼び出す役目を持っているだけである (図 2.4)。

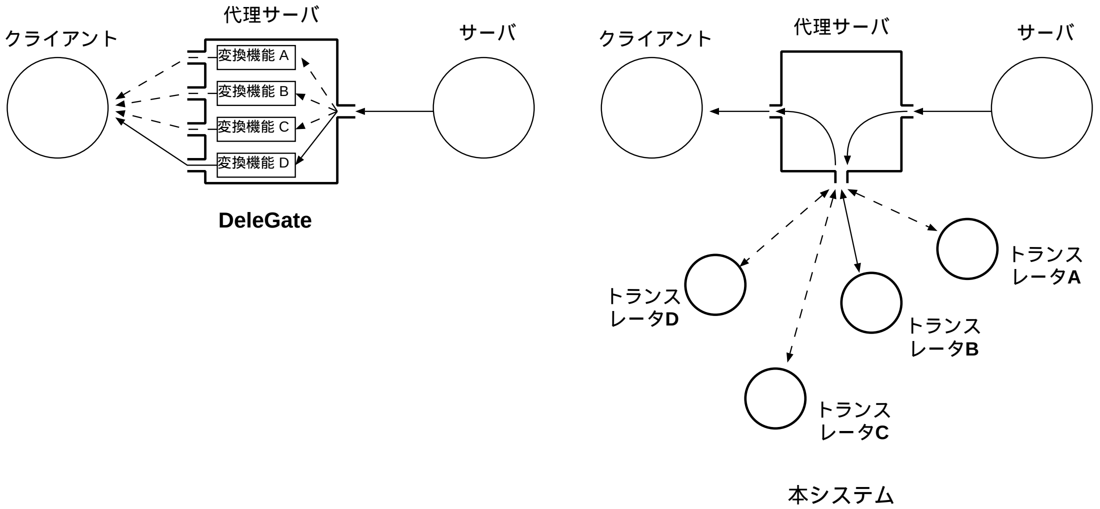


図 2.4: DeleGate と本システムの相違

このように変換機能をサーバ、クライアントから独立させ、代理サーバとトランスレータで役割分担させることによって以下のような利点が生まれる。

- ユーザからは代理サーバが万能変換システムに見える。目的の変換を行なうトランスレータがどこにあるか意識する必要はない。
- トランスレータはだれでも立ち上げることができ、それをインターネット上で立ち上がっているすべての代理サーバから利用できる。
- 新たな情報表現形式から既存の情報表現形式への変換に対応するためには、インターネット上のどこかでそれに対応したトランスレータが立ち上がればよい。

- Macintosh、Windowsなど特定のアーキテクチャのマシンでしか利用できない変換ソフトウェアでも、トランスレータとして利用することができればインターネットに接続されているすべてのアーキテクチャのマシンから利用できる

- 少数の情報表現形式にしか対応していない単純なクライアントでも、多くの情報表現形式に対応可能になる

- 代理サーバとトランスレータに分かれ、トランスレータは代理サーバとのみ通信を行なうので、FTP、WWWなどのプロトコルへの対応は代理サーバが行ない、トランスレータ-代理サーバ間のプロトコルは、サーバ-クライアント間のプロトコルに依存しない。新たに別のプロトコルの情報提供システムに対応するためにはそのプロトコルに対応した代理サーバを作成すればよく、トランスレータに変更を加える必要はない

- トランスレータとして、既存のほとんどの情報表現形式変換ソフトウェアのほとんどを利用することが可能であるため、容易に機能拡張することができる。

第 3 章

設計

3.1 本情報提供システムのモデル

インターネット上の情報提供システムはたいてい、サーバに情報が置かれており、クライアントを使ってユーザは必要な情報を取得する、クライアント・サーバモデルを採用している。クライアントが情報を要求し、サーバが情報を送信する。これを拡張し、ユーザの環境で正しく表示/再生できる情報表現形式で情報を取得することができるシステムとして、サーバ、クライアントにトランスレータを加えたモデルを考案した (図 3.1)。

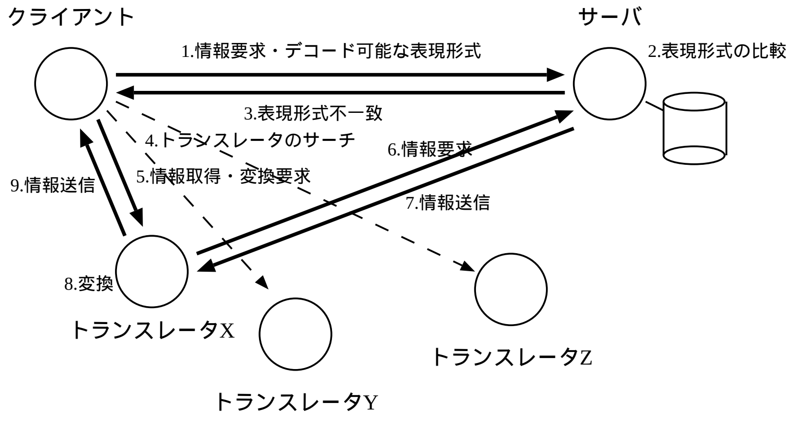


図 3.1: 本研究の情報提供システムの基本モデル

このモデルでの通信手順は以下のようになる。

1. クライアントはサーバに情報を要求するとともに、自分の環境で正しく表示/再生できる情報表現形式のリストを送信する。
2. サーバは、要求のあった情報の表現形式とクライアント側で表示/再生できる形式を比較し、一致すればそのまま送信する。

3. 一致しなかった場合、クライアントは、サーバ上にある情報の表現形式からクライアント側で解読できる表現形式への変換ができるトランスレータを探す。

4. 適切なトランスレータが見つかった場合、トランスレータがサーバからデータを取得し、クライアントの要求する表現形式に変換してクライアントに送信する。

しかし、これをそのまま実現しようとするとサーバ、クライアントも全く新たに設計しなおさなくてはならない。それよりも現在普及している情報提供システムのサーバ、クライアントをそのまま利用できる方が好ましい。それで既存のシステム、とくに広く使われている、FTP と WWW のサーバ、クライアントはそのままで、トランスレータを介在させる方法を考案した。

上のモデルを既存のシステムを利用して実現するときの問題点としては以下のようなものがある。

- クライアントで解読できる情報表現形式をサーバに伝え、サーバが自分のところにある情報の表現形式を知って、それがクライアントで解読可能かどうか判断することができない。
- クライアントに外部のトランスレータを呼び出す機能がない。

これらの問題を解決するために、代理サーバ (Proxy サーバ) を利用するモデルを考案した。クライアントは常にこの代理サーバを介してサーバにアクセスする。代理サーバは、クライアントで解読できる情報表現形式、クライアントから要求のあったサーバ上の情報の表現形式を判断し、それらが一致しない場合は、トランスレータを探して変換を依頼する (図 3.2)。

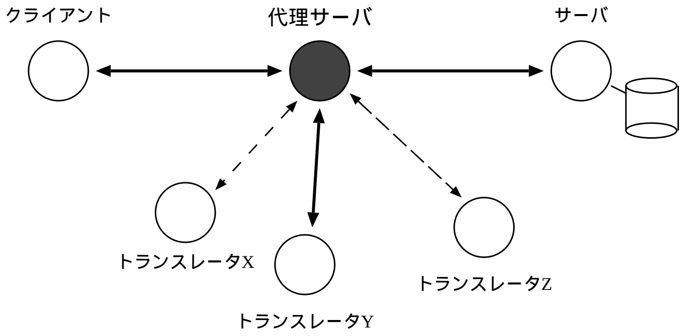


図 3.2: 代理サーバを利用した情報提供システムのモデル

情報表現形式変換に関わることは代理サーバを用いることによって、クライアントとサーバは既存のものをそのまま用いることが可能となる。また、既存のクライアント–サーバ間で用いている通信プロトコル (FTP や HTTP) への対応は代理サーバで行なうことがで

き、代理サーバ-トランスレータ間のプロトコルは、クライアント-サーバ間のプロトコルに依存しない。つまり、別のプロトコルの情報提供システムでトランスレータを利用できるようにするためには、そのプロトコルに対応した代理サーバを作成すればよく、個々のトランスレータに変更を加える必要はないという利点もある。さらに代理サーバでサーバ上の情報のキャッシュを持たせたり、セキュリティへの配慮からファイヤ・ウォールを設定している組織内からファイヤ・ウォールを越えた先のサーバにアクセスする機能を付加させることも可能である。反面、最初の1回だけではあるが、クライアントが明示的に代理サーバを指定しなければならないことになる。また、代理サーバとトランスレータ間を情報が往復することになるので、情報の送信経路が基本モデルよりさらに冗長になるという欠点がある。

3.2 代理サーバの設計

本研究では、FTPプロトコル[5]とHTTPプロトコルに対応した代理サーバを設計した。代理サーバは、以下の機能を持たねばならない。

1. クライアント側で解読可能な情報表現形式と、サーバから送られてくる情報の表現形式を判断する(情報表現形式の判断)
2. 適切なトランスレータを発見する(トランスレータの発見)
3. サーバから送られてきた情報をトランスレータに転送し、変換結果を受け取り、クライアントに転送する(トランスレータとの通信)

これらの機能の設計について以下で述べる。

3.2.1 FTPの場合の情報表現形式の判断

クライアント側で解読できる情報表現形式の判断

FTPプロトコルには、クライアント側で解読できる情報表現形式を判断するのに助けとなるような情報をサーバ側に伝えるメッセージは何も規定されていない。それで、ユーザ自身が手動で解読できる情報表現形式を伝える必要がある。

サーバ上の情報の表現形式の判断

FTPサーバから送られる情報の表現形式を代理サーバで判断する方法としては、以下のようなものが考えられる。

1. FTPサーバから目的の情報(もしくはその一部)を受け取り、その内容を調べて表現形式を判断する

2. 目的の情報のファイル名の拡張子部分から判断する
3. クライアント側からユーザが代理サーバに伝える

1の方法は、かならずしも情報表現形式を正確に判断できるとは限らない。2の方法は、たいていの場合、ファイル名の最後に付けられる拡張子がそのファイルの情報表現形式を表わしていることを利用したものだが、テキストファイルの場合、‘manual.txt’ など、それがテキストであることは分かるが、どのような言語のどんな文字コードで記述されているかまでは分からないことが多い。3は、ユーザ自身に情報表現形式の判断を任せる方法である。これは自動的とはいえない方法だが、他の方法に比べて確実な方法である。それで本研究で設計した FTP 用代理サーバは 3の方法を採用することにした。

3.2.2 HTTP の場合の情報表現形式の判断

クライアント側で解読できる情報表現形式の判断

クライアントからの HTTP リクエストメッセージには以下のようなフィールドがある。

- ‘Accept:’...クライアントが受け入れられるメディア形式
- ‘Accept-Language:’...クライアントが受け入れられる自然言語

メディア形式は、

メディアの種類/サブタイプ

という形式で表現される。たとえば、

`text/plain`	...プレインテキスト
`text/html`	...HTML(HyperText Markup Language) 形式のテキスト
`image/gif`	...GIF エンコーディングの画像
`image/jpeg`	...JPEG エンコーディングの画像
`video/mpeg`	...MPEG エンコーディングの動画
`audio/x-wav`	...WAV エンコーディングの音声

のように記述される。この表記方法は、RFC1700[7] に従っている。また、自然言語の表記方法は、ISO639[8] に従う。メディア形式には、

`text/plain;charset=ISO-8859-1`

のようにさらにパラメータを付加することも可能である。

これらが十分な情報を含んでいれば、情報表現形式の判断に利用できる。しかし、これらが厳密に実装されている例は少ない。実際にクライアントから送信されるリクエストメッセージのダンプを表 3.1、3.2、3.2に示す。

それで、‘Accept:’ および ‘Accept-Language:’ フィールドでクライアントの解読できる情報表現形式を判断することはあきらめた。その理由は以下の通りである。

表 3.1: 各種 WWW クライアントのリクエストメッセージ:その 1

- **Netscape 2.0Beta4 (X11 用)**

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/2.0b4 (X11; I; BSD/OS 2.0 i386)
Host: localhost:9999
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

- **Netscape 2.0Beta4 (Win32 用)**

```
GET / HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/2.0b4 (Win95; I)
Host: rinnote
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

- **Microsoft Internet Explorer**

```
GET / HTTP/1.0
Accept: */*, image/x-xbitmap, image/jpeg, image/gif
Accept-Language: jp , en
User-Agent: Microsoft Internet Explorer/4.40.425 (Windows 95)
```

- **lynx 2.3.7Beta**

```
GET / HTTP/1.0
Accept: */*
Accept: application/x-wais-source
Accept: application/html
Accept: text/plain
Accept: text/html
Accept: www/mime
Accept: video/mpeg
Accept: image/jpeg
Accept: image/x-tiff
Accept: image/x-rgb
Accept: image/x-xbm
Accept: image/gif
Accept: application/postscript
User-Agent:  Lynx/2.3.7 BETA  libwww/unknown
```

表 3.2: 各種 WWW クライアントのリクエストメッセージ:その 2

- **NCSA Mosaic 2.4 (X11 用)**

```
GET / HTTP/1.0
Accept: text/plain
Accept: application/x-html
Accept: application/html
Accept: text/x-html
Accept: text/html
Accept: audio/basic
Accept: audio/x-aiff
Accept: image/gif
Accept: image/jpeg
Accept: image/tiff
Accept: image/x-portable-anymap
Accept: image/x-portable-bitmap
Accept: image/x-portable-graymap
Accept: image/x-portable-pixmap
Accept: image/x-rgb
Accept: image/rgb
Accept: image/x-xbitmap
Accept: image/x-xpixmap
Accept: image/xwd
Accept: image/x-xwd
Accept: image/x-xwindowdump
Accept: video/mpeg
Accept: application/postscript
Accept: application/x-dvi
Accept: message/rfc822
Accept: application/x-latex
Accept: application/x-tex
Accept: application/x-texinfo
Accept: application/x-troff
Accept: application/x-troff-man
Accept: application/x-troff-me
Accept: application/x-troff-ms
Accept: text/richtext
Accept: text/tab-separated-values
Accept: text/x-setext
Accept: */*
User-Agent:  NCSA Mosaic for the X Window System/2.4 (L10N-2.4.0)  libwww/2.12 modified
```

表 3.3: 各種 WWW クライアントのリクエストメッセージ:その 3

- **Emacs w3-mode**

```
GET / HTTP/1.0
MIME-Version: 1.0
HTTP-Version: 1.0
Extension: Security/Digest
Session-ID: <RIN.199614225902@timothy.wide.sfc.keio.ac.jp>
From: rin@timothy.wide.sfc.keio.ac.jp
Accept-encoding: base64, x-gzip, 7bit, 8bit, binary, x-compress, x-hqx, smalld-printable
Accept-language: */*
Accept: multipart/alternative, multipart/mixed, multipart/*
Accept: application/octet-stream, application/dvi
Accept: application/dvi, application/emacs-lisp, application/x-tar
Accept: application/latex, application/tex, application/texinfo
Accept: application/x-gzip, application/zip
Accept: application/postscript, application/postscript
Accept: application/x-www-pem-reply, application/x-www-pgp-reply, audio/*
Accept: audio/*, image/x-xwd, image/x11-dump, image/windowdump
Accept: image/*, text/plain, text/plain, text/plain
Accept: text/enriched, text/html, video/mpeg, archive/tar
User-Agent: Emacs-W3/2.1.100 URL/1.255 (X11 ; i386-ibmpc-bsdi2.0)
Message-ID: <RIN.199614230314@timothy.wide.sfc.keio.ac.jp>
```

- **CERN-LineMode**

```
GET / HTTP/1.0
Accept: */*; q=0.300
Accept: application/octet-stream; q=0.100
Accept: text/plain
Accept: text/html
Accept: image/jpeg
Accept: image/x-tiff
Accept: image/gif
Accept: application/postscript
From: rin@timothy.wide.sfc.keio.ac.jp
User-Agent: CERN-LineMode/unspecified  libwww/unknown
```

- ほとんどのクライアントは‘Accept:’フィールドに、‘*/*’、つまりすべてのメディア形式を受け入れると指定している
- ‘Accept-Language:’フィールドをメッセージに含まないクライアントも多い
- どちらのフィールドでも、受け入れられる文字コード体系は分からない
- HTTP Version 1.0の最新の仕様ドラフト[3]では、‘Accept:’、‘Accept-Language:’フィールドの規定はなくなった

現状で、クライアントの受け入れられる情報表現形式を判断する方法としては、クライアントからのリクエストに含まれる、‘User-Agent:’フィールドを利用する方法が考えられる。ここにはクライアントの種別が記述される。代理サーバはこのフィールドを見て、そのクライアントの受け入れられる情報表現形式を推測することができる。

さらに、ユーザの要求、たとえば英語のホームページを日本語に翻訳してほしいといった要求をクライアントから代理サーバに伝える方法が必要である。これはクライアントからのリクエストメッセージのあるフィールドを使うという方法では不可能である。これを可能にするためにはクライアント自体に改良を加えなくてはならない。これは本研究での設計方針に反する。そこで、何らかの方法でユーザから前もって、「サーバからの情報が表現形式Xであれば、表現形式Yに変換してから送信する」という要求、たとえば、「サーバから送られてきたホームページが英語で記述されていたら日本語に変換してクライアントに送信する」といった要求を受け付けておくという方法が考えられる。しかし、HTTPはステートレスのプロトコルであり、クライアントが代理サーバにリクエストを出し、代理サーバがクライアントにレスポンスを返すと通信のコネクションはすぐに切れてしまう。そのため、代理サーバがクライアントに関する情報を保持することが難しい。DeleGateなどでは、変換機能(任意の日本語文字コードからシフトJISコードへの変換、任意の文字コードからgifイメージへの変換など)ごとにポート番号を変えてクライアントからのコネクションを待ち受けるという方法をとっている。この場合、ユーザは目的の変換を行なうポート番号を代理サーバのポート番号として指定する。しかし、ユーザの一時的な要求に対応する場合、クライアントによっては代理サーバのポート番号の変更は、クライアントの再起動が必要になるのであまり好ましいとはいえない。

そこで、ユーザは前もって代理サーバに変換要求を伝え、代理サーバはそれを保存し、代理サーバはクライアントからの接続があるたびに、クライアントのホスト名を調べ、保持されている変換要求の中にそのホストのユーザからの要求があれば、それにしたがって変換を行なうことにした。クライアントがUNIXの場合、複数のユーザが同一のホストからWWWクライアントを利用して代理サーバにアクセスしている可能性があるのでこの方法は完全とはいえないが、PCやMacintoshでは1台のマシンのユーザは1人であるし、UNIXマシンでもたいていの場合、1台のマシンからWWWクライアントを用いているのは1人だけであると仮定してこの方法を採用することにした。

ほかにも、リクエストメッセージの‘From:’フィールドを用いれば、マシンだけでなく、ユーザ名まで区別することができるが、すべてのクライアントがこのフィールドをつけ

てリクエストを送ってくるわけではない。また、一部のWWWクライアント((Netscape Version 2.0Beta4など)では、リクエストメッセージに‘Proxy-Connection: Keep-Alive’というフィールドがあり、代理サーバ側も対応していれば、コネクションをずっと保持することができる。これが利用できればユーザの変換要求を保持することも容易だが、いまのところほとんどのクライアントではサポートされていない

サーバ上の情報の表現形式の判断

サーバからのHTTPレスポンスメッセージのヘッダには以下のようなフィールドがある。

- ‘Content-Type:’...送られる情報のメディア形式
- ‘Content-Encoding:’...送られる情報の符合化形式

‘Content-Encoding:’に記述できる符合化形式として、HTTP Version 1.0のドラフトでは以下の2つだけが定義されている。

- x-gzip ... gzipコマンドによる圧縮形式
- x-compress ... compressコマンドによる圧縮形式

以前のHTTP Version 1.0の仕様ドラフトでは、これに加えて

- ‘Content-Language:’...送られる情報の自然言語

というフィールドがあったが、これを送出するhttpサーバはほとんどない。

それで、テキスト情報以外では、ほとんど‘Content-Type:’フィールドで判断できるのでこれを利用することにした。しかし、テキスト情報の場合、どの言語のどんな文字コードで表現されているかを示す情報がヘッダにはなにもない。これは情報の内容を直接調べるか、もしくはユーザに指定してもらうという方法で対処することにした。

3.2.3 トランスレータの発見

情報表現形式を判断し、変換が必要であることが分かったら、その変換を行なうことのできるトランスレータを発見しなくてはならない。トランスレータは、インターネット上のさまざまなところで任意に立ち上げられるものと仮定する。代理サーバが適切なトランスレータを発見する方法としては以下のようないくつかの方法が考えられる。

1. 代理サーバに人手で登録する: トランスレータが立ち上げられるたびにニュース、メールなどでアナウンスされ、代理サーバの管理者がそれを見て代理サーバの設定ファイルにトランスレータのアドレスと対応している変換などを登録する。代理サーバはその設定ファイルを参照してトランスレータを探す。

2. **トランスレータの階層化:** すべてのトランスレータが登録されているルートトランスレータをいくつか定め、新たに立ち上げられたトランスレータはルートトランスレータに自分のアドレス、対応している変換などを登録する。代理サーバは、ルートトランスレータに問い合わせて適切なトランスレータを知る。

3. **IP マルチキャスト [9] を用いる:** トランスレータはみなひとつのマルチキャストグループに属する。つまり、トランスレータはみな同じマルチキャスト IP アドレスを持つ。代理サーバはそのマルチキャストグループに変換要求を送信すると、その変換に対応できるトランスレータが応答する。

ここでは、以下の理由により IP マルチキャストを用いる方法を採用した。

- 新たなトランスレータの立ち上げ時に何らかの登録作業は不要である
- マルチキャストグループへ送信するメッセージは、発信者 (代理サーバ) を中心にネットワークを放射状に広がるので、同じ変換機能を持つトランスレータが多数存在したとしても、ネットワーク的に代理サーバにより近い位置にあるトランスレータが見つかる可能性が高い

反面、以下のような欠点もある。

- IP マルチキャストに対応したオペレーティングシステム (以下 OS) は多くない
- MBone[10][11] (現在のユニキャストのインターネット上で、トンネリングなどの技術を用いて仮想的に構築されたマルチキャストバックボーン) に接続されていなければならない
- UDP なので信頼性が低い

しかし、サーバ、クライアントはマルチキャストに対応している必要はなく、代理サーバとトランスレータだけが対応していればよいので現状でもそれほど問題にならないと判断した。また、マルチキャストはマルチメディアデータの送信のための有望な通信技術として注目されているので近い将来にはより多くの OS でサポートされると考えられる。

3.3 トランスレータの設計

トランスレータには、

1. 代理サーバからの変換要求を待ち受ける
2. 代理サーバから情報を受け取り、変換して送り返す

という機能が必要である。

1については、前述のようにマルチキャストを用いるので、トランスレータは起動時に共通のマルチキャストアドレスをつけて、代理サーバからの要求を待ち、変換要求があったらその変換が自分に行なうことができるかどうかチェックし、可能な場合は代理サーバに応答メッセージを返す。

2のフェーズは、マルチキャストで行なわれる必要はないので、代理サーバによって選択されたトランスレータは、1対1のコネクションを張り、そのコネクション上で情報のやりとりをすればよい。コネクションの張り方と情報の受渡しの方法としては以下のようなものが考えられる。

- 1本のコネクションで、まず代理サーバからトランスレータに情報をすべて送り、同じコネクションで変換結果をトランスレータから代理サーバへ送り返す。
- 1本のコネクションで、代理サーバ-トランスレータ間で、情報をある大きさに区切ったブロック単位で転送する。
- 複数のコネクションを張り、1つのコネクションで代理サーバからトランスレータに情報を送り、別のコネクションでトランスレータから代理サーバへ変換結果を送り返す。

情報表現形式変換プログラムは、

1. 入力されたデータストリームを順次変換し、出力できるもの
2. いったんすべてのデータを取り込んでから変換を行ない変換結果を出力するもの

の2種類に分けられる。前者の例としては、同じ自然言語の文字コード変換(JISからシフトJISへの変換など)があり、後者の例としては、アーカイブ形式変換(tarによるアーカイブからlhaによるアーカイブへの変換など)がある。ただでさえ情報の転送経路が冗長になるので、代理サーバ-トランスレータ間の情報転送はできるだけ効率よく行なえるようにしたい。そこで代理サーバからトランスレータへの情報転送、トランスレータから代理サーバへの変換結果の転送をそれぞれべつのコネクションで行なうことにした。

3.4 代理サーバ-トランスレータ間通信プロトコル

3.4.1 通信手順の概要

代理サーバが、サーバ上にある情報の表現形式Xからクライアントで解読可能な情報表現形式Yへの変換が必要であると判断したときに起きる代理サーバ-トランスレータ間の通信手順の概要を以下にまとめる。通信は、大きく分けて3つのフェーズに分けられる。

1. トランスレータ発見フェーズ

    (a) 代理サーバは、トランスレータの属するマルチキャストグループに、X から Y への変換要求を送信する
    (b) 変換要求を受け取ったトランスレータは、自分に X から Y への変換ができる場合は、自分が変換できることをユニキャストで代理サーバに知らせる

2. 変換要求フェーズ
    (a) 代理サーバは、最も早く応答のあったトランスレータに TCP のコネクションを張り、あらためて変換要求を出す
    (b) コネクションが張られ、変換要求を受け取ったトランスレータは、代理サーバに確認応答を返す

3. 変換フェーズ
    (a) 代理サーバは、トランスレータへデータ送信用と受信用の 2 つのコネクションを張る (最初のコネクションは制御用に用いる)
    (b) 代理サーバは、データ送信用コネクションへサーバから送られてきた情報を転送する
    (c) トランスレータはデータを受け取り、変換結果を (代理サーバから見た) データ受信用コネクションへ順次送信する
    (d) 代理サーバは、トランスレータから受け取った表現形式変換されたデータを順次クライアントへ転送する

3.4.2 メッセージフォーマット

メッセージ中の特殊文字は以下のように表現する。

`<SP>` ASCII 文字のスペース (16 進数で 0x20)
`<CR>` ASCII 文字の復帰 (16 進数で 0x0D)
`<LF>` ASCII 文字の改行 (16 進数で 0x0A)

また、メッセージ中の ASCII アルファベットは大文字小文字の区別をしない。

最初のトランスレータ発見フェーズで代理サーバ-トランスレータ間で送受信されるメッセージは以下の通りである。

- 代理サーバがトランスレータ発見のためにマルチキャストする要求メッセージ

  `WHOCAN<SP>元の情報表現形式<SP>変換後の情報表現形式<CR><LF>`

- 変換可能なトランスレータから代理サーバへの応答メッセージ。これは UDP データグラムで送られる

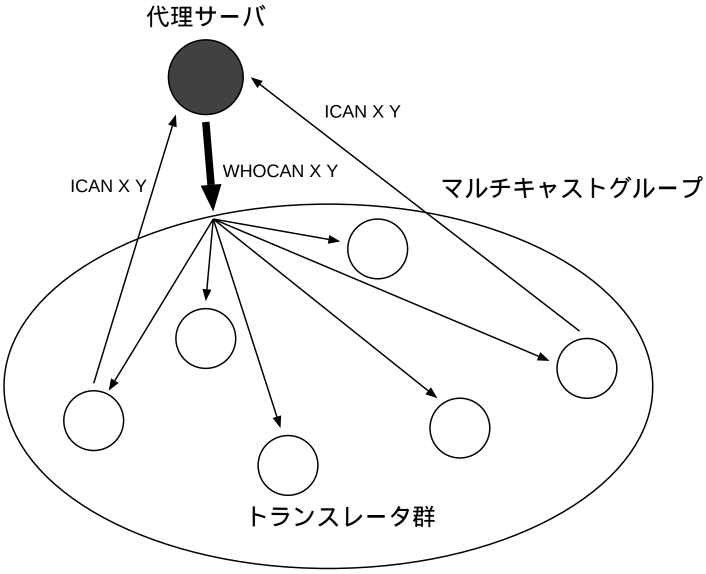


図 3.3: トランスレータ発見フェーズの通信

`ICAN<SP>元の情報表現形式<SP>変換後の情報表現形式<CR><LF>`

変換要求フェーズで送受信されるメッセージは以下の通りである。これらは代理サーバからトランスレータ発見フェーズで選択されたトランスレータへ張られた TCP コネクション上で送受信される。

- 代理サーバからトランスレータへ変換を要求するメッセージ

  `TRANS<SP>元の情報表現形式<SP>変換後の情報表現形式<CR><LF>`

- トランスレータから代理サーバへの応答メッセージ。
  - 変換要求受理メッセージ

    `ICAN<SP>ポート番号<CR><LF>`

    ポート番号は、トランスレータが代理サーバからのデータ送信用のコネクション要求を待ち受けるポートの番号である。
  - 変換要求拒否メッセージ

    `ICANT<CR><LF>`

    これが用意されているのは、代理サーバでトランスレータのアドレスがキャッシュされているなどの理由で、トランスレータ発見フェーズと変換要求フェーズの間にかなりの時間が経過した場合、その変換機構が無効になっている可能

性があるためである。この場合、代理サーバはトランスレータ発見フェーズからやり直すことになる。

続く変換フェーズではまず、代理サーバはトランスレータからの ‘ICAN’ メッセージで指定されたポート番号に対して、データ送信用コネクション、続いてデータ受信用コネクションを張る。代理サーバ-トランスレータ間には、合計 3 つの TCP コネクションが張られることになる。最初に張られたコネクションは制御用に用いられる。変換フェーズで制御用コネクション上で送受信されるメッセージは以下の通りである。

- 代理サーバからトランスレータへ変換中止を指示するメッセージ

  ```
  ABORT<CR><LF>
  ```

  これを受信したらトランスレータは変換を中止し、すべてのコネクションをクローズする。

- トランスレータから代理サーバへ変換時にエラーが生じたことを知らせるメッセージ

  ```
  ERROR<CR><LF>エラーメッセージ<CR><LF>
  ```

  これを受信したら代理サーバはトランスレータへのデータ送信を中止し、すべてのコネクションをクローズする。

- トランスレータから代理サーバへ変換が正常に終了したことを知らせるメッセージ

  ```
  OK<CR><LF>
  ```

変換フェーズでは、サーバ、クライアント、代理サーバ、トランスレータの接続関係は図 3.4のようになっている。

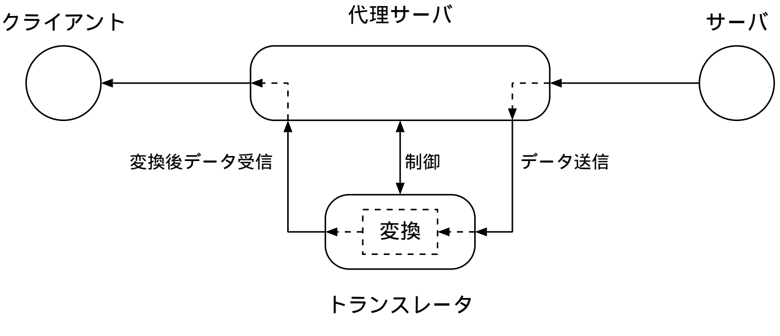


図 3.4: 変換フェーズでの各エンティティの関係

代理サーバの状態遷移図は図 3.5のようになる。

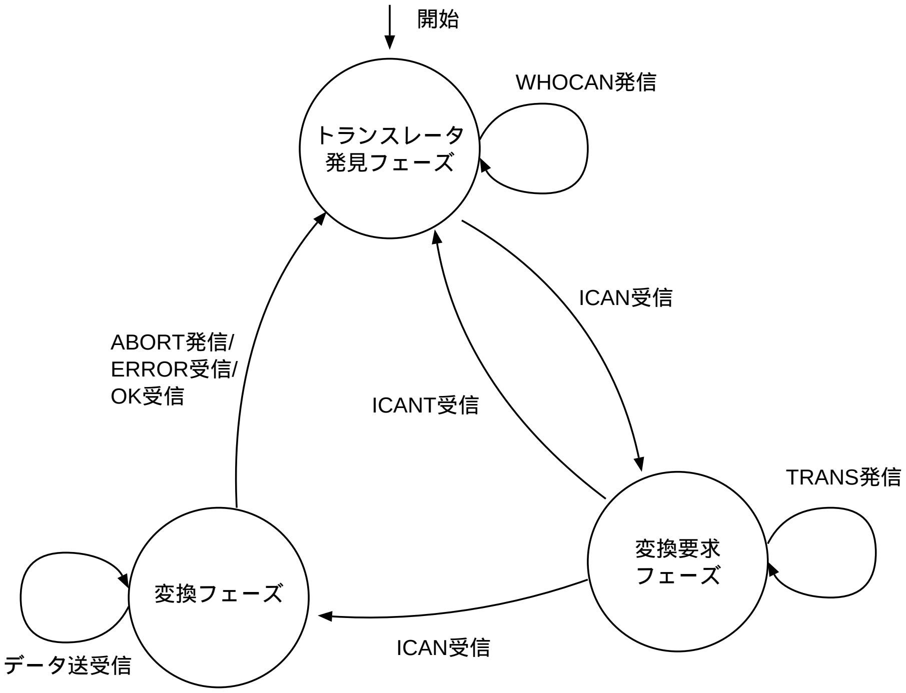


図 3.5: 代理サーバの状態遷移

第 4 章

実装

代理サーバ、トランスレータの実装について説明する。 実装はすべて BSD/OS Version 2.0.1 上で行なった。

4.1　代理サーバの実装

代理サーバは、サーバ-クライアント間のプロトコルに依存する部分と依存しない部分とに分けることができる。サーバとの通信を行なう部分、クライアントとの通信を行なう部分、情報表現形式の判断を行なう部分は、サーバ-クライアント間プロトコルに依存するが、トランスレータとの通信を行なう部分はこれに依存しない (図 4.1)。

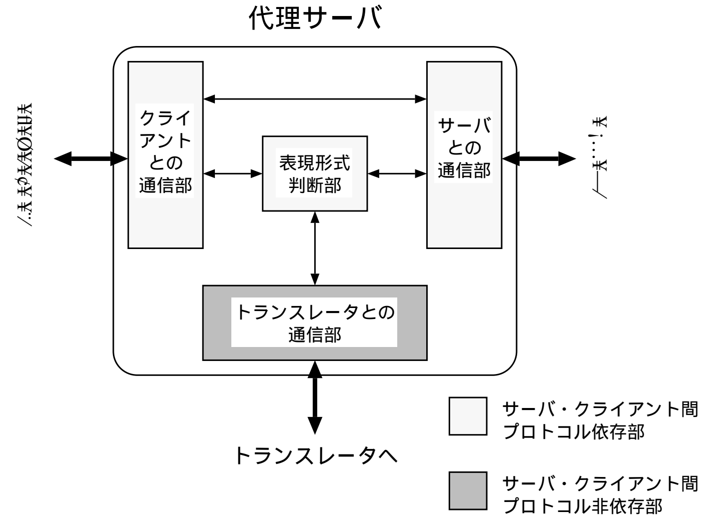


図 4.1: 代理サーバのアーキテクチャ

4.1.1 トランスレータとの通信部

この部分は、どのプロトコルに対する代理サーバでも共通なので、ライブラリ関数の形で実現することにした。用意した関数は、以下の 2 つである。

- `int find_translator(char *server_mt, char *client_mt, unsigned char ttl, long timeout, struct sockaddr_in *tr_addr);`
  - 機能:
    変数 `server_mt` に指定された情報の表現形式を変数 `client_mt` に指定された情報の表現形式へ変換できるトランスレータを探す。見つかった場合は、変数 `tr_addr` に見つかったトランスレータに関する情報が収められる。
  - 内部動作:
    トランスレータのマルチキャストグループへ ‘`WHOCAN`’ メッセージを送り、トランスレータからの ‘`ICAN`’ 応答を待つ。IP マルチキャストでは、マルチキャストパケットの到達限界を TTL(Time To Live) で指定できる。これを変数 `ttl` に指定する。‘`WHOCAN`’ メッセージを発信してから、トランスレータからの応答が返ってくるまで待つ最大時間を変数 `timeout` に指定する。

    ‘`WHOCAN`’ メッセージをマルチキャストする部分のコードは以下のようになる。`setsockopt` システムコールで TTL をセットする。

```
int s;
char buf[1024];
struct sockaddr_in saddr;
  ...
s = socket(PF_INET, SOCK_DGRAM, 0);
setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
bzero(&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_port = htons(トランスレータのポート番号);
saddr.sin_addr.s_addr = inet_addr(マルチキャスト IP アドレス);
len = sprintf(buf, "WHOCAN %s %s", server_mt, client_mt);
sendto(s, buf, len, 0, (struct sockaddr *)&saddr, sizeof(saddr));
```

  - 引数:

引数	型	意味
server_mt	char *	元の情報の表現形式
client_mt	char *	変換後の情報の表現形式
ttl	unsigned char	マルチキャストパケットの到達する最大 hop 数
timeout	long	発見にかかる最大待ち時間 (単位:秒)
tr_addr	struct sockaddr_in *	見つかったトランスレータに関する情報

  - 戻り値:

値	意味
0	目的の変換を行なうトランスレータが発見された
−1	目的の変換を行なうトランスレータが発見されなかった

- `int connect_translator(struct sockaddr_in *tr_addr, char *server_mt, char *client_mt, int sockv[]);`

  - 機能:

    変数 `tr_addr` に指定されたトランスレータに、変数 `server_mt` に指定された情報の表現形式から変数 `client_mt` に指定された情報の表現形式への変換のための接続を要求する。成功した場合は、変数 `sockv` に 3 つのソケットディスクリプタの値がセットされる。`sockv[0]` はトランスレータへデータを送信するためのソケット、`sockv[1]` はトランスレータから変換結果を受信するためのソケット、`sockv[2]` は通信制御用のソケットである。

  - 内部動作:

    指定されたトランスレータに TCP コネクションを張り、'`TRANS`' メッセージを送る。'`ICAN`' 応答が返ってきたらさらにデータ送信用のコネクション、データ受信用のコネクションを張る。これらのコネクションに対するソケットディスクリプタの値を変数 `sockv` に収める。最初に張ったコネクションへのソケットディスクリプタが `sockv[2]` となる。

  - 引数:

引数	型	意味
tr_addr	struct sockaddr_in *	接続先トランスレータに関する情報
server_mt	char *	元の情報の表現形式
client_mt	char *	変換後の情報の表現形式
sockv	int *	トランスレータとの通信用ソケット

  - 戻り値:

値	意味
0	トランスレータへの接続が成功した
−1	トランスレータへの接続が失敗した

4.1.2 FTP 用代理サーバの実装

クライアント、サーバとの通信部

FTP プロトコルは代理サーバを介して FTP サーバにアクセスすることは考慮されていない。そのためクライアントは、いったん FTP 用代理サーバにログインし、そこからさらに目的の FTP サーバにログインするという手順を踏むことになる。クライアントから代理サーバへの接続は、通常の FTP サーバへのログインと同様の手順で行なうことができる。

さらに代理サーバから目的のFTPサーバへ接続するための方法はいくつか考えられる。たとえば、DeleGateではFTPプロトコルのCWDリクエスト(クライアントで‘cd’コマンドを入力したときに送信される)の引数の書式を拡張し、‘cd //FTPサーバのホスト名’でFTPサーバへ接続する。本実装ではSITEリクエストを利用することにした。SITEリクエストは、クライアントからサーバ特有の機能を実行するためにFTPプロトコルで規定されているリクエストである。ユーザは、クライアントに、‘site 引数1 引数2 ...’の書式で入力することによって送信される。引数はすべてそのままサーバに送られる。それで、‘site open FTPサーバのホスト名’で、代理サーバからFTPサーバに接続ができるようにした。

FTPクライアントと代理サーバ、そして代理サーバとFTPサーバの接続が完了した後は、代理サーバはFTPクライアントから受け取るメッセージをそのままFTPサーバへ転送し、FTPサーバからの応答もそのままFTPクライアントに転送する。しかし、FTPサーバからファイル転送が行なわれるときだけは情報表現形式の変換が必要かどうか判断して、トランスレータを呼び出す。FTPサーバからファイル転送が行なわれるのは、RETRリクエストがクライアントから送られてきたとき(ユーザがFTPクライアントからgetコマンドが入力したとき)である。

情報表現形式判断部

設計の章で述べた通り、FTPサーバ上の情報の表現形式、クライアントでデコードできる情報表現形式の指定は、ユーザが手動で行なう。UNIXの標準的なFTPクライアントで、サーバ上のファイルを取得する場合は、

```
> get サーバ上のファイル名 クライアント側でのファイル名
```

のように指定する(‘>’は、ftpクライアントが出すプロンプト)。それでたとえば、

```
> get foo.gif foo.tiff
```

のように指定されたなら、代理サーバはファイル名の拡張子を見て、「サーバ上にあるファイルの情報表現形式は、画像のGIF形式であり、クライアント側で欲しているのは、画像のTIFF形式である」と判断し、トランスレータを呼び出すという仕組みにすればよいと考えた。しかし、実際にgetコマンドを入力したときにクライアントから送られるのはFTPプロトコルの‘RETR’コマンドであり、このコマンドの引数はgetコマンドで指定された第1引数のみである。上の例で実際にクライアントからサーバに送られるメッセージは‘RETR foo.gif’であり、getコマンドの第2引数は送られない。これではクライアント側で要求している情報表現形式が判断できない。これに対処する方法として以下のようなものが考えられる。

1. 代理サーバがクライアントに関する情報を保存する: 前節で説明したサーバ固有の機能を呼び出す‘SITE’コマンドを利用する。たとえば、クライアントがgetコマンドでファイルを取得する前に、

```
> site translate gif tiff
```

というコマンドを送ると、代理サーバには、‘SITE translate gif tiff’ が送られてくる。代理サーバはこの SITE コマンドに関してはサーバに送らずに自分で解釈し、現在アクセスしているクライアントに対しては、GIF から TIFF への変換が必要であるという情報を保存する。その後、‘RETR foo.gif’ といったコマンドが送られてきたら引数の拡張子部分を見て TIFF への変換が必要だと判断してトランスレータを呼び出す。

2. **代理サーバにログインするときに指定する:** クライアントは一旦代理サーバにログインしてから、さらに目的のサーバにログインするという手順を踏むことになるが、代理サーバへのログイン時のユーザ名もしくはパスワードの代わりにクライアントの希望する情報表現形式を入力する。

3. **RETR の引数の記述方法を拡張する:** たとえば、クライアントで

```
> get foo.gif>gif>tiff foo.tiff
```

と入力すれば、代理サーバには ‘`RETR foo.gif>gif>tiff`’ という文字列が送られてくる。代理サーバは、‘`>`’ をファイル名と情報表現形式の区切りであると判断し、サーバには ‘`RETR foo.gif`’ だけを送り、サーバ上の情報を取得する。残りの部分 (‘`>gif>tiff`’) からサーバ上にあるファイルの情報表現形式は GIF であり、クライアント側では TIFF 形式を要求していると判断してトランスレータを呼び出す。

今回実装した FTP 用代理サーバでは、3の方法を採用した。この方法だと、ファイル名の拡張子によって情報表現形式の判断ができない場合でも情報表現形式を明示することができる (たとえば、サーバ上の foo.txt が EUC コードで、これを JIS コードに変換して送ってほしい場合は、‘`get foo.txt>jis>euc`’ という指定ができる) という利点がある。しかし、グラフィカルユーザインタフェースの FTP クライアントではこのような指定は難しい。このようなクライアントへの対応は今後の課題とする。実際の利用例を図 4.2に示す。

```
client% ftp david 8888 ←代理サーバのホスト名とポート番号を指定
Connected to david.sfc.wide.ad.jp.
221 welcome to ftp-translator proxyserver.
Name (david:rin): ←ログイン名は何でもよい
331 Any username OK. Please input your address as password.
Password: ←パスワードは何でもよい

230 logged in. Please use 'site open' to connect ftp server.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> site open ftp.iij.ad.jp ←目的のサーバへログイン
200- Connected to ftp.iij.ad.jp
>> 220 ftp.iij.ad.jp FTP server (Version wu-2.4(6) Thu Oct 5 05:30:35 JST 1995) ready.
200 Command OK. Please use 'user' command to login server
ftp> user ftp ←ログイン名入力
331 Guest login ok, send your complete e-mail address as password.
Password: ←パスワード入力

230-
230-* Welcome, IIJ archive user! This is an FTP server of IIJ. If have any
...
230 Guest login ok, access restrictions apply.
ftp> get tput-1.0.tar.gz>tar.gz>lzh foo.lzh ← tar.gz から lzh への変換
227 Entering Passive Mode (133,4,46,5,7,67)
150 Opening BINARY mode data connection for tput-1.0.tar.gz (39302 bytes).
226 Transfer complete.
52847 bytes received in 35 seconds (1.5 Kbytes/s)
ftp> quit
211 Good bye!
client%
```

図 4.2: FTP 用代理サーバの利用例

4.1.3 HTTP 用代理サーバの実装

クライアント、サーバとの通信部

ほとんどの WWW クライアントには、代理サーバを通してサーバにアクセスする機能が備えられている。WWW クライアントは、代理サーバのホスト名とポート番号を環境変数や設定ダイアログなどへセットする。代理サーバを利用するように設定されたクライアントからは、たとえば、ユーザが見たい WWW のページの URL(Uniform Resource Loctor)[12] として、

```
http://www.mag.keio.ac.jp/˜rin
```

を指定すると、

```
GET http://www.mag.keio.ac.jp/˜rin/ HTTP/1.0
```

というリクエストが代理サーバに送られる (この後に、‘Accept:’ などのヘッダフィールドが続く)。代理サーバは、このなかでホスト名を意味する ‘www.mag.keio.ac.jp’ の部分を取り出し、そのホストの HTTP サーバ (通常ポート番号 80 番) に接続する。そして、クライアントからのリクエストから ‘`http://www.mag.keio.ac.jp`’ の部分を取り除いた、‘`GET /~rin/ HTTP/1.0`’ をサーバへ送信する。サーバからは同じコネクションを通して、レスポンス (ヘッダと情報の内容) が送られてくるので、変換が必要ない場合は、代理サーバはそれをそのままクライアントへ転送する。

情報表現形式判断部

設計の章で述べたように、情報表現形式を行なうサーバから送られてきた情報の表現形式とクライアントで受け取りたい情報表現形式の組をクライアント側のユーザが事前に登録するという方法をとる。

ユーザが代理サーバにそのような登録をするためのインタフェースとして、WWW クライアントを用いることができるようにした。こうすればユーザは、別のクライアントを用意する必要はない。それで HTTP 用代理サーバは、図 4.3のような構成となる。

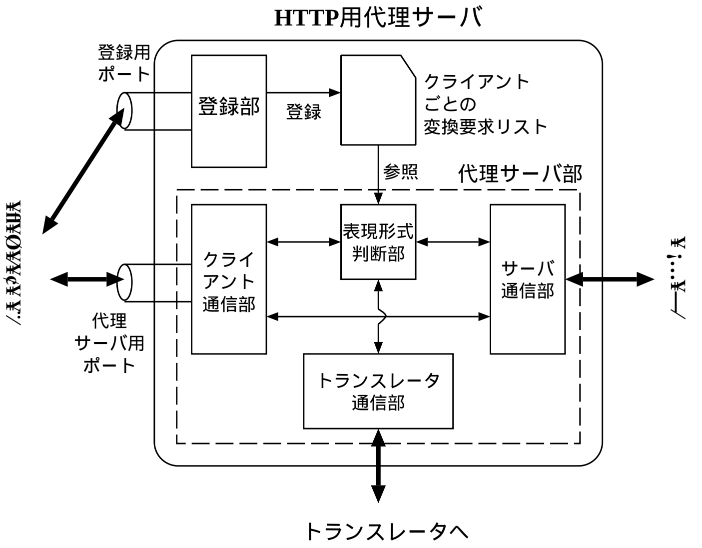
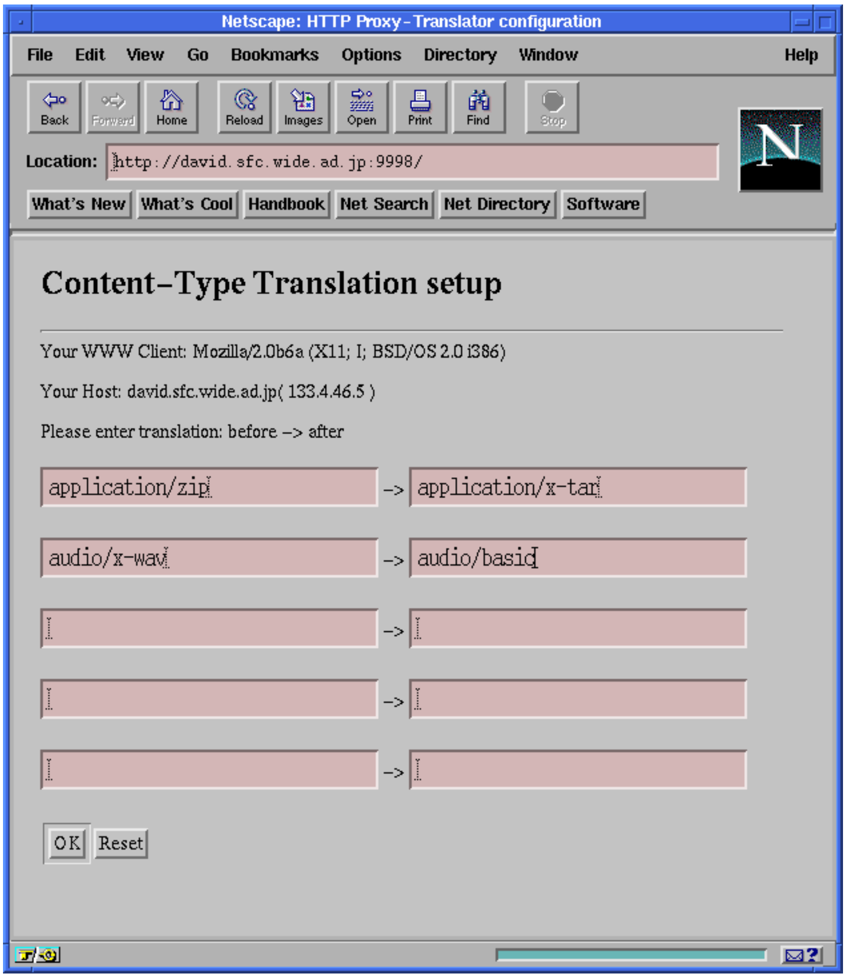


図 4.3: HTTP 用代理サーバの構成

代理サーバは、代理用とは別の登録用のポートでクライアントからのリクエストを待つ。ユーザは、WWW クライアントから、

`http://`代理サーバのホスト名:登録用ポート番号/

を指定してページを開くと、図 4.4のような登録用の画面が現われる。



Location: http://david.sfc.wide.ad.jp:9998/

Content–Type Translation setup

Your WWW Client: Mozilla/2.0b6a (X11; I; BSD/OS 2.0 i386)

Your Host: david.sfc.wide.ad.jp(133.4.46.5)

Please enter translation: before –> after

application/zip –> application/x-tar

audio/x-wav –> audio/basic

図 4.4: 代理サーバへの変換要求登録画面

ここで入力された各項目の値が、ユーザの利用している WWW ブラウザ名 (HTTP リクエストの ‘User-Agent:’ フィールドから取得する) とホストの IP アドレスとともにクライアントからの変換要求データとして以下のような構造体に保存される。

```
struct client_info {
    struct in_addr addr;    /* クライアントの IP アドレス */
    char *user_agent;       /* WWW ブラウザ名 */
    char *tr_list[2*MAX_TR_ENTRY]; /* 情報表現形式変換要求リスト */
    struct client_info *next;
};
```

`tr_list` には、サーバから送られてくる情報表現形式とその形式の情報が送られてきた場合、クライアントに送るときに変換してほしい表現形式が交互に収められている。

このデータは、コネクションが張られるたびに参照されるので、高速化のため登録時にクライアントのIPアドレスをキーにしてハッシュした表としてメモリ上に保持される。同じホストでも複数のネットワークインタフェースを持っている場合は、コネクションごとに別のIPアドレスでアクセスされる可能性がある。それでgetpeernameシステムコールでクライアント側のIPアドレスを取得した後、gethostbyaddrシステムコールでさらにそのクライアントホストの他のIPアドレスも取得して、それもハッシュ表のキーにして登録している。

代理サーバは、代理用ポートを通してWWWサーバへのアクセスがあるたびに登録されたクライアントごとの変換要求データを参照する。それで、情報表現形式変換が必要かどうかの判断のアルゴリズムは以下のようになる。

1. クライアントからのリクエストが届いたら、リクエストの‘User-Agent:’フィールドからWWWクライアントの名前を取得する

2. getpeernameシステムコールでクライアント側のIPアドレスを取得する

3. IPアドレスをキーにしてハッシュ表を検索する

4. 一致するエントリが見つかった場合、さらにWWWクライアントの名前が一致するかどうか調べる

5. IPアドレスとWWWクライアント名が両方とも一致したら、情報表現変換要求リストを参照するフラグを立てる

6. サーバへクライアントからのリクエストを転送し、サーバからレスポンスを受け取る

7. フラグが立っていない場合は、レスポンスをそのままクライアントに転送する。フラグが立っている場合は以下の手順を踏む

8. レスポンスのヘッダ部から‘Content-Type:’フィールドの値を取り出す

9. このフィールドの値に一致するエントリが、情報表現変換要求リストの中にあれば、その情報表現形式に対応するクライアントの要求する情報表現形式のエントリ(‘Content-Type:’フィールドに一致したのが `trlist`[n]+であれば、`tr_list`[n1]+)を取り出し、この組の変換ができるトランスレータを探す。

10. 発見できれば、‘Content-Type:’フィールドをクライアントの要求する情報表現形式に書き換えてレスポンスのヘッダ部をクライアントに送る。

11. レスポンスのボディ(ヘッダを除いた部分)をトランスレータに送り、トランスレータからの変換結果をクライアントに転送する

4.2 トランスレータの実装

4.2.1 メッセージ待ち受け部

トランスレータは、代理サーバからのトランスレータ発見メッセージ (マルチキャストUDP) と変換要求メッセージ (TCP) の待ち受けをしなければならない。それでデフォルトのマルチキャストアドレスとして 224.0.1.100、マルチキャストと TCP で共通の待機ポート番号として 7777 を選択した。マルチキャストデータグラムを待ち受けるためには、BSD/OS を含む 4.4BSD UNIX[13] では、通常のユニキャストデータグラムの待ち受けをする場合と同じように、

```
s = socket(PF_INET, SOCK_DGRAM, 0);
```

でソケットを生成して、bind システムコールでローカルのアドレス (INADDR_ANY) とポート番号に結び付ける。そのあと、setsockopt システムコールを用いて、

```
struct ip_mreq mreq;

mreq.imr_multiaddr.s_addr = inet_addr(マルチキャストアドレス);
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

のように指定するとそのマルチキャストアドレスに向けて送られてくるデータグラムを受け取ることができるようになる。

4.2.2 変換機構の実現

情報表現形式変換には、既存の変換フィルタプログラムを利用することにした。つまり、トランスレータは、代理サーバとの通信機能と外部の変換フィルタプログラム起動の機能だけを持ち、実際の表現形式変換機構はもたない。変換フィルタプログラムは標準入力からデータを取り込み、変換結果を標準出力へ出力するように作られていることを前提する。トランスレータの設定ファイルに、図 4.5のように記述する。

```
jis|euc|text/plain  sjis|text/plain;charcode=sjis  /usr/local/bin/nkf -s
gif|image/gif       ppm|image/x-ppm                /usr/X11/bin/giftoppm
au|audio/basic      wav|audio/x-wav                /usr/local/bin/au2wav
```

図 4.5: トランスレータの設定ファイルの例

設定ファイルの各行の書式は、

元の形式名リスト 変換後の形式名リスト 変換ツールの絶対パスと引数

であり、形式名リストは‘|’で区切って複数の情報表現形式名を列挙することができる。設定ファイルはトランスレータの起動時に読み込まれる。

代理サーバからの変換要求（WHOCAN もしくは TRANS）メッセージを受け取ったら、トランスレータは変換要求メッセージに含まれる、元の情報表現形式と変換してほしい情報表現形式の組と一致する組を設定ファイルの内容から探す。一致するものがあれば、変換できることを代理サーバへ伝える（ICAN メッセージを送る）。

変換フェーズでは、代理サーバからトランスレータへ制御用、データ送信用、データ受信用の3つのTCPコネクションが張られるが、トランスレータは3つのパイプを生成し、変換フィルタプログラムを起動する。1つのパイプを通して代理サーバから受け取ったデータを変換フィルタプログラムの標準入力へ送り込み、もう1つのパイプで変換フィルタプログラムの標準出力からの出力を受け取って代理サーバへの送信コネクションへ送る。さらにもう1つのパイプを通して変換フィルタプログラムの標準エラー出力を得る。標準エラー出力からの出力は、変換フィルタが正常終了しなかった場合に ERROR メッセージの引数として代理サーバへの制御コネクションを通して送られる（図4.6）。

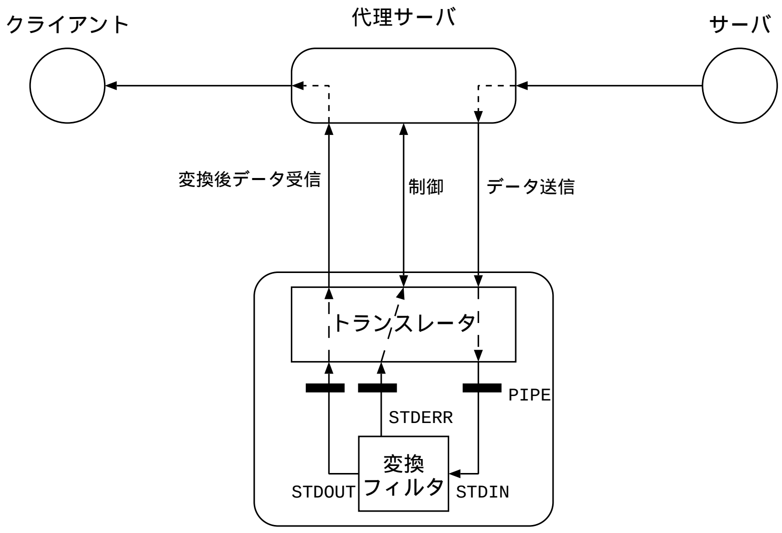


図4.6: トランスレータと変換フィルタプログラム

このようにトランスレータ自身は、代理サーバとフィルタプログラムとの間のインタフェースだけを持ち、実際の変換は、外部のフィルタプログラムを起動して変換を行なうように実装することによって、既存の豊富な情報形式変換フィルタプログラムが利用できるようになり、トランスレータの機能拡張が容易になった。外部の変換フィルタプログラムは、標準入力からデータを受け取って標準出力に処理結果を出力する、UNIXのごく標準的なプログラムであればなんでもよく、シェルスクリプトなども用いることができる。

4.3 情報表現形式名の統一

代理サーバとトランスレータで用いられている情報表現形式名が統一されていなければ、代理サーバは適切なトランスレータを発見することはできない。そこである程度の規則を設けて統一をはかることにした。

- 基本的には、そのエンコーディング形式を表わすのに一般的に用いられている記法に従う。ファイル名の拡張子につけられているものなど。
- 大文字小文字の区別はしない
- 情報表現形式の記法の標準として、HTTP の ‘Content-Type:’ フィールドの表記法としても採用されている、MIME(Multipurpose Internet Mail Extensions)[14] の Content-Type の表記方法に従う

MIME の Content-Type の表記法は、テキストの文字コードを識別するための情報を含んでいない。それで HTTP の場合に、ユーザがクライアントにテキスト形式の情報が送られてくるときの文字コードを指定するときには、

```
text/plain;charcode=jis
```

のように指定することにした。さらに、HTTP 用代理サーバで、HTTP サーバから送られてくる情報の表現形式の判断のためにレスポンスヘッダの ‘Content-Type:’ フィールドの値が用いられているが、ここにはテキストの場合は ‘text/html’、‘text/plain’ などといった値が入るのみで、文字コードを識別するのに足りる情報がない。それで、HTTP 用代理サーバへの変換要求設定のとき (図 4.4) に、文字コードの変換を設定したいときには、サーバ上の情報形式として、‘text/plain’、これの変換後の形式として ‘text/plain;charcode=jis’ のように指定するしかない。つまり、元のテキストの文字コードがなにかは、トランスレータによって自動判別できなければならない。

それで、図 4.5にあるように、トランスレータのフィルタ設定ファイルには、一つの情報表現形式を表すのに、一般的に用いられているいくつかの表記法と Content-Type の表記法を列挙しておく。

第 5 章

実験と評価

5.1 実験環境

図5.1のような研究室のネットワークの上で実験を行なった。host A、host B、host C はマルチキャストに対応しており、その間のルータもマルチキャストのパケットを通すことができる、もしくはトンネリングによってマルチキャストパケットを通している。

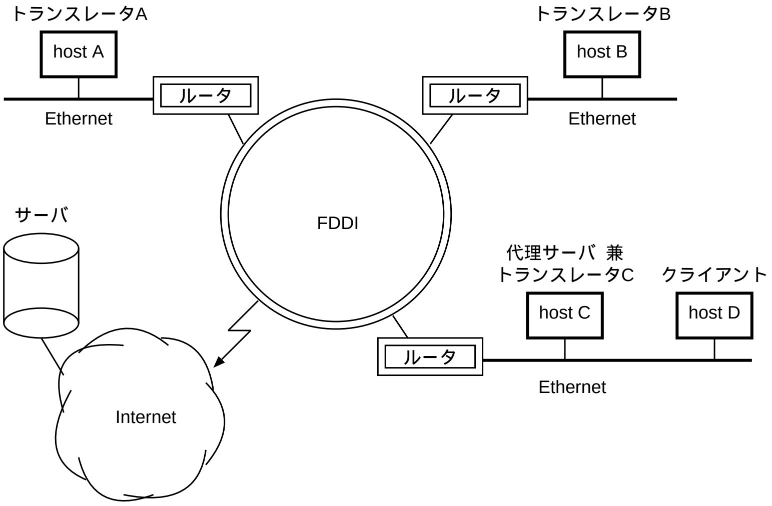


図 5.1: 実験環境

この環境で、host D のクライアントは host C の代理サーバを介してインターネット上のサーバにアクセスし、必要なときに代理サーバは host A、B、C のトランスレータを使って情報表現形式変換を行なうという基本的な動作は確認できた。

5.2 実現できたトランスレータ

本システムは、トランスレータに登録する変換フィルタとして、標準入力からデータを受け取り、標準出力に変換結果を出力するプログラムやシェルスクリプトであればほとんどのものを利用できる。以下に実現できたトランスレータの例を挙げる。

5.2.1 文字コード間の変換

日本語の文字コードとして広く用いられているものとして、JIS コード、シフト JIS コード、EUC コードがある。トランスレータにこれらの間の変換を行なわせるには、nkf、jc といった変換フィルタを利用できる。

5.2.2 画像の符合化形式間の変換

画像の符合化形式としては、GIF 形式、JPEG 形式、TIFF 形式、X window system のビットマップ形式 (xbm) など、さまざまな形式がある。これらの間の変換プログラムも非常にたくさん存在する。これら変換ツールで標準入出力を利用できるものであればそのままトランスレータに登録できる。しかし、複数のプログラムを UNIX のパイプ機能を用いて接続して利用する場合は、シェルスクリプトにしておかなければならない (将来的には、トランスレータが自動的に、登録されたフィルタの中から適切な組み合わせを見つけてパイプによる接続ができるようにすることを検討している)。例えば、GIF 形式から XPM 形式に変換するフィルタを giftoppm と ppmtoxpm というプログラムで実現したい場合は、以下のようなシェルスクリプトを作成し、登録する。

```
#!/bin/sh
giftoppm|ppmtoxpm
```

5.2.3 音声の符合化形式間の変換

音声の符合化形式として、au 形式、wav 形式、aiff 形式などがよく用いられる。これらの間の変換を行なうプログラムとして、sox というプログラムがある。しかし、sox は標準入出力を利用するプログラムではない (より正確には、標準入力からデータを受け取ることはできるが、出力はファイルのみ)。それで、標準入力から受け取ったデータを一時的にファイルに保存し、そのファイルを sox に処理させるシェルスクリプトを書けばよい。たとえば、UNIX(主に Sun Microsystems の機種) で標準的に用いられている au 形式から、Windows 標準の音声符合化形式である wav 形式へ変換するフィルタは以下のようになる (図 4.5の au2wav)。

```
#!/bin/csh
set tmpfile=/var/tmp/tmpsound.$$
/usr/contrib/bin/sox -t au - -t wav $tmpfile rate
/bin/cat $tmpfile
/bin/rm $tmpfile
```

5.2.4 アーカイブ形式間の変換

アーカイブ形式もOSごとに標準的に使われるものに違いがある。UNIXでは主にtarとgzipが利用され、Windowsではzip形式とlha形式が標準である。Macintoshでは、stuffitやcompact proというアーカイバによるアーカイブ形式がよく用いられる。たとえば、以下のようなシェルスクリプトを、tarとgzipによるアーカイブ形式からlhaによるアーカイブ形式に変換するフィルタができる。

```
#! /bin/csh
set tmpdir=/tmp/tmpdir$$
mkdir $tmpdir
cd $tmpdir
gzip -cd | tar xf -
lha a - .
rm -rf $tmpdir
```

5.2.5 日本語テキストを画像へ変換

日本語のフォントがないような環境で日本語のテキストを表示させるためには、トランスレータでテキストを画像にして送ればよい。テキストをPostscript形式へ変換するa2psと、Postscriptインタプリタで、さまざまな形式への変換ができるgs(ghostscript)を利用して、以下のようなシェルスクリプトを作成してトランスレータに登録しておけばそのような機能を実現できる。

```
#!/bin/sh
PATH=/usr/local/bin
nkf | a2ps | gs -q -sDEVICE=gifmono -sOutputFile=- -
```

もちろん、この場合、a2psとgsは日本語に対応したものでなければならない。このシェルスクリプトが、/usr/local/bin/jp2gifだとすると、トランスレータの設定ファイルには以下のように記述する。

```
txt|jis|euc|sjis|text/plain gif|image/gif /usr/loal/bin/jp2gif
```

5.3 実現可能なトランスレータ

適当なフィルタプログラムが存在しないなどの理由で、まだ実現はできていないが、適当なプログラムが用意できれば容易に実現可能なトランスレータとしては以下のようなものがある。

- 日本語と英語、中国語と日本語などの自然言語間の翻訳

  これを行なうトランスレータも、適当な翻訳ソフトウエアが利用できればすぐに作成することができる。しかし、ユーザから辞書の指定ができない点は問題になる場合もある。

- テキストから音声への変換

  読み上げソフトがあれば実現可能である。

- **HTML** テキストを画像に変換

  HTML(HyperText Markup Language) 形式のテキスト中の文字を画像に変換して表示する機能である。これは DeleGate が CII(Character by Inline Image) 機能として実現している。この機能があれば、ある言語の文字コードが表示できない WWW クライアントでもその文字を GIF 形式の画像として表示できる。HTML テキストは、一種のスクリプトなので、プレインテキストの場合のように単純にすべてをビットマップに変換することはできない。HTML のコマンドでない部分だけを一文字一文字画像にする必要がある。これは、HTML の書き換えを行なうフィルタプログラムと、文字ごとのビットマップをおいた HTTP サーバの組み合わせで実現できる。

- 動画の符合化形式間の変換

  MPEG 形式、QuickTime 形式、AVI 形式などが一般に広く用いられている。これらの符合化形式間の変換のための適当なプログラムが入手できればトランスレータとして利用可能である。しかし、動画の場合は形式間の変換には情報落ちが生じる可能性が高い。Macintosh 上で実現されている、MPEG 形式と QuickTime 形式の間の変換プログラムなどもあるが、かなり多くのパラメータを指定する必要がある。トランスレータとして実現した場合、パラメータの指定が可能なように代理サーバ-トランスレータ間プロトコルを拡張する必要がある。

5.4 実現不可能なトランスレータ

本研究のシステムでは、1 回のクライアントからのリクエストに対して送ることができるのは、1 つのファイルのみである。このため、変換によって複数のファイルに分かれる

場合には対応できない。例えば、アーカイブを展開するという変換は、展開によって複数のファイルとなるため不可能である。

5.5 転送のオーバヘッド

図 5.2のような環境で、FTP サーバからクライアントへ約 1.2M バイトの JIS コードのテキストファイルを転送し、EUC コードに変換するのにかかる時間を計測した。トランスレータには、nkf という非常に広く用いられている日本語文字コード変換ツールを登録した。

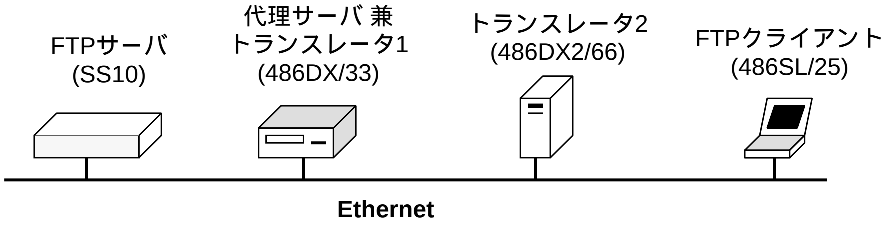


図 5.2: 転送時間測定の環境

条件を変えて、それぞれの条件で 6 回実行し、その平均時間を求める。転送時間の測定には、FTP クライアントが転送終了時に表示する転送時間を用いた。クライアント上での nkf 実行にかかる時間は、BSD/OS の time コマンドの実時間の出力を用いた。

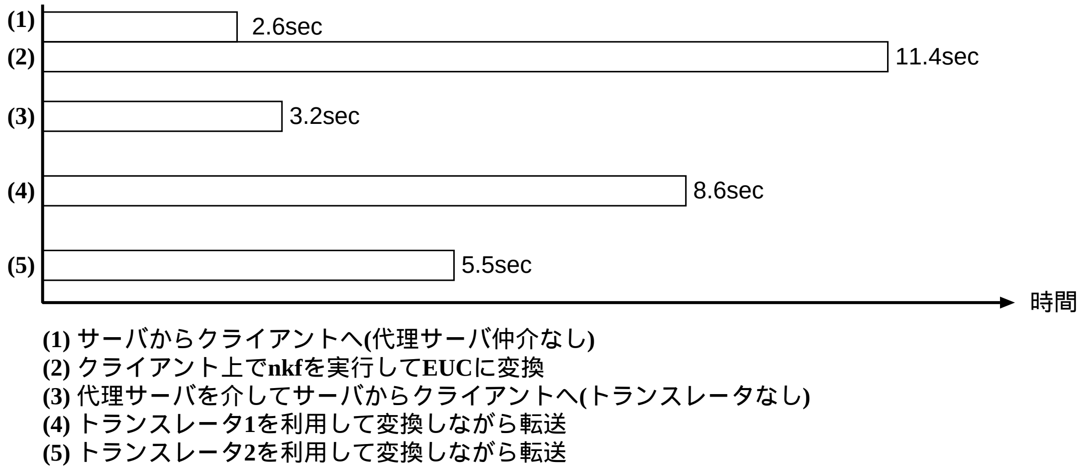


図 5.3: 実行時間の比較

図 5.3を見ると、FTP でサーバからファイルを取得し、クライアント上で変換を行なう (図中の (1)+(2)) より、変換しながら転送する (図中の (4)、(5)) の方がより早く、目的の形式で、情報を取得することができている。現実には、サーバ、クライアントと代理サーバの間のネットワーク的距離、代理サーバとトランスレータ間のネットワーク的距離はもっと大きくなるのでより多くの時間がかかることになるが、この実験から、代理サーバ、トランスレータを中継する方が、ユーザが情報を取得してから自分で適切な変換ツールを選択し、変換を行なう場合より早く目的の情報を適切な形式で入手することが可能であるといえる。

5.6 評価

本システムを利用することによってインターネットの利用において以下のような点を改善することができる。

- ユーザがインターネットで、ある情報を取得してもそれが自分の利用している環境では解読できない表現形式だった場合、まずその表現形式の解読ができるソフトウェア、もしくは自分の利用している環境で解読できる形式に変換するソフトウェアを入手することから始めなくてはならない。使用している OS などによっては、解読不可能な場合もある。本システムを利用することによってそのような場合にも、ユーザは、サーバから送られる情報の表現形式と自分の環境で解読できる表現形式を代理サーバに伝えるだけで解読可能になる。
- ある情報表現形式から別の情報表現形式へ変換するソフトウェアは、他数存在する。しかし、ユーザが、どの変換ソフトウェアで自分の目的とする変換が行なえるのか、その変換ソフトウェアはどこから入手できるのかを調べ、入手し、インストールするのは大変である。本システムのトランスレータとして各変換ソフトウェアを登録すれば、ユーザは、本システムの代理サーバを利用するだけで、さまざまな変換に対応できる。
- 情報提供者はさまざまな環境からのアクセスを考慮して、同じ情報を複数の形式でサーバ上に置くことが少なくない。例えば、
    - 同じ動画を MPEG、QuickTime、AVI の各形式で置く
    - 同じ画像を GIF、JPEG の各形式で置く
    - 同じ音声を au、wav、AIFF の各形式で置く
    - 同じアーカイブを tar+gzip、lha、shar の各ツールでまとめた形式で置く

  などである。これはサーバのリソースの無駄使いである。本システムが利用できることを前提にすれば、1 つの情報について 1 種類の情報表現形式で置いておけばよく、リソースの節約になる。

- DeleGateでは、新たな変換機能を付加するためには、DeleGate自身をコンパイルし直して、起動し直さなくてはならない。本システムの代理サーバで新たな変換機能を利用できるようにするためには、インターネット上のどこかでその変換機能を持つトランスレータが立ち上がればよい。
- 適切な代理サーバを作成すれば、機能の制約の多い端末装置、例えば、PDA(Personal Digital Assistant)やファクシミリでも、インターネット上の情報を取得し、利用することが容易になる。

第 6 章

考察

6.1　代理サーバ-トランスレータ間通信プロトコルについて

今回実装したプロトコルでは、単純にデータを送り、変換結果を送り返すことしか考慮していない。しかし、実際には変換に際し、いくつかのパラメータを送ってやる必要があることもある。たとえばテキストを画像に変換するトランスレータであれば、解像度何DPI(Dot Per Inch)の画像にするか、といったパラメータも必要である。変換に必要とされるパラメータとしてどんなものがあるか調査し、プロトコルヘッダとして付加できるようにしなければならない。

また、変換による情報の劣化が避けられない場合もある。そのような場合、どれくらいの劣化までを許容するかというパラメータも必要である。HTTP Version 1.1の以前のドラフトではそのようなパラメータの規定があった。

さらに、新たなパラメータにも対応できるような拡張性を持たせる必要がある。

6.2　HTTP Version 1.1への期待

HTTPのヘッダのフィールド(クライアントからのリクエストの‘Accept:’、サーバからのレスポンスの‘Content-Type:’など)は本来、本研究で考えるようなクライアント-サーバ間で情報表現形式のネゴシエーションをすることを考慮して設けられたものであった。しかし、設計の章で述べたように、実際のWWWクライアント、サーバの実装ではこれらのフィールドが十分な機能を果たしていない。そのため本研究のHTTP用代理サーバは、サーバから送られてくる情報の表現形式、クライアントがデコードできる情報表現形式の判断のためにそれらのフィールドを部分的にしか用いることができなかった。

HTTPの次のバージョン(Version 1.1)のドラフト[6]では、‘Accept:’、‘Accept-Encoding:’、‘Accept-Language:’、‘Content-Language:’の各フィールドの規定が復活しており、より厳密に定義されている。さらに、‘Accept-Charset:’(クライアントの受け入れられる文字集合)フィールドが新たに付け加えられている。これらが仕様どおりに実装されれば、本研究のHTTP用の代理サーバの情報表現形式判断機構はより容易に実現できるようになる。

6.3　Windows、Macintosh上でのトランスレータの実現

実際に実装は行なわなかったが、WindowsやMacintoshでもトランスレータを実現することは可能である。これらのOS上でトランスレータが実現できれば、特定のOSでしか動作しない情報形式変換プログラム(英語以外のあまり一般的でない言語から日本語への翻訳ソフトウェア、Macintosh固有のアーカイブソフトウェアなど)をネットワーク上のすべてのアーキテクチャのマシンからその変換サービスを受けることができることになる。これらのOSではUNIXのような標準入出力はないが、アプリケーション間通信の方法は規定されているのでそれを利用できる。しかしWindowsもMacintoshもIPマルチキャストにまだ対応していないので、代理サーバからのメッセージ受け付け部はマルチキャストに対応しているUNIX上で実現しなければならない。

6.4　クライアント、サーバを含めた情報提供システムの再設計

本研究では、クライアント、サーバは既存のものをそのまま用いることを前提として、代理サーバ、トランスレータを設計した。しかし、サーバ、クライアントもユーザが望みの情報表現形式で情報を取得することを可能にするように設計しなおすとすると以下のようになる。

- クライアント–サーバ間プロトコル

  サーバとクライアントで情報表現形式のネゴシエーションを行なうのに十分な情報を付加できるようにする。そのためにはすくなくとも、HTTP Version 1.1のヘッダの仕様を完全に満たしている必要がある。また、クライアントからのメッセージヘッダには、クライアントがデコードできる情報表現形式だけでなく、クライアントを利用しているユーザの要求する情報表現形式が記述できるフィールドが必要である。たとえば、英語の情報を日本語に翻訳して送ってほしいとか、画像はGIF形式でもTIFF形式でも表示できるクライアントを利用していても、後からの利用を考えてどんなエンコーディング形式の画像でもTIFF形式に変換して送ってほしいといった要求は、ユーザが与えることができなければならない。

- クライアント

  以下のような機能を持つ必要がある

  - ユーザの変換要求を受け付ける機能
  - クライアントのマシンでデコードできる情報表現形式を認識する機能
  - トランスレータ発見機能

- サーバ

ネゴシエーションに十分な情報を提供するため、自分の保持しているひとつひとつの情報の表現形式を知っているべきである。そして、クライアントもしくはサーバに、情報表現形式判断機構とトランスレータ発見機構を持たせる。これにより代理サーバは必要なくなる。

- トランスレータ

  トランスレータは、クライアントから直接呼び出され、クライアントから変換前と変換後の情報表現形式に関する情報だけでなく、目的の情報の URL も受け取り、それをもとにサーバから情報を受け取って、変換してクライアントへ転送する。

このようなシステムが実現できれば、ユーザの望む形式で情報を受け取ることがより容易になる。また、現状では同じ情報でもユーザの便宜をはかるために異なったいくつかの表現形式でサーバ上に置かれることがある (同じ内容のテキストを JIS コードとシフト JIS コードで置く、同じ音声ファイルを WAV 形式と AU 形式で置く、同じ動画を MPEG 形式と QuickTime 形式と AVI 形式で置くなど) が、このシステムではサーバ上には 1 つの表現形式だけで置き、その他の形式が必要な場合はトランスレータに任せることができるのでディスク使用量の節約になる。

第 7 章

結論と今後の課題

7.1 結論

本研究では、インターネット上の情報提供システムのための非常に拡張性の高い情報表現形式変換システムの設計と実装について述べた。本システムによって、ユーザが目的の変換を行なう変換ツールを選択するという作業を自動化することができる。また、変換は情報がサーバからクライアントに送られてくる間に行なわれる。

本システムを利用することによって、ユーザがインターネット上で取得した情報が、そのユーザの環境ではデコードできず、結局活用することができないという事態が生じることを非常に少なくすることが可能である。

本システムは、サーバから送られてくる情報の表現形式とクライアントの要求する情報表現形式を判断する代理サーバと、情報表現形式の変換を行なうトランスレータによって構成される。それぞれ固有の機能を持ったトランスレータがインターネット上に分散して存在し、代理サーバは、変換が必要であると判断したときに、適切なトランスレータを探して、変換を依頼する。トランスレータの発見のためにマルチキャストを用いることによって、代理サーバは前もってどこにどのような変換機能を持つトランスレータがあるか知る必要はない。また、トランスレータも何の登録などの作業も不要で、インターネット上の複数の代理サーバから利用可能になった。これらの仕組みによって、代理サーバはさまざまな情報表現形式間の変換に、柔軟に対応することが可能となった。

本システムの代理サーバと類似したシステムとして DeleGate があるが、DeleGate は変換機構を内蔵しているのに対して、本システムの代理サーバは、変換機構を外部のトランスレータに分離することによって、より柔軟にさまざまな情報表現形式変換に対応することが可能である。

ある情報表現形式から別の情報表現形式へと変換するソフトウェアは、非常に多く存在し、多くがインターネット上で入手可能である。しかし、ユーザが自分の目的とする変換を行なってくれるソフトウェアが何で、それはどこで入手できるのかを探すのはかなり大変な作業である。また、ユーザの用いている環境でその変換ソフトウェアが利用できるとは限らない。本研究のトランスレータとしてそのような変換ソフトウェアを利用すること

は容易であり、トランスレータとして利用できれば、ユーザは、目的の変換を代理サーバに指示するだけでその変換機能を利用できる。どのようなソフトウェアがその変換に必要なのかは全く意識する必要はなく、ユーザの利用しているクライアントのOSに依存することはない。

7.2 今後の課題

以下のような項目が今後の課題として残されている。

- **UNIX以外のOS上でのトランスレータの開発**...本研究で開発したトランスレータは、UNIX上のものだけであったが、他のOSたとえばWindowsやMacintoshなどの上で稼働するトランスレータを開発する。これが実現できれば、特定のOS上でしか動かない情報形式変換プログラムもトランスレータとして利用することが可能になる。
- 広域分散環境での運用実験...今回は、いくつかのネットワークで分割されているとはいえ、研究室内のLocal Area Netoworkのなかでのみの実験であった。トランスレータがインターネット上に分散した状態での実験を行なう必要がある。
- 代理サーバ-トランスレータ間プロトコルの表現力向上...考察の章で述べたような改良を加える必要がある。

謝辞

研究、論文の指導をしていただいた村井純助教授、徳田英幸助教授、斉藤信男環境情報学部長、苗村憲司教授、中村修助手、楠本博之講師、植原啓介研究員の各氏に感謝いたします。また、さまざまなアドバイスを与え、励ましてくれた、モービル広域ネットワークプロジェクトのメンバ、オープンメディアプロジェクトのメンバ、徳田村井研究室のメンバに感謝の意を表します。

参考文献

[1] Sun Microsystems, Inc.: RFC1014 XDR: External Data Representation Standard, June 1987

[2] 佐藤豊: 多目的プロトクル中継システム DeleGate, 電子技術総合研究所彙報 第59巻 第6号 (1995)

[3] T. Berners-Lee, R. Fielding and H. Frystyk: Internet-Draft Hypertext Transfer Protocol – HTTP/1.0, October 1995

[4] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. John, D. Torrey and B. Alberti: RFC1436 The Internet Gopher Protocol (a distributed document search and retrieval protocol), March 1993.

[5] J. Postel and J. Reynolds: RFC959 FILE TRANSFER PROTOCOL (FTP), October 1985

[6] R. Fielding, H. Frystyk and T. Berners-Lee: Internet-Draft Hypertext Transfer Protocol – HTTP/1.1, November 1995

[7] J. Reynolds and J. Postel: RFC1700 ASSIGNED NUMBERS, October 1994

[8] International Organization for Standardization (ISO): ISO 639:1988 (E/F). Code for the Representation of Names of Languages, April 1988

[9] S. Deering: RFC1112 Host Extensions for IP Multicasting, August 1989

[10] JP MBone 運用グループ: JP MBone 実験参加の手引, February 1995

[11] JP MBone 運用グループ: JP MBone 実験運用のガイドライン, February 1995

[12] T. Berners-Lee, L. Masinter and M. McCahill: Uniform Resource Locators (URL), December 1994

[13] S. Leffler, R. Fabry, W. Joy and P. Lapsley: An Advanced 4.4BSD Interprocess Communication Tutorial, June 1993

[14] N. Borenstein and N. Freed: RFC1521 MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, September 1993