Vol.2011-UBI-32 No.11
2011/11/24

ネットワークコンピューティングのための包括的マッシュアップフレームワークの検討

放地宏佳[†1]　三好健文[†1]
入江英嗣[†1]　吉永努[†1]

スマートフォンや情報家電といったネットワーク接続可能なデバイスにより実世界の多種多様な場面で情報処理を活用できるようになった．それに伴って，多種多様な用途を対象としたアプリケーションが求められるようになっている．そこで，多様化するアプリケーションをユーザ自身が開発できるよう，プログラミングに慣れていないユーザであっても，自らのアプリケーションへの要求を自らで満たすことが可能なマッシュアップフレームワーク IDUMO を提案する．

IDUMO フレームワークでは，マッシュアップに必要不可欠な機能である，(1) 統一的な入出力インタフェースの定義，(2) プログラム実行モデルの差異の吸収，(3) 容易な開発方法，を提供する．本論文では，IDUMO フレームワークの設計について述べ，アプリケーション開発のケーススタディにより有用性を示す．

IDUMO : A Study of The Comprehensive Mashup Framework for Network Computing

Hiroyoshi HOUCHI ,[†1] Takefumi MIYOSHI ,[†1]
Hidetsugu IRIE [†1] and Tsutomu YOSHINAGA[†1]

As smartphones and information appliances are widely used, we can employ information processing in all areas of day-to-day lives. Based on this, various applications are required for each of our activities. In order to make users to develop such required applications easily, a mashup framework IDUMO is proposed.

IDUMO framework provides three features to support mashup; 1) unified interfaces for input/output data, 2) adaptability for various program execution models 3) friendly implementation method. In this paper, the design of IDUMO framework is described and the availability is shown by several case studies to implement application with the framework.

1. はじめに

現在，ネットワークインフラの普及により，ネットワークを通じて様々なサービスにアクセス可能となった．例えば，天気予報や電車の遅延情報をリアルタイムに知ることができ，ネットワーク上の計算資源を用いることでデータの保管や計算処理を行うこともできる．加えて，パソコン以外の，スマートフォンや情報家電といったデバイスも普及しており，これらが直接ネットワークに接続され，ネットワーク上の資源の活用や，ネットワーク上からのデバイスの操作といった，多種多様な場面で情報処理を活用できるようになった．

しかしながら，様々なデバイスやネットワーク越しの豊富なサービスが発展したゆえに，新たな問題が発生する．アプリケーション開発者はユーザの要求に応じたアプリケーション開発が求められるが，デバイスやサービスの増加により，ユーザの要求自体が多様化してしまっている．例えば，ネットワーク上の天気予報を取得し表示するアプリケーションを開発することを考える．従来であれば，単に，一日一回天気予報を表示するアプリケーションがあればユーザの要求を満たしていた．しかしながら，現在の持ち運びが可能で，かつネットワーク越しの様々なアプリケーションが活用できる状況においては，以下のように天気予報情報を活用するケースも想定される．

- 自宅の天気予報と勤務先の天気予報を同時に表示したい
- ３時間毎のリアルタイムな天気予報を表示させたい
- 天気予報の結果を他の処理に利用したい

これらユーザの要求全てに対応したアプリケーションを開発することは困難であり，そのためアプリケーションで提供される機能とユーザの要求との間にギャップが生じる．また，ユーザがアプリケーションを実行させるプラットフォームも，パソコンやスマートフォン，情報家電など様々である．そのため，たとえ多種多様な要求に答えられるアプリケーションが開発可能であっても，そのユーザが持っているプラットフォームに対応させるためには移植する必要があり，移植コストが発生する．

従って，多様化した要求を解決できるアプリケーションを，種々のプラットフォームに向けてアプリケーション開発者が開発するのではなく，自らの要求を満たすアプリケーションをユーザ自身で開発するというアプローチが求められる．このアプローチに対する一つの解としてマッシュアップがある．マッシュアップとは，複数のサービスを組み合わせて新しいサービスを作成する技術を指す．例えば，食べログ[1)] では，店の位置と口コミを地図上に

†1 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications.

描画するという要求を，自身の持つ店の口コミ情報をまとめたサービスと，Google Maps[2)]の地図サービスを組み合わせることで実現している．このように，ある要求に対し，それそのものを満たすアプリケーションがない場合でも，要素となる複数のアプリケーションを組み合わせることで，要求を解決することができる．

しかし，自分自身の要求を満たすアプリケーションを，多種多様なネットワークサービスやデバイスの持つ機能を組み合わせて実現することは簡単ではない．従って，簡単なプロセスで自らが求めるアプリケーションを記述可能なプログラミング開発用フレームワークの実現が必要となる．そこで，手軽にマッシュアップを行うことができるアプリケーション開発フレームワーク IDUMO を提案する．IDUMO フレームワークでは，デバイスの機能やネットワーク上のサービスを機能毎にモジュール化し，それらのモジュールを選択，接続することでアプリケーションの開発を可能とする環境を提供する．モジュール化された機能やサービスをブラックボックス的に利用することができるため，一般的なプログラミングの知識やアルゴリズムの修得が不要であり，プログラミングに不慣れな人であっても，自らが求めるアプリケーションを簡単に開発することができる．

本論文では，IDUMO フレームワークを実現するにあたっての課題と IDUMO フレームワークの実現可能性について述べる．以降，第 2 節で IDUMO フレームワークの設計課題を示し，第 3 節で IDUMO フレームワークの実装方法について述べる．第 4 節では IDUMO フレームワークを用いたアプリケーション開発のケーススタディを示し，第 5 節でケーススタディを通じての IDUMO フレームワークの有用性について考察する．第 6 節で関連研究について述べ，最後に第 7 節で本論文をまとめる．

2. IDUMO フレームワーク実現のための課題と設計方針

IDUMO フレームワークでは，(1) ネットワークに接続されたデバイス上の機能とネットワーク上のサービスを自由に組み合わせた，(2) 各自の持つデバイスの上で実行可能なアプリケーションとして，(3) 簡単に構築可能な，アプリケーション開発環境を提供することを目指す．このようなフレームワークには以下に述べる設計課題がある．

統一的な入出力インタフェースの定義　一般的に，マッシュアップを行うためには，マッシュアップの要素として扱う対象を統一的な入出力インタフェースを有するモジュールとして扱い，そのモジュール化した機能間でデータを自由に授受するための手法が必要である．しかしながら，IDUMO フレームワークではマッシュアップを行う対象を不特定多数のデバイスやサービスとしなければならない．そのため，まず，それぞれ独自のインタフェースあるいはアクセス規約をもつデバイスやサービスを，統一した入出力インタフェースによって定義することが課題として挙げられる．

プログラム実行モデルの差異の吸収　それぞれのユーザが所望するアプリケーションとユーザが持っている端末が異なる場合，プログラム実行モデルの差異より，アプリケーションを実行することができない．マッシュアップを行った際，特定のデバイスのプログラム実行モデルに依存したアプリケーションを生成してしまうと，その他のデバイスを持つユーザがアプリケーションを利用できないという問題があげられる．

容易な開発方法の提供　マッシュアップを行う行為が難しい場合，利用できる人が限定されてしまう．プログラムに精通していないユーザが，自身の要求に合わせて簡単にアプリケーションを構築するためには，使用したいモジュールの宣言と，モジュール同士の接続を簡単に示せる手段を提供しなければならない．

本節ではこの 3 つの問題点について具体的な内容について述べ，IDUMO フレームワークでの解決方法について述べる．

2.1 統一的な入出力インタフェースの定義

ネットワークに接続されたデバイス上の機能や，ネットワーク上のサービスをマッシュアップの対象とする際，多種多様なリソースを取り扱うことになるが，リソースへのアクセス方法はそれぞれ異なる．例えば，ネットワーク上のサービスを用いる場合は REST を用いた XML ベースのアクセスを行い，センサデバイスを用いる場合はシリアル通信によって生ビットストリームを取得するといった，それぞれの異なったリソースへのアクセス方法が存在する．実際のプログラミングにおいてこれらリソースを扱う際には，それぞれのリソースの仕様を確認しながら開発が行われる．しかしながら，複数のリソースを組み合わせるマッシュアップを行う際に，リソースごとの複雑な仕様を逐次確認することは，開発コストの増加と，メンテナンス性の低下を招く．すなわち，自由にマッシュアップを行いアプリケーションを作成することが妨げられる．

この問題を解消するためには，他のリソースとの接続部分は共通のインタフェースを用いつつ，それぞれのリソースへの取り扱いには個別のプロトコルを扱えるようにする必要がある．IDUMO フレームワークにおいて，他のリソースとの接続は「データを受け渡す」，「データを受け取る」といった簡単なインタフェースを実装したモジュール単位として扱う．接続部分の共通インタフェースは，データの受け渡し方法を定義しているだけに過ぎない．そのため，ストリーミングデータの取得や，ネットワーク上のデータの取得といったリソース特有の処理はモジュール内に隠蔽化し記述することが可能である．接続されるモジュールが，どのようなデータ形式を取り扱い互いに接続可能であるかは，別途確認を行うことで，モジュール間の接続に矛盾がないことを確認できる．モジュール化を行うことにより，リソース間の接続を「複雑な仕様の確認」から，データ形式の確認という「簡単な仕様の確認」にすることができる．

2.2 プログラム実行モデルの差異の吸収

多種多様なデバイス上でアプリケーションを実行可能とするためには，各々の実行デバイスにおける，プログラム実行モデルの差異について考えなければいけない．例えば，パソコン上のコンソールでは，プログラムは逐次実行されるものである．しかしながら，AndroidOS上のプログラムの実行は，システムの発行するイベントにより制御される．このようなプログラム実行モデルの差異は，実行デバイスにおけるシステムの制約により発生するため，プログラム実行モデルを変更することは不可能である．そのため，マッシュアップで作成されたアプリケーションを実行するにはその制約上で動作させる必要がある．

IDUMO フレームワークでは，それぞれの実行デバイス上で「マッシュアップで作成されたアプリケーションを実行する」アプリケーション（Controller）を作成することでこの問題を解決する．Controller はデバイスのプログラム実行モデルをラッピングし，マッシュアップで作成されたアプリケーションを実行可能な環境を構築する．実行デバイスごとに固有の Controller を作成する必要は出てくるが，Controller によりプログラム実行モデルの差異の吸収が行われ，マッシュアップで作成されたアプリケーションの実行を保証することができる．

2.3 容易な開発方法の提供

マッシュアップによるアプリケーション開発を行う際には,「どのような機能を持ったアプリケーションを作成したい」と,「どのようにアプリケーションを動作させる」といった情報を記述できれば良い．なぜなら，必要最低限以上の情報の提示や，難しい構文規則を持つマッシュアップ環境は，アプリケーション作成以外の事を考えさせられ，アプリケーション開発に集中できない為である．

IDUMO フレームワークでは，マッシュアップ環境に必要な情報をアプリケーションの機能を表す「使用するモジュールの定義」,「モジュール間の接続情報」，アプリケーションの動作を表す「繰り返し回数」,「繰り返し間隔時間」のみを扱い，簡単な記述方法を提示し容易な開発環境を提供する．

3. IDUMO フレームワークの実装

本節では，第 2 節で述べた課題に対する IDUMO フレームワークにおける解決方法について述べる．なお，現在の IDUMO フレームワークは Java によって実装しているため，本節での説明には Java の用語を用いる．

3.1 リソース処理の差異問題の解決手法

IDUMO フレームワークでは，デバイス上の機能やネットワークサービスといったリソースを統一的に扱うために，すべてを IDUMOItem インターフェイスを実装するクラスとする．さらに，IDUMOItem のうち,「データを受け渡す」機能に相当するモジュールのインタフェース（Sender）と「データを受け取る」機能に相当するモジュールのインタフェース（Receiver）に分割する．これにより，データ授受に関する各モジュールの役割を明確に記述することができる．

図 3.1 データの受け渡しに関する機構

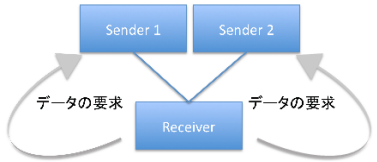

図 3.2 データの要求に関する機構

IDUMOItem 間のデータの受け渡しは，どのようなデータも入れることができるデータ構造 (パイプ) によってやり取りする（**図 3.1**）．パイプ自体はただ単純にデータの受け渡しを規定するインタフェースであり，データ授受が可能であるかの責任は負わない．このことにより，特定のデータ授受に依存した仕組みを持つことなく実際のデータを隠蔽化できる．特定の IDUMOItem 同士が接続可能であるかの確認は，Sender が送信するデータと Receiver が受信可能なデータのマッチングをとることで行う．Sender と Receiver に接続可能なデータの定義を行うことで，モジュール間接続の妥当性を持たせつつモジュール間の接続性を確保している．

データ取得の流れは，Receiver からの要求によって Sender がデータを受け渡す方式を採用する．この方式により，Receiver で Sender から受け取るデータの同期が可能であり，複数の Sender のデータを受け取るマッシュアップアプリケーションの構築が可能となる（**図 3.2**）．

3.2 実行デバイスの差異問題の解決手法

実行デバイス毎に非依存なアプリケーションの開発を実現するため，IDUMO フレームワークではアプリケーションを，マッシュアップにより記述されたアプリケーション手順を表すデバイス非依存な部分と，そのアプリケーション手順を実際にデバイス上で実行させるデバイス依存の部分に分離する．デバイス非依存な部分は Component インタフェースを，デバイスに依存する部分は Controller インタフェースを実装するクラスとして定義する．

Component にモジュールとモジュールの接続，モジュールの処理を何回実行するかといった実行デバイスに非依存な処理を定義する．Controller には，Component で定義された処理を「デバイス上の実行モデル」に合わせて実行できるよう処理を定義する．Controller を用いる事により，デバイス毎に異なるシステム制御方式を解消しつつ，異なるデバイス間

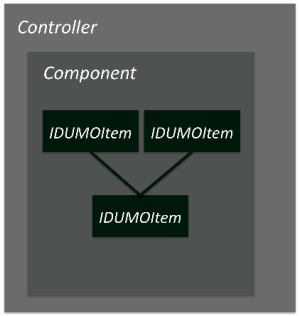

図 3.3 実行デバイスに応じたラッピング

でも同様のアプリケーションを実行できることが保証される（**図 3.3**）．このようにインタフェースを明確に分離することで，特定のデバイスで利用可能なモジュールを用いて設計された Component に対して，Controller を変更するだけで他のデバイスへ容易に移植することを可能とする．

3.3 容易な開発方法の提供方法

IDUMO フレームワークでは，マッシュアップを容易に行うために XML による簡単な構成ファイルでアプリケーションを開発できるようにする．この XML にはマッシュアップに必要となる，使用するモジュールの宣言，モジュール同士の接続についての定義および，処理の繰り返し回数，繰り返しの間隔時間を記述する．記述された XML ファイルは，コンパイラによって実行対象とするデバイス用のプログラムに変換され，実行対象となるデバイス用のコンパイラを用いて実行可能なアプリケーションに変換される．XML の例を**図 3.4** に示す．図 3.4 の 1 行目は XML のメタ情報である．2 行目から IDUMO フレームワークによるマッシュアップ情報が記述されており，2 行目自身には IDUMO フレームワークのメタ情報が記述されている．メタ情報の内容は `ver` が IDUMO フレームワークのバージョン情報，`type` が実行デバイスの指定，`name` がマッシュアップによって作成するアプリケーション名，`package` がアプリケーションを保存する場所となっている．このメタ情報の内容は，IDUMO フレームワークによってコードの自動生成をする際に重要な情報とはなるものの，補助的な情報でありマッシュアップ情報とは異なる．3 行目から 6 行目までの`<items>`でモジュールの宣言の定義を行い，7 行目から 8 行目の`<connections>`でモジュール同士の接続の定義を行なっている．10 行目から 14 行目までの`<executions>`は実行方法を示しており，11 行目の`<loop count>`で処理の繰り返し回数の設定，12 行目の`<sleep time>`で繰り返しの間隔時間の設定をしている．繰り返し回数の設定を-1 とすることで，処理を無限

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <idumo ver='0.1' type='android' name='Accelerometer' package='com.hixi_hyi.idumo.android.auto.app'>
3 <items>
4 <item class='AccelerometerProvider' name='a1' option='AccelerometerProvider.Type.X' init='activity'/>
5 <item class='TextViewReceiptor' name='textView' init='activity'/>
6 </items>
7 <connections>
8 <connect src='a1' sink='textView'/>
9 </connections>
10 <executions>
11 <loop count='-1'/>
12 <sleep time='1000'/>
13 </executions>
14 </idumo>
```

図 3.4 加速度センサ x の値を画面に出力する XML

回ループさせることができる．

4. ケーススタディ

第 3 節で述べた IDUMO フレームワークを Java で実装し，実際のアプリケーション作成を行った例をケーススタディとして示す．これらのケーススタディを通して，第 5 節で IDUMO フレームワークの有用性を述べる．

4.1 ネットワークサービスの利用

まず，ケーススタディとして，ネットワークサービスとして提供されている天気予報の情報を取得し，パソコンのコンソール上に文字列として表示するアプリケーションの作成を考える．アプリケーションを作成する際，パソコンなどの閉じたデータのみならず，ネットワーク上のデータを活用することが考えられるためである．

このアプリケーションを実現するためには，(1) ネットワークサービスからの天気予報を取得する IDUMOItem（LivedoorWeatherProvider）と（2）PC のコンソール上に文字列を表示する IDUMOItem（ConsoleViewReceiptor）の二つを用意し，これらをマッシュアップすればよい．

ここで，IDUMOItem の実装単位について述べる．マッシュアップという特性上，一つのモジュールは一つの機能を実現する単位で構成する方が，他のアプリケーションを作成する際に再利用しやすいと考えられる．また，逆に，あまりに小さい単位でモジュールを分割して実装すると，モジュール間でのデータ授受のオーバヘッドや，マッシュアップの組み合わせコストがいたずらに増加すると考えられる．そのため，センサや出力装置といった物理的なハードウェアモジュールに合わせて，IDUMOItem の単位を決めるのは自然な発想である．以降のケーススタディでも同様に自然な単位でのモジュール設計を行うこととする．

アプリケーションを構築するために，これらのモジュールの宣言と接続関係から成るマッシュアップ情報を XML で記述する．記述した XML ファイルは**図 4.1** の通りである．3 行目

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <idumo ver='0.1' type='console' name='TodaysWeather' package='com.hixi_hyi.idumo.console.auto.app'>
3  <items>
4  <item class='LivedoorWeatherProvider' name='w1' option='LivedoorWeatherProvider.Type.WEATHER' init='63'/>
5  <item class='ConsoleViewReceiptor' name='consoleview'/>
6  </items>
7  <connections>
8  <connect src='w1' sink='consoleview'/>
9  </connections>
10 <executions>
11 <loop count='1'/>
12 <sleep time='1000'/>
13 </executions>
14 </idumo>
```

図 4.1 天気予報アプリケーションのマッシュアップ情報を記述した XML

から 6 行目の<items>で LivedoorWeatherProvider と ConsoleViewReceiptor のモジュールの宣言がされていることがわかる．また，7 行目から 9 行目の<connections>で LivedoorWeatherProvider と ConsoleViewReceiptor の接続の定義がされている．10 行目から 13 行目の<executions>内の<loop count>より，このアプリケーションは一度だけ実行されることがわかる．この XML から自動生成されたプログラミングコードを**図 4.2** に，最終的に生成されたアプリケーションのスクリーンショットを**図 4.3** に示す．図 4.2 の 25 行目から 29 行目で図 4.1 の XML のモジュール定義が Java コード上のモジュール定義に置き換わっていることがわかる．また，30 行目の connect より，それぞれのモジュール同士の接続が宣言されていることがわかる．34 行目の setLoopCount より，実行が一度だけであることを宣言されていることがわかる．図 4.3 より，適切にネットワーク上の情報を取得し，利用できていることがわかる．

4.2 デバイス情報とネットワークサービスの連携

次に端末のデータをネットワークサービスを利用して解析するアプリケーションを作成するケーススタディとして，Android 端末の GPS 情報から，逆ジオコーディングサービス[3]を用いて現在地に変換し表示するアプリケーションの作成を考える．スマートフォンのアプリケーションを作成する上で，端末データとネットワークサービスの連携を行う頻度が高いと考えたためである．

このアプリケーションを作成するためには，Android の GPS 情報を取得する IDUMOItem（GPSProvider）と，GPS 情報から現在地に変換する IDUMOItem（ReverseGeocordingHandler）と Android 端末上に文字列を表示する IDUMOItem（TextViewReceiptor）を考えるとよい．これらをマッシュアップで組み合わせることにより GPS 情報から現在地を表示させるアプリケーションを構築することができる．

アプリケーションを構築するために記述した XML ファイルを**図 4.4** に示す．3 行目から 8 行目の<items>で GPSProvider と ReverseGeocordingHandler，TextViewReceiptor のモジュール定義がされている．また，9 行目から 13 行目の<connections>でそれぞれのモジュール接続の定義がされている．14 行目から 17 行目の<executions>の<loop count>と<sleep time>より，このアプリケーションは 10 秒ごとに何度も実行されることがわかる．この XML から自動生成されたプログラミングコードを**図 4.5** に，最終的に生成されたアプリケーションのスクリーンショットを**図 4.6** に示す．図 4.5 の 23 行目から 32 行目で図 4.1 の XML で定義したモジュールが Java コード上のモジュール定義に置き換わっていることがわかる．また，33 行目から 35 行目の connect より，それぞれのモジュールの接続が定義されている．40 行目と 41 行目からアプリケーションの繰り返し回数と，繰り返し間隔時間が定義されている．図 4.6 より，端末のデータとネットワークサービスの連携ができていることがわかる．

```
1  package com.hixi_hyi.idumo.console.auto.app;
2  import com.hixi_hyi.idumo.console.*;
3  import com.hixi_hyi.idumo.console.exec.*;
4  import com.hixi_hyi.idumo.console.provider.*;
5  import com.hixi_hyi.idumo.console.handler.*;
6  import com.hixi_hyi.idumo.console.receiptor.*;
7  import com.hixi_hyi.idumo.core.*;
8  import com.hixi_hyi.idumo.core.exec.*;
9  import com.hixi_hyi.idumo.core.provider.*;
10 import com.hixi_hyi.idumo.core.handler.*;
11 import com.hixi_hyi.idumo.core.receiptor.*;
12 public class TodaysWeatherMain extends AbstractConsoleMain {
13   @Override
14   public void init() {
15     setExecutionWithComponent(new TodaysWeatherComponent());
16   }
17   public static void main(String[] args){
18     TodaysWeatherMain main = new TodaysWeatherMain();
19     main.exec();
20   }
21 }
22 class TodaysWeatherComponent extends AbstractExecutionComponent {
23   @Override
24   public void onIdumoMakeFlowChart() throws IdumoException {
25     LivedoorWeatherProvider w1 = new LivedoorWeatherProvider(63);
26     w1.setOption(LivedoorWeatherProvider.Type.WEATHER);
27     add(w1);
28     ConsoleViewReceiptor consoleview = new ConsoleViewReceiptor();
29     add(consoleview);
30     connect(w1, consoleview);
31   }
32   @Override
33   public void onIdumoPrepare() {
34     setLoopCount(1);
35     setSleepTime(1000);
36   }
37 }
```

図 4.2 天気予報アプリケーションのプログラミングコード

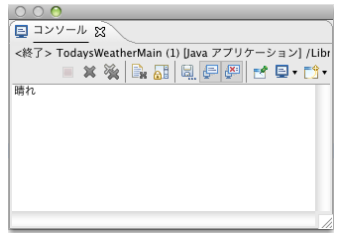

図 4.3 天気予報アプリケーションの実行例

4.3 マッシュアップにより構成されたアプリケーションの再利用

特定のデバイスを対象に作ったアプリケーションを他のデバイスでも使いたいというケースを考える．ケーススタディとして，パソコン上の動作を対象として開発された「データを TCP で送信する」アプリケーションに対して，Android 端末で再利用することを考える．

まず，パソコンで動作するデータを TCP で送信するアプリケーションの説明を行う．このアプリケーションは特定の値を常に送り続ける IDUMOItem（StringProvider）と受け取っ

```
<?xml version='1.0' encoding='UTF-8'?>
<idumo ver='0.1' type='android' name='ReverseGeoLocation' package='com.hixi_hyi.idumo.android.auto.app'>
<items>
<item class='GPSProvider' name='gps1' option='GPSProvider.Type.LATITUDE' init='activity'/>
<item class='GPSProvider' name='gps2' option='GPSProvider.Type.LONGITUDE' init='activity'/>
<item class='ReversedGeocordingHandler' name='rgh'/>
<item class='TextViewReceiptor' name='textview' init='activity'/>
</items>
<connections>
<connect src='gps1' sink='rgh'/>
<connect src='gps2' sink='rgh'/>
<connect src='rgh'  sink='textview'/>
</connections>
<executions>
<loop count='-1'/>
<sleep time='10000'/>
</executions>
</idumo>
```

図 4.4 現在地取得アプリケーションのマッシュアップ情報を記述した XML

たデータを TCP で送信する IDUMOItem（SendTCPReceiptor）によって構成される．

アプリケーションを構築するために，パソコン用に記述された XML を**図 4.7** に示す．StringProvider と SendTCPReceiptor は，節 4.1 で想定した天気情報取得アプリケーションや節 4.2 で想定した現在地取得アプリケーションとは異なり，Android 端末のセンサ情報や特定のデバイスに依存した画面出力などは行わない．今回利用する IDUMOItem はデバイスに依存しないため，これらの IDUMOItem を用いて作成されたマッシュアップ情報は再利用可能である．

これらの IDUMOItem を用いて Android 端末用に記述された XML を**図 4.8** に示す．図 4.7 と図 4.8 の比較を行うと，2 行目に定義してある IDUMO フレームワークのメタ情報部分 `type` 及び `package` 以外の XML 情報が同一であることがわかる．これら XML から自動生成されたプログラミングコードを**図 4.9** と**図 4.10** に示す．図 4.9 の 24 行目から 42 行目までと図 4.10 の 19 行目から 37 行目までの TCPSendComponent クラス部分が同一であることがわかる．また，最終的に生成されたアプリケーションによって送信されたデータを受け取ったアプリケーションのスクリーンショットを**図 4.11** に示す．図 4.11 により Android 端末とパソコン端末から通信が行えていることがわかる．

5. 考　　察

IDUMO フレームワークを実現するに当たっての設計課題は，第 2 節で述べた，(1) 統一的な入出力インタフェースの定義，(2) プログラム実行モデルの差異の吸収，(3) 容易な開発方法の提供，である．この設計課題が，本論文で示した実装で解決されていることをケーススタディの事例を通じて考察する．

ケーススタディ1 では，ネットワークサービスとして提供されている天気予報の情報を取得

```
package com.hixi_hyi.idumo.android.auto.app;
import com.hixi_hyi.idumo.android.*;
import com.hixi_hyi.idumo.android.exec.*;
import com.hixi_hyi.idumo.android.provider.*;
import com.hixi_hyi.idumo.android.handler.*;
import com.hixi_hyi.idumo.android.receiptor.*;
import com.hixi_hyi.idumo.core.*;
import com.hixi_hyi.idumo.core.exec.*;
import com.hixi_hyi.idumo.core.provider.*;
import com.hixi_hyi.idumo.core.handler.*;
import com.hixi_hyi.idumo.core.receiptor.*;
public class ReverseGeoLocationActivity extends AbstractAndroidActivity {
  @Override
  public void init() {
    setExecutionWithComponent(new ReverseGeoLocationComponent());
  }

}
class ReverseGeoLocationComponent
  extends AbstractAndroidExecutionComponent {
  @Override
  public void onIdumoMakeFlowChart() throws IdumoException {
    GPSProvider gps1 = new GPSProvider(activity);
    gps1.setOption(GPSProvider.Type.LATITUDE);
    add(gps1);
    GPSProvider gps2 = new GPSProvider(activity);
    gps2.setOption(GPSProvider.Type.LONGITUDE);
    add(gps2);
    ReversedGeocordingHandler rgh = new ReversedGeocordingHandler();
    add(rgh);
    TextViewReceiptor textview = new TextViewReceiptor(activity);
    add(textview);
    connect(gps1, rgh);
    connect(gps2, rgh);
    connect(rgh, textview);
  }

  @Override
  public void onIdumoPrepare() {
    setLoopCount(-1);
    setSleepTime(10000);
  }
}
```

図 4.5 現在地取得アプリケーションのプログラミングコード

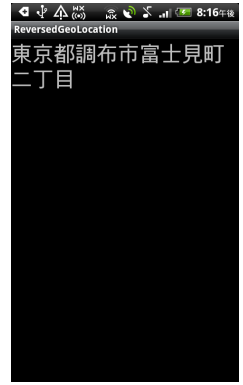

図 4.6 現在地取得アプリケーションの実行例

し，パソコンのコンソール上に文字列として表示するアプリケーションの作成を考えた．ケーススタディ2 では，端末の GPS センサのデータをネットワークサービスを利用し現在地に変換するアプリケーションを作成を考えた．ケーススタディ1 の LivedoorWeatherProvider は，ネットワークサービスに対してリクエストを発行しデータのレスポンスを待つという同期的なデータのアクセス方式を持つ IDUMOItem であり，ケーススタディ2 の GPSProvider は，Android のイベント発生のタイミングで更新されるデータであり，アプリケーションのリクエストとは関係なくデータが生成される非同期なデータのアクセス方式を持つ IDUMOItem である．これらは，それぞれデータに対するアクセス方法が異なる．また，ケーススタディ1 の ConsoleViewReceiptor は，文字列をパソコンのコンソール上に表示する機能を持つ IDUMOItem であり，ケーススタディ2 の TextViewReceiptor は，文字列を Android 端末上に表示する機能を持つ IDUMOItem である．これらは，それぞれデバイスごとに固有な出力をもつ IDUMOItem である．これら異なったデータのアクセス方式や，デバイス特有の機能などをまとめて IDUMOItem として定義できていることから，(1) 統一的な入力イ

```
<?xml version='1.0' encoding='UTF-8'?>
<idumo ver='0.1' type='console' name='TCPSend'
 package='com.hixi_hyi.idumo.console.auto.app'>
<items>
<item class='StringProvider' name='s'
 init='"IDUMO"'/>
<item class='SendTCPReceiptor' name='r'
 init='"192.168.12.2",10000'/>
</items>
<connections>
<connect src='s' sink='r'/>
</connections>
<executions>
<loop count='-1'/>
<sleep time='1000'/>
</executions>
</idumo>
```

図 4.7 パソコンの TCP 通信の
マッシュアップ情報を記述した XML

```
<?xml version='1.0' encoding='UTF-8'?>
<idumo ver='0.1' type='android' name='TCPSend'
 package='com.hixi_hyi.idumo.android.auto.app'>
<items>
<item class='StringProvider' name='s'
 init='"IDUMO"'/>
<item class='SendTCPReceiptor' name='r'
 init='"192.168.12.2",10000'/>
</items>
<connections>
<connect src='s' sink='r'/>
</connections>
<executions>
<loop count='-1'/>
<sleep time='1000'/>
</executions>
</idumo>
```

図 4.8 Android の TCP 通信の
マッシュアップ情報を記述した XML

ンタフェースの定義ができていることがわかる．

ケーススタディ1 ではパソコン上で動く天気予報取得アプリケーションの作成を考えたケーススタディ2 では Android 端末上で動く現在地取得アプリケーションの作成を考えた．これらのケーススタディより，マッシュアップ情報より自動生成されたアプリケーションが異なる実行デバイスにおいても動作していることがわかる．また，ケーススタディ3 ではマッシュアップアプリケーションの再利用を考えた．ケーススタディ3 において，図 4.9 の 22 行目から 41 行目と，図 4.10 の 19 行目から 39 行目の SendTCPComponent(マッシュアップ情報の記述部分) が同一であることがわかる．SendTCPComponent が図 4.9 の 12 行目から 23 行目の SendTCPMain 及び図 4.10 の 12 行目から 18 行目の SendTCPActivity により実行され，図 4.11 により同一の機能を持ったアプリケーションが動いていることが確認できる．このことから，(2) プログラム実行モデルの差異が吸収されていることがわかる．

ケーススタディ 1 では天気予報取得アプリケーションを，ケーススタディ 2 では現在地取得アプリケーションを，ケーススタディ 3 では Android 端末上とパソコン上で動く TCP 通信アプリケーションの作成について考えた．これらのアプリケーションは，XML に記述されたマッシュアップ情報を解析することでプログラミングコードを生成する．また生成されたプログラミングコードをコンパイルし，実行することでアプリケーションを実際に動かしている．このことから，XML の記述のみで実際に動作するアプリケーションの開発が可能であることがわかる．利用者は XML のフォーマットと IDUMOItem それぞれが提供するデータ型さえ知っていれば，プログラミングのような複雑な言語仕様などを覚えることなく，アプリケーションの開発が可能となる．したがって，(3) 容易な開発方法の提供が行える．

しかしながら，利用者が XML のフォーマットを知る必要があるについては，課題として挙げられる．この課題はブロック同士の組み合わせができるグラフィカルインタフェースを用いることで解消され，利用者は XML のフォーマットといった仕様を知ることなく，もっと容易な開発方法の提供を行うことができる．

```
package com.hixi_hyi.idumo.console.auto.app;
import com.hixi_hyi.idumo.console.*;
import com.hixi_hyi.idumo.console.exec.*;
import com.hixi_hyi.idumo.console.provider.*;
import com.hixi_hyi.idumo.console.handler.*;
import com.hixi_hyi.idumo.console.receiptor.*;
import com.hixi_hyi.idumo.core.*;
import com.hixi_hyi.idumo.core.exec.*;
import com.hixi_hyi.idumo.core.provider.*;
import com.hixi_hyi.idumo.core.handler.*;
import com.hixi_hyi.idumo.core.receiptor.*;
public class TCPSendMain
 extends AbstractConsoleMain {
  @Override
  public void init() {
    setExecutionWithComponent
   (new TCPSendComponent());
  }
  public static void main(String[] args){
    TCPSendMain main = new TCPSendMain();
    main.exec();
  }
}
class TCPSendComponent
 extends AbstractExecutionComponent {
  @Override
  public void onIdumoMakeFlowChart()
  throws IdumoException {
    StringProvider s
     = new StringProvider("IDUMO");
    add(s);
    SendTCPReceiptor r
     = new SendTCPReceiptor("192.168.12.2",10000);
    add(r);
    connect(s, r);
  }
  @Override
  public void onIdumoPrepare() {
    setLoopCount(-1);
    setSleepTime(1000);
  }
}
```

図 4.9 パソコンの TCP 通信アプリケーションの
プログラミングコード

```
package com.hixi_hyi.idumo.android.auto.app;
import com.hixi_hyi.idumo.android.*;
import com.hixi_hyi.idumo.android.exec.*;
import com.hixi_hyi.idumo.android.provider.*;
import com.hixi_hyi.idumo.android.handler.*;
import com.hixi_hyi.idumo.android.receiptor.*;
import com.hixi_hyi.idumo.core.*;
import com.hixi_hyi.idumo.core.exec.*;
import com.hixi_hyi.idumo.core.provider.*;
import com.hixi_hyi.idumo.core.handler.*;
import com.hixi_hyi.idumo.core.receiptor.*;
public class TCPSendActivity
 extends AbstractAndroidActivity {
  @Override
  public void init() {
    setExecutionWithComponent(new TCPSendComponent());
  }
}
class TCPSendComponent
 extends AbstractAndroidExecutionComponent {
  @Override
  public void onIdumoMakeFlowChart() throws IdumoException {
    StringProvider s
     = new StringProvider("IDUMO");
    add(s);
    SendTCPReceiptor r
     = new SendTCPReceiptor("192.168.12.2",10000);
    add(r);
    connect(s, r);
  }

  @Override
  public void onIdumoPrepare() {
    setLoopCount(-1);
    setSleepTime(1000);
  }
}
```

図 4.10 Android の TCP 通信アプリケーションの
プログラミングコード

```
% java TCPServer
IPv4:[192.168.12.2] PORT:[10000]
[192.168.12.4] IDUMO
[192.168.12.5] IDUMO
[192.168.12.4] IDUMO
[192.168.12.5] IDUMO
[192.168.12.4] IDUMO
[192.168.12.5] IDUMO
[192.168.12.4] IDUMO
```

図 4.11 TCP 通信アプリケーション実行結果

6. 関 連 技 術

関連技術として，データのマッシュアップ環境を提供する Yahoo!Pipes[4] と，ユーザの簡単なアプリケーション開発環境を提供する Blocco[5]，モジュールのマッシュアップ環境

を提供する Plagger[6] がある．

Yahoo!Pipes はグラフィカルユーザインタフェース上で，モジュールのブロック間を線でつなぐことによりデータのマッシュアップを簡単にすることができる Web アプリケーションである．様々なソースから取得されたデータを，合成，フィルタリングなどの操作を行い，ユーザが求めるデータに変換することが可能である（**図 6.2**）．Yahoo!Pipes はマッシュアップという点で IDUMO と似ているが，対象とするリソースがデータのみであるという点で異なる．

Blocco は AndroidOS 上で動作する，イベントドリブンプログラミングを簡単に行うことが可能な Android アプリケーションである．「○○が発生したら」という予め定められた条件を判定し，その条件が満たされた場合にサービスを実行するといったイベントドリブンなアプリケーションを自ら作ることが可能である（**図 6.1**）．開発者は公開されている Blocco SDK を用いることで，条件やサービス等のモジュールを自ら開発可能である．Blocco はサービスとデバイスのサービス協調ができる点で IDUMO と似ているが，イベントドリブンという制約により，ユーザの実現に対して制約が発生する場合がある．また，対象とするデバイスが AndroidOS のみという点で異なる．

Plagger は Perl モジュールのマッシュアップにより，アプリケーションを開発できるアプリケーションである．モジュールの組み合わせにより様々なアプリケーション開発を可能とする．また，Plagger のモジュールは Perl を用いて作成されていることから，Perl を扱える人であれば自由にモジュールを作成することが可能であり，柔軟性が高い．Plagger は IDUMO と非常に似ているアプリケーションであるが，実行環境に Perl を入れなければいけないこと，また設定に Perl の知識が必要であることから，導入するための敷居が高い．

これらの関連技術に対して，IDUMO フレームワークでは，データやデバイスというリソースを限定せず様々なリソースを包括的に，誰でも簡単に扱うことができるフレームワークを実現している点で新規性と有用性を有する．

7. ま と め

本論文では，情報処理の活用分野の広がりによる多種多様なアプリケーションへのニーズを解消するためのマッシュアップフレームワーク IDUMO の提案を行った．IDUMO フレームワークを実現するにあたって（1）統一的な入出力インタフェースの定義，(2）プログラム実行モデルの差異の吸収，(3）容易な開発方法という課題が存在した．これらの課題を解決するため IDUMO フレームワークでは（1）Sender や Receiver を用いたモジュール間の接続の抽象化，(2）Component と Controller を用いた実行デバイスの抽象化，(3）XML を用いた簡単な記述を用いたプログラミングの設計，実装を行った．ケーススタディを用いた考察では，IDUMO フレームワークの設計により課題が解決されたことを確認し，IDUMO フレームワークの有用性を示した．

今後の課題として，考察であげたグラフィカルインタフェースの実装を行い，より容易な開発方法の提供を行う．また，フレームワークを利用して様々なアプリケーションを開発することで，IDUMO フレームワークの有用性を示す．

図 6.1 Blocco のアプリケーション作成画面

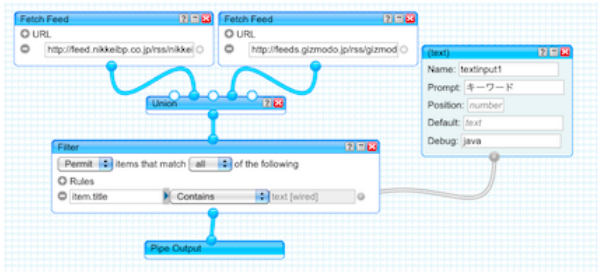

図 6.2 Yahoo!Pipes のマッシュアップ画面

参 考 文 献

1) グルメ・レストランガイド [食べログ] : http://r.tabelog.com/
2) Google マップ - 地図検索 : http://maps.google.co.jp/
3) 簡易逆ジオコーディングサービス / Finds.jp Web サービス : http://www.finds.jp/wsdocs/rgeocode/index.html.ja
4) Pipes: Rewire the web : http://pipes.yahoo.com/pipes/
5) Blocco : http://www.Blocco.jp/
6) Plagger - Trac : http://plagger.org/trac