社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ライフログのためのマッシュアップ API の DB 実装と Web サービス化

下條　彰†　まつ本真佑†　中村　匡秀†

† 神戸大学　〒 657–8531 神戸市灘区六甲台町 1–1
E-mail: †shimojo@ws.cs.kobe-u.ac.jp, ††{shinsuke,masa-n}@cs.kobe-u.ac.jp

あらまし　我々は先行研究において，異なるライフログをマッシュアップするための標準データモデルと (LLCDM)，標準化されたデータにアクセスするためのマッシュアップ API (LLAPI) を提案・実装している．しかしながら，これまでの実装においては，LLCDM はローカルファイルシステムに保存された XML ファイル群であり，LLAPI はファイルにアクセスする perl ライブラリであった．そのため，実行性能と環境依存の問題があった．本稿では，LLCDM を関係データベース (MySQL) で管理し，LLAPI をデータベースにアクセスする Web サービスとして実装しなおし，問題の解決を図る．性能評価の結果，約 23〜275 倍の応答時間の改善が見られた．

キーワード　ライフログ, マッシュアップ, 標準データモデル, Web サービス, データベース

Implementing Database and Web Services for Life-Log Mashup APIs

Akira SHIMOJO†, Shinsuke MATSUMOTO†, and Masahide NAKAMURA†

† Kobe University　Rokkoudai-cho 1–1, Nada-ku, Kobe, Hyogo, 657–8531 Japan
E-mail: †shimojo@ws.cs.kobe-u.ac.jp, ††{shinsuke,masa-n}@cs.kobe-u.ac.jp

Abstract　We have previously proposed the LifeLog Common Data Model (LLCDM) to aggregate heterogeneous lifelog data, and the LifeLog mashup API (LLAPI) to access the standardized data. The LLAPI was implemented as a set of Perl libraries that access the local file system storing XML data of the LLCDM. Therefore, it had a performance bottleneck, and was poor in the portability. To cope with these problems, we re-engineer the LLCDM and the LLAPI with the relational database MySQL and the Web services, respectively. The experimental evaluation shows that the performance of the new implementation achieves 23 to 275 faster than the previous one.

Key words　Lifelog, Mash up, Common data model,Web service, Database

1. はじめに

近年，人間の行動をデジタルデータとして記録に残す**ライフログ**が注目を集めている．Web 上にはいくつものサービスが存在し，様々な種類のライフログを Web 上で記録・共有できるようになっている．現在こうしたサービスの多くは，ライフログの「記録」と「利活用」はサービス内に閉じた形で行われている．しかし，様々なサービスでばらばらに記録されたライフログを，集約・連携 (**マッシュアップ**と呼ぶ) する事で，より付加価値の高い情報・サービスへの発展が期待できる．

既存のライフログ・サービスの中には，マッシュアップ用の API やブログパーツを用意しているものも存在し，外部プログラムからログデータにアクセスすることができる．しかしながら，これら API の仕様はサービス毎に異なっており，連携するサービスの組み合わせ毎に異なるプログラムロジックが必要となる．マッシュアップの効率化，生産性を向上するには，ある程度標準的なデータモデルおよびサービス API が望まれる．

我々は先行研究 [1] [2] において，異なるライフログを横断的に扱うための標準データモデル (**Life Log Common Data Model - LLCDM**) およびそれにアクセスするための API(**Life Log API - LLAPI**) を提案している．LLCDM は，5W1H の観点から様々なライフログを分析し，特定のサービスに依存しないデータ項目を抽出，依存データは意味を解釈せずに埋め込むというアプローチをとっている．具体的なデータ項目は，日付，時刻，場所，ユーザ，コンテンツ，アプリケーション，デバイス等となっている．プロトタイプとして，LLCDM を正規化した XML ファイル群を格納するファイルシステムとして実装した．また，LLAPI を XML ファイルにアクセスする Perl ライブラリとして実装した [2].

このプロトタイプを用いて，我々はライフログサービスをマッシュアップする実際のアプリケーションを開発する実験を行った [3]．その結果，LLCDM と LLAPI を利用してマッシュアップを行うと，利用しない場合に比べて，その開発効率は格段に向上した．しかしながら，ファイルシステムへのアクセス

がボトルネックとなり，実行速度に関して大きな改善が必要であることがわかった．さらにマッシュアップ API を Perl ライブラリとして開発したため，実行プラットフォームへの依存性も問題となった．

これらの問題を解決すべく，本稿では LLCDM および LLAPI の再実装を行い性能の評価を行う．具体的には，まず LLCDM 形式に標準化されたライフログデータを全てデータベースで管理し，検索やアクセス速度の向上をねらう．性能と信頼性を確保するため，関係データベース MySQL を利用し，従来 XML ファイルで保存していたデータを収納するためのテーブル設計を行う．また，LLAPI をこのデータベースにアクセスするラッパープログラムとして再実装する．汎用性を向上するため，LLAPI を SOAP および REST 形式の Web サービスとして公開する．LLAPI の Web サービス化により，様々な言語やプラットフォームから LLCDM へのアクセスが可能となる．

有効性を検証するため評価実験を行った．LLAPI の実行速度は，SOAP アクセスで約 20 倍，REST アクセスで約 200 倍の性能向上が確認できた．これにより，今まで無視できなかったオーバーヘッドを大幅に下げ，さらに開発者の様々な用途，要求に対応することが確認できた．

2. 準　　備

2.1 ライフログのマッシュアップ

最近，ネットワーク上に存在する様々なライフログを集約・連携することがさかんに行われている．複数のライフログの集約・連携による付加価値創造を，本稿では**ライフログのマッシュアップ**と呼ぶ．代表的なものに，“mixi” [4] や “Facebook” [5] のように，ブログやミニブログ，写真や動画，さらにはスケジュール等を 1 つに集約したサービスがある．他のアイデアとして，例えば，“Twitter” [6] と “ねむログ” [7] を連携することで，日々の精神状態と睡眠時間の相関が取れるかも知れない．

ライフログ・サービスの中には，外部サービスから自身のライフログへのアクセス手段を，サービス API や Web 部品 (ブログパーツ等) という形で公開しているものも存在し，マッシュアップに利用することができる．しかし，こうした API や API によって得られるライフログのデータ構造には統一的な標準が無く，サービスごとにまちまちである．つまり，**ライフログへのアクセス手段がサービスに強く依存**している．したがって，マッシュアップの開発においては，連携するライフログの組み合わせごとに，異なるプログラムロジックが必要となる．

2.2 先行研究：標準データモデル (LLCDM) とマッシュアップ API(LLAPI)

ライフログの高度かつ柔軟なマッシュアップを支援するため，我々は先行研究 [1] [2] において，**ライフログのための標準データモデル (Life Log Common Data Model - LLCDM)** と**マッシュアップ API(Life Log API - LLAPI)** を提案している．図 1 に示すとおり，各ライフログ・サービスのデータは，あらかじめ特定のサービスに強く依存しない標準的なデータモデル (LLCDM) に変換され格納される．この LLCDM の上では汎用的な API が定義され，様々なライフログへの標準的かつ横断的なアクセスが可能となる．

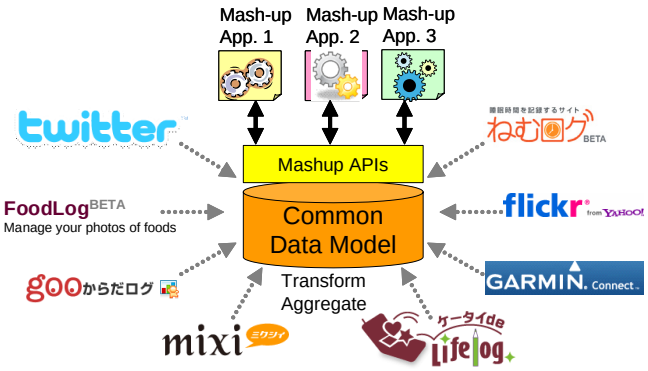

図 1　Life Log Common Data Model-LLCDM

表 1　LLCDM のスキーマ

観点	データ項目	説明
WHEN	<date>	日付
	<time>	時刻
WHO	<user>	ユーザ
	<party>	共に行動したユーザ
	<object>	対象ユーザ
WHERE	<location>	場所
HOW	<application>	ライフログサービスを実現するアプリケーション
	<device>	ログ記録に用いたデバイス
WHAT	<content>	ログの内容
	<ref_schema>	外部スキーマへの参照
WHY	-------	必要なら<content>内へ

表 1 に LLCDM のデータスキーマを示す．LLCDM は 5W1H の観点からライフログが備えるべきデータ項目を整理し，特定のサービス・アプリケーションに依存しないデータストアを定義する．LLCDM の特徴は，日付や時刻，ユーザといったおおよそどのライフログにも付与される情報を標準化し，アプリケーションに依存するログの内容に関しては自ら解釈せず，外部のスキーマに任せるという構造をとっている．

また，LLAPI は LLCDM 形式に変換されたライフログデータにアクセスするための標準的なインターフェースを定義する．これは LLCDM のデータ項目に関するクエリを入力とし，合致するライフログデータを LLCDM 形式の配列で返す．

```
getLifeLog(s_date, e_date,  s_time, e_time,  user,
                location, application, device, select)
```

■引数

- s_date:　検索を開始する日付に関するクエリ
- e_date:　検索を終了する日付に関するクエリ
- s_time:　検索を開始する時刻に関するクエリ
- e_time:　検索を終了する時刻に関するクエリ
- user:　ユーザに関するクエリ
- location:　場所に関するクエリ
- application:　サービスに関するクエリ
- device:　デバイスに関するクエリ
- select:　抽出データに関するクエリ

2.3 現状の課題

上記の先行研究において我々は，LLCDM のリポジトリおよび LLAPI のプロトタイプを開発した．このプロトタイプは，LLCDM 形式に変換したライフログデータを XML ファイルとしてファイルシステムに保存する．LLAPI はファイルシステムにアクセスしてデータ検索を行う Perl ライブラリとして実装した．マッシュアップアプリケーションの開発実験 [2] [3] においては，提案手法による開発効率の大幅な向上が証明された．その一方で，いくつかの実用上の課題が浮かび上がった．

1 つ目の課題は，マッシュアップ API の実行が非常に遅く，ライフログデータをオンラインでマッシュアップするには無視できないオーバーヘッドとなることがわかった．これは，ライフログが 1 件ごとに XML ファイルとして保存されているため，ファイルシステムへのアクセスが頻繁となり，これがボトルネックとなったと考えられる．今後データが増加するにあたり，さらにオーバーヘッドが大きくなるのは自明である．

2 つ目として挙げられるのは，Perl ライブラリとして提供された LLAPI のプラットフォーム依存性である．ライブラリを配布する提供方式は利用者の開発言語や実行環境に依存し，またバージョン管理の難しさというデメリットが生じる．

これら 2 つの課題を解決するのが本研究の目的である．

3. 提 案 手 法

3.1 研究の目的とスコープ

前節で挙げた 2 つの課題を解決すべく，本研究では以下の二つを達成することを目的とする．

目的 G1： LLAPI の性能改善，および

目的 G2： LLAPI のポータビリティの向上

まず目的 G1 を達成するために，従来ファイルシステムで管理していたライフログデータをデータベースで管理し，検索やアクセスの高速化を図る．LLAPI はこのデータベースにアクセスするラッパープログラムとして再構築する．本研究では，信頼性と性能の両方を確保するため，関係データベース (RDB) を用いることにする．

また，LLAPI を Web サービス [8] として公開することにより，目的 G2 の達成を図る．Web サービスは，オブジェクトを XML にシリアライズして，標準的なプロトコル (REST, SOAP/HTTP) でサーバプログラムの呼び出し，データ交換を行う仕組みである．様々な言語やプラットフォーム用にミドルウェアが整備されており，LLAPI のポータビリティ向上に貢献できる．より具体的には，以下の項目を順に実施する．

(1) LLCDM のデータベース設計

(2) ライフログデータの蓄積方式の検討

(3) LLAPI の実装

(4) Web サービスによる LLAPI の公開

3.2 LLCDM のデータベース設計

LLCDM 形式のライフログデータを，関係データベースで管理するためのデータベース設計を行う．具体的には，表 1 の LLCDM の各項目を漏れなく格納するためのテーブル設計を行う．図 2 に文献 [9] の表記法に従った ER 図を示す．四角はテーブル (エンティティ) を表しており，テーブル名，CRUD，主キー，属性のスキーマが並ぶ．各テーブルの下部にはインスタンスを併記している．エンティティ間の関連として，(＋—∈) は親子関係を，(＋—…) は参照関係を表す．紙面の都合上，部分的に正規化を崩したモデルを掲載している．今回，データベースには以下の 4 つのテーブルを設けた．

(a) ユーザ情報：UserInfo

データを取得するユーザの情報を管理する．このテーブルを設けたのは，ユーザ情報は，様々なライフログで共通して参照されるものであり，マッシュアップの際に頻繁に使われる情報の一つであると考えたためである．属性には，UserID，ユーザ名，パスワード，メールアドレスがある．UserID は同一の人物でもアプリケーションごとに異なる場合，別に登録する必要がある．

(b) アプリケーション：Application

ライフログデータの取得元のアプリケーションに関する情報を管理する．このテーブルは，アプリケーション固有の設定やリファレンス URL などを管理するために設けた．アプリケーション情報は多くのライフログデータで共通に参照されるため，一つのテーブルとして独立させたほうが効率が良いと考えたためである．属性としてはアプリケーション名，会社名，URL，アプリケーションに関する説明，リファレンス URL からなる．

(c) ライフログ：LifeLog

取得したライフログデータを保管するメインのテーブルである．属性は標準データの各項目を保管するフィールドからなる．ただし，`<Application>`および`<Location>`のデータは，それぞれ別テーブルに保存されたデータへ紐づけるための ID を外部キーとして保管する．主キーは，UserID，日付，通し番号 (seq) を組とする複合キーとしている．これはライフログデータをマッシュアップする際，UserID と日付をクエリとしてデータの検索・取得をすることが非常に多いと考えられるからである．また，データの重複防止や管理を容易にするためでもある．

(d) 場所：Location

各ライフログの取得場所`<Location>`に対応するデータを保管する．複数のログが同一場所で取られる可能性があることと，場所に関するマッシュアップ・クエリを想定し，独立したテーブルとして定義した．このテーブルの各組には一意に割り当てられた LocationID によって，対応する LifeLog テーブルの組と紐付けられている．`<Location>`データはライフログデータを生成した場所を示すデータ項目であり，緯度情報，経度情報，もしくは住所が保管されている．

3.3 ライフログデータの蓄積方法の検討

次に，LLCDM のデータベースにライフログデータを蓄積する方法をついて述べる．図 3 に本研究で構築した蓄積方法を示す．ライフログサービスからデータの変換まで，以下の 3 ステップを踏む．

(Step1) 生データの取得

まず各ライフログサービスからオリジナル生データ (図中 Raw Data XML) を取得し，XML で保存する．この作業を定

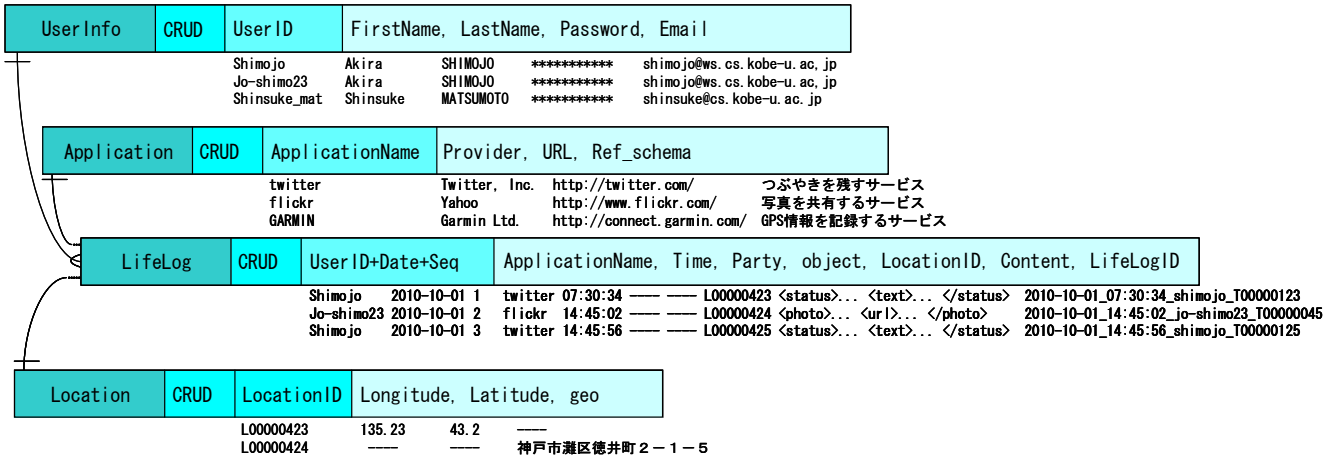

図 2　LLCDM のためのデータベース設計

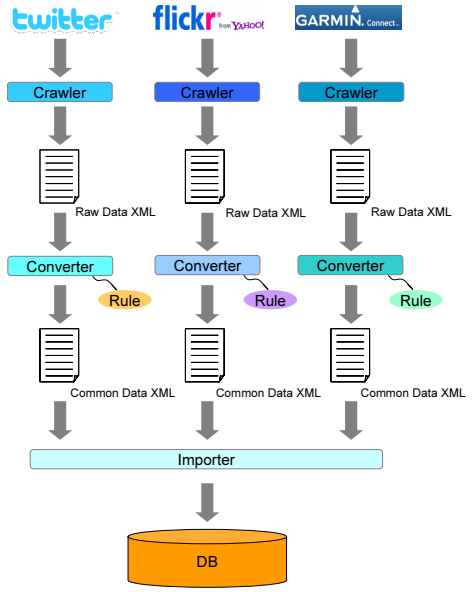

図 3　ライフログデータの蓄積方法

期的に行うのが，Crawler と呼ぶプログラムであり，各ライフログサービスが公開している固有の API を実行してデータを取得する．Crawler はライフログ毎に固有のプログラムである．

(Step2) LLCDM への変換

次に，Crawler が取得した生データを，LLCDM 形式のデータ (図中 Common Data XML) に変換する．この作業は，Converter プログラムが行う．ここでは各ライフログサービスごとに定義された LLCDM への変換ルール (図中 Rule) を用いて，生データを LLCDM の各項目にマッピングしている．変換方法については先行研究 [2] を参考にされたい．

(Step3) データベースへの挿入

最後に Common Data XML をデータベースに挿入する．このときに使用するプログラムが Importer である．Importer は LLCDM の各項目を対応するデータベースのテーブルに代入する．LLCDM に変換されたデータは統一された形式となっているため，Importer は Crawler や Converter のようにアプリケーションごとに設ける必要はない．

以上の 3 ステップを行って，MySQL データベースへライフログデータの蓄積を行うシステムを perl および cron システムを用いて実装した．3 ステップに区切ったのは，システムの改訂や保守を容易にするためである．例えば，もし TwitterAPI の仕様が変更したとしても，該当する Twitter の Crawler のみ変更するだけで良い．また，データベースに挿入するまでに二つの XML ファイルを生成している．これはもしデータベースサーバが落ちたときに対応するためのバックアップであり，また再利用を想定してのものである．これらの XML ファイルは一つのリポジトリに保管されている．

3.4　LLAPI の実装

全てのライフログデータが LLCDM に従ってデータベースに蓄積されているならば，それにアクセスするための LLAPI は，単なるデータベースの問い合わせプログラムとして実装できる．2.2 に示した LLAPI の getLifelog メソッドは，内部でデータベースに対して以下の SQL 文を発行し，検索結果をオブジェクトにまとめて配列で返せばよい．

```
SELECT * FROM LifeLog WHERE
  Date BETWEEN $d_dateq AND $e_dateq
  AND Time BETWEEN $s_timeq AND $e_timeq
  AND UserID LIKE $userq AND Application LIKE
  $applicationq AND Device LIKE $deviceq
```

LLAPI の実装には Java 言語を用い，O/R マッパーである iBATIS を用いてオブジェクト／タプルのマッピングを行った．iBATIS を用いることで，SQL 文を Java ソースコードから分離でき，変更・再利用に強いプログラムが実現できる．

3.5　Web サービスによる LLAPI の公開

最後に，LLAPI を Web サービスとして公開する．Web サービス化することの最大のメリットは，LLAPI の利用者の言語環境・実行環境に非依存になるという点である．Web サービスへのアクセス方法さえ知っていれば，多くのプログラミング言語から利用することができる．もう一つのメリットとして，API の改訂・保守を行っても，再配布する必要が無いことである．これにより，利用者は安定したサービスを享受できる．

我々は Java で実装した LLAPI を，Apache Axis2 [10] Web

```
<ns:getLifelogResponse
  xmlns:ns="http://mashupAPI.hns.cs27.kobe_u.jp"
  xmlns:ax276="http://bean.hns.cs27.kobe_u.jp/xsd"
  xmlns:ax273="http://sql.java/xsd"
  xmlns:ax278="http://mashupAPI.hns.cs27.kobe_u.jp/xsd">
   <ns:return type="jp.kobe_u.cs27.hns.mashupAPI.LifeLog">
      <ax278:application>Twitter</ax278:application>
      <ax278:content>
         <?xml version="1.0" encoding="utf-8" ?>
         <id>27597746132</id>
         <text>AP試験撃沈なう(T_T)</text>
         <user>
            <name>Akira Shimojo</name>
            <location>Kobe, Hyogo, Japan</location>
            <screen_name>shimojo</screen_name>
            < time_zone>Osaka</time_zone>
              ......
         </user>
            ......
      </ax278:content>
      <ax278:date>2010-10-17</ax278:date>
      <ax278:device>Web</ax278:device>
      <ax278:lifelogID>
               2010-10-17_02:30:24_meruten_T0000921
      </ax278:lifelogID>
      <ax278:time>02:30:24</ax278:time>
      <ax278:user>shimojo</ax278:user>
     .....
   </ns:return>
</ns:getLifelogResponse>
```

図 4　Web サービス化した LLAPI の実行結果

サービスフレームワークを用いて，デプロイ (配置) した．Axis2 でデプロイされた Web サービスは，SOAP 形式と REST 形式の両方に対応する．SOAP 形式の場合，ミドルウェアを使用するため実行時間が遅くなったり，また初期の学習コストが高くつくという欠点がある．しかし高機能であるため取得したデータを自動的に解析し，オブジェクトに変換するという利点がある．そのため拡張性が高く，大規模なデータの取得・解析などに向いている．一方 REST 形式の場合，HTTP の GET/POST メソッドで XML ベースの情報を受け取れるため，非常に簡単に実行でき，レスポンスが速いという利点を持つ．しかし取得したデータの解析はクライアント側で行わなければならないため，大規模かつ複雑なデータには向いていない．

図 4 に REST 形式で Web サービスを呼び出した場合の実行結果を示す．これは LLAPI に「ユーザ “shimojo” が “2010-10-17” に残したライフログのうち，アプリケーション “twitter” のもの」をクエリとして投げた場合の結果 (一部) である．

4. 評価実験

先行研究で開発したで述べた LLAPI のプロトタイプ (2.3 参照) と，今回開発した新たな LLAPI の性能を比較する．

4.1 評価方法

旧 LLAPI と新 LLAPI に対して，以下のクエリを実行して，ライフログデータを取得する．

Case 1: 2010-10-15 から 2010-10-16 までのデータを全て取り出す

Case 2: 2010-09-01 から 2010-09-30 のデータを取得する

表 2　LLAPI の実行時間の比較

Case	1	2	3	4
SOAP(sec)	0.131	1.006	0.281	0.422
REST(sec)	0.015	0.100	0.019	0.025
OLD(sec)	4.238	4.028	4.254	0.581
SQL(sec)	0.007	0.005	0.003	0.001

表 3　LLAPI で取得したデータ件数とサイズ

Case	1	2	3	4
# of items	36	119	195	449
data size (kB)	118	381	1,450	630

Case 3: 09:00:00〜10:15:00 の間のデータを全て取り出す

Case 4: ユーザ shimojo のデータを全て取り出す

各ケースに対しクエリを 5 回ずつ呼び出し，実行時間を計測，その平均値をとる．また新 LLAPI は，SOAP 呼び出しと REST 呼び出しの 2 通りで実行する．さらに参考までに，SQL でデータベースから直接呼び出す方法も試し，時間計測を行う．

4.2 実験環境

評価実験で使用した計算機は以下の通りである．

データベースサーバ： AMD Athlon(tm) 64 Processor 3500+, 3GB メモリ，Vine Linux, MySQL 5.1.36

Web サーバ： AMD Athlon(tm) 64 FX-51 Processor，2GB メモリ，Vine Linux，Tomcat 5.5, Apache Axis 1.4.1

クライアント： Intel(R) Core(TM)2 Duo CPU 2GB メモリ，Perl SOAP::Lite，XML::Simple モジュール

4.3 実行結果

表 2，表 3 に実行結果を示す．なお，今回評価対象となるライフログデータは 1591 件であり，あらかじめ MySQL データベースに蓄積されている．また旧 LLAPI のために，同じデータを XML ファイルとしてファイルシステムに蓄積している．実験計測を行うクライアントプログラムは，SOAP 版，REST 版，旧 LLAPI 版の 3 種類とも，全て Perl 言語で実装している．

表 2 に各 LLAPI の実行時間を秒で示す．表 2 の 2 行目は，新 LLAPI を SOAP 形式で呼び出した場合 (**SOAP**)，3 行目は REST 形式で呼び出した場合 (**REST**)，4 行目は旧 LLAPI で呼び出した場合 (**OLD**)，最終行は SQL でデータベースから直接呼び出した場合 (**SQL**) である．

表 2 から提案手法は従来型に比べ，約 23 倍〜約 275 倍の実行効率の向上が見られた．これは MySQL のデータ検索の性能によるところが大きい．また，SOAP と REST を比べてみると，どのケースにおいても REST のほうが実行時間が短くなっている．おおよそ 8〜16 倍の差が出ていることがわかった．大きな要因は，SOAP 形式においてミドルウェアが XML ／オブジェクトの変換（マーシャリング）を行うのに要するオーバーヘッドである．しかしそのオーバーヘッドも旧 LLAPI の性能と比較すれば，十分実用に耐えうるものと考えている．OLD を見てみると，ケース 4 においては他のケースに比べ実行時間が非常に短くなっているが，これは実装によるものと考えられる．

以上の結果から，ライフログデータの保管をデータベースへ移行したことにより，格段に実行効率が増したと考えられる．

表 3 に各 LLAPI を実行した際にヒットしたライフログデータの件数と各ケースで取得したデータの大きさを示す．各 LLAPI を実行した結果，どのケースにおいても 5 回とも全て同じ件数がヒットし，データの内容も同一であった．このことからどのマッシュアップ API も正確な検索ができているといえる．今後検索クエリが増加したり，フリーワード検索などに対応するようになる場合，過不足なくデータを取得できるかは大きな課題となるだろう．また，データの大きさはケースによってばらばらであるが，今後より膨大なデータを扱うようになれば，サーバへの負担は大きくなるものと考える．大規模データに対してスケールアウトする仕組み (例えば KVS [11]) を考える必要がある．

5. 関 連 研 究

これまで，Web サービスのマッシュアップに関する研究は数多く行われている．例えば Lizcano ら [12] や Lorenzo ら [13] の研究は，エンドユーザでも Web サービスのマッシュアップができるようにそのフレームワークを提案するものである．また，Maximilien ら [14] の研究は，マッシュアップコンポーネントの共有と再利用を手助けするために IBM のシェアブルコードオンラインサービスを提案している．さらに Abiteboul ら [15] の研究では，マッシュアップのグローバル性を高めるために，mashlet と呼ばれるコンポーネントを提案している．

これらの研究は既存の Web サービスをマッシュするという点では本稿と共通点があるが，いずれもそのデータの形式そのものをを統一したモデルに変換するという点で異なる．また，本稿ではライフログに焦点を当て，それに特化しているため，その点においても上記の研究とは異なる．

データモデルの標準化においても Amato ら [16] の研究があげれられる．この研究は，ネットワーク上に分散した様々なセンサーシステムのデータやアクセス方法を統一的な形式で表現するというものである．この研究は様々なデータを統一するという点において共通点があるが，センサーに特化しているという点で本稿とは異なる．

6. おわりに

本稿では，先行研究で開発したライフログマッシュアップ API(LLAPI) の性能およびポータビリティに関する問題を克服するため，関係データベースおよび Web サービスを用いて再構築を行った．本研究の有効性を確かめるため，旧 LLAPI とこのたび開発した新 LLAPI との比較実験を行った．その結果，検索性能が格段に向上したことがわかった．さらに LLAPI を Web サービス化することにより，利用者の環境に依存せず，保守が容易なライフログサービスを提供できることを確認した．

今後の課題として，まずは対象となるライフログサービスを増やし，マッシュアップ API のバリエーションを広げることである．そのためにどのような API が必要なのか検証し，その有効性を確かめる必要がある．さらに LLAPI での検索方法をフリーワード検索などに対応させるなどにより，ユーザにより検索しやすい API を提供するべきである．これを実現した上で，さまざまなライフログのマッシュアップにより，ユーザにどのような付加価値が提供できるのか検討・検証をしていきたい．

謝　　辞

この研究の一部は，科学技術研究費 (若手研究 B 20700027, 21700077，研究活動スタート支援 22800042) の助成を受けて行われている．

文　　献

[1] 中村匡秀，下條　彰，井垣　宏，“異なるライフログを集約するための標準データモデルの考察，” 電子情報通信学会技術研究報告，vol.109，no.272，pp.35–40，2009-11-05.
[2] 下條　彰，福田将之，井垣　宏，中村匡秀，“異なるライフログをマッシュアップするためのデータ変換・集約アクセス api の実装，” 電子情報通信学会技術研究報告，vol.109，no.450，pp.85–90，2010-03-04.
[3] 鎌田早織，坂本寛幸，井垣　宏，“マッシュアップ api を用いた異なるライフログサービスの連携 (ライフインテリジェンスとオフィス情報システム)，” 電子情報通信学会技術研究報告，vol.109，no.450，pp.91–96，2010-03-04.
[4] mixi, Inc., “mixi”. http://mixi.jp/.
[5] Facebook, Inc., “Facebook”. http://www.facebook.com/.
[6] Twitter, Inc., “twitter”. http://twitter.com/.
[7] BOstudio, Inc, “ねむログ”. http://www.nemulog.jp/.
[8] W3C, “W3c web service activity”. http://www.w3.org/2002/ws/.
[9] 渡辺 幸三，販売管理システムで学ぶモデリング講座，翔泳社，2008.
[10] The Apache Software Foundation, “Apache axis2”. http://http://www.apache.org/.
[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” SOSP ’07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, pp.205–220, New York, NY, USA, ACM, 2007.
[12] D. Lizcano, J. Soriano, M. Reyes, and J.J. Hierro, “Ezweb/fast: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services,” iiWAS ’08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, pp.15–24, New York, NY, USA, ACM, 2008.
[13] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah, “Data integration in mashups,” SIGMOD Rec., vol.38, no.1, pp.59–66, 2009.
[14] E.M. Maximilien, A. Ranabahu, and K. Gomadam, “An online platform for web apis and service mashups,” IEEE Internet Computing, vol.12, pp.32–43, 2008.
[15] S. Abiteboul, O. Greenshpan, and T. Milo, “Modeling the mashup space,” WIDM ’08: Proceeding of the 10th ACM workshop on Web information and data management, pp.87–94, New York, NY, USA, 2008.
[16] F. Amato, V. Casola, A. Gaglione, and A. Mazzeo, “A common data model for sensor network integration,” Complex, Intelligent and Software Intensive Systems, International Conference, vol.0, pp.1081–1086, 2010.