DBSJ Journal, Vol.7, No.4
March 2009

マッシュアップデータの選択的閲覧における効率的な部分更新

Partial Data Reconstruction for Selective Browsing of Mashups

長嶺 貴一♥　池田 宗平♥
鎌田 十三郎♦　草野 直樹♠
Takakazu NAGAMINE　Sohei IKEDA
Tomio KAMADA　Naoki KUSANO

現在我々が作成中のマッシュアップフレームワークでは，様々なデータリソースの組み合わせ方を定義しておけば，必要に応じたデータ構築がクライアント側で行われ，Web アプリケーション内では一つのデータイメージが共有される．利用者は興味のある部分をアプリケーション上で自由に閲覧可能であり，その際，閲覧対象が利用者の関心により移り変わっていく．本論文ではデータソースの変化に追従するための，部分更新手法を提案する．本手法では，効率的に更新を行うために，変化の影響を受ける領域のうち表示に必要な部分のみ再計算する．影響領域をシンボリックに扱うことで，領域チェックの低コスト化を図っている．また，結果が現在の表示に必要ない通信も削減する．

We are developing a framework to build flexible mashup applications where the user can browse mashup items selectively from various mashup items. The application developer only has to prepare a configuration of web service combinations, and our mashup engine presents a combined data structure that are constructed in a demand-driven manner. This paper proposes a partial data reconstruction method on receiving changes of data sources. Our system invalidates only data influenced by the changes, and constructs data currently needed for browsing. The method uses an efficient check mechanism of influenced area.

1. はじめに

近年，様々なWebサービスが公開され，それらを利用したマッシュアップが盛んになっている．我々はマッシュアップで得られたデータ(以下，マッシュアップデータ)を閲覧するWebアプリケーションのための，開発用フレームワークを提案している[1,2]．このフレームワークで作られた Web アプリケーション(以下，Web アプリ)では，フレームワークが提供するGUIコンポーネントを通して，利用者は現在関心のある部分について選択的に閲覧を進めていくことができ，その間，システムは閲覧に必要なデータのみを作成する．

♥ 学生会員　神戸大学大学院工学研究科情報知能学専攻博士前期課程 {nagamine,ikeda}@cs26.scitec.kobe-u.ac.jp
♦ 非会員　神戸大学大学院工学研究科 t_kamada@acm.org
♠ 非会員　神戸大学大学院工学研究科情報知能学専攻博士前期課程　kusano@cs26.scitec.kobe-u.ac.jp

開発者は，Web サービス[1]の組み合わせ方と表示方法を設定ファイルに記述するだけで，ブラウザ上で動作する Web アプリを作成することができる．

本論文では，我々のフレームワークで用いている部分更新手法を紹介する．この部分更新手法は，利用者の入力や Web サービスの結果などのデータソースの変化に追従し，効率の良い選択的閲覧を可能にするもので，変化の影響を受け，かつ表示に必要な部分についてのみ更新を行う．利用者が様々な入力を試しながら継続的に閲覧を行うような Web アプリにおいては，データソースの変化に自動的に追従し，新たなデータを表示に反映すべきであろう．その際，利用者の興味に応じて閲覧を進めている場合には，現在興味のある部分のみ再構築すれば十分である．そこで，我々のフレームワークの基本モデルに基づいて，マッシュアップデータ中の影響領域を求め，表示に必要かを判定し再構築を行う．その際，影響領域を変化のあったデータからの相対的な位置関係として静的に求めておくことで，表示に必要か否かの判定を，低コストに行うことができる．

2. 対象とするアプリケーション例

本節では，アプリ例を通して，選択的閲覧における部分更新と，その課題について説明する．図 1 は，ホテルを検索する単純なアプリ例である．利用者は地名，日付を入力し，ホテル検索サービスから取得したホテル一覧を左テーブルで閲覧する．マッシュアップ情報として，ホテル住所から駅検索を利用して得た最寄り駅までの距離などを，テーブルに列として追加することが可能である．他に，指定日における過去数年間の平均気温情報も上部に表示される．また，クリックしたホテルについて，指定日の空室情報が右テーブル上で閲覧できる．このようなアプリで，利用者が日付だけを変更した場合は，影響を受ける平均気温や空室情報のみを更新すれば十分である[2]．

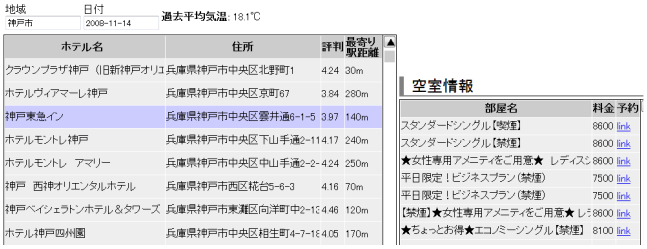

図 1 Web アプリケーション例(1)
Fig.1 Sample Application of our Framework(1)

我々のフレームワークは，開発者が与えた Web サービスの接続情報及び表示方法から，上記のようなアプリケーションを実現する．その際，マッシュアップに必要な計算を要求駆動で進めることで，利用者による選択的な閲覧を効率的に行う．例えば，左のテーブルでは，利用者は列の追加や，スクロールなどの操作が可能であり，システムは現在の表示部に必要なデータ取得のみを行う．加えて，取得した結果を随時計算・表示することで高い応答性能を達成している．

[1] 現在は HTTP GET リクエスト形式を持ち，XML 文書として結果を返す Web サービスを対象としている．
[2] 今回ホテル・空室情報の検索には[3]を，最寄り駅の検索には[4]を利用した．平均気温情報は，我々が用意した Web サービスを用いている．

さらなるデータ取得は，ユーザの閲覧操作に応じて自動的に進められる．また，利用者は列にフィルタ条件を設定することができる．その場合には，システムは条件に応じたデータ構築を行い，効率化を図る．例えば，ホテルの基本情報をもとに条件を設定している場合は，条件を満たさないホテルについて駅検索などを行わないことで，問合せ時間の短縮を行う．このように，システムはユーザの閲覧に必要なデータのみを作成している．本研究の目的は，データソースの変化に際しても，表示に必要な部分についてのみ更新を行うことである．

次に，より複雑かつ実際的なアプリ例を紹介する．図2は，観光用途に図1を拡張した例であり，ホテル検索以外に，同地域の観光地情報も検索され，地図上に表示される．観光地について，旅行予定日と同時期に撮影された写真を画像共有Webサービスから取得して表示しており，利用者は季節の写真を見ながら旅行日程や宿泊施設の検討を行うことができる[3]．このように高機能なインターフェイスを持つWebアプリでは，様々な入力・表示要素を含む可能性があり，ユーザ操作に応じて閲覧対象も簡単に変化しうる．

図 2 Web アプリケーション例(2)
Fig.2 Sample Application of our Framework(2)

また，部分更新は，ユーザ入力の変化だけでなく，ネットワーク上のデータソースの変化によって起こる場合もある．例えば，先の例における空室情報は，日付が最繁期に近い場合は，頻繁にデータ更新を行うべきであろう．あるいは，先のアプリケーションを複数の利用者間で共有して使えるように拡張する場合，各利用者の入力情報(日程や観光地・ホテルの選択情報)がネットワークを通じて共有されることになる．つまり，他のユーザの入力変化に応じて，マッシュアップ内容も更新されることになる．提案手法は，ユーザ操作によって不規則かつ動的に変化する表示対象範囲に対処しながら，各種の変化しうるデータソースや多様な表示形態に対応することを目指している．

3. 基本モデル

本節では，2章のWebアプリ(図2)を例に，本フレームワークにおける基本モデルについて概説する．基本モデルやアプリ事例の詳細については，文献[1, 2]を参照されたい．我々のフレームワークでは，マッシュアップデータを表現するためのデータモデル部と，データを閲覧するための表示部に分離したモデル化を行っており(図 3)，開発者は XML 形式の設定ファイルに，マッシュアップ定義と表示設定をそれぞれ記述する(図 5)．

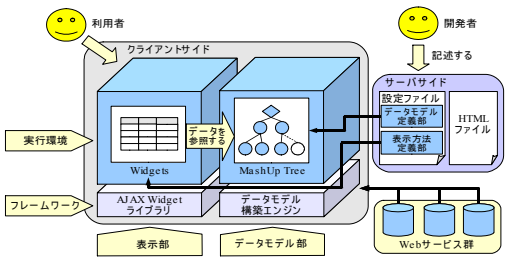

図 3 システム概観
Fig.3 System Overview

3.1 データモデル部

本フレームワークでは，マッシュアップデータを図 4 のような木構造(**Mashup Tree**)として扱う．データの流れは図中矢印で表現されており，例えば入力された地名(図中のplace)を元にホテル検索(hotelSearch)を行い，その結果から各ホテルの情報(hotel)を得るといった一連の流れが表現されている．各 node は，入力や Web サービス問合せ，文書処理などの操作を表しており，その結果取得されたデータを保持している．ホテル一覧のようなリスト形式データは，リスト要素をまとめる **list connector** と，要素を表す複数の子 node(**list element** と呼ぶ)の形で表現される．図中実線で表現される階層関係は，各ホテル(hotel)と，その名前(name)，空室検索(roomSearch)，空室情報(room)のリストといったデータ間の親子関係を表現している．

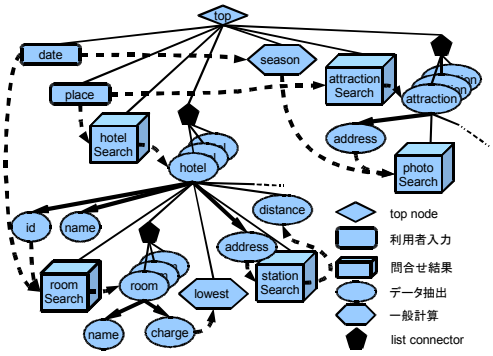

図 4 Mashup Tree
Fig.4 Mashup Tree

開発者は，node の構築規則を設定ファイルに記述する．各 node の構築規則を **node class** と呼び，利用者入力(記述例では input)，Web サービス問合せ(request)，XML 文書／要素からのデータ抽出(extract, extractList)，一般計算(operation)などの種類がある．node class は Mashup Tree と同様の階層構造を持つが，リスト形式データは一つの extractList から構築される．加えて，11−12 行目のように extractList を入れ子として記述することで，リスト形式データ中に更にリストを持つものも表現することができる[4]．また，記述する際には必ず，識別のための **node class 名**を

[3] 観光地情報，画像共有 Web サービスは，我々が構造化しなおしたものを使用している．地図の表示には Google Maps [5] を用いている．

[4] 現時点では，マッシュアップ定義の関数化には対応できていない．結果として，同じ node class が階層的に現れるような再帰構造も存在していない．

name 属性として与える．以降，ある node class c によって生成された node（リスト形式の場合は list element)を，c に属する node と表現する．

```
 1 <mashUpDef>
 2   <input name="place"/> <input name="date"/>
 3   <request name="hotelSearch" baseURL="http://...">   モデル部
 4     <refParam key="place" src="place"/>
 5   </request>
 6   <extractList name="hotel" src="hotelSearch" path="Hotel">
 7     <extract name="id" path="HotelID"/>
 8     <request name="roomSearch" baseURL="..." update="300">
 9       <refParam key="id" src="id"/> <refParam key="date" src="!.date"/>
10     </request>
11     <extractList name="room" src="roomSearch" path="Room"> ...
12     </extractList> ...
13   </extractList> ... </mashUpDef>
14 <widgetDef>
15   <embed htmlID="hotelTable">
16     <table id="hotelTable" target="TREE" foreach="hotel"> <columns>
17         <column nodeName="name"/>                          表示部
18         <column nodeName="address"/>
19         ... </columns></table></embed>
20   <embed htmlID="roomTable">
21     <table id="roomTable" target="WIDGET::hotelTable.selected" ...>
22       ... </table></embed>
23   <embed htmlID="maps">
24     <maps id="map"> <markers>
25       <marker target="WIDGET::hotelTable.onView"/>
26       <marker target="WIDGET::attractionTable.onView"/>
27       ... </markers></maps></embed>
28   ... </widgetDef>
```

図 5 設定記述例(一部省略)
Fig.5 Configuration Example

図 4 中矢印で表現されたデータ間の依存関係は，ソースとなる node の node class を指定することで行い，node class の src 属性を用いて指定する(図 5,9 行目など)．指定には，親 node class からの相対 Path を用い，祖先を辿る場合には，親 node の参照を記号“!”で表現する．例えば，空室検索(roomSearch)の問合せパラメータとなる日付(date)は“!.date”と指定する．なお，下向きパス中に extractList class を指定した場合，その全 list element をアクセス対象とみなす．これは，charge から lowest を計算しているように，最小値や合計値を求める operation class に引数列を渡す際などに利用する．

本論文で取り扱うデータソースの変化は，利用者入力もしくは Web サービス問合せの値変化に相当する．Web サービス問合せについては，定期的問合せを指定することが可能であり，ユーザは update 属性を用いて問合せ間隔を指定する．今回は roomSearch(8 行目)に対して指定している．

3.2 表示部

マッシュアップデータの表示には，フレームワークが提供する GUI コンポーネント(以下 **widget**)を使用する．図 2 のテーブル，地図などが widget である．widget は，設定にしたがってデータの表示や入力の受付を行う．

widget の表示対象指定は，Mashup Tree 中の部分木単位に行う．設定として，表示対象となる部分木の選び方と，部分木中のどの node を表示するかを指定する．ホテル一覧テーブル(16－19 行目)の場合，表示対象は Mashup Tree 中の hotel 全てである．このような場合，target 属性に TREE を指定することで，Mashup Tree 全体を表示対象とし，foreach 属性に hotel という相対 Path を指定することで，top node 下の hotel node class に属する node をテーブルの各行として扱うことができる．また，列指定(17－18 行目)では，各行に相当する（hotel）node 下の name，address node を列として表示することを指示している．

一方，空室一覧テーブル(21－22 行目)の場合，ユーザの選択動作に応じて，表示対象とする部分木は動的に変化する．このような場合は，他の widget の属性情報を用いた部分木指定を行う．具体的には，target 属性に WIDGET::hotelTable.selected を指定することで，ホテル一覧テーブル(id 属性が hotelTable)の選択行に相当する node に追従し，その node を頂点とする部分木を表示対象とする．選択行が変化すれば，連動して表示対象部分木も変化する．また，地図(24－27 行目)の場合には，各 marker 要素で複数の変化する表示対象を指定している．例えば 1 つ目の marker 要素の target 属性(WIDGET::hotelTable.onView)は，現在ホテル一覧テーブルに表示されているホテル群をマーカー表示するための記述である．

3.3 要求駆動によるデータ構築

ここでは，要求駆動によるデータ構築について概要を説明する．フレームワークは実装上もデータモデル部と表示部に分離しており，上記設定記述に基づいて，インクリメンタルにマッシュアップデータを構築する．widget は利用者の操作に応じて，自身の(再)描画を行うが，その際描画ルーチンにおいて，表示に必要になったデータをデータモデル部に対して要求する．データモデル部は要求に応じ，top からパスを辿って目的の node を取得し，その値を widget へと返す．パス中の node や目的の node の実体が存在しない場合には，node class に設定された依存関係に従ってソースとなる node を取得し，その値から新たに node を作成する．ソースとなる node も存在しない場合には，再帰的に手順を繰り返し，必要な node を順次構築していく．その際，Web サービス問合せが必要な場合には，一旦構築を中断して，結果到着後に再開するような機構を持っている．この仕組みは，今回提案する部分更新手法に統合されているため，詳細は 4 章において説明する．

4. 部分更新手法

ある node に値変化があった際，提案手法では，その node の**影響領域**のうち，**表示領域**に必要なデータ再構築のみを行う．ある node の影響領域とは，その node の値変化の影響を受ける node 群のことであり，本システムでは，値が変化した node 自身とその子孫 node，値変化した node をソースとして作成された node 群，および上記 node 群の影響領域の和として定める．例えば，前述のホテル検索の Web アプリ(図 4)では，date node が変更された場合，roomSearch や room, photoSearch などが影響領域に含まれる．また，ホテル全体の最安値を各 lowest から計算する operation を top 下に追加したような場合には，値の変化したリスト中の roomSearch node より上の階層に影響が及ぶこともある．

一方，表示領域とは，widget が自身の描画を行うためにアクセスする node 群のことである．widget は，与えられた設定に従って表示を行うが，利用者の操作により表示領域は動的に変化し，その領域は不規則な形状になりうる．例えば，テーブルに対してスクロール操作やフィルタ条件の設定を行った場合，テーブル widget の表示領域に含まれる list element の範囲は図 6 のようになる．また，別の widget 上での行選択などに応じて，表示対象の部分木を切り替える widget も存在する．このように，ユーザの興味が閲覧過程において移り変わっていくため，入力の変化に応じて表

示領域に必要なデータについてのみ再構築を行う．これにより，すでに閲覧範囲から外れた部分に，無駄な計算・通信コストを払わないようにしている．

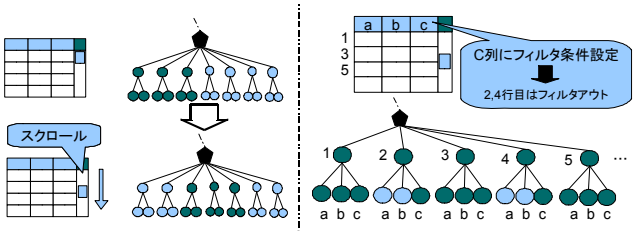

図 6 表示領域の変化
Fig.6 Transition of Display Targets

4.1 実装方針

我々のフレームワークにおける部分更新機能の実現は，以下のように行われている．まず，データモデル部が影響領域に含まれる node の削除を行い，その後，widget による再描画を行う．こうすることで，widget は現在の表示領域に基づいた node アクセスを行い，結果，現在必要な部分に関してのみデータ再構築が行われることになる．但し，表示領域全てを再描画するのでは無駄が多いので，表示領域と影響領域の重複部分に限定した再描画を行う．4.2 節で述べるように，影響領域をシンボリックな形で扱うことで，影響領域の削除ならびに表示領域との重複チェックは効率的に行うことができる．

Web サービス問合せ結果を widget 描画に反映させる際も，上記の部分更新機能を用いて実現する．データモデル部が widget からのデータ要求を受けた際，途中 Web サービス問合せが必要な場合は，widget には当該 node を構築中である旨を示す値を即座に返し，並行して非同期問合せを開始する．そして，結果取得後に，結果を保持する node について，部分更新時と同様にその影響領域を求め，widget に再描画を依頼する．

このような widget からの pull ベースの実装法には利点がある．一つには，widget の種類によって異なる表示領域の管理を widget 側に集約でき，widget への再描画依頼 API も統一することができる．また，非同期問合せを行う間に起こった widget の状態変化などについて，気にする必要がない．

加えて，表示領域と影響領域の重複チェックが簡単にできることで，別のメリットも存在する．まず，widget は更新領域を把握することで，例えば，その部分の点滅表示させユーザに知らせることができる．あるいは，リスト状データ全てが新しくなった場合にスクロールバーを初期位置に戻すといったことも可能であり，実際に行っている．

重複チェックは，最新の表示領域では不必要となってしまった通信の削減にも利用されている．本フレームワークでは，Web 問合せの際に，同じ Web サービスに短時間に何度もアクセスすることを防ぐために，通信を一時待機する．この時，問合せを再開する際は，その結果が来た場合の影響領域と widget の表示領域の重複を確認する．重複がない場合，つまり現在の表示に必要がなくなっていれば，通信をキャンセルする．また，定期的 Web サービス問合せを設定している場合についても，現在の表示に不必要であれば，通信をキャンセルする．

4.2 影響領域の表現と導出

影響領域は，node 実体の集合として扱うのではなく，node class を用いてシンボリックに表現する．具体的には，図 7a のように，top node から更新対象 $node_q$ に至るパス上の $node_{0,\cdots,q-1}$ (list connector は含まない)に対して，$node_i$ を頂点とする部分木中に存在し，かつ node class の集合 $cset_i = \{c_{i,0}, \cdots, c_{i,k}\}$ のいずれかに属する node という形で，影響を受ける node 群を表すことができる．例えば，図 4 のアプリでは，ある 1 つの roomSearch node($node_{q=2}$)の値が到着あるいは変化したときの影響領域は，親である hotel node ($node_1$) を頂点とする部分木中の，node class 群 $cset_1$ (room, name, charge, lowest) のいずれかに属する node として表現できる．この場合，$cset_0$ には node class 群は含まれず，hotel リスト中の他の要素には影響を与えない．これらの node class は，設定記述の依存関係から静的に求めることができる．このため，widget が表示対象とする部分木を下降しながら，部分木中の影響領域を効率的に特定することが可能である．

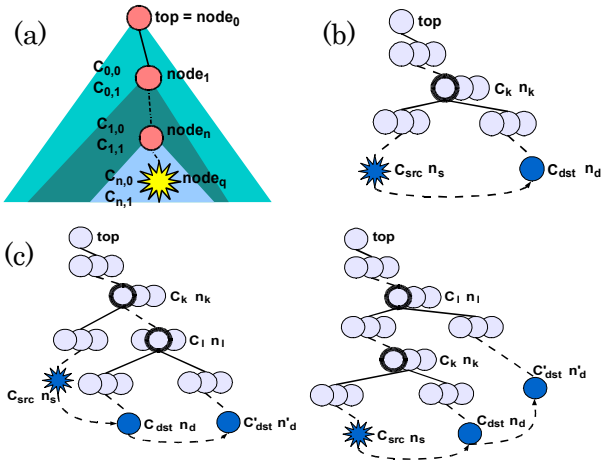

図 7 影響領域
Fig.7 Dependent Area

ある node が更新された際，その子孫 node は当然影響領域として更新されるため，親子関係にないソースが指定されていた場合の影響領域について考える(図 7b)．node class C_{dst} において，ソース node class として C_{src} が指定されていた場合，C_{src} に属する node n_s の値が変化すれば，node n_k を頂点とする部分木中の，C_{dst} に属する全ての node n_d が影響を受けることになる．

n_d の値が変化すれば，C_{dst} をソースとする C'_{dst} に属する node 群も影響を受けることになる．C_{src}, C_{dst}, C'_{dst} の位置関係には，図 7c 左右の可能性が存在する．左図のように，C_{src}, C_{src}, C_{dst} の LUB (least upper bound) C_k が，C_{dst}, C'_{dst} の LUB C_l より上位に位置する場合，n_k を頂点とする部分木中の全 C'_{dst} node が影響範囲となる．一方で，C_l が C_k の上位に位置する場合，node n_l を頂点とする部分木中の全 C'_{dst} node が影響範囲となる．上記手続きを，node class 間の各依存関係に対して順次適用することで，ある node class に属する node に更新があった場合の影響領域を冒頭に述べたような形で求めることが可能である．

この影響領域を用いて，各 widget の再描画の際に重複判

定を行う．我々のフレームワークでは widget は木構造データを下降しながらデータを取得するが，その際部分木中に影響領域が存在するかを判定することで，不要な再描画を省略している．例えばテーブル widget ならば，行単位，セル単位で判定を行いながら必要な部分のみ再描画を行う．

5. 評価

本節では，部分更新手法の基本性能及び実アプリ上での評価を行い，その後，他の実装手法との比較検討を行う．以下の結果はいずれも，Firefox 3.0.1 を対象に Firebug 1.2.1 を使用し，Intel Core2 Duo T7250 2GHz，メモリ 3GB，Windows Vista Business (32bit)上で測定した．

まず，部分更新処理の基本性能を見るため，シンプルな widget において更新時間を測定した．widget は，5行×4列のテーブルを用い，入力の変化により，そのうち1列が全て更新される．内部的には表示領域中の 21 個の node について重複判定を行っている．また，影響領域による変化も確認する目的で，影響領域に含まれる node class 数を意図的に変化させた．今回，node class 数を 20,30,40,50 としてそれぞれ測定を行ったが，これは 2 章における 2 つめのアプリ例(図2)において，影響領域に含まれる node class 数が最大で 23 個であったことから設定した値である．また，データの構築の際には Web サービス問合せは発生しないものとし，XML 文書からの切り出しのみ行っている．表1が測定結果である．更新時間は，いずれも 140 から 150 ミリ秒程度であり，これは秒単位で値変化が起きたとしても十分追従可能な値であるといえる．また，より詳しく見ていくために，本手法の各処理時間を計測したが，重複の判定には 1 個の node あたり平均して 0.20 ミリ秒程度で行われていた．これらは，描画処理コストに比べて非常に小さく，widget に様々な機能を実装する際，オーバーヘッドをあまり気にせずに重複判定を利用することができると言える．また，影響領域中の node class 数の変化によるコストの増加は，全体として大きな影響を及ぼさなかった．

表 1 更新時間
Table 1 Time for Update Processing

node class 数	更新時間 [i] (ms)	内部処理相当 [ii] (ms)	判定時間 [iii] (ms)
20	143.10	15.52	4.39
30	139.72	15.87	4.06
40	149.58	18.00	5.15
50	147.60	19.68	5.21

i 値の変化から描画完了まで
ii 更新時間と描画処理との差，データの削除・構築・重複判定時間に相当
iii 更新中の重複判定処理のみに要する時間

次に，実アプリ上での挙動を見るために，2 章における図2と同等のアプリを作成し，入力変化時の反応時間を測定した．利用者入力である日付を変更した場合，空室一覧，平均気温，地図上の写真などが更新対象となる．空室一覧については，Web サービス問合せ結果到着後，約 0.17 秒程度の時間で表示に反映された(描画時間を含む)．気温情報も空室一覧と同様に Web サービス問合せが発生し，問合せ結果到着後，約 0.03 秒で更新が行われた(気温データを一箇所反映させるだけであるため)．地図上の写真については，表示までの一連の流れは以下の通りである．まず，各観光地毎に順次 Web サービス問合せを行い，画像の場所を示す URL を取得する．今回の地図 widget は Google Maps をラップする形で実装されており，URL 到着時に再描画を促された widget は，データモデル部から URL を取得し，Google Maps API を通じて画像の URL を設定する．個々の URL 到着から URL 設定までに要した時間は平均 0.07 秒であった．その後，ブラウザは画像データの取得・表示を行うことになる．

参考までにそれぞれ Web サービス問合せに要した時間は，平均気温，写真(URL 取得)が 0.3 秒程度，空室一覧については 0.5 から 2 秒程度であった[5]．更新処理は，次々と到着する Web サービスの結果取得に応じて行われるが，いずれも利用者の操作や Web サービス問合せに要した時間に比べて小さく，閲覧の妨げにならない程度であった．

他の実装手法の比較検討: ここでは部分更新手法を用いずに，同様のアプリを実現した場合の検討を行う．以下では本手法以外に3つの実現方法を考え，それぞれを用いた場合の計算範囲と，不向きなケースについて議論する．

最初の対案は，取得しているデータのうち，変化の影響を受ける部分を全て再計算する方法である．その場合，例えば 100 件目までのデータを取得しているが，5 件分のみ表示している場合には，20 倍のデータ再構築のコストが必要になる．表 1 の node class 数 20 の場合，提案手法ではデータ構築に 11 ミリ秒程度要しており，この方法ではデータ構築に 220 ミリ程度を要すると考えられる．また，データ構築の際に Web サービス問合せも必要な場合，応答性能はさらに大きく低下してしまう．

影響するデータを計算しなおすのではなく，全データを削除した上で各 widget を再描画する方法も考えられる．この場合には影響領域に関係なく，現状表示に必要な部分についてのみ計算が行われる．Web サービス問合せ結果をキャッシュしておけば，不要な問合せは行われないが，データ構築に余分なコストを払うことになる．また，この方法であっても，非同期通信の結果到着時には，全再描画を行うのは不適当である．そのため別途非同期通信用の表示処理を行うためのコードが必要になってしまう．

また，データの変化時に，widget のどこを更新するかと言うことを，ハンドラなどを用いて開発者が明示的に指定するという方法も考えられる．しかしこの場合，依存関係を考えながら指定を行わなければならず，マッシュアップ定義を変更した場合に再利用が困難である．

6. 関連研究

既存のマッシュアップツールには，サービス接続関係の簡単な指定に注力したものが多い．例えば Yahoo! Pipes[6] や Popfly[7]，Afrous[8]では，GUI 環境の上で上記接続関係をデータフローとして表現することができ，出来上がったものを Web サービス(Pipes)や Web アプリ(Popfly,Afrous)として公開可能である．これらのツールは，指定されたサービス接続関係に従って，全てのデータを構築し結果を得るものであり，利用者の選択に応じて部分的にマッシュアップを進めていくという使い方はできない．入力変化に応じたデータ更新に関しては，Popfly が指定データソースに対して定期的に再取得を行う機能を持っているが，影響を受ける部分全ての再計算を行ってしまう．

また，利用者がインクリメンタルにマッシュアップを進

[5] 画像取得に要する時間は，提供サイト毎に異なるのでここでは省略する．

めるためのツールとして Marmite[9] や Mash Maker[10,11,12] がある．これらは，ブラウザのアドオンとして提供され，利用者自身がその場でマッシュアップ項目を追加していくことで，我々のフレームワーク同様，利用者の興味に応じたデータ構築を行うことが可能である．作成したマッシュアップを他の利用者と共有することもできる．Mash Maker は，関数型に基づいたマッシュアップ定義や，データを木構造として表現するなど，我々のフレームワークと共通点が多い．しかし，これらのツールは，利用者の操作に従ってボトムアップにマッシュアップを進めるものであり，第三者が構築したマッシュアップに対して，部分的に閲覧するといった使い方はされない．このため，入力変化に応じたデータ更新についても，利用者の現在の閲覧範囲を考慮せず，変化の影響を受けるデータが全て更新されることになる．

マッシュアップアプリの開発環境である Google Mashup Editor[13]では，HTML，CSS，JavaScript などを記述してアプリケーションを構築する．マッシュアップに必要な各種処理や Ajax で動作する GUI コンポーネントが，独自の HTML タグや JavaScript ライブラリの形で提供されており，開発者はそれらを利用することができる．しかし，本研究で対象としたような部分更新を実現するには，必要なデータに関しての再計算・再描画などの処理を，開発者自身が明示的に記述する必要がある．

リレーショナルデータベースの materialized view では，各テーブルの変更差分に対して，インクリメンタルに view の更新を行うための技術[14,15]が知られている．しかし，我々のフレームワークが対象としている Web サービス群は，独立して運営されていることが多く，同様の技術を用いることは難しい．また，我々は利用者に対し，選択的な閲覧形態を提供しているが，表示可能なデータのうち実際に表示される範囲は部分的であり，加えて操作に応じて動的に変化する．このような状況において，不要な計算を省いている点が，materialized view とは異なる．

7. まとめ

本論文では，要求駆動型マッシュアップフレームワークにおいて，データソースが変化した際に，その影響領域を効率的に更新する手法を提案した．我々のフレームワークでは，Web サービスの組み合わせ方とデータの表示方法を指定すれば，利用者は興味のある部分を選択的に閲覧することが可能な Web アプリを作成できる．今回提案した手法では，入力の変更や Web サービス問合せ結果の変化に際して，ユーザのデータ閲覧に必要な領域に関してのみ，データ再構築を行う．これにより，ユーザの入力操作や入力値の変化に柔軟に追従する Web アプリケーションが実現可能となった．

[謝辞]

本研究は，科学研究費補助金若手研究 (B)18700029 の補助を受けたものである．

[文献]

[1] 池田宗平, 草野直樹, 長嶺貴一, 鎌田十三郎: “要求駆動によるマッシュアップと AJAX Widget による効率的な閲覧”, コンピュータソフトウェア (2008 年 4 月採録決定, 掲載日未定).

[2] Sohei IKEDA, Takakazu NAGAMINE, and Tomio KAMADA: “Application framework with demand-driven mashup for selective browsing”, In Proc. of iiWAS 2008, pages 33-40 (2008).

[3] 楽天株式会社. 楽天トラベル系 API, http://webservice.rakuten.co.jp/

[4] HeartRails. HeartRails Express 駅情報取得 API, http://express.heartrails.com/

[5] Google. Google Maps API, http://code.google.com/apis/maps/index.html

[6] Yahoo! Yahoo Pipes, http://pipes.yahoo.com/pipes/

[7] Microsoft. Popfly, http://www.popfly.com/

[8] Afrous.dev, http://afrous.com/

[9] Jeffrey Wong and Jason Hong: “Marmite: enduser programming for the web”, In CHI ’06 extended abstracts on Human factors in computing systems, pages 1541-1546 (2006).

[10] Intel Mash Maker, http://mashmaker.intel.com/

[11] Robert J. Ennals and Minos N. Garofalakis: “Mashmaker: mashups for the masses”, In Proc. of SIGMOD’07, pages 1116-1118 (2007).

[12] Rob Ennals and David Gay: “User-friendly functional programming for web mashups”, In Proc. of ICFP ’07, pages 223-234 (2007).

[13] Google. Google Mashup Editor, http://code.google.com/gme/

[14] Ashish Gupta and Inderpal Singh Mumick: “Maintenance of materialized views: Problems,techniques and applications”, IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing, 18(2):3-18 (1995).

[15] Jingren Zhou, Per-Ake Larson and Hicham G. Elmongui: “Lazy maintenance of materialized views”, In Proc. of VLDB'07, pages 231-242 (2007)

長嶺 貴一 Takakazu NAGAMINE

2008 年，神戸大学工学部情報知能工学科卒業．同年から同大学大学院工学研究科情報知能学専攻博士前期課程に在学．Web アプリケーション開発環境，マッシュアップ開発環境に興味を持つ．日本データベース学会，電子情報通信学会各学生会員．

池田 宗平 Sohei IKEDA

2007 年，神戸大学工学部情報知能工学科卒業．同年から同大学大学院工学研究科情報知能学専攻博士前期課程に在学．マッシュアップアプリケーション開発及び，その為の開発環境に興味を持つ．日本データベース学会，日本ソフトウェア科学会各学生会員．

鎌田 十三郎 Tomio KAMADA

1970 年生．1995 年，東京大学大学院理学系研究科情報科学専攻修士課程修了．1998 年より神戸大学工学部助手．2007 年より同助教．並列・分散処理，プログラミング言語処理系などに興味を持つ．ACM，日本ソフトウェア科学会，情報処理学会各会員．博士(工学)．

草野 直樹 Naoki KUSANO

2007 年，神戸大学工学部情報知能工学科卒業．同年から同大学大学院工学研究科情報知能学専攻博士前期課程に在学．プログラミング言語およびその型検証に興味を持つ．