マッシュアップ型アプリケーション開発モデルの提案

2005MT075 中村 浩二　2005MT130 横井 公紀

指導教員 青山 幹雄

1. はじめに

近年，Web APIの公開が増加しており，複数のWeb APIを組み合わせ，手軽にサービスを開発する手法であるマッシュアップが注目を集めている[1].

本研究では，マッシュアップに焦点を当て，リッチクライアントサービスの実現を目的とする．Web APIを用いたアプリケーション開発を分析し，体系的な開発モデルを提案する.

2. マッシュアップ型開発の問題点

2.1. マッシュアップ型開発

マッシュアップ型開発とは，複数のWeb APIを重ね合わせ， 新たなサービスを開発することである[5]．コンテンツを容易に作成でき，短期間でアプリケーションを開発できる.

2.2. MVCモデルに基づくマッシュアップ型開発の分析

MVC モデルの観点で考えると，マッシュアップ型開発は Web ブラウザ上の View の重ね合わせで成り立つ(図 1)．View のみの作成で済むため，開発工程が削減できる[2].

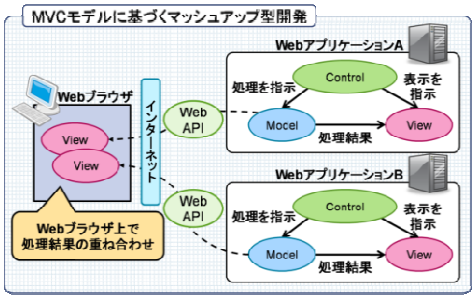

図 1 MVC モデルに基づくマッシュアップ型開発

しかし，マッシュアップ型開発は Model と Control の開発が不要であるため，Web API の処理の実行制御ができない．これを実現するには，View の重ね合わせが行われる前に Model の制御を行う仕組みが必要である.

3. 関連研究

開発手法が類似する Web サービス連携と既存のマッシュアップ型開発である MashMaker[4]を MVC モデルに基づき分析し，マッシュアップ型開発と比較を行う.

なお，本稿における重ね合わせとは，Web ページ上で取得したデータを直接貼り合わせて表示することであり，組み合わせとは，処理を制御して結果を表示することを指す.

3.1. Web サービス連携

Web サービス連携では，プロバイダによってあらかじめ組み合わせるサービスが決まっており，サービス実行前に組み合わせが行われる.

プロバイダは Web サービスの Model を制御するための Control を保持する(図 2)．これにより，サービスの処理の制御が可能となる．組み合わせが可能である点が，重ね合わせに限定されるマッシュアップ型開発と異なる.

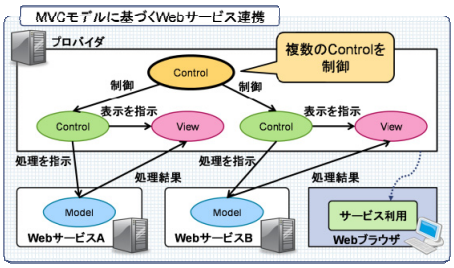

図 2 MVC モデルに基づく Web サービス連携

3.2. 重ね合わせによるマッシュアップ型開発

MashMaker の構造を MVC の観点で考えると，データを取り出す Widget が Control の役割となり，Model と View を制御する．Control は取り出したデータを直接 View に反映する．しかしサービスの連携を行うことはできないため，MashMaker は View の重ね合わせとなる.

4. アプローチ

マッシュアップ型開発は View の重ね合わせで成り立つ．この重ね合わせに着目し，MVC の観点から対象と方法を決定する.

4.1. 前提条件

(1) 開発に利用するWeb APIが公開されている.
(2) 複数のWeb APIを用いた連携を前提とした開発を行う.
(3) 利用するWeb APIはあらかじめ決まっている.
(4) Web APIへのアクセスはREST で行う.

4.2. MVCに基づく重ね合わせ対象の分析

4.2.1. Model の重ね合わせ

Modelの重ね合わせとは，公開されている複数のWeb APIが持つ機能を一つに統合した状態を指す．Modelを統合すると，統合前の個々のサービスが利用できない．よって，Modelの統合自体が困難である.

4.2.2. Control の重ね合わせ

Controlの重ね合わせにより，複数のModelの制御を考える．Controlはユーザからのリクエストを仲介して，複数のWeb APIを制御する．また，Controlはリクエストの仲介から，処理結果を受け取り，Viewを書き換えるまでの一連のロジック(実行順序)に従って行われる．Controlに各制御のタイミングを記述することで，定義した通りにModelとViewの両者を制御する．

4.2.3. View の重ね合わせ

View の重ね合わせは，従来のマッシュアップ型開発と同様，Web ページ上でデータを重ね合わせる開発である．

4.2.4. 重ね合わせ対象の決定

これまでの議論から対象を決定する．Model の重ね合わせは困難である．View の重ね合わせは，従来のマッシュアップ型開発と同じである．Control の重ね合わせでは，複数の Model を処理し，View の制御が可能と考えられる．以上より，本研究では Control の重ね合わせが有効と考える．

4.3. 重ね合わせ方法の決定

Controlの重ね合わせには，サーバあるいはクライアントでマッシュアップを行う方法がある(図3)．サーバサイドの場合，リクエスタとWeb APIの間でブローカが双方の情報を仲介する．クライアントサイドの場合，リクエスタへ制御プログラムを組込む方法がある．

(1) アプリケーションサーバを用いたブローカによる制御

アプリケーションサーバ内に複数のアプリケーションの制御順序を記述したプログラムを設置する方法である．リクエスタ側のプログラムに変更を加えることなく利用できる．ブローカを置くことはControlの特性に合致していると考えられる．

(2) リクエスタ内への制御プログラム組込み

リクエスタ内に制御プログラムを組込む方法である[3]．リクエスタがプログラムの使用をサポートしている必要がある．また，制御の規模が増大するとリクエスタの処理量が増加するため，多くのリソースが必要となる．

このため，本研究では(1)の Control の特徴を考慮し，(1)のブローカを用いた制御の仲介を提案する．

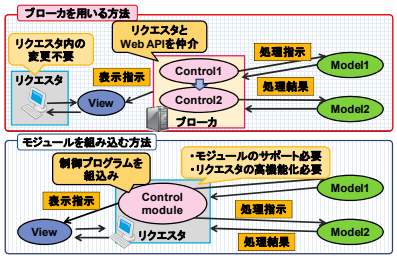

図 3 Control の重ね合わせ方法

5. 提案する手法

5.1. Controlブローカの提案

ブローカで複数の Model の制御を重ね合わせる．ブローカに置かれた制御プログラム中には，Web API と View の制御順序が記述される．また，リクエスタからのリクエスト情報に基づき，Web API へのリクエストを生成する．レスポンスが返却されるとロジックによる解析を行う．

5.2. ブローカ構築の共通条件

ブローカを成り立たせるための構成要素となり，任意のアプリケーションを開発する場合に必要な部分である．下記の共通部分の定義を行う．

(1) 全体の実行制御
(2) View 制御部
(3) Model 制御部

また，Control ブローカのアーキテクチャを図 4 に示す．以降で共通部分の概要とメッセージ交換について述べる．

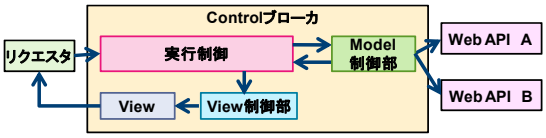

図 4 Control ブローカのアーキテクチャ

5.2.1. 実行制御

制御順序に従って各制御部に指示を出す．直接 Model や View に対する制御は行わない．

5.2.2. View 制御部

実行制御によって呼び出され，Viewの制御を行う． 出力に関するクラスがこの部分にあたる．

5.2.3. Model 制御部

実行制御によって呼び出され，Web API の制御とレスポンスの処理を行う．実行制御から送られた情報に基づき，Web API へリクエストする．また，そのレスポンスを実行制御に返す．

5.2.4. ブローカ全体の処理の流れ

ブローカ全体の処理の流れをシーケンス図に表す．Web API のレスポンスを直接 View に表示する場合のメッセージ交換を図 5 に示す．

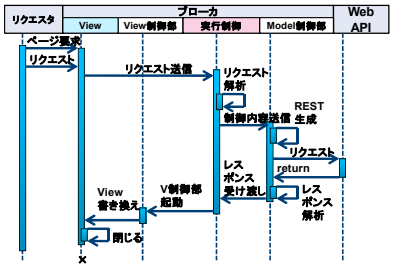

図 5 ブローカ全体の処理の流れ

複数の Web API の制御時は，実行制御による制御内容の送信からレスポンスの受け渡しまでの処理が複数回存在する．リクエスト対象の Web API は制御内容送信時に決定される．制御内容はその Web API に対応したものとなる．そのため，複数の Web API を制御するアプリケーション開

発の場合は，Model 制御部は任意の Web API を処理できなければならない．

5.3. Web APIの仕様に関する考慮点

5.3.1. Web API 構成要素の区別と意味定義

ブローカ構築の際には Web API の仕様に関する問題がある．Web API ごとに仕様の差異があり，個別に対応しなければならない．プロトコルやレスポンス形式の違いを「Web API の仕様差異部分」と表現する．これらは任意の Web API に対する再利用性が求められる．

また，リクエスト URL やレスポンスフィールドのタグ名のような Web API 特有の仕様を「Web API 固有の部分」と表現する．仕様差異部分と固有の部分の例を図 6 に示す．

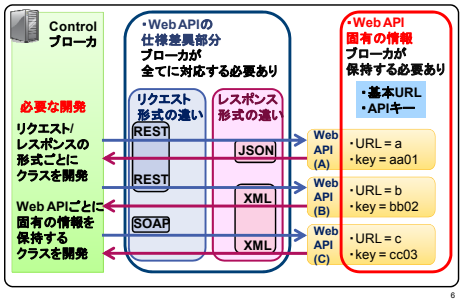

図 6 Web API の仕様差異部分と固有部分

5.3.2. Web API 仕様差異部分に関する問題

(1) リクエスト形式に関する問題

リクエスト形式が異なる Web API を用いた開発の場合は，Web API と通信を行う Model 制御部が形式の差異に対応しなければならない．例えば，REST と SOAP の双方に対応する場合は，各リクエストクラスの開発が必要である．

(2) レスポンス形式に関する問題

XML 形式のレスポンスを解析する場合，DOM による処理が考えられる．JSON 形式のレスポンスを解析する場合，Java 言語に JSON に関するインタフェースが存在しないことから，外部で公開されたパーサを利用する．

このため，双方の形式に対応するアプリケーション開発の場合は，各処理を行うクラスを開発する必要がある．

5.3.3. Web API 固有の部分に関する問題

(1) レスポンスフィールドに関する問題

第一に階層構造に関する問題がある．XML と JSON ではデータの参照方法が異なる．利用する Web API ごとにレスポンスの階層構造は異なり，常に同一の手順で全てのデータを取り出せるとは限らない．

次にレスポンスフィールドのタグ名に関する問題がある．直接名前を指定してデータ参照を行う場合，レスポンスに含まれる Web API 独自の固有名詞に対応する必要がある．

また，キー名の重複に関する問題がある．子要素が同一の場合，どの親の要素かを特定できなくてはならない．

(2) リクエスト URL，API キーの保持に関する問題

基本 URL と API キーはリクエストの際に必要不可欠なものであるが，Web API 固有のものである．ブローカが必ず保持し，リクエスト時に取り出せなくてはならない．

5.3.4. 仕様差異部分と固有部分の記述方法

仕様差異部分記述クラスを集約した固有部分記述クラスを開発する方法が考えられる．この例を図 7 に示す．これにより仕様差異部分と固有部分は明確に分離される．同一の Web API を用いた開発を行う場合には集約されたクラスを再利用できる．

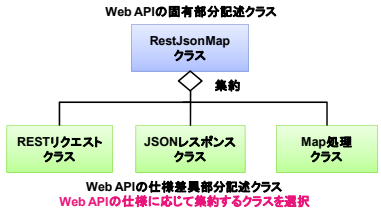

図 7 集約による固有部分記述クラス構成例

6. プロトタイプによる評価と考察

6.1. プロトタイプのアーキテクチャとクラスの定義

プロトタイプにより提案手法の妥当性を評価する．プロトタイプのアーキテクチャを図 8 に示す．

実装は，アプリケーションサーバに Apache Tomcat (6.0.18)を採用し，Java Servlet(JDK 1.6.0, Servlet 2.5, JSP 2.1)を用いて開発した．プロトタイプを構成するクラスと，開発したクラスのメソッド数(NOM)と規模(LOC)を表1に示す．

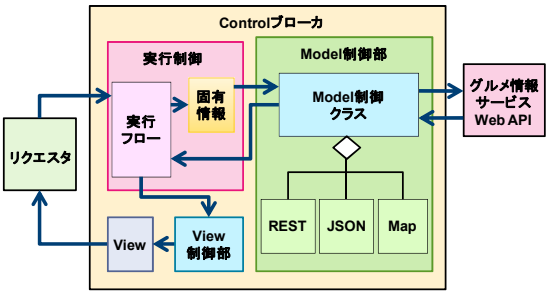

図 8 プロトタイプのアーキテクチャ

表 1 開発したクラスのメソッド数と規模

クラス	NOM	LOC
実行フロークラス	3	68
Web API 固有情報保持クラス	4	86
View 制御クラス	1	25
Model 制御クラス	6	112
REST リクエストクラス	1	59
JSON 処理クラス	1	45
Map 処理クラス	1	23
総合	**17**	**418**

6.2. 検証対象と条件

例としてリクルート社が提供するグルメ情報サービス ”HotPepper” の Web サービス(http://webservice.recruit.co.jp/)の中から，グルメサーチ Web API を用いたプロトタイプを開発した．検証条件を以下に示す．

(1) リクエスタからブローカへのリクエストは 1 回とする．

(2) ブローカから Web API へのリクエストも 1 回とする.
(3) Web API は JSON 形式のレスポンスを返す.

6.3. プロトタイプの実行例

以下のシナリオを実行フローに記述し, 動作を確認した.
(1) リクエストを受け取る.
(2) 固有情報クラス, Model 制御クラス, レスポンス参照用 Map, View 制御クラスのオブジェクトをそれぞれ生成.
(3) リクエストに必要なパラメータのセット.
(4) リクエスト URL を生成.
(5) URL に基づき, Web API にリクエストを送信.
(6) レスポンスを Map 形式へ変換する.
(7) Map をそのまま表示する.

このシナリオに基づいた実行フローの記述を図 10 に示す. また, プロトタイプの実行例を図 11 に示す. 提案手法に基づいたプロトタイプにより, ユーザのリクエストを基に Web API から処理結果を受け取り View に表示するまでのシナリオの実行が確認できた.

```
try {
    // (2) 固有情報クラス,Model制御クラス, レスポンス参照用Map
    //      View制御クラスのオブジェクトをそれぞれ生成
    HotPepperGourmetRequest hpp = new HotPepperGourmetRequest();
    HotPepperGourmetResponse hpgr = new HotPepperGourmetResponse();
    HashMap ResponseMap[] = hpgr.GetResponseMap();
    MakeView mv = new MakeView();

    // (3) 固有情報オブジェクトがリクエストに必要なパラメータのセット
    hpp.setAPIparameters("key", hpp.getAPIkey());
    hpp.setAPIparameters("keyword", request.getParameter("keyword"));
    hpp.setAPIparameters("format", "json");

    // (4) 固有情報オブジェクトがリクエストURLを生成
    hpp.makeRequestURL();
    // (5) 固有情報オブジェクトが生成したURLに基づき,
    //      Model制御オブジェクトがWeb APIにリクエスト
    hpgr.getAPIContent(hpp.getReqUrl());

    // (6) Model制御オブジェクトがレスポンスをMap形式へ変換
    hpgr.ParseJSONAndSet(hpgr.getContent());

    // (7) View制御部がMapを表示する
    int length = hpgr.getlength();
    mv.PrintResponseMap(length, ResponseMap, out);
```

図 10 実行フロークラスのロジック記述

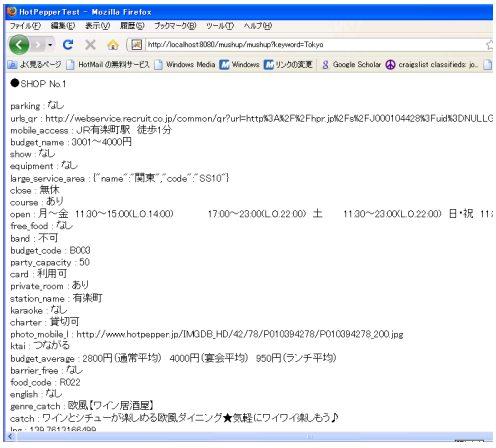

図 11 プロトタイプ実行例

6.4. 評価と考察

(1) クライアントサイドマッシュアップをサーバに移行

提案手法ではサーバサイドでマッシュアップを行うことで, JavaScript による開発において問題であったコードの秘匿性を確保した. また Java Servlet による開発で, ブラウザの種類に依存しない動作を実現した.

(2) Web API の Control の重ね合わせを実現

複数の Web API の Control を重ね合わせるブローカアーキテクチャを提案した. 提案したアーキテクチャに基づいたプロトタイプの実装を行い, 制御の変更が可能であることを確認した.

(3) Web API のインタフェースに対する再利用性の確保

提案手法では Web API の仕様差異部分と固有部分を明確に分離した. 仕様差異部分を記述したクラスを集約した新しいクラスに固有部分を記述し, 処理の正常性を確認した. さらに, 集約したクラスは同一の Web API を用いた開発であれば再利用可能である.

(4) ロジックを持つマッシュアップ開発の簡易化

固有部分を分離したことにより, 実行フローの記述はこれらの手続き呼び出しで実現できる. 固有部分を記述したクラスを揃えることにより, 実行フローの複雑なビジネスロジックの記述が不要となった.

7. 今後の課題

本研究ではリッチクライアントサービスの実現を目標としたが, 未達成に終わっている. 今後は実現に向けて, リクエスタとブローカ間のメッセージ交換および, View に関するモデルを定義する必要がある. また, Web API 単位での共通部分の抽出等について, より小さな粒度において検証する必要がある.

8. まとめ

本研究では Web API の重ね合わせ方法と場所に着目し, MVC モデルに基づいた分析からブローカによる Control の重ね合わせを提案した. 提案手法により, 複数の Web API を実行順序に従って制御することに成功した. 今後はさらに View の制御に関するアーキテクチャの詳細化を行うことにより, マッシュアップによるリッチクライアントサービス実現への応用が期待される.

参考文献

[1] 本田 正純, マッシュアップかんたんA to Z, シーアンドアール研究所, 2007.
[2] J. Wong, J. Hong, What Do We "Mashup" When We Make Mashups?, Proc. WEUSE IV/Proc. ICSE 2008, ACM, May 2008, pp. 35-39.
[3] 中村 一仁, 価値モデルに基づくWebサービスの動的選択と価値保証, 2005年度南山大学大学院数理情報研究科修士論文, 2006.
[4] R. Ennals, et al, MashMaker: Mashups for the Masses, Proc. 2007 ACM SIGMOD, Jun. 2007, pp. 1116-1118.
[5] X. Liu, et al, Towards Service Composition Based on Mashup, Proc. 2007 IEEE Congress on Services, Jul. 2007, pp. 332-337.