Babascript: 人とコンピュータの協調による処理実行環境

馬場匠見　　橋本翔　　増井俊之 *

概要.　プログラムに人力処理を簡単に組み込むことのできるシステム「Babascript」を提案する。Babascript は、シンプルな人力処理命令構文と、命令に対する返り値を返せるスマートフォンアプリケーションを組み合わせることで実現する、プログラム上で人力処理を実現するための仕組みである。Babascript では、通常のプログラムにおける関数呼び出しと同じ記法で人への処理を命令することができる。人とコンピュータがプログラム上で協調して動作していくことによって、コンピュータだけでは実現しなかった処理の実現や人の行動そのもののプログラム化が可能となる。具体的な応用例として、「仕事のプログラム化」と「人を使った実世界プログラミング」について示す。

1　はじめに

コンピュータには処理が困難なタスクであっても、人ならば処理できることがある。例えば、その場の雰囲気を数値化・文字列化することはコンピュータには処理は困難だが、人なら処理可能だ。このような、コンピュータだけでは実現が困難な処理を実現するために、人を計算資源として利用する手法はヒューマンコンピュテーション [13] と呼ばれ、様々な研究が行われている。

人の処理能力は有用であり、人力処理をプログラムから扱うことが出来れば、コンピュータだけでは実現しなかった処理を実現することができる。しかし、既存のプログラミング言語の処理実行対象はコンピュータであるため、人に対する処理命令構文は含まれていない。プログラムから人力処理を利用できるようにする技術は、クラウドソーシング等の分野において研究が進んでいる ([5], [8], [11] 等)。クラウドソーシングは、インターネットを経由して不特定多数の群衆にタスク処理を依頼する手法だ [10]。

プログラムから人力処理を実行するためには、既存の手法ではクラウドソーシングプラットフォームの利用が必要となる。そのため、例えば、家族や職場の人、研究室のメンバーのような、特定の人物をプログラムに組み込み、処理実行命令を送るといったことは困難だ。また、処理命令の記法も複雑である。既存の研究事例では、クラウドソーシングプラットフォームへのアクセスキーや複雑な設定を記述したり、SQL 文のようなものをプログラム上に直接記述するといった記法のものが存在する。これらの事例では、記法が複雑化することが多く、シンプルな記述で実装することはできない。

本論文では、Babascript プログラミング環境について述べる。Babascript 環境は、関数呼び出しによって人に処理命令の通知ができる Babascript(プログラム例 図 1) と、Babascript からの命令を受け取り、処理結果を入力することのできるスマートフォン向け web アプリケーションの Babascript Client(図 2) から構成される。Babascript 環境では、特定の人物を実行対象として指定するため、不特定多数の人を扱うクラウドソーシングプラットフォームを利用しない。Babascript 環境によって、既存のプログラミング言語上で人への処理命令を、通常のプログラムにおける関数呼び出しとほぼ同じ記述で実現できる。

本論文では Babascript 環境の応用の例として、「仕事のプログラム化」と「人を使った実世界プログラミング」について述べる。

```
1  #!/usr/bin/env baba
2  // ライブラリ読み込み
3  Babascript = require("babascript");
4  // 処理命令構文を持ったオブジェクトを宣言
5  baba = new Babascript("baba");
6
7  // 「雨はふりそうですか」が命令内容
8  baba.雨はふりそうですか({format: 'boolean'}, function(result){
9    // 人から処理命令の結果が返ってくると、以下を実行する
10   // result.value に返り値が代入される
11   if(result.value === true){
12     // ....
13   }else{
14     // ...
15   }
16 });
```

図 1. Babascript プログラム例

2　BabaScript プログラミング環境

BabaScript プログラミング環境は、人に対する処理命令を記述するための Babascript と、処理命令を受け取り、返り値を返すことのできる Babascript Client から成り立つ。

* 慶應義塾大学大学院政策・メディア研究科, 慶應義塾大学環境情報学部

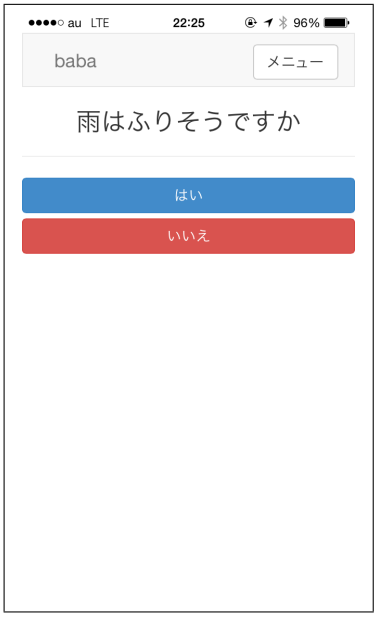

図 2. Babascript Client インタフェース

2.1 Babascript

Babascript は、Javascript(V8 JavaScript Engine[1]) 上に実装された、人に対して処理命令を送る機能に特化した DSL(Domain Specific Language) である。図 1 のようなプログラムによって、人に処理命令を送ることができる。Babascript は、通常の関数を実行しているのと同じようなシンプルな記法で、人力処理のための記述を実現する。

2.1.1 基本仕様

Babascript では、定義されていないメソッドが実行されると、エラーを返さずに、人への命令構文として解釈する。定義されていないメソッドは全て人への命令として実行され、人に通知される。例えば、「toString」や「call」等のメソッドは、javascript においてはほぼすべてのオブジェクトが持つメソッドだ。一方で、「clean_up_your_room」や「bake_bread」のようなメソッドは定義しない限りは存在しないメソッドである。Babascript は、この定義されていないメソッドをエラーとして評価せず、人への命令構文として評価する。

人への命令構文として評価されたメソッドは、そのメソッド名と引数を元にしたタスク情報を生成し、Babascript Client へと送信する。この際、メソッド名部分がユーザに命令として提示される文となる。メソッド名が自由に設定できるため、内容は命令文ではなく、質問のようなものもあり得るが、本研究では文章としては命令ではないものも統一して命令と呼ぶ。人への命令構文の第一引数にはオプション情報としてオブジェクトを、第二引数には人力処理の実行後に実行するコールバック関数を指定する。この際、コールバック関数が省略されると、タスク情報のみを Babascript Client に通知する。

命令の実行結果を Babascript Client から受け取ると、人への命令構文の実行を終了し、指定したコールバック関数が実行される。コールバック関数実行時に引数として、処理に成功していれば処理実行者が入力した結果が、処理失敗であればエラー文が代入される。人への処理命令は、通常の関数呼び出しとほぼ同じ記法で実行することができ、新たに記法を学習するといった必要がない。

人オブジェクトはインスタンス生成時に id を指定する必要がある。人への命令構文は、この id を元に命令配信先を決定する。例えば、id=baba に命令を送りたければ、人オブジェクト宣言時の第一引数には baba という文字列を指定する。指定した id に命令が配信されるため、Babascript Client 側でも同じ id を指定する必要がある。この際、指定した id に接続したクライアントが複数あった場合、後述する broadcast オプションを指定していないと、各クライアントに順番に命令が配信される。

2.1.2 オプション情報

人への命令構文の第一引数には、Babascript Client に伝送したい情報の入ったオブジェクトをオプション情報として与える。例えば、返り値のフォーマットを指定するオプションが考えられる。現在対応しているフォーマットは Boolean,String,Int,List の 4 つが存在する。例えば、Boolean を指定すれば、処理実行者には true と false のどちらかを選択するインタフェースが提示される。

List を返り値のフォーマットとして指定する場合のプログラム例は図 3 の通りである。フォーマット情報以外にも、選択肢の候補情報などをオプション情報として付加する。

```
#!/usr/bin/env baba
// ライブラリ読み込み
var Babascript = require('babascript');
// 人への命令構文を持つオブジェクトを宣言
var baba = new Babascript("baba");

// formatにList指定する.
// list配列の中身が選択肢として提示される.
option = {format: 'list', list: ['良い', '悪い', '普通', 'わからない']}
baba.会場の雰囲気はどうですか(option, function(result){
  // result.value に返り値が入る
  // 返り値の値に応じて処理を分岐させる
  if(result.value ===  '良い'){
    // ....
  }else if(result.value === '悪い'){
    // ...
  }else if(result.value === '普通'){
    // ...
  }else{
    // ...
  }
});
```

図 3. オプション情報例 (List)

特別なオプション情報として、broadcast オプショ

[1] https://code.google.com/p/v8/

ンが存在する。broadcast オプションを付与することで、指定した ID に接続する全てのクライアントへと命令を配信し、指定した数の返り値を得られるまで待機し、コールバック関数を実行する。broadcast オプションは、個人ではなく、特定のグループへの命令を想定した実装となっている。例えば、所属研究室のメンバーに処理命令を送り、必要な数の返り値を得たら実行を終了する、といった用途に使うことが出来る。

2.2 Babasciript Client

Babascript Client は、Babascript からの命令を受け取り、値を返すためのスマートフォン向け web アプリケーションである。Babascript との通信を担うサービス部分と、返り値の入力等を担うインタフェース部分から構成される。

2.2.1 サービス

サービス部分では、主に Babascript との通信をおこなっている。命令受信のイベントに対してコールバック関数を指定することで、処理命令を受け取ることができる。基本的な利用方法は、図 4 に示す。

クライアントオブジェクトの宣言時、id を指定することによって、指定 id に対して処理命令が発行された際にタスクを受信することができる。クライアントオブジェクトには returnValue というメソッドが定義されており、このメソッドを利用することで返り値を命令発行元に返すことができる。

サービス部分は、インタフェース部分とは分離した実装となっている。本論文においては後述のスマートフォン向け Web アプリケーションとしてインタフェース部分を実装したが、サービス部分は Web アプリケーション以外にも適応可能である。

2.2.2 Web アプリケーション

送られてきた命令内容をユーザに提示するためのインタフェースとして、スマートフォン向け Web アプリケーションを実装した。図 2 のような画面を持つ。

Babascript からの命令を受け取ると、インタフェースが変化し、命令内容と指定されたフォーマットに沿ったフォームをユーザに提示する。例えば、フォーマットに Boolean を指定していた場合、ユーザには true ボタンと false ボタンが提示され、どちらかを押すと、その結果が返り値としてプログラムに返される。Boolean 以外には、String、Number、List に対応している。String と Number であれば、文字・数字の入力フォームと投稿ボタンが提示され、投稿ボタンを押した際にフォームに入力されていた内容が返り値としてプログラムに返される。List であれば、選択フォームと投稿ボタンが提示され、同様に、選択していた要素が返り値としてプログラム

```
// Babascript Client のライブラリ読み込み
var Client = require('babascript-client');
// clientオブジェクトの宣言
var client = new Client("takumibaba");

client.on("get_task", function(task){
  // ユーザに命令内容を示す.
  // task.key に命令本文が入る
  $("#task-body").html(task.key);
  var format = task.format;
  if(format === 'boolean'){
    // formatがbooleanの場合.
    $("#true-button").click(function(e){
      // trueボタンが押されたら返り値trueを返す
      client.returnValue(true);
    });
    $("#false-button").click(function(e){
      // falseボタンが押されたら返り値falseを返す
      client.returnValue(false);
    });
  }else if(format === 'string'){
    //  formatがstringの場合
    //  <input>要素の値を返り値として返す
    $("#submit-button").click(function(){
      var value = $("#string-input-form").val();
      client.returnValue(value);
    });
  }else if(format === 'number'){
    // formatがnumberの場合
    //  <input>要素の値を返り値として返す
    $("#submit-button").click(function(){
      var value = $("#number-input-form").val();
      client.returnValue(value);
    });
  }else if(format === 'list'){
    // formatがlistの場合
    for(var i=0;i<list.length-1; i++){
      // <select>要素に返り値候補を追加する
      var option = "<option>"+list[i]+"</option>";
      $("#list-input-select").append();
    }
    // <select>要素で洗濯されている値を返す
    $("#submit-button").click(function(){
      var value = $("#list-input-select").val();
      client.returnValue(value);
    });
  }

});
```

図 4. Babascript Client サービス部分のプログラム例

に返される。フォーマットが指定されていない場合は、Boolean のインタフェースを提示する。

3 実装

上記システムは全て JavaScript で実装した。Babascript は JavaScript のサーバサイド実行環境である Node.js 上で動作する。また、Babascript Client は Web ブラウザ上及び Node.js 上で動作する。

全体図は図 5 のとおりだ。

Babascript と Babascript Client 間の通信を実現するために、Node-Linda[2] を利用した。Node-Linda は、Linda[9] モデルを拡張し、socket.io[2] 上に実装したものである。

[2] http://socket.io

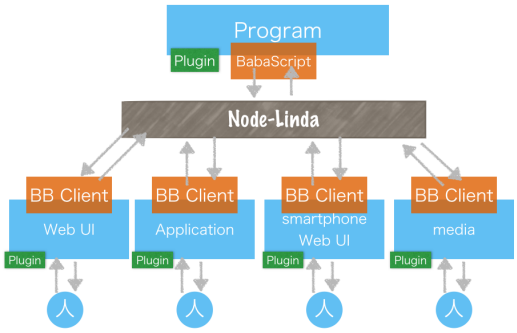

図 5. システム全体図

3.1 処理の流れ

Babascript は、以下のような流れで人への命令構文が実行される。

1. 人への命令構文を実行する
2. 命令が Node-Linda サーバを経由してクライアントへと配信される
3. 命令を受け取ったクライアントがユーザに処理を促す
4. 命令実行者が、処理結果を入力する
5. Node-Linda サーバを経由して実行元プログラムに入力された処理結果が送信される
6. プログラム側で指定されたコールバック関数が実行され、処理が継続される

3.2 タスク情報の構成

人への処理命令構文の実行によってタスク情報が生成される。このタスク情報は、図 6 のような json オブジェクトとなる。

```
task = {
  "name": "baba", // 命令配信先ID
  "type": "eval", // 配信タイプ
  "key": "部屋の温度は適切ですか", // 処理命令の内容
  "cid": "1409227153_0.869041177676", //タスクID
  "option": { // オプション情報
    "format": "boolean",
  }
};
```

図 6. タスクの JSON オブジェクト

4 応用例

以下のような応用が考えられる。

4.1 仕事のプログラム化

人への命令がプログラムとして記述可能になることで、仕事をプログラム化することができると考える。人の仕事は、例えば、マニュアルのような形で記述されることが多い。マニュアルはどのように行動すべきかが構造的に記述された手順書であり、「A という条件の時には B の処理を実行する」といったことが文章として記述されていることが多い。コンピュータにとっての手順書であるプログラムとは類似点多いと言え、プログラムに変換可能であると考える。

プログラム化して実行できるようにすることで、仕事における状態をプログラムで管理することができる。状態を可能な限りプログラムで管理し、経験や知識によって可能な条件判断などもプログラム化することができれば、プログラムに指示されたことのみを実行するだけである程度の仕事を実行できると考える。指示されたことのみを実行するだけでも良いということは、仕事の引き継ぎなどが最低限となり、人の代替をも容易にする可能性がある。プログラム経由にすることで、詳細な実行ログや実行状況を把握することも可能である。仕事の実行量の定量化や、状況監視、状況の可視化などに応用可能であると考える。

また、人同士がコミュニケーションを取ることなく、複数人を協調させるといったことも可能となる。通常、複数人を協調させるためには、人同士が相談したり、上位の意思決定者が必要となる。しかし、コミュニケーションはコストのかかるものであり、適切に行われない場合、問題が生じることもある。意思決定をプログラムに委ねることは、複数人を効率よく協調させることに繋がるとも考えられる。

4.2 人を使った実世界プログラミング

人をセンサーやアクチュエータとして利用することで、人を使って実世界を操作することが可能となる。現在のセンサーやアクチュエータでは、実世界に干渉するのには限界がある。例えば、その場の雰囲気を数値化したり文字列化するといった、コンテキスト情報を伴ったセンシングは困難である。しかし、人力処理を組み込むことのできる本提案手法ならば、コンテキスト情報を伴うようなセンシングであっても、実現可能である。また、人を汎用的に利用可能なアクチュエータとして利用することもできる。人をアクチュエータとして利用する、Human as Actuator と言えるような事例は少なく、汎用的に使える事例はない。両者を組み合わせることによって、既存の仕組みでは困難であったより柔軟な実世界へと干渉可能なプログラミングが可能となると考えられる。

また、本提案手法では通常のプログラムと人力処理のプログラムを混合させることができるため、条件次第で人とコンピュータのどちらに処理させるかをプログラムが判断し、適切な方に処理させるということも可能である。

5　考察

本研究について、以下のように考察する。

5.1　処理に対する人の積極的な貢献

Babascript によって、人に対してもプログラムから処理実行命令を送れるようになった。今まではコンピュータにのみ実行命令を送れたため、処理に対して明示的に人が貢献することはあまりなかった。意思決定などをコンピュータに委ね、人がその決定に従う動くことで幸せになれる

5.2

Babascript によって、コンピュータだけに仕事させるのではなく、人もプログラムによって仕事させられるようになった→人がコンピュータではできない処理について、積極的カバーしていく→コンピュータだけだった時よりも、色んな処理が可能となる→そこで処理の実現を諦めるのではなく、コンピュータが得意なことはコンピュータが、人が得意なことは人がやるようにすれば良い。人が積極的に処理に貢献し、人とコンピュータが相互に補完しあって処理を実行していくことで、全体的な処理の正確性の向上等に寄与する可能性も存在する。それはまた、人にとってもメリットになりうるのではないかと考える。

5.3　処理単位としての人

本研究では、人は処理を命令され実行する存在となり、プログラム上においてコンピュータやセンサーと同等となる。こういったことに対して心理的な拒否感を覚えることも考えられる。しかし、一処理単位として扱うことは、大きなメリットでもある。命令を実行するだけのノードであるということは、ただ処理内容を実行することにのみ集中すれば良いということだ。ただやるべきことだけが提示され、その通りに動けば良いということは、深く考える必要がなく、楽であるということが考えられる。もちろん、全ての処理がただ提示する通りに動けば良いものではないと考えられるが、作業を楽にするための一つのアイデアであると言える。

5.4　タスク実行の遅延と実行保障性

Babascript による人への命令構文が実行されても、命令を受け取った人がすぐに命令に対する処理を実行し値を返すことを完全に保証することはできない。タスク受信端末を見ていない、受信しても実行できないといった状況の場合、すぐに値を返すことはできない。Babascript による人への命令構文は全て非同期実行する実装となっているため、人力処理の待ち時間に他の処理を実行させておくことが可能であるが、人力処理の実行待ちが全体の処理を遅延させる可能性は存在する。

また、労働関係にあるなど、強制力がある場合命令は実行されると考えられるが、強制力がない場合はそもそもタスクを無視するといったことも考えられる。タスク実行に強制力がない場合は、金銭などのインセンティブを与えるといった手段によって、実行保障性を確保するといったことが考えられる。

5.5　例外処理

Babascript において、命令が想定する状況と現実の状況との乖離によって適切な返り値を選択・記述ができなくなるといった可能性がある。これは、現実が刻々と変化していることなどから、完全に避ける事の出来ない問題であると考える。この際、無理やり値を返すといった処理をしてしまうと、本来の状態とは違った判断がなされてしまう危険性がある。命令文とは明らかに現実が異なっている場合などは、タスク実行者から例外としてプログラムに通知出来るような仕組みの実装によって、問題の解決へと繋げられると考える。

5.6　命令内容の粒度

Babascript では、人への命令構文の記述には制限がない。そのため、命令の抽象度はプログラマに依存する。抽象度が高すぎるメソッド名にしてしまうと、タスク実行者にとって理解しづらい文面となり得る。その結果、想定外の処理が実行され、意図しない結果を招く恐れがある。

具体的過ぎる命令は、全体の処理内容にもよるが、プログラム自体が冗長となり得る。プログラムとタスク実行者の間のやりとりが増え、通信や待機時間などがボトルネックとなる可能性がある。また、タスク実行者にとっても、やりとりが増えることで負担増になると考えられる。処理ごとに異なると考えられるが、命令文は適切な抽象度に設計しなくてはならない。

5.7　複数命令の同時実行

複数のプログラムから同時に一人のタスク実行者へとタスクが配信される可能性がある。この際、異なるコンテキストにある命令が交互に配信され、タスク実行に大きな障害をもたらす可能性がある。例えば、料理プログラムと掃除プログラムが同時に実行された場合、鍋で煮ている途中に「洗剤を投入しろ」などといった命令が配信されることが考えられる。

この問題は、命令実行者は一つのプログラムからのみ連続して命令を受信できるような仕組みを用意することによって、解決可能であると考えられる。また、応用アプリケーションでの実装になるが、コンテキストを明示し、どの処理系におけるタスクなのかを命令実行者に示すといった手段によっても解決可能である。

6 関連研究

計算機では処理が難しいようなタスクを解決するために、人を計算資源として利用する手法はヒューマンコンピュテーション [13] と呼ばれ、様々な研究が行われている。インターネットを介して不特定多数の群衆にタスクを実行させるクラウドソーシングと組み合わせた研究事例も多く存在する。クラウドソーシングのプラットフォームとしては、Amazon Mechanical Turk[1] が存在する。Barowy らは、CrowdProgramming という概念を提唱し、プログラミング言語内においてクラウドソーシングによる計算とコンピュータによる計算の統合を実現した [5]。Ahmad らは、Jabberwocky[3] という、ソーシャルコンピューティングのためのプログラミング環境を提案している。Jabberwocky は、様々なクラウドソーシングプラットフォームを統合して管理できる Dormouse と、Dormous とのインタラクションに特化した Dog,MapReduce[3] を人力処理に適応した ManReduce から構成される。Franklin らは、機械だけでは答えられないような DB へのクエリに対する応答を、クラウドソーシングを用いることで返答可能にする CrowdDB を提案している [8]。Morishima らは、人をデータソースとしてプログラムの中で利用する手法を提案している [12]。これらの研究は、人を計算資源やデータソースとしてシステムに組み込むことを想定しているが、本研究では、計算資源やデータソースに限らず、実世界への干渉等も考慮に入れている。また、本研究では実世界環境の操作を実現するために、家族や職場の人のような、特定の人物を対象としており、クラウドソーシング利用を前提とした既存研究とは異なる。

Dog[4] は Web アプリケーションにおけるユーザとのインタラクションの記述に特化したプログラミング言語として提案されている。本研究も、人とのインタラクションの記述に特化したものであり、類似している。Dog は Web アプリケーションの実装時によくあるユーザとのインタラクションの記述に特化したものであるが、Babascript はユーザをプログラムの処理ノードとして扱うことを想定している。

ユビキタスコンピューティングの研究分野においては、Human as Sensor という、人や人が持つスマートフォンをセンサーとして利用する概念も提唱されている。PRISM は、スマートフォンを利用したセンシングプラットフォームだ [7]。Liu らは、ソーシャルメディア上の人をセンサーとして扱った Q&A サービス MoboQ を提案し、その検証を行った。Human as Sensor に類する研究では、人をセンサーとして扱うことを対象としているが、本研究ではセンサーのみを対象としていない。

Cheng らは、モーションプラットフォームにおけるモーターやメカニカル機構の代替として人を利用した Haptic Turk を提案している [6]。Haptic Turk はゲームでの利用に特化したものである。本研究は、用途を限らない汎用的な仕組みとなっている。

加藤らは、人とロボット間でのタスク共有システム Sharedo を提案した [14]。人とロボットのタスク実行における協調についても述べられており、本研究で指摘している「処理実行において人とコンピュータは協調すべき」という考えと類似している。

[3] http://research.google.com/archive/mapreduce.html

7 おわりに

本論文では、プログラムに人への処理命令を組み込むことのできるプログラミング環境 Babascript を提案した。既存の仕組みでは、プログラムから人に処理命令を送るためには、クラウドソーシングプラットフォームが必要であったり、複雑な構文の実行が必要である。本論文の手法では、身近な特定の人物を命令実行対象とし、シンプルな記法での人への処理命令を実現した。

また、Babascript によって実現が見込まれる応用例について言及した。今後は、考察で述べた Babascript の問題点についてシステムの改良などを行う。

参考文献

[1] Amazon Mechanical Turk. https://www.mturk.com/mturk/.

[2] Linda implementation on Socket.IO. https://github.com/node-linda/linda.

[3] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. The Jabberwocky Programming Environment for Structured Social Computing. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pp. 53–64, New York, NY, USA, 2011. ACM.

[4] S. Ahmad and S. Kamvar. The Dog Programming Language. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pp. 463–472, New York, NY, USA, 2013. ACM.

[5] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor. AutoMan: A Platform for Integrating Human-based and Digital Computation. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pp. 639–654, New York, NY, USA, 2012. ACM.

[6] L.-P. Cheng, P. Lühne, P. Lopes, C. Sterz, and P. Baudisch. Haptic Turk: A Motion Platform Based on People. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pp. 3463–3472, New York, NY, USA, 2014. ACM.

[7] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma. PRISM: Platform for Remote Sensing Using Smartphones. In *Proceedings of the 8th International Conference on Mo-*

bile Systems, Applications, and Services, MobiSys '10, pp. 63–76, New York, NY, USA, 2010. ACM.

[8] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pp. 61–72, New York, NY, USA, 2011. ACM.

[9] D. Gelernter and N. Carriero. Coordination Languages and Their Significance. *Commun. ACM*, 35(2):97–107, Feb. 1992.

[10] J. Howe. The Rise of Crowdsourcing. *Wired Magazine*, 14(6), 06.

[11] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut. CrowdForge: Crowdsourcing Complex Work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pp. 43–52, New York, NY, USA, 2011. ACM.

[12] A. Morishima, N. Shinagawa, T. Mitsuishi, H. Aoki, and S. Fukusumi. CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing. *Proc. VLDB Endow.*, 5(12):1918–1921, Aug. 2012.

[13] L. Von Ahn. *Human Computation*. PhD thesis, Pittsburgh, PA, USA, 2005. AAI3205378.

[14] 淳 加藤, 大介 坂本, 健夫 五十嵐. Sharedo: To-do リストによる人-ロボット間のタスク共有. WISS '11: 第 19 回インタラクティブシステムとソフトウェアに関するワークショップ, pp. 102–107, 2011.