アドホックネットワークにおけるＷＥＢ環境シミュレータの開発

廣木 大地＊　　李 睿棟＊

A Development of WEB Environment Simulator for Mobile Ad hoc Network

Daichi Hiroki＊, Li Ruidong＊

キーワード：Ajax ,WEB Application , Ad hoc Network
Keywords：Ａｊａｘ,ＷＥＢアプリケーション,アドホックネットワーク

1. はじめに

近年、無線ネットワーキング技術の発達に伴い、無線通信端末をルータのように振舞わせることでマルチホップ通信を動的に実現する技術として、アドホックネットワークが注目を浴びている(1)。このアドホックネットワーク技術は災害時のインフラを必要としない通信基盤や軍事用の暫定的な通信基盤、グループコミュニティや自動車の相互通信を実現する ITS(Intelligent Transport Systems)などとしての利用が考えられる。しかし、このような応用アプリケーションのための基盤として研究が進められているアドホックネットワーク技術であるが、応用アプリケーションの性能評価のためのシミュレーション環境が整備されているとはいいがたい。これは応用アプリケーションよりも無線同士の相互通信の難しさや効率向上のためのネットワークレベルでのシミュレーションに多くの関心が研究者の中で寄せられていたからという背景もある。

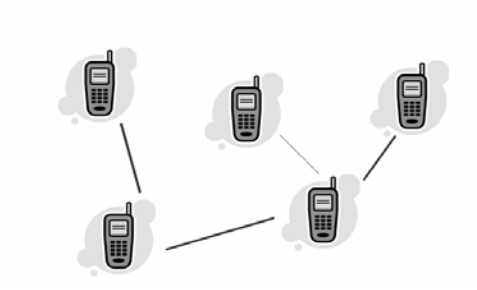

図 1　アドホックネットワークとは

また、応用アプリケーションの利用者である無線端末保持者の移動によって変動を起こす通信形態であるにもかかわらずその利用者の作り出すネットワークがいかような形態になるのかということとその形態によってどの程度シミュレーションの精度が変わるのかというユーザ目線からの通信評価というものは少ない。一方、これらの新しいネットワーク技術への技術者や研究者の関心の高さに対して、その利便性やそれのもたらすユーザエクスペリエンスを広く社会に広め還元するということが非常に難しくなっている。論文や技術解説といったものは一般の人々には敷居が高く、それによって将来性への投資なども滞ることになってしまう。また、セキュリティ関連技術では目に見える安全性の保障が技術を詳しく知らない人々への社会的な貢献としても「わかりやすく」「ビジュアルに」伝える努力が必要であると考える。

本稿では高レイヤアプリケーションの動作実験を目的とした WEB ブラウザ上で動作するシミュレーションツールの作成とその技法について述べる。ブラウザ上での実行のためにはいくつかの課題がある。その課題とは、

- シミュレーションのように通常の WEB アプリケーション以上に時間のかかる処理を行うこと。
- 結果のアニメーションなどベクタグラフィックを扱うための処理。
- 画面遷移の無い動的なインタフェースの実装。
- ブラウザからの１ユーザあたり TCP セッション数の制限。

などがあげられる。これらが問題となる理由はそれぞれの技術の解説などで述べる。

WEB ブラウザ上で、という技術的な制約を設けたのは現在のネットワークシミュレータは専門研究を行う研究者のみに高額なパッケージソフトとして販売され、利用方法が難しい。しかし、それらのパッケージのネットワークシミュレータが提供する動作アニメーションはビジュアルでわかりやすく、専門家が口頭で技術を解説するよりも雄弁にその動作を語る。従来であれば研究結果のアニメーションを見せるためには同様のソフトウェアをインストールしてもらい、研究の重要な知的財産であるシミュレーションファイルを提供し、そしてそれを動作させる必要があった。

しかし WEB ブラウザ上で実装することで非常に簡単に URI を教えるだけで見せることができる。ネットワーク環

＊ 筑波大学システム情報工学研究科ＣＳ専攻ＯＳＤＰ研究室
〒305-8573 茨城県つくば市天王台 1-1-1
筑波大学電子情報工学棟 E 棟 107

境のあるプレゼン会場であれば実際の動作をプレゼンテーションすることが可能になる。本稿では WEB 上でのユーザーインターフェース実現のための技術とシステムの構成技術を解説し、アーキテクチャ、シミュレータの機能解説、シミュレーションの実現のアルゴリズムを解説する。

2. 関連技術

シミュレータ実装のために必要な技術を前項で述べた課題に則しながら、紹介、解説する。

〈2･1〉 Rich User Interface 技術

（1） Dynamic HTML

WEB ブラウザ上での表示はすべて HTML と CSS の組み合わせで表現される。WEB アプリケーションとしてインタフェースを実装する場合、この HTML を動的に書き換える必要性が出てくる。この際にブラウザ側（クライアントサイド）で動作するスクリプトを実行することで、書き換えを実現する技術の総称を DHTML(Dynamic HTML)という。以前は、WEB ブラウザの仕様に統一性はなく、独自の実装となっていたため、多くの人が閲覧することが想定されるサイトでは避けられる傾向があった。だが。現在ではこのための技術の多くが、次に紹介する DOM(Document Object Model)[(2)]という規格に基づいたインタフェース､及びメソッドを利用することが一般化している。

しかし､DOM API 群の実装はブラウザごとやそのバージョンごとに若干の違いを見せ、WEB 上でのアプリケーション開発の難しさの一因になっている。

他方では、DHTML を用いたインタフェース開発の場合、サーバを利用することや画面遷移などをする必要なく、多くの操作を行うことができるため、WEB 上のリッチインターフェースアーキテクチャとして再度注目を集めている。

（2） DOM: Document Object Model

W3C が XML 仕様に基づいた文章構造をプログラムで利用するための API として DOM(Document Object Model)というものを標準化している。XML 文書（XHTML も含む）をプログラム上で扱う際にそれらを Tree 構造として扱うモデルである（図 1 参照)。分散オブジェクトを扱うための技術である CORBA2.2 で定義されている OMG IDL を用いて定義されているので、オブジェクト指向言語との親和性が高い。本稿で述べるシミュレータではクライアントサイドでは JavaScript、サーバサイドでは Ruby による DOM の実装を利用している。またベクタグラフィックの制御のためにも XML 準拠の SVG を用いているため DOM による操作を用いて実装している。この仕様が普及する多くのブラウザに実装されたために現状の Dynamic HTML の利用の選択肢の幅が広がり、リッチインタフェース技術として認知されるようになった。図２に DOM によって XML テキストから生成されるオブジェクトを示す。

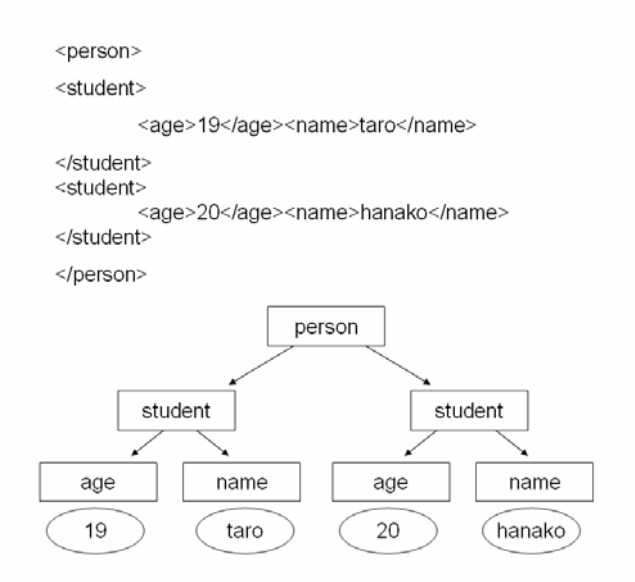

図 2 XML と DOM Tree

〈2･2〉 画面遷移のない通信のための技術

従来の WEB アプリケーションではブラウザに HTML で定義されている FORM やリンク URL を用いて、リクエストを送信していた。このため、サーバサイドからの情報を利用するためにユーザには必要の無い画面遷移が押し付けられるという状況であった。標準的なブラウザの多くに画面遷移に依存しないサーバサイドとの通信方法が搭載されるようになりこのような状況から改善され、既存の WEB アプリケーションの限界を超え、リッチインタフェースに適合したアプリケーションの実装が可能となった。そのために用いられる技術についての解説をする（図３)。

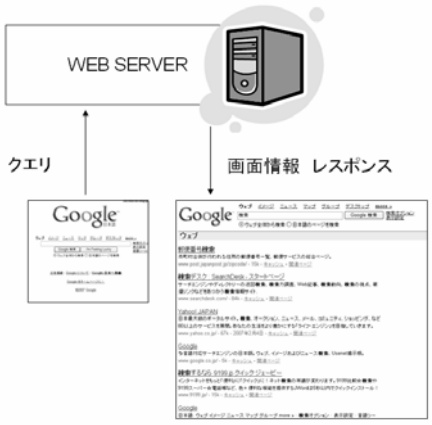

図 3 画面遷移のある WEB アプリケーション

図３は、画面遷移の存在するアーキテクチャにおけるサーバプログラムとの連携を示す。画面全体の情報を通信のたびに必要とするために全体的にスムーズな操作を実現することが難しい。もっとも、一般に認知されている WEB アプリケーションの多くがこのような形態を持っているため、図４にユーザアクションが画面の遷移によって阻害される様子を示す。ユーザアクションとして、データ入力する場合には、クライアントサイドで UI のフィードバックが

あり、ユーザの行動は阻害されることはない。しかし、サーバサイドにデータを送信するために Submit 動作をする場合には画面情報を取得し、そのすべてをレンダリングする間、次のデータ入力などを行うことができず、ユーザのアクションは阻害されてしまう。そのため、既存の WEB アプリケーションのユーザアクションは画面の遷移を基本の単位としたものとなり、クライアントアプリケーションとは異なる操作性を要求することになってしまっていた。

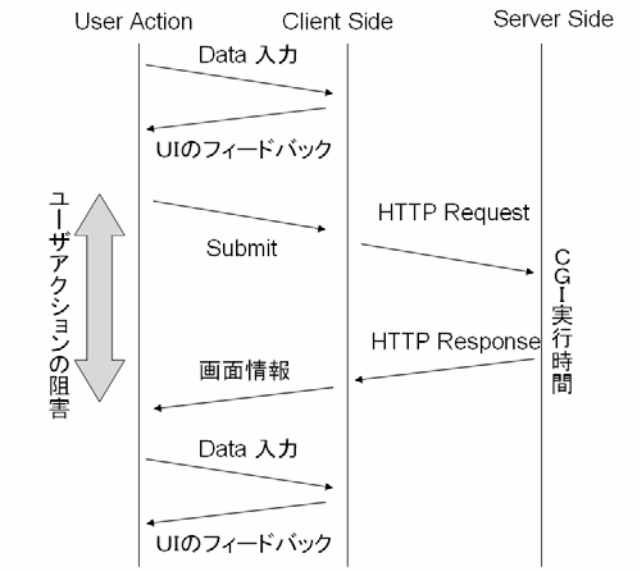

図 4　画面遷移のある WEB アプリケーションの動作シーケンス

（1）　CGI

CGI(Common Gateway Interface)とは WEB サーバ上でプログラムを起動するための仕組みである。HTTP リクエストの内容やクエリーのデータを取得可能にし、標準出力と HTTP レスポンスの Body を接続することで平易に WEB サーバ経由でのプログラミングを可能にしたインタフェースであり、画面遷移を用いるＷＥＢアプリケーションで頻繁に利用されていた。しかしシミュレーションなど、結果のレスポンスまでに長い時間を要するプログラムでは、この仕組みを単純に利用することは難しい。なぜなら、CGI の呼出し毎に Http 通信のためのソケットと CGI プログラムの標準入出力を接続して独立したプロセスを fork し、実行を行うからだ。その結果、通信の切断とプロセスの終了は同義となるため、長時間の処理をサーバサイドに要求するとレスポンスが変える前にブラウザを閉じれば相手のプロセスも強制的に終了してしまう上、ブラウザのタイムアウト時間に依存してしまう。WEB ブラウザ上で長時間の処理の依頼をするために本研究では CGI をインタフェースとして利用した特殊なアーキテクチャを用いている。

（2）　Ajax[3]

Ajax（エイジャックス）とは、Asynchronous JavaScript + XML の略称と 2005 年 2 月 18 日に Jesse James Garrett により定義され、状況によっては CSS や DOM などの技術も含むユーザーインターフェース構築技術の総称である。近年のブラウザに標準的に搭載されるようになった XMLHttpRequest（HTTP 通信を行うための JavaScript 組み込みクラス）や、その他非同期通信を可能にする DOM API などを用いることによる非同期通信を利用し、通信結果に応じてダイナミック HTML で動的にページの一部を書き換えるというアプローチを取るものである。広義においては XML を利用することに必然性は無く、後述する JSON（JavaScript Object Notation）などのデータ形式も用いられることが多い。

Ajax による非同期通信の様子について図 5 に示す。ユーザアクションを阻害せずにサーバの情報や CPU 資源が利用可能になる。図 4 と異なり、サーバサイドに情報を送信した直後であっても、ユーザアクションは阻害されること無く次の動作が可能である点に注目してもらいたい。また、受信したデータを元に、画面全体でなく UI の一部のみを書き換えることによって連続したユーザアクションが期待できる。この操作性は既存のクライアントアプリケーションと同等のものであり、WEB アプリケーションの可能性を広げることに一役を買っている。

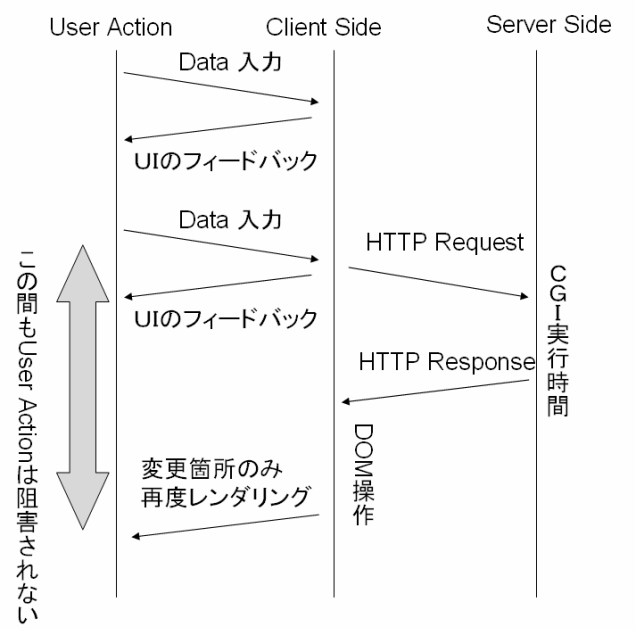

図 5　Ajax を用いたアプリケーションの動作シーケンス

（3）　JSON[4]

JSON とは JavaScript Object Notation の略語であり、JavaScript で用いられるデータ記述形式のサブセットとして定義されているが、その利用は JavaScript だけにとどまらず、多くの言語で、通信のオブジェクト記述言語として利用されている。その性質上から JavaScript 上で利用する際には eval 関数でデータを再生するだけで利用できるため、WEB アプリケーションで非常に良く利用されている。本シミュレータにおいても軽量なデータ記述形式であることなどからマークアップ言語のように扱っている。図６に JSON の例を示す。

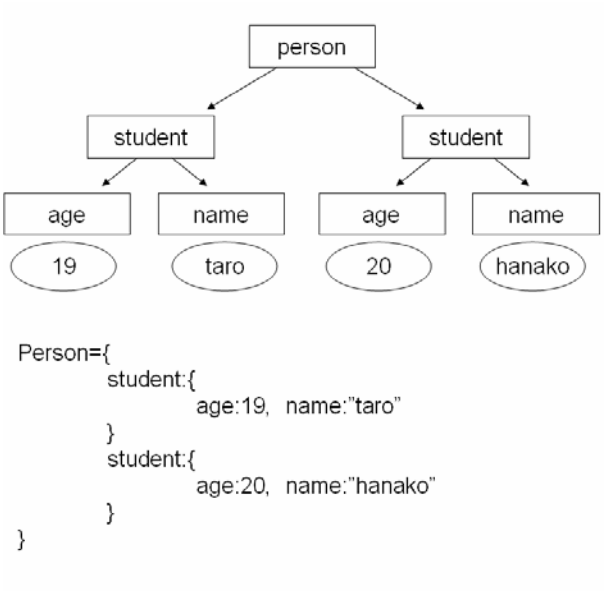

図 6　Object と JSON 表現の例

〈2･3〉　TCP セッションコントロール技術

Ajax を用いたアプリケーションで一般に問題となることは HTTP リクエストごとに発生する TCP のセッション数の問題である。サーバ側からリアルタイムに情報をやり取りしたい場合にその変化をポーリングするため、数百 msec ごとに HTTP リクエストを発行する必要性があるなどし、TCP セッション数が増え、サーバ不可が増大するなどの問題が存在する。また、ポーリング間隔を長くすることでレスポンスの悪化などインタフェースとしての欠点を抱えることになってしまう。その問題を解決するために現在ではチャットアプリケーションの作成などのために Comet という技法が用いられている。本研究では Comet の技法を発展させた手法を用いて、サーバサイドとクライアントサイドの通信セッション数や頻度などを調整している。

(1)　Comet[5]

Comet（コメット）とは、クライアントサイドによるポーリングの負荷を極力少なくするための技術であり、従来では不可能であったサーバ上で発生したイベントを即座にクライアントに送信することが可能になる技術である。一般的に、Web サーバはクライアントからのリクエストを受け取るとすぐにレスポンスを返す用に設計を行う。しかし、Comet を利用した Web アプリケーションでは、サーバはクライアントからのリクエストに対してすぐに応答せず、保留状態にしておき、サーバ上でなんらかのイベント(チャットで誰かが発言した等）が発生したときにレスポンスを返す。こうすることによって、サーバで発生したイベントを即座にクライアントに送信することができる。技術的な手法として一般的なものについて詳細に述べる。図７はポーリングを用いたサーバ監視プログラムの動作シーケンスである。Tp はポーリング間隔の時間をあらわし、☆の図はサーバ上でのデータ更新などの変化を表す。図に示すとおり、サーバのアクション発生から、クライアントサイドでその変化を通知されるまでには Tp に依存する長い Delay が発生する。Tp を小さくすることで、Delay を少なくすることが出来るのだが、その文、HTTP リクエストの発生数は多くなり、サーバの負荷が増大する。図８に Comet を用いたアプリケーションの動作シーケンスを示す。典型的な Comet の手法では、サーバの動作監視にポーリングの手法はとらず、サーバからの応答があるまで、sleep 状態で待機し続ける。しかし、前述したとおり、サーバサイドのプログラムは独立したプロセスとして起動する CGI として実装されることが多いため、外部の状態を監視しながら sleep 状態で居続けることは難しい。そこで、sleep に入る前に CGI 自身の PID を記録しておき、サーバ側のプログラムの変化発生時にその PID を持つプロセスに対して USER 定義の SIGNAL を発行する。これによって sleep 状態から脱し、レスポンスを返すといった動作を行う。これによってイベント発生からクライアントサイドへのレスポンスまでの delay は格段に少なくなり、ポーリングと異なりむやみやたらに HTTP リクエストを発行することなくサーバとのデータの同期などを行うことが可能になっている。このような利点から、Comet 技術は、複数のユーザと編集情報を同期するなどの用法のほか、チャットなどの高速なレスポンスが要求されるアプリケーションで用いられることが期待されている knowhow といえる。一方で実装の複雑化と従来の WEB アプリケーションエンジニアに要求されていた知識と異なる種類の知識が必要になるため、一部の優秀なアプリケーションを除いては実装が少ない。

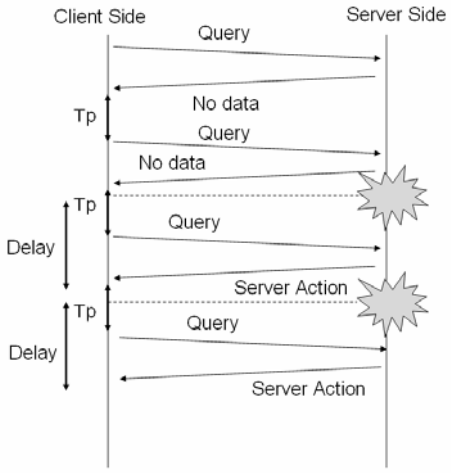

図 7　ポーリングを用いたサーバの監視

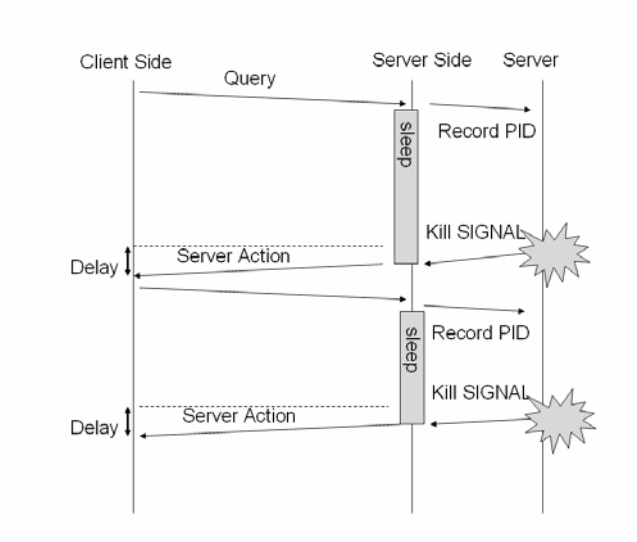

図 8　Comet を用いたサーバの監視

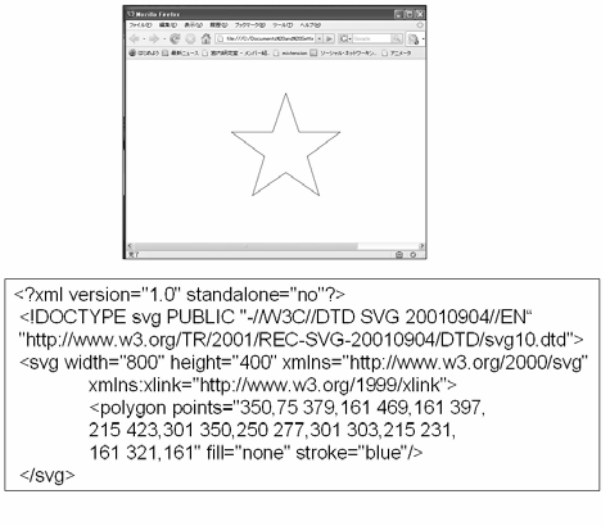

図 9　SVG の表示とソース

〈2･3〉　ベクタグラフィック技術

HTML や CSS にはベクタグラフィックを扱うための定義が存在しない。シミュレーションで用いるアニメーションのような柔軟なベクタグラフィックを扱うための方法論として一般的なものはいまだ存在していない。本シミュレータでは SVG という W3C の策定した XML によってベクタグラフィックを扱う形式を利用するものとした。これには２つの理由がある。１つは、SVG 自体が XML で構成され、JavaScript の DOM API が利用できるという点である。もう１つは Firefox や Opera といった一般に良く利用されるブラウザにデフォルトで SVG を扱うためのエンジンが実装されつつあるという情勢である。一方、ブラウザ市場で最大のシェアを持つ Internet Explorer には SVG を扱うためのエンジンが実装されておらず Adobe の提供するプラグインを利用する必要があるといった難点が存在している。Flash のように完全なプラグイン型でベクタグラフィックを構成するものも存在するが、DOM を用いた標準的な手法による操作ができないことや、独自の Action Script 言語を習得する必要があるため、開発時のリソースという問題を抱えている。

(１)　SVG[6]

SVG(Scalable Vector Graphics)は、XML によって記述されたベクタグラフィック言語のこと、或いは、SVG で記述された画像フォーマットのこと。W3C で標準として勧告されている。画像とその XML 例を図９に示す。SVG は画像フォーマットとして定義されながら、HTML のようなマークアップ言語で構成されている。このため、人間でも読みやすく、手作業で編集することも可能である。また、XML でもあるため,内部にスクリプトを埋め込むことで動的に内容を書き換えることもでき、非常に多彩な表現が可能である。図９の SVG では XML 宣言と Document Type の宣言のあと、Document Root として SVG タグの中に画像のサイズなどが記載され、その子として Polygon タグが存在している。Polygon タグの Points 要素によって指定された点をつなぐ多角形を構成している。

3.　シミュレータの実装

シミュレータの実装について詳細に述べる。最初にシステム全体を構成する要素とその連携のためのアーキテクチャについて述べ、次に Comet 技術の応用として、非同期通信のためのセッションコントロールの技法について解説をする。シミュレーションそのものの手法について述べ、インタフェース上の機能について解説をする。これらの技法はこのシミュレータに独自のものも多く、WEB アプリケーションの可能性を広げるための試みでもあるといえる。

〈3･1〉　システムアーキテクチャ

本研究のシミュレータは主に３つの要素で構成されている。１つがクライアントサイドで動作する WEB Front-end。もう１つがシミュレーションを実行する Simulation Server、最後にこれらをつなぐ Session Bridge である。以降はこれらについて述べる。

(１)　WEB Front End

WEB フロントエンドは、図 10 のような複数の要素から構成されている。基本的には HTML+CSS といった WEB ページの構成であるが、特殊な class を持つタグを複雑な機能をもつ UI パーツとして再構築する Widget ライブラリや後述するセッションコントロールを行うための Session Control Engine ライブラリ、Widget ごとの挙動を定義するライブラリなどに分割され設計している。また、

Widget ライブラリとして UI パーツの構成を分離する利点は、blog などに貼り付け可能にする場合にユーザに特殊なプログラムコード自体を貼り付けてもらうことは、ユーザフレンドリで無く、煩雑であると考え、シミュレータ利用のための script タグと Widget class を持つ div タグを追加するのみで、WEB 技術に詳しくない blog 利用者にも使ってもらうことが可能になるだろうと考えているからだ。また JavaScript を有効でないユーザが閲覧をした場合にこれらのコンテンツが表示されず、通常のサイトとして自然に表示

されるようにすることが可能であるという点も、このような設計の重要な点である。このような WEB アプリケーションデザインの原則として”Graceful Degradation”(品のある劣化)という言葉がある。Widget ライブラリの挙動の例として図 11 を示す。Session Control Engine については、次項のセッションコントロールで詳しく触れる。API Wrapper はクライアントサイドから、サーバ側の API を呼び出す際の Wrapper として実装され、Node Manager や Animator などから容易にメッセージを送信することができる。

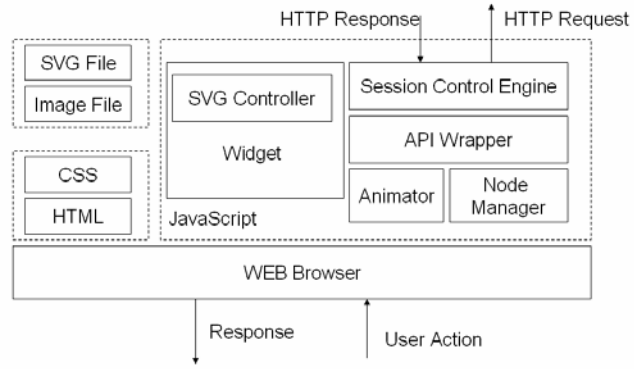

図 10　Web Front End の構成

図 11　Widget ライブラリの動作

（2） Simulation Server

Simulation Server の構成を図 12 に示す。Simulation Server の接続は Session Bridge からのみに限定されており、TCP サーバとして実装された Session Management 要素が通信をコントロールし、シミュレーションのための Thread を管理する要素を経由してシミュレーションを起動する。Node に関連したプロトコルをレイヤ毎に管理し、特定のフォルダにあるファイルを自動的にアドオンすることで Node 管理に利用する。また、シミュレーションは Event の Queue を基本単位とするため、その処理を行うルーティンとして Event Management 要素を用いている。

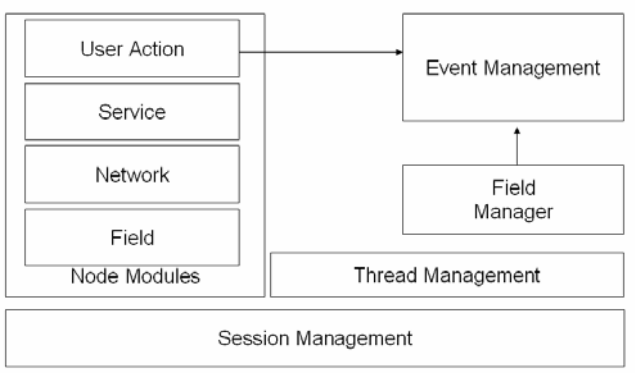

図 12　Simulation Server の構成

（3） Session Bridge

WEB Front End からの要求をシミュレーションサーバに送信し、その結果を返すなどの処理を行う CGI として実装されている。これらのセッションコントロールについては独立して次項で述べる。これは Comet の技術を拡張した処理技術として実装しており、WEB 上でのアプリケーションの可能性を広げる新しいアーキテクチャと言えるだろう。

〈3･2〉　セッションコントロール

本シミュレータでは頻繁に Simulation Server と WEB Front End での情報のやりとりを行う。そのため、効率的な通信のコントロールが必須となる。また、シミュレーションや動画像のエンコーディング処理といった時間のかかる処理をサーバに依頼するといった場合、通常の CGI やサーブレットといったアプローチでは限界がある。この手法を Two Way Connection Method と名づけ、その実装について解説する。

図１３に Server と Bridge、クライアントライブラリの通信の様子を示す。WEB Browser と WEB Server の通信は HTTP を用いて行われる。Bridge と Server は Raw な TCP 通信を用いて行われる。Simulation Server を複数展開することで計算量の負荷を軽減するためにもネットワーク通信を用いることは重要であると思われる。次に図１４にログイン時のシーケンスを示す。HTTP リクエストとしてクライアントサイドの Session Control Engine が WEB Server 上の Session Bridge に問い合わせをすると、指定された Simulation Server に通信を開始する。Simulation Server は Session Bridge にセッションＩＤを発行し、その ID を Session Control Engine に返信する。以降の通信はこのセッションＩＤを利用して、セッションを同定する。図１５にユーザアクションから、そのレスポンスが変えるまでのフローを示す。

クライアントはサーバに返信用のバックグラウンドのコネクションを発行する。次にユーザが通信のためのアクションを起こすと Tw の時間の間、待機する。その間に次のユーザアクションがあればそのデータも発行し、一度の通信で複数のメソッドの対応に対抗する。この通信は即時応答される。応答はすべてバックグラウンドのセッションを通じて行われる。サーバ側からの状態変化の通知などがあれ

ばその後、Tw の間待機し、すべてのレスポンスをまとめて返信することで通信量やクライアントの負荷、サーバの負荷を軽減できる。

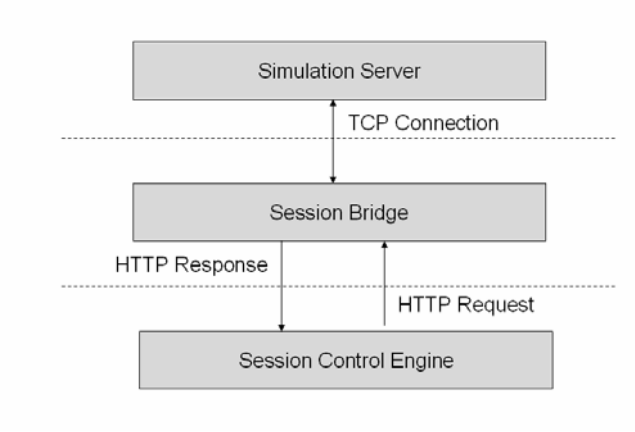

図 13　セッションコントロールのための要素

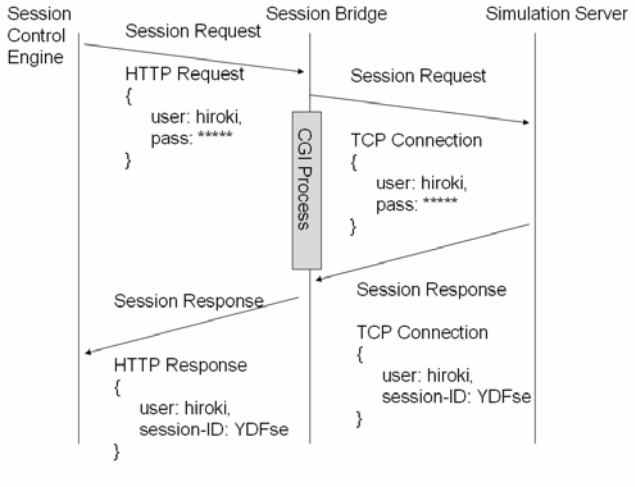

図 14 ログインフェーズのシーケンス

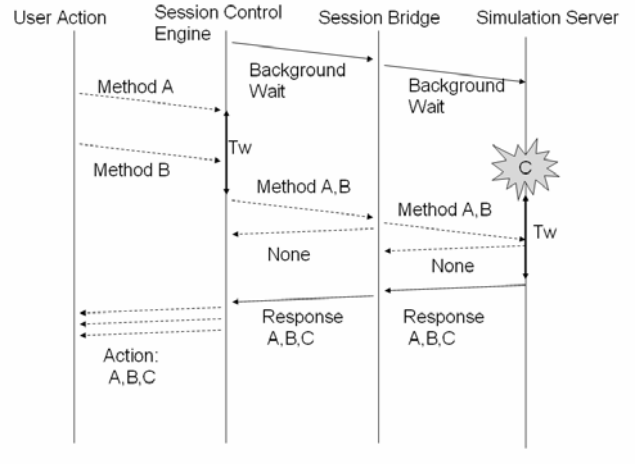

図 15　データ通信のシーケンス

〈3･3〉　シミュレーション

Ad hoc ネットワークでの基本単位はノードと呼ばれるルータ機能をもった無線端末である。これらを定義し、その定義に基づいてシミュレーションを行う。アプリケーションの動作のためのシミュレーションのため、物理層は数的解析のために単純なモデルを採用した。また、時間単位を区切ることで非同期なシステムを同期的に取り扱うという手法を採用している。これにともない、ルーティングアルゴリズムなどやデータリンク層の最適化などのために用いられるシミュレータに比べて、低レイヤの精度は高くは無い。しかし、アプリケーション全体の動作の効率やユーザの行動などを中心に検証できるシミュレーションを可能にすることに開発の主目的をおいた。以降でその特徴とアルゴリズムについて述べる。

（1） Layer

シミュレーションで利用するノードを定義する際にネットワークのモデルとして、頻繁に用いられるレイヤ構成を採用する。しかし、具体的な TCP/IP 実装などをシミュレーションするのではなく、あくまで提供アプリケーションの実装のための構成を用いた。その構成は、

・User Action
・Service
・Network
・Field

の順にユーザから遠ざかって扱われるレイヤと定義する。順にその意味を解説する。

User Action)　ユーザの行動を定義するレイヤ。端末を所持しているユーザがどのような原理で行動するかなどを定めるほか、どのようなタイミングでアプリケーションを実行するかなどを設定することができる。このような定義によってユーザの行動を中心にアプリケーションの状態を把握することが可能になる。

Service)　ユーザが利用可能なアプリケーションに対する定義を行うことができる。サービスといってもサーバ機能をもつアプリケーションだけでは無く、クライアント機能も定義できる。User Action 層と Network 層のインタフェースを利用することでレイヤの分割を行う。

Network) ルーティングやネットワークを利用するサービスへパケットを渡すための層。この層に新しいルーティングアルゴリズムなどを定義することで、サービスに特化したルーティングアルゴリズムなどを調査することが出来る。

Field) ユーザが行動するフィールドに関する定義を行う層。無線の受信範囲に入っているかどうかを調べ、上位レイヤにパケットをバッファし、一定の無線範囲にパケットを送出することを担当する。

（2） Event Driven

Event Driven とはプログラムの実行をイベント発生時まで待機し待ち続けることで、対話的なプログラムを実現する実装方法のことであるが、シミュレーションにおいては、非同期のアプリケーションのシミュレーションのために、時間軸にそったイベントを定義し、その Event Queue の逐次的な解釈によってシミュレーションを実現する手法であ

る。たとえば、ノードごとに１つの Thread を担当させ、性能評価をおこなうことはそれぞれのプログラムの容易さを向上させるが、負荷の調整や、任意時間でのステートの確認などが難しくなる。本シミュレータではノードが多様な状況でのシミュレーションを行うためにそれぞれのレイヤのプログラムがEvent Queueの中にメッセージを残すといった手法を採用している。

（3） Step

非同期のシミュレーションを行うに際し、完全な再現性を確保することは難しい。また、アプリケーションの構成するネットワークを分析する際に任意のタイミングに一般化することも困難である。そこで、本シミュレータではシミュレーション後の数理的解析をサポートするために Step という時間単位を定め[7]、すべての処理をその時間単位の整数倍によって表現することで様々なシミュレーションを実現している。

（4） Disk Graph[8]

Ad hoc ネットワークの通信をモデル化する際に、無線ネットワークのモデル化は難しい問題となる。ここでは受信範囲を半径固定の円として定義し、その円のなかにノードが属していた場合に受信可能とするような単純なモデルを採用した。これによってアプリケーションのシミュレーション精度に影響が出るような場合も想定し、Field レイヤに新しいプロトコルを定義することで、より柔軟なモデル化も行うことが出来る。しかし、柔軟な物理レイヤのモデルやネットワークレイヤのモデルの採用は数理的解析を困難にし、一般性をもったアプリケーションの有用性を示すことは難しい問題になってしまうだろう。シミュレータユーザはこの定義に対して費用対効果をどれだけ求めるのかといった問題を考えた上で採用することが望ましい。

〈3･4〉 アニメーションインタフェース

（1） 拡大縮小

フィールドに表示されたノードの配置などをマウス操作のみで移動させ、スライドバーの操作によって画面の拡大及び縮小を実現している。これによって、アニメーション時も対話的に特定のノードの位置を確認することや、全体図を見るなどといった操作が可能になっている。この操作性は Google Maps[9]などの既存のアプリケーションと同様であるため、ユーザは直感的な操作でアニメーションを閲覧することが可能になっている。スライドバーは図１６の①。

（2） ローディングと逐次実行

アニメーションを開始するとデータのロードを行いなら、通信データや移動データを解釈してアニメーションを実行する。図１６の②のようなインタフェースで、再生及び停止を制御することが出来る。

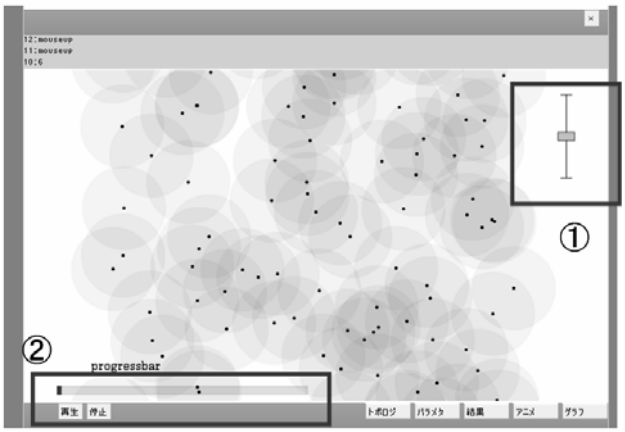

図 16 アニメーションインタフェース

4. まとめ

ＷＥＢ環境という非常に制約の多い環境で、アドホックネットワークのシミュレータを実現するための、技術や実装について示した。研究者の発想した新しいアルゴリズムや研究のイメージが世界中のインターネットユーザに容易に閲覧可能になることは教育的にも啓蒙的にも、また研究者自身のフィードバックにもよい影響をあたえられるものと信じる。

文　　献

（1） MANET WG,http://www ietf org/html charters/manet-charter html

（2） DOM: http://www w3 org/DOM/

（3） Ajax: A New Approach to Web Applications Adaptive Path (2005-02-18),http://www adaptivepath com/publications/essays/archives/000385 php

（4） (RFC: 4627)The application/json Media Type for JavaScript Object Notation

（5） Dr Rohit Khare,
Beyond AJAX:Accelerating Web Applications with Real-Time Event Notification,http://www knownow com/products/docs/whitepapers/KN-Beyond-AJAX pdf,2005

（6） W3C SVG, http://www w3 org/Graphics/SVG/

（7） 内田 真人, 佐藤 大輔,コンピュータウィルス拡散過程の数理モデル化と解析,電子情報通信学会 情報セキュリティ研究会, 信学技法 Vol 104, No 200, pp 1-6,2004

（8） Fabian Kuhn, Roger Wattenhofer,Ad-Hoc Networks Beyond Unit Disk Graphs,International Conference on Mobile Computing and Networking ,2003

（9） Google Maps, http://maps google co jp/