データ入出力統合フレームワークによる複数の SNS と連携するアプリケーション開発の支援

岡崎亮介[†1]　毛利公美[†2]　白石善明[†1, †3]

SNS（Social Networking Service）の普及に伴って，災害時における情報提供のための被災者支援システムも SNS を用いて提供される見込みがある．システムに用いられる SNS が利用者の日常的に使っている SNS と異なる場合，利用者は使い慣れない SNS を操作することになる．システムの利用を促すために，日常的に使っている SNS を模したクライアントアプリケーション（以下，クライアント）からシステムに用いられている SNS を操作できればよいが，複数の SNS と連携するクライアントを開発するにはそれぞれで異なる API を把握する必要がある．本稿では，クライアント開発を支援するために，複数の SNS のデータ入出力を統合するフレームワークを提案する．フレームワークが再利用性および拡張性を備えていること，および，フレームワークによって機能追加が容易にできることを示している．

Supporting the Development of Application Cooperating with Plural SNSs by Input-Output Integrating Framework

RYOSUKE OKAZAKI[†1]　MASAMI MOHRI[†2]　YOSHIAKI SHIRAISHI[†1, †3]

1. はじめに

度重なる災害の経験から，被害を未然に防ぐための防災だけでは不十分であり，被害を最小限に留めるための減災もまた重要であると認識されるようになってきている．減災の取り組みのひとつに，被災者へ正確な情報を伝えることで適切な行動を支援するものがあり，そのための被災者支援システムがある．このようなシステムは，主に政府機関や自治体などの公共機関が提供している．

被災者支援システムは，SNS（Social Networking Service）の普及にともなって，SNS を用いて提供される見込みがある．このとき，被災者支援システムが用いる SNS は，利用者が日常的に使っている SNS とは異なることがある．被災者支援システムは，利用者情報の登録時や災害発生時に利用する機会が限られているため，災害が発生した際には，利用者がシステムを使い慣れていない状態で使う場面が出てくる．使い慣れない SNS で提供されるシステムは，使いにくいと感じやすい．災害時のような非常時にあっては，その心理的な傾向は増すであろうと考えられる．こうした状況は，システムが利用されないことにつながる．

図 1 に示すように，システムの利用者が日常的に使っている SNS を模したクライアントアプリケーション（以下，クライアント）から，日常的に使っている SNS に加えて，被災者支援情報が提供される SNS を操作できれば，使い慣れた GUI でシステムを使うことができる．このようなクライアントを提供することを考えると，SNS は多数存在することから，複数の SNS に対応することが望ましい．しかし，各 SNS はそれぞれ異なる形式でデータをやり取りし，クライアント開発にはそれらデータ形式の把握が必要になることから，複数の SNS に対応したクライアントを開発するには，対応する SNS の分だけコストがかさむ．

本稿では，複数の SNS に対応したクライアント開発を支援するための，SNS のデータ入出力を統合するフレームワークを提案する．フレームワークを用いてクライアントを試作し，試作クライアントのフローズンスポットとホットスポットについて考察を与え，再利用性と拡張性について評価する．

以下，2 章では，SNS の活用事例から，SNS が被災者支援システムの情報基盤として利用される見込みがあることを説明する．3 章では，SNS と連携するクライアント開発の課題を説明し，課題を解決するための SNS のデータ入出

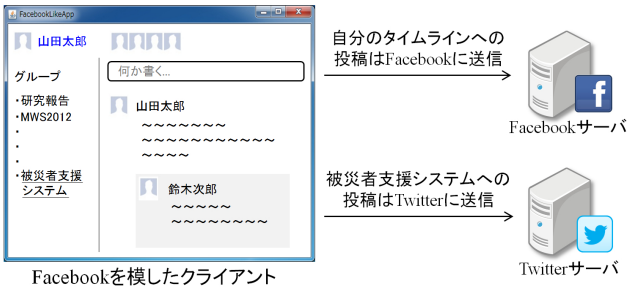

図 1　使い慣れた GUI による被災者支援システムの操作
Figure 1　Operation of the disaster information system by familiar GUI.

†1 名古屋工業大学大学院工学研究科
Graduate School of Engineering, Nagoya Institute of Technology
†2 岐阜大学
Gifu University
†3 名古屋工業大学高度防災工学センター
Advanced Disaster Prevention Engineering Center, Nagoya Institute of Technology

力を統合する仕組みについて述べる．4 章では，3 章で述べた仕組みを実現する方法について議論する．5 章では，フレームワークの基本的な性質を説明し，その性質を満たすように定めた設計について述べる．6 章では，フレームワークの実装，および，フレームワークを用いて試作したクライアントについて述べる．7 章では，実装したフレームワークについて，再利用性，拡張性の観点から評価する．最後に 8 章でまとめる．

2. 被災者支援システムと SNS

2.1 災害時における SNS の活躍

2011 年 3 月に発生した東日本大震災において，Twitter や Facebook などの SNS が活用されたことが報告されている．家族・知人の安否確認や被災状況など，主に情報入手を目的として利用された．SNS は日常的に利用されるサービスであり，使い慣れていた人により，災害時のような非常時に積極的に使われたと考えられる．SNS を通じたやり取りは電話が不通のときでも可能であったり，メールや災害伝言板よりも早く情報を入手できた[1]ことも，SNS が活用された一因である．

また，震災後に公共機関のソーシャルメディアへの注目が高まった[2]という指摘もある．公共機関は災害時におけるソーシャルメディアの有効性に注目していると考えられる．

2.2 SNS 利用者の増加

総務省[3]は，スマートフォンやタブレットなどの携帯端末が普及することで，ソーシャルメディアの利用が広がる可能性を指摘している．実際に，総務省の調べでも SNS の利用者は年々増加している．また，大手 SNS に利用者年齢制限緩和の動き[4][5]もあることから，今後幅広い世代に浸透してゆくことで，より利用者が増加することが予想される．

2.3 公共機関による SNS の利用

公共機関が提供するシステムに SNS が用いられるようになってきている．

総務省消防庁は Twitter 上で災害情報タイムライン[6]を提供している．また，SNS を通じて 119 番通報する仕組みを検討しており，実証実験の準備を進めている[7].

佐賀県武雄市[8]をはじめとした，複数の自治体は Facebook 上でホームページを運用している．特に佐賀県武雄市は，ホームページの完全移行や，Facebook 上で通信販売ページを展開[9]するなど，SNS の活用がめざましい．

SNS を用いることには，システムの構築・運用・管理にかかるコストが減少することや，情報が人から人へと伝わりやすく，広がりやすいこと，既存の会員基盤を利用できることなどのメリットがあることから，今後，公共機関が SNS を用いてシステムを構築する事例が増加することが考えられる．

2.4 被災者支援システムの SNS 利用

2.1 節から 2.3 節までに述べたことから，被災者支援システムもまた SNS を用いて提供される事例が増加すると考えられ，本研究でもそのような状況を想定する．このとき，利用者が普段使っていない SNS をシステムが用いていると，システム利用の阻害要因となる．

そこで，使い慣れた SNS の GUI を模したクライアントを提供することでシステムの利用につなげることを以下では考える．

3. クライアントアプリケーションの開発支援

3.1 クライアントアプリケーション開発の課題

SNS と連携するクライアントを開発するには，各 SNS が公開している API を用いて SNS サーバとやり取りしなくてはならない．クライアント開発者は，それらの API の仕様を把握することが必要になる．SNS 運営事業者は，クライアント開発者に API を提供するとともに，API の仕様をリファレンスマニュアルとして Web サイトなどを通じて公開している．API の仕様はしばしば変更されることがあり，仕様を把握し直した上で，クライアントに反映させる作業が発生することがある．素早く仕様変更に対応するには，クライアント実装コードの変更が最小限となるような構成となっていることが望ましい．

ここで言及している API とは，一般に Web API と呼ばれるものであり，図 2 のように，主に HTTP 通信を用いて呼び出すものである．呼び出しに対する応答は JSON（JavaScript Object Notation）形式や，XML（Extensible Markup Language）形式などで返される．したがってクライアント開発では，HTTP 通信機能や，JSON あるいは XML データのオブジェクトへの変換機能を実装することになる．

一方，クライアント開発を支援するため，HTTP 通信やオブジェクトへの変換を自動で行うライブラリが各 SNS それぞれで公開されている．ライブラリによって機能実装の作業コストは削減されるが，SNS ごとに扱いの異なるライブラリを用いるため，学習コストは対応する SNS の分だけ必要となることに変わりはない．

以上を踏まえると，クライアント開発において以下を満たすことが求められる．

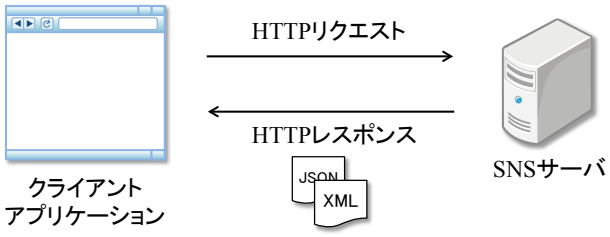

図 2 Web API の流れ
Figure 2 Flow of Web API.

- API を容易に呼び出すことができる
- API の仕様変更に柔軟に対応できる
- 新たな SNS に対応する際でも，API の仕様やライブラリの仕様を把握することなく対応できる

3.2 データ入出力の共通化

API の仕様を把握することは，入出力データを把握すること，および，どの API を用いれば目的を果たせるかを把握することである．図 3 に示すように，API と入出力データが SNS ごとで異なることが，複数の SNS に対応したクライアントの実装を難しくしている．各 SNS のデータ入出力を共通化できれば，複数の SNS への対応が容易になり，クライアントの開発を支援することができる．

データ入出力の共通化を実現する方法として，図 4 に示すような仕組みが考えられる．まず，クライアントと SNS サーバの間に，データのやり取りを仲介するインターフェースを挿入する．クライアントからの入力も，SNS サーバからの出力も，データは入出力が規定されたインターフェースを通じてやり取りされる．クライアントから入力するデータは共通形式とし，入力されたデータは対象となる SNS 専用の通信モジュールによって SNS 固有の形式に変換される．変換されたデータを用いて API が呼び出され，SNS サーバへ入力される．SNS サーバから出力されるデータは SNS 固有の形式であるが，通信モジュールによって共通形式へと変換された後，クライアントへ出力される．

このような仕組みがあれば，クライアント開発者は統合されたインターフェースが提供するひとつの API のみ理解すれば，複数の SNS に対応したクライアントを開発することができる．すなわち，新たな SNS に対応することになったときでも，対象 SNS の API を把握しなくともよくなり，学習コストがかからずに済む．

4. データ入出力統合フレームワーク

災害の発生は容易には予測できない．いつ起こるかわからない災害に対しては，迅速にシステムを構築し，災害が発生したときに備えた運用がなされているべきである．そのために，開発の工程をできるだけ短縮できることが求められる．

ソフトウェア開発を支援するものの代表にライブラリがある．ライブラリは，ソフトウェアに必要となる機能をひとまとめにして提供するものであり，コーディング作業を削減することができる．一方で，コーディングよりも，設計作業の方がソフトウェア開発の中で時間のかかる工程とされている．したがって，設計も支援できることが望ましい．

本研究で開発対象とするクライアントは，各自治体などが提供する被災者支援システムに合わせて複数開発されることが想定されるが，どのクライアントにもある程度共通

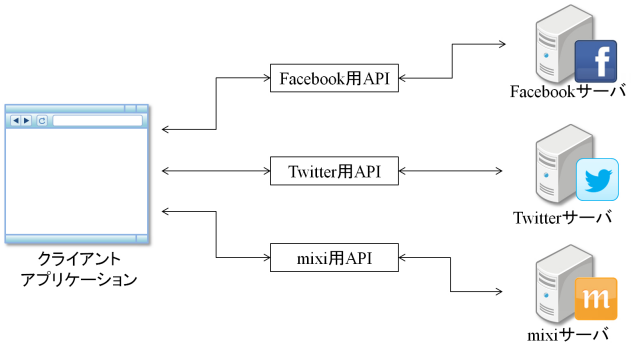

図 3　SNS ごとに異なる API とデータ入出力

Figure 3　API and input-output different in each SNS.

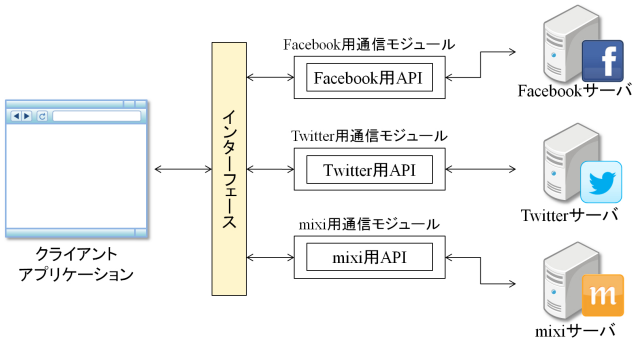

図 4　SNS のデータ入出力を統合する仕組み

Figure 4　A mechanism of integrating SNSs' input-output.

の設計が現れてくると考えられる．

本稿では，コーディングだけでなく，設計も支援することを考え，SNS のデータ入出力統合フレームワークを提案する．フレームワークを用いた開発では，ライブラリのようにまとまった機能を提供するだけでなく，ソフトウェアの設計の一部がフレームワークによって規定されていることで，設計作業の削減が可能となる．

5. フレームワークの設計

5.1 設計方針

5.1.1 フレームワークの基本的な性質

ソフトウェアの開発において，一度作成された設計やモジュール，コーディング技法などを別の開発の際に再利用することで，開発効率や品質を高める取り組みがなされてきた．フレームワークは，そうした中で見出されてきた，複数あるソフトウェア再利用技術のひとつである．

一般的な実装では，アプリケーション開発者の記述するコードがアプリケーション全体の制御を行う．フレームワークを用いた実装では，フレームワークがアプリケーションの制御を行い，アプリケーション開発者が記述するコードはフレームワークに呼び出される形になる．この現象は制御の反転（Inversion of Control）と呼ばれ，フレームワークの重要な性質のひとつとされる．同じく再利用技術であるライブラリとは，この点において異なる．

アプリケーション開発者は，フレームワークが提供する共通機能を利用してアプリケーション固有の機能を実装することになる．共通機能は，別のアプリケーションを開発するときにもそのまま使えるように，すなわち，再利用性が高くなるように設計されるべきである．一方で，アプリケーション固有の機能は実装がしやすいように,すなわち，拡張性が高くなるように設計されるべきである．

5.1.2 フローズンスポットとホットスポット

再利用性と拡張性を持たせるためには，ユーザ（フレームワーク利用者）の要求を実現しやすい構成としなければならない．そこで，ユーザが「どのような機能を実現したいと考えているか」という要求を分析する．

Pree[10]は, フレームワークの構成をフローズンスポットとホットスポットという概念で表している．クライアント共通の機能として用意し，今後，追加・変更をしないと定めた箇所をフローズンスポットという．クライアント開発者は，フローズンスポットにある機能を利用してクライアントを開発する．フローズンスポットは，クライアントに依らず常に利用される，つまり再利用される箇所になる．対して，クライアントごとに異なる機能の追加・変更を容易にできるよう定めた箇所をホットスポットという．クライアント開発者が実装する箇所はこの部分になる．ホットスポットは，それぞれのクライアントで独自に実装，つまり拡張される箇所になる．

本稿では，被災者支援システムのクライアント開発者の要求を以下のように想定する．

(1) GUI を自由にデザインできること
クライアント開発者は，ある SNS を模したクライアントを開発するために，GUI を自由にデザインできなければならない．

(2) 画面表示するデータの取り扱いが容易であること
クライアント開発では，SNS 上のデータを画面表示するように実装する．SNS ごとに形式の異なるデータの取り扱いが容易になれば,迅速なクライアント開発が可能になる．

(3) 被災者支援システム独自の機能追加が容易であること
クライアントは SNS 上の情報を表示するだけでなく，被災者を支援する機能を追加されることが期待される．

要求が明確化されることで，再利用する部分と，拡張する部分が切り分けられる．(1)，(3)はクライアントごとに異なるものであるから，ホットスポットとすることが望ましい．(2)はデータ形式を共通化することで実現できることから，常に固定の機能として，フローズンスポットとすることが望ましい．

5.1.3 パターン

再利用性と拡張性を高めるためには適切な設計が求められる．そのためにはパターンの適用が不可欠とされている[11]．

パターンは, ソフトウェア開発の過程で発見されてきた，開発上で頻繁に現れる特定の問題を解決するための方法論である．これまでに様々なパターンが提案・主張されてきたが，それらはすべて同じレベルの抽象度ではない．Buschmann ら[12]は，抽象度レベルに応じて 3 つにパターンを分類している．抽象度の高い順に，アーキテクチャパターン, デザインパターン, イディオムである．このうち，アーキテクチャパターン，およびデザインパターンが設計に関わるパターンである．本研究では，抽象的な設計から段階的に具体化することで，設計品質を高めることができるという考えのもと，はじめにアーキテクチャパターン，その後デザインパターンを適用して設計している．なお，イディオムはコーディングの際に適用するパターンであり，フレームワークの性質に関わるものではないため，本稿では言及しない．

5.2 設計

5.2.1 フローズンスポットとホットスポットの設定

フレームワークの構成は図 5 のようになっている．

SNS サーバから取得したデータを保持するデータ保管部はフローズンスポットとする．データ形式を統一することで, 本来 SNS によって異なる形式のデータを画面表示しやすくなる．その他，SNS に応じた通信処理を記述するデータ通信部や，SNS データを整形するデータ整形部，データ処理の完了通知をやり取りする通知送信部および通知受信部，データ処理を制御する制御部も同様にフローズンスポットとし，クライアント開発に必要な機能をあらかじめ実装した状態で提供する．

ホットスポットとして，GUI にあたる画面入力部と画面出力部を設ける．これにより，クライアント開発者が GUI を自由にデザインできるようにする．また，フローズンスポットとして前述したデータ整形部は，拡張可能なように一部をホットスポットとする．SNS データの取り扱いを容易にすることで，SNS データを利用した独自機能の追加が容易になる．

以上のようにすることで，クライアント開発者は，制御の流れや API の呼び出し，データの取り扱いを考えずに GUI の設計・実装ができるようになる．

5.2.2 パターンの適用

フローズンスポットとホットスポットを設定したことにより，再利用する部分と拡張する部分が定まったことになる．さらに，パターンを適用することで再利用性と拡張性の向上を図る．

ソフトウェアの基礎的な構造を定めるアーキテクチャパターンには，対話型アプリケーション（GUI アプリケーション）の構築に適している MVC（Model-View-Controller）パターンを適用した．MVC パターンは，既存の GUI アプ

リケーションのためのフレームワークでも多く用いられている．処理（Model），入出力（View），入力に応じた処理の制御（Controller）を分離することで M，V，C 間の依存性を下げることができる．依存性が下がることで，M，V，C のいずれかひとつを変更した際でも，他のふたつへの影響を減らすことが可能になる．特に，GUI を構成する View はしばしば変更されるが，その場合に Model や Controller の変更は最小限で済み，再利用性が高まる．図 5 のフレームワーク構成図に MVC パターンを適用したものが図 6 となる．

デザインパターンは，アーキテクチャを構成するサブシステムが持つさらに小さなアーキテクチャである[12]．Gamma ら[13]がまとめたデザインパターンを含む，ソフトウェアの機能を構造化するためのパターンが集められている．本フレームワークでは，Observer パターンと Strategy パターンを適用した．

View（GUI）は，Model で処理されたデータを取得し，出力（画面表示）する．View と Model は相互に干渉するため密結合になりやすい．一方で，GUI は変更されやすい箇所であり，それに伴う Model への影響は最小限に留めたい．Observer パターンは，処理（Model）の完了の GUI（View）への通知を疎結合にするパターンであり，MVC パターンを適用したフレームワークでよく用いられている．クラス間の関係は図 7 のようになる．SNS サーバより受け取ったデータの整形が完了すると，通知クラスによって，Observer インターフェースを通じて GUI（View）へ通知されるようになっている．クライアント開発者は，整形されたデータをどのように表示するかを画面表示クラスに記述するだけでよい．以下に，Observer パターンの実現例を示す．Java を用いた例で示しているが，一般的なオブジェクト指向言語でも同様にして記述できる．

(1)通知受信時の処理インターフェースを規定する

```
/* 監視クラス */
public interface Observer{
    public abstract void update();
}
```

(2)通知のひな形を規定する

```
/* 通知クラス */
public abstract class Notifier{
    private ArrayList observers = new ArrayList();
    public void notifyObservers(){
        Iterator itr = observers.iterator();
        while(itr.hasNext()){
            (itr.next()).update();
        }
    }
}
```

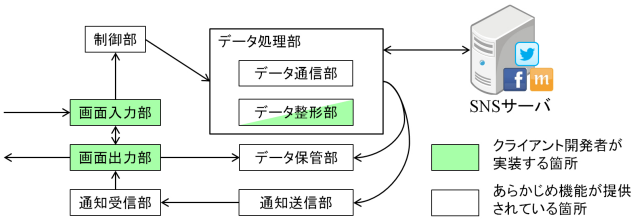

図 5　フレームワークの構成

Figure 5　Structure of the framework.

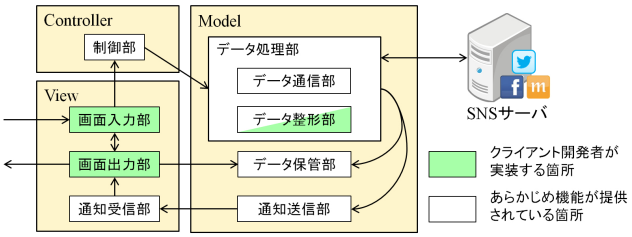

図 6　MVC パターンを適用したフレームワークの構成

Figure 6　Structure of the framework which applied MVC pattern.

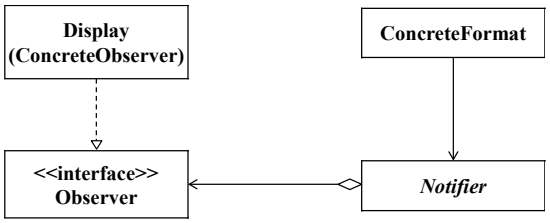

図 7　Observer パターンを適用した Model から View への通知

Figure 7　Notice applied observer pattern from a model to a view.

(3)データ整形処理後，GUI へ通知する

```
/* データ整形処理クラス */
public class FormatA extends Notifier implements Format{
    @Override
    public void format(){
        // データ整形処理
        notifyObservers();
    }
}
```

(4)通知受信時の画面表示処理を記述する

```
/* GUI クラス */
public class Display implements Observer{
    @Override
    public void update(){
        // データの画面表示処理
    }
}
```

SNS に応じた処理を容易に切り替え可能にして，クライアントを複数の SNS との連携に対応するために次のように Strategy パターンを適用する. Strategy パターンはアルゴリズムを切り替えるためのパターンであり, これによって, Controller より呼び出される各 SNS に応じたデータ処理（Model）を容易に切り替えることとする．クラス間の関係は図 8 のようになる．あらかじめインターフェース（抽象クラス)としてメソッド名, および引数を規定しておき, それにしたがって各 SNS サーバとの通信機能や SNS サーバから受け取ったデータの整形機能を具象クラスとして記述することで，フレームワーク利用者は，本フレームワークで規定された API を通じて各機能を使用できるようになる．これにより API が統合され，SNS ごとの API の差異が吸収される. 以下に, 各 SNS サーバとの通信機能を Strategy パターンによって実現する方法を, Java を用いた例で示す.

(1) はじめにインターフェースを規定する.

```
/* 通信用インターフェース */
public interface Communicator{
    public abstract void sendMessage(String message);
}
```

(2) 抽象メソッドに対して, SNS に応じた処理を記述する

```
/* Twitter 用通信クラス */
public class ComForTwitter implements Communicator{
    @Override
    public void sendMessage(String message){
        // Twitter 用メッセージ投稿処理を記述
    }
}
/* Facebook 用通信クラス */
public class ComForFacebook implements Communicator{
    @Override
    public void sendMessage(String message){
        // Facebook 用メッセージ投稿処理を記述
    }
}
```

(3) インターフェースを通じて, 目的の通信クラスを利用する

```
/* 制御クラス */
public class Controller{
    private Communicator twitter;
    private Communicator facebook;

    public Controller(){
        twitter = new ComForTwitter();
        facebook = new ComForFacebook();
    }

    // 何らかの処理（対 Twitter）
    public void processingA(){
        twitter.sendMessage("message");
    }
    // 何らかの処理（対 Facebook）
    public void processingB(){
        facebook.sendMessage("message");
    }
}
```

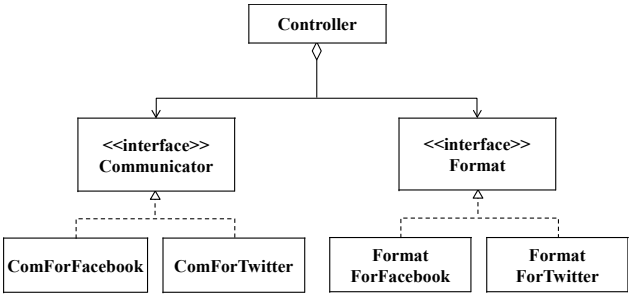

図 8 Strategy パターンを適用した各 SNS 固有の処理

Figure 8 A peculiar processing of each SNS which applied strategy pattern.

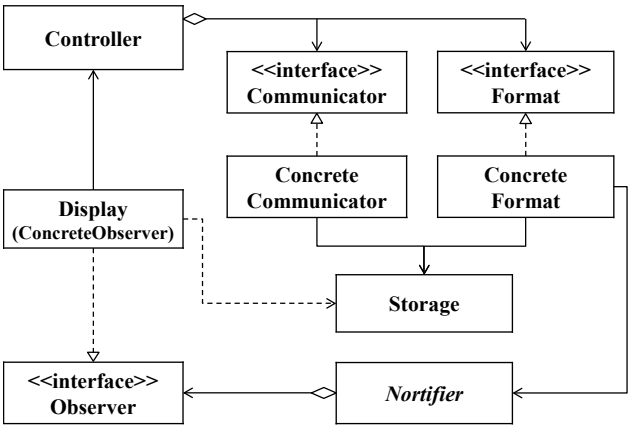

図 9 フレームワークのクラス構成

Figure 9 Class constitution of the framework.

表 1 フレームワークの開発環境

Table 1 A framework development environment.

言語	Java (JDK 1.7.0_09)
ライブラリ	Twitter4J 3.0.3
	RestFB 1.6.11

6. 実装

6.1 実装したフレームワーク

5 章で述べた設計に基づき，データ入出力統合フレームワークを実装した. 全体のクラス構成は図 9 のようである. 開発言語には Java を用いた．開発環境を表 1 に示す.

本稿では，Facebook と Twitter に対応した実装について

述べるが，API が公開されている任意の SNS を追加可能である．ここでの API の呼び出しにはオープンソースのライブラリを用いている．

本フレームワークによるデータ入出力統合の例を以下に示す．ここでは，SNS サーバへのメッセージ投稿における API 呼び出しの例を示している．

(1)　各 SNS が公開している Web API の呼び出し

Twitter：

[HTTP リクエスト]

https://api.twitter.com/1.1/statuses/update.json

[POST データ]

message=TestMessage

Facebook：

[HTTP リクエスト]

https://graph.facebook.com/me/feed

[POST データ]

status=TestMessage

(2)　ライブラリによる API の呼び出し

Twitter（Twitter4J[14]）：

```
String message = "TestMessage";
twitter.updateStatus(message);
```

Facebook（RestFB[15]）：

```
String message = "TestMessage";
Parameter param = Parameter.with("message", message);
facebook.publish("me/feed", FacebookType.class, param);
```

(3)　本フレームワークによる API の呼び出し（*は SNS に応じたオブジェクト）

SNS に依らず共通：

```
String message = "TestMessage";
*.sendMessage(message);
```

SNS が公開する Web API を呼び出すには，(1)のようにHTTP リクエストによって各 SNS サーバにアクセスすることになるが，HTTP 通信の実装が別途必要になる．一般的には，(2)のようにライブラリを利用して実装する．しかし，SNS ごとに Web API は異なっている．本フレームワークを用いると，どの SNS に対しても(3)のようにコードを記述することで，それぞれの SNS サーバとやり取りすることができる．したがって，クライアント開発者は，本フレームワークによる API の呼び出し方を覚えるだけで複数の SNS に対応したクライアントを開発することが可能になる．API の呼び出しを行う通信モジュールは，Strategy パターンの適用により容易に差し替え可能であり，API の仕様が変更された際の対応は最小限で済む．

6.2　評価用クライアントアプリケーション

災害発生時には，外出していることや，避難のため移動することなどがあり，携帯端末で情報を入手することが多

表 2　クライアントアプリケーションの開発環境

Table 2　A client application development environment.

言語	Java (JDK 1.7.0_09)
SDK	Android SDK rev. 21.0.1

図 10　試作クライアントの操作画面

Figure 10　Operation screen of prototype client application.

表 3　フローズンスポットとホットスポットの比率

Table 3　The ratio of frozen spots and hot spots.

フローズンスポット	66.9%
ホットスポット	33.1%

くなると考えられる．そこで，Android 上で動作する Java クライアントを実装したフレームワークを用いて開発した．クライアントの開発環境を表 2 に，操作画面を図 10 に示す．

本稿では 2 種類のクライアントを開発している．Facebook の GUI を模したものと，Twitter の GUI を模したものであり，どちらも Facebook と Twitter の両方を同時に操作可能である．

7.　評価

7.1　フローズンスポットとホットスポットの比率

試作した 2 つのクライアントについて，フローズンスポットとホットスポットの比率を計測した．2 つのクライアントの比率の平均を表 3 に示す．

フローズンスポットはアプリケーションに依らない共通の部分であり，ここで提供される機能をもとにクライアントを開発する．フローズンスポットの内，フレームワークとして構成するために規定したインターフェース部分はオーバーヘッドであり，その割合は 3.4%であった．したがって，クライアントを試作するのに 63.5%のコードを記述せずに済んだことになり，再利用性を備えているといえる．また，フローズンスポットを変更する必要があった場合は，フレームワークの設計に不備があったことになる．今回の

試作ではフローズンスポットを変更することはなかったため，設計は適切であったといえる．

ホットスポットはアプリケーションごとに固有の部分であり，それぞれで異なる実装をするものである．したがって，ホットスポットが存在することが拡張性を備えていることの条件となる．今回の試作では，33.1%がホットスポットとなった．

GUI は作り込むことのできる余地が大きく，作り込むのにともなってコード量が増す部分である．GUI によってコード量が増すことになればホットスポットの割合も高まることになるが，過度に実装すると，有意な比率が計測されなくなる．試作クライアントでは，GUI は最低限の実装をしており，フローズンスポットの割合が多くなること，すなわち再利用可能な部分が多くなることが確認された．

7.2 被災者支援システム独自の機能の追加

実際に被災者支援システムが提供される際には，それぞれのシステムごとに，単純な情報提供以外の独自の機能を追加されることが考えられる．本フレームワークでは，そうした独自の機能を容易に追加できるような構成としていた．以下では，救援要請機能を機能追加の例として示す．

救援要請機能の操作画面を図 11 に示す．機能の概要は次の通りである．

(1) 救援を要請したいユーザは，クライアントのメニューから救援要請ボタンを押すことで，「タスケテ！」というメッセージとともに現在地が送信される．

(2) 「タスケテ！」というメッセージを見た別のユーザは，メッセージの横に表示される「Help me！」と表示されたボタンを押す．

(3) 地図アプリケーションが立ち上がり，救援要請者の現在地を知ることができる．

このような機能を，Facebook と Twitter の双方で動作するように実装したところ，ライブラリのみを用いて実装した場合では，SNS にかかわらない共通部分に 27 行，Facebook サーバとの通信に 9 行，データの整形に 59 行，Twitter サーバとの通信に 9 行，データの整形に 33 行のコードを要した．一方，本フレームワークを用いた実装では，Facebook サーバとの通信のために 1 行のコードを記述するが，この 1 行のコードを書き換えるだけで Twitter サーバとの通信が可能になる．つまり，一度作成された機能は，最小限の書き換えで新たな SNS へ対応することができる．なお，ここで示しているコード行数は，救援要請機能の機能部分を追加するのに必要となったコード行数であり，GUI の実装コードは含まれていない．

また，ライブラリを用いた場合，本機能の追加のために SNS サーバと通信するのに必要なコードは，Facebook と Twitter でそれぞれ 9 行と示したが，これらのコードを記述

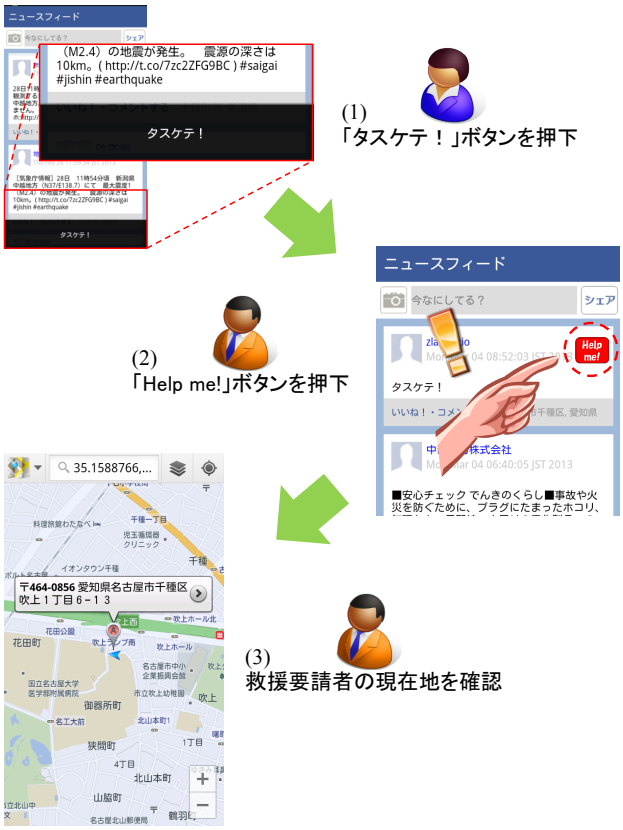

図 11 救援要請機能の操作画面

Figure 11 Operation screen of a function of rescue request.

することは容易でないこともある．なぜなら，ライブラリには，SNS が公開する API の仕様を把握していないと使えないものもあり，API の仕様が複雑であると必要とする API が見つけられないことがある．また，「API の仕様が詳細に記載されていない」，「記載されている情報が古い」，「記載そのものがない」など，リファレンスマニュアルが十分に整備されていないことがあると，試行錯誤によって目的の API やデータを入手することになることも二つ目の理由としてあげられる．さらに，SNS によって API の仕様が異なることから，別の SNS へ新たに対応するときには上記二点の困難さが顕在化する．これらの開発の遅滞の要因に対して，本フレームワークでは，各 SNS で公開されている API の仕様をアプリケーション開発者がすべてを把握する必要なく SNS サーバとやり取りすることができることから，本フレームワークを利用することで開発を迅速に進めることができる．

以上のように，本フレームワークを用いることで，機能を拡張するのにかかる作業コストを抑えることができる．この評価用クライアントアプリケーションの実装においては，従来では，Facebook と Twitter の双方について API を把握しなければならなかったが，本フレームワークを用いれば，フレームワークで規定されている API さえ理解しておけばよい．これは，Facebook と Twitter 以外の SNS につ

いてもフレームワーク開発者が対応すれば同様であり，対応する SNS が増えれば増えるほど，アプリケーション開発者の学習コストの削減効果は大きくなる．

8. おわりに

SNS の普及から，自治体などの公共機関が提供する被災者支援システムも SNS を基盤として提供される見込みがある．システムの利用を促すため，クライアントは利用者にとって使い慣れた GUI で提供されることが望ましい．システムの利用者が日常的に使っている SNS を模したクライアントによって達成できるが，一方で，複数の SNS と連携するクライアントを開発するには，SNS ごとに異なる API を呼び出さなければならないため，対応する SNS の分だけコストがかさむ．

本稿では，被災者支援システムのクライアント開発を支援するため，SNS のデータ入出力を統合するフレームワークを提案した．提案フレームワークにより，どの SNS に対しても同じ入出力でデータをやり取りできるようになる．

フレームワークの再利用性と拡張性を高めるため，ソフトウェアパターンに基づき設計した後，フレームワークを実装した．

評価として，試作クライアントのフローズンスポットとホットスポットについて計測し，再利用性と拡張性を備えていることを確認した．また，被災者支援システム独自の機能追加の例を示し，拡張が容易にできることを示した．

参考文献

[1] ウェザーニューズ：「東日本大震災」調査結果，入手先〈http://weathernews.com/ja/nc/press/2011/pdf/20110428_2.pdf〉（参照 2013-04-08）．

[2] 伊藤智久，安岡寛道：自治体のソーシャルメディア活用とその指標～オープンガバメントの促進に向けた民間プラットフォーム活用～，入手先〈http://www.nri.co.jp/publicity/mediaforum/2011/pdf/forum159.pdf〉（参照 2013-04-08）．

[3] 総務省：平成 24 年版 情報通信白書，入手先〈http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h24/html/nc123220.html〉（参照 2013-04-08）．

[4] Facebook Explores Giving Kids Access, THE WALL STREET JOURNAL, available from <http://online.wsj.com/article/SB10001424052702303506404577444711741019238.html> (accessed 2013-04-08).

[5] Google Japan Blog, 入手先〈http://googlejapan.blogspot.jp/2012/01/google.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+GoogleJapanBlog+(Google+Japan+Blog)〉（参照 2013-04-08）．

[6] 総務省消防庁 (FDMA_JAPAN) on Twitter，入手先〈https://twitter.com/FDMA_JAPAN〉（参照 2013-04-08）．

[7] 日本経済新聞：SNS で「119 番」、夏にも実証実験　消防庁，入手先〈http://www.nikkei.com/article/DGXNASDG1203R_S3A310C1CR8000/〉（参照 2013-04-08）．

[8] 武雄市役所，入手先〈http://www.facebook.com/takeocity〉（参照 2013-04-08）．

[9] F&B 良品，入手先〈http://www.facebook.com/FunBuytakeo〉（参照 2013-04-08）．

[10] Pree, W.（著），佐藤啓太，金澤典子（訳）：デザインパターンプログラミング，トッパン，（1996）．

[11] Johnson, E. R.，中村宏明，中山裕子，吉田和樹：パターンとフレームワーク，共立出版，（1999）．

[12] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal M.（著），金沢典子，桜井麻里，千葉寛之，水野貴之，関富登志（訳）：ソフトウェアアーキテクチャーソフトウェア開発のためのパターン体系，近代科学社，（2000）．

[13] Gamma, E., Helm, R., Johnson, E. R. and Vlissides, M. J.（著），本位田真一，吉田和樹（訳）：オブジェクト指向における再利用のためのデザインパターン，ソフトバンクパブリッシング，（1999）．

[14] Twitter4J，入手先〈http://twitter4j.org/ja/index.html〉（参照 2013-04-08）．

[15] RestFB, available from <http://restfb.com/> (accessed 2013-04-08).