# 移動を伴うマッシュアップアーキテクチャの提案と Google Maps を用いたカーナビゲーションシステムの開発

2006MI067 桂川 健　2006MI091 丸山 剛　2006MI147 櫻井 利家

指導教員 青山 幹雄

## 1. はじめに

現在，カーナビゲーションシステムが多くの自動車に搭載されるようになった．しかし，現在のカーナビゲーションシステムはプラットフォームに依存する問題がある．本研究ではネットワーク通信によるプラットフォームに依存しないカーナビゲーションシステムの実現技術を提案する．

## 2. 研究の目的

前述した問題点を解決するために，ネットワークを通じて公開されている Google Maps[1]を用いて，GPS により取得した位置情報をマッシュアップ[2]したカーナビゲーションシステムの実現技術を提案する．また，様々な Web サービスを付加サービスとしてマッシュアップし，機能性拡張を行う．その際，自動車のスピードに追随できるようなアーキテクチャを実現する必要がある．

## 3. 研究課題

GPS による位置データと Google Maps の地図データ，Web API によるマッシュアップを行うためには，以下の問題点が挙げられる．

(1) マッシュアップの形態 ： 一般的にマッシュアップはクライアントサイドで行われる．機能を提供するプロバイダの Web API に依存してしまい，プロバイダ側のルールに従って，リクエストを送り，レスポンスで用いる XML や JSON を解釈する必要がある．

(2) 位置情報の処理遅延 ： GPS レシーバで受信したデータの処理時間がかかり，現在位置の表示に誤差が生じる．

(3) JavaとJavaScriptの非互換性 ： Google Maps APIを制御する JavaScript はブラウザ上の処理であり，GPS データを制御している Java のプログラムを呼び出すことが不可能である．

## 4. アプローチ

### 4.1. 前提条件

(1) 自動車から高速なネットワーク接続が容易である．

(2) 開発に利用する Web API が公開されている．

(3) ネットワーク障害は起きないものとする．

(4) Web API へのアクセスは REST で行う．

### 4.2. 拡張 MVC モデル

本研究では，MVC モデル[3]に基づき問題点を解決することが可能である，拡張 MVC モデルを提案する(図 1)．

従来の MVC モデルとは異なり，Google Maps API を用いるため，View がクライアント側へ移行し，HTTP によるメッセージの交換が必要となる．

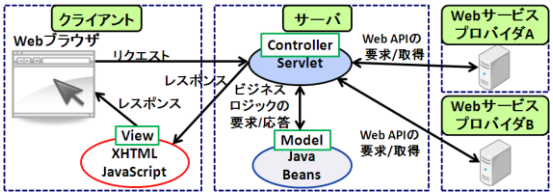



図 1 ： 拡張 MVC モデル

### 4.3. 位置情報と地図情報の移動を伴うマッシュアップ

Java と JavaScript には互換性がないが，両者を連携させる方法を提案する．

(1) JSON ファイルを介した連携方法

Java から JSON ファイルに測位データを出力し，その JSON ファイルを JavaScript が読み込む(図 2)．

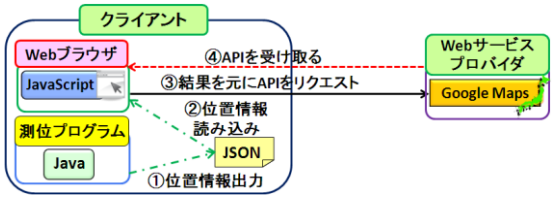



図 2 ： JSON ファイルを介した連携方法

(2) サーバを介した連携方法

Java からサーバに測位データを送信し，JavaScript がサーバを呼び出して結果を受け取る(図 3)．

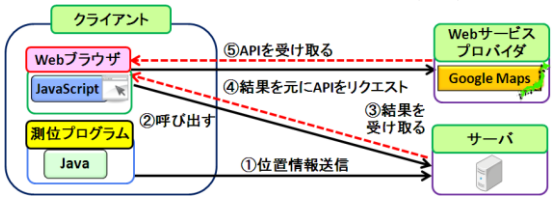



図 3 ： サーバを介した連携方法

### 4.4. 付加サービスにおけるサーバサイドマッシュアップ

マッシュアップの形態として，クロスドメイン制限を解決するためにプロキシサーバを経由する方法や JSONP を用いる方法で行うクライアントサイドマッシュアップが挙げられる．しかし，研究課題を解決するためにはサーバで処理を行うサーバサイドマッシュアップが望ましい(図 4)．

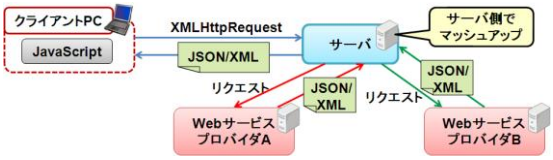



図 4 ： サーバサイドマッシュアップ

## 5. 提案アーキテクチャ

前述のアプローチを用いて，GPS から取得した位置情報と Google Maps の地図情報をマッシュアップした Google Maps ナビゲーションシステムを提案する．クライアントサイドマッシュアップと MVC モデルを拡張したサーバサイドマッシュアップの 2 つのアーキテクチャを提案する．

### 5.1. システムの構成要素

(1) ナビ制御システム ： 地図情報と位置情報をマッシュアップし，Web ブラウザに表示させるシステムである．XHTML と JavaScript で実装する．

(2) GPS コントローラ ： GPS レシーバから測位データを受信して位置情報変換システムに測位データを送る．GPS コントローラ内では Java で扱える値に変換する．Java で実装する．

(3) サービスマッシュアップシステム ： 天気情報や宿泊情報をマッシュアップし，その処理結果をナビ制御システムに送る．サーバを用いた付加サービスのマッシュアップでのみ使用する．Java で実装する．

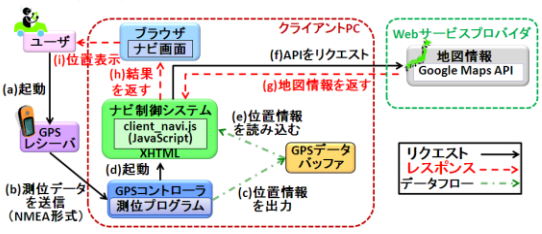
(4) 位置情報変換システム ： GPS コントローラから測位データを受け取り JSONP に変換する．その後，ナビ制御システムを起動する．Java で実装する．

(5) GPS データバッファ ： GPS コントローラから測位データを書き込む．位置情報がナビ制御システムでも扱えるように JSON 形式のテキストファイルとする．

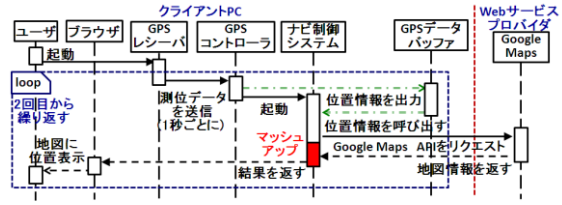
### 5.2. カーナビゲーションのアーキテクチャの実現方法

地図情報と位置情報をクライアントサイドのみで処理を行うアーキテクチャ(図 5)とサーバサイドで処理を行うアーキテクチャ(図 6)を提案する．また，図 6 は図 5 に示したクライアントサイドマッシュアップの処理(c)~(g)に対応する部分のみ図示する．

(1) 地図情報と位置情報のクライアントサイドマッシュアップ(C-M)




(a) システム構成

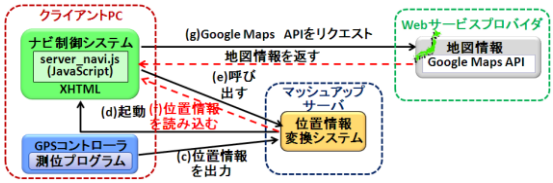



(b) シーケンス

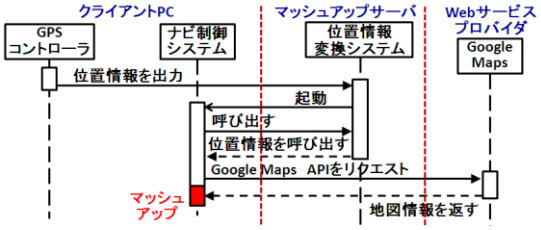
図 5 ： 地図情報と位置情報の表示(C-M)

(2) 地図情報と位置情報のサーバサイドマッシュアップ(S-M)

クライアントでの処理の場合は，GPS コントローラが変換処理を行っていたが，サーバで処理を行うことによってクライアントの負荷が軽減できる．




(a) システム構成

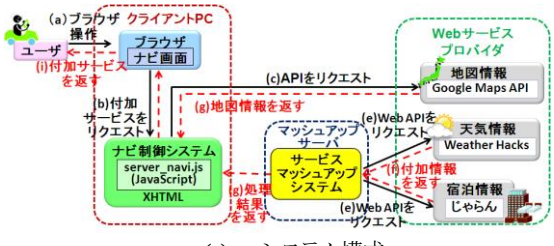



(b) シーケンス

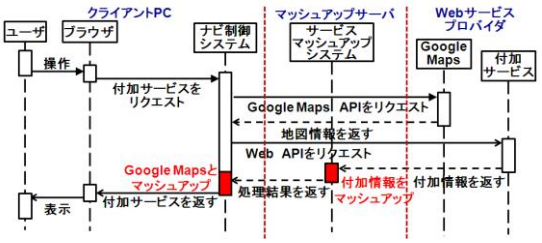
図 6 ：地図情報と位置情報の表示(S-M)

(3) 付加サービスのサーバサイドマッシュアップ

サーバサイドで地図情報と天気情報，宿泊情報のマッシュアップを行うアーキテクチャとその振る舞いを図 7 に図示する．図 7-(b)では付加サービスに天気情報と宿泊情報を用いる．




(a) システム構成




(b) シーケンス

図 7 : 付加サービスとのマッシュアップ

# 6. プロトタイプの実装

## 6.1. 実行環境

プロトタイプの実行環境を以下に示す(表 1).

表 1 : 実行環境

(1) クライアント環境

| | |
|---|---|
| 実行環境 | Java SE 6 |
| OS | Microsoft Windows XP Home Edition SP3 |
| メモリ | 1.23GB |
| CPU | Intel(R) Celeron(R) M processor 1.00GHz |
| GPS レシーバ | GARMIN GA 25MCX |
| Web ブラウザ | Internet Explorer 8 |

(2) サーバ環境

| | |
|---|---|
| 実行環境 | Java SE 6 |
| OS | Microsoft Windows XP Professional SP3 |
| メモリ | 0.99GB |
| CPU | Intel(R) Pentium(R) 4CPU 2.60GHz |
| Web サーバ | Apache Tomcat 6.0 |

## 6.2. 実装方法

前章で提案したクライアントサイドのみで処理を行うアーキテクチャとサーバサイドで処理を行うアーキテクチャを実装した. また, 前提条件として, 複数のユーザからのサーバへの同時アクセスはないものとする.

## 6.3. 実装結果

実装した 2 つのアーキテクチャにおける, Java のクラス数とメソッド数(NOM)の合計とコメントを含む行数(LOC)を示す(表 2). また, JavaScript のファイル数と関数の数(NOF)と LOC を示す(表 2).

表 2 : 開発したプログラムのメトリクス

(1) クライアントサイドのみで処理を行うアーキテクチャ

| クライアント | | | | | | サーバ | | |
|---|---|---|---|---|---|---|---|---|
| Java | | | JavaScript | | | Java | | |
| クラス数 | NOM | LOC | ファイル数 | NOF | LOC | クラス数 | NOM | LOC |
| 3 | 10 | 192 | 2 | 14 | 134 | 2 | 4 | 229 |

(2) サーバサイドで処理を行うアーキテクチャ

| クライアント | | | | | | サーバ | | |
|---|---|---|---|---|---|---|---|---|
| Java | | | JavaScript | | | Java | | |
| クラス数 | NOM | LOC | ファイル数 | NOF | LOC | クラス数 | NOM | LOC |
| 2 | 7 | 183 | 1 | 15 | 140 | 4 | 10 | 322 |

## 6.4. 位置情報における遅延時間の評価

クライアントサイドのみで処理を行うアーキテクチャでは, クライアントが位置情報を取得するために GPS データバッファに測位データを出力し, ナビ制御システムが GPS データバッファを読み込んで結果を受け取る.

サーバサイドで処理を行うアーキテクチャでは, 位置情報変換システムに測位データを送信し, ナビ制御システムが位置変換システムを呼び出して結果を受け取る.

それぞれのアーキテクチャでは位置情報を Google Maps 上に表示するまでに処理遅延が発生するため, 現在位置に誤差が生じる.

## 6.5. 付加サービスのマッシュアップ処理の評価

付加サービスのマッシュアップ処理における通信時間は通信を行うタイミングが任意で, 頻繁に情報も更新されることがないため, 位置情報と地図情報とのマッシュアップとは切り離して考える.

## 6.6. 実験結果

2 つのアーキテクチャに対する位置情報取得の遅延時間を, 取得回数 100 回毎に取得時間の平均値をグラフにした(図8, 図9). また, 位置情報の取得回数100回毎にじゃらんにリクエストを送り, 付加サービスの取得時間をグラフにした(図 10). また, 各時間要素の定義を表 3 に示す.

表 3 : 各時間要素の定義

| 時間 | 定義 |
|---|---|
| $T_1$ | GPS コントローラから GPS データバッファに測位データを出力するまでの時間 |
| $T_2$ | ナビ制御システムから GPS データバッファを呼び出して結果を受け取るまでの時間 |
| $T_3$ | GPS コントローラから位置情報変換システムに測位データを送信して起動させるまでの時間 |
| $T_4$ | ナビ制御システムから位置情報変換システムを呼び出して結果を受け取るまでの通信時間 |
| $T_5$ | 位置情報変換システムの処理時間 |
| $T_6$ | 2 つのアーキテクチャにおける付加サービスの取得時間 |

(1) クライアントサイドのみで処理を行うアーキテクチャの位置情報取得による遅延時間

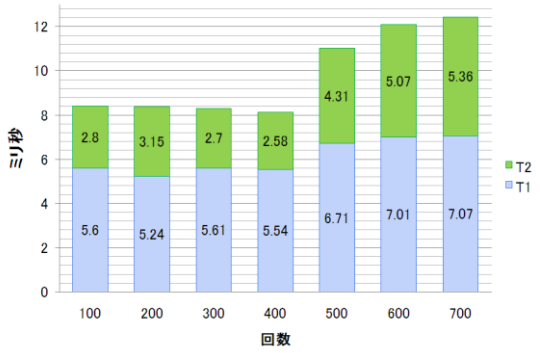



図 8: $T_1$, $T_2$ の推移

(2) サーバサイドで処理を行うアーキテクチャの位置情報取得による遅延時間

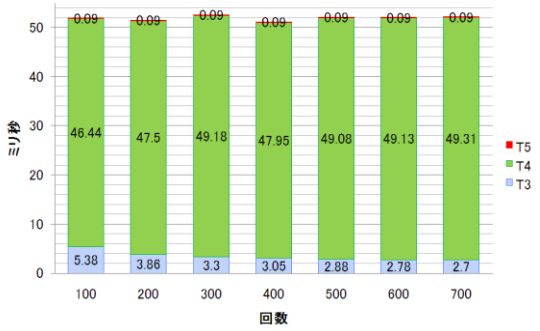



図 9 : $T_3$, $T_4$, $T_5$ の推移

(3) 付加サービスのマッシュアップの処理時間

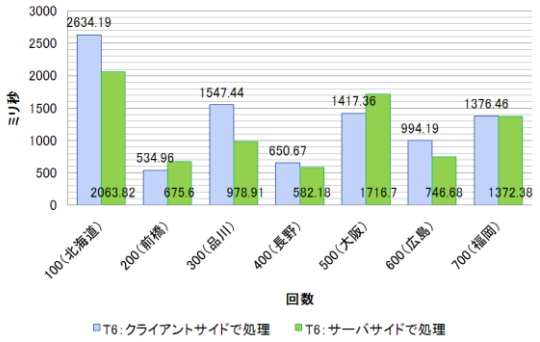



図 10 : $T_6$ の推移

## 7. 評価と考察

プロトタイプの挙動と，2 つのアーキテクチャの遅延時間及び付加サービスの取得時間の評価と考察を行う．

### 7.1. 位置情報表示における遅延時間の評価と考察

60km/h の自動車が進む遅延時間による距離の誤差が，クライアントサイドのみで処理を行うアーキテクチャでは 0.16m であり，サーバサイドで処理を行うアーキテクチャでは 0.86m であるため，遅延時間による距離の誤差はほとんどないといえる．図 8 で，100 回から 400 回までの遅延時間と 400 回目以降の遅延時間に変化があったのは，Google Maps との通信を連続的に行うことで，ブラウザのキャッシュへの負荷が大きくなったためと考えられ，キャッシュへの負荷を考慮する必要がある．また，図 9 で，回数毎の遅延時間に変化がなかったのは，$T_4$ が大きいため，相対的にキャッシュへの負荷による遅延時間の影響が少なかったためと考えられる．

遅延時間による両者の距離の誤差はほとんどなく，今後は付加サービスと位置情報をマッシュアップして多様なサービスを提供することが重要となるため，サーバサイドで処理を行うアーキテクチャが適切である．

### 7.2. 付加サービスのマッシュアップ処理の評価と考察

図 10 で，それぞれの取得平均時間は 1307.9 ミリ秒と 1160.9 ミリ秒で，取得する地域によってデータ量が異なるために，取得時間に差が生じた．最大で 2634.2 ミリ秒かかった場合もあったが，付加サービスの処理時間は秒単位で更新されることはなく， 60km/h の自動車が 2634.2 ミリ秒で進む距離は 43.9m であり，地図による画面遷移がほとんど行われないため，付加サービスのマッシュアップは自動車のスピードに追随できる．

## 8. 今後の課題

(1) ネットワーク障害に対する処理 : キャッシュを利用し，位置情報とのマッシュアップを一時的にクライアント側で行うアーキテクチャを提案する．

(2) 複数クライアントからのリクエストに対する処理 : 複数クライアントを考慮したアーキテクチャを提案する．

(3) メッセージの相互運用性の考慮 : SOAP にしか対応していない Web サービスも存在するため，SOAP にも対応するアーキテクチャを提案する．

## 9. まとめ

本研究では， 2 つのアーキテクチャを提案し，検証比較した．プロトタイプを実装，比較した結果，サーバを用いることによる通信の遅延時間による距離の誤差はほとんどないことが分かった．また，付加サービスをマッシュアップして多様なサービスを提供することが重要なため，サーバサイドで処理を行うアーキテクチャが適切であるといえる．

## 参考文献

[1] Google Maps API - Google Code, http://code.google.com/intl/ja/apis/maps/.

[2] 本田 正純，マッシュアップかんたん AtoZ，C&R 研究所，2007.

[3] 山田 祥寛，独習 Java サーバサイド編，翔泳社，2009.