Babascript:

Babascript:

匿名で査読を行うため著者名なし *

Summary.

Babascript という、プログラムに人力処理を簡単に組み込むことのできる手法を提案する。現状のプログラムでは、コンピュータを制御することはできるが、人は制御できない。本論文の提案手法では、通常のプログラムで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。プログラムの中に人という要素を組み込むことによって、その場の雰囲気の数値化・可視化といったコンピュータだけでは実現出来なかったような処理を実現させることができる。

1 はじめに

人はコンピュータには処理が困難なタスクであっても、処理することができる。例えば、その場の雰囲気がどのようなものかを数値化・文字列化することはコンピュータには難しいが、人ならば実行可能だ。このような、計算機では処理が難しいよう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 [9] と呼ばれ、様々な研究が行われている。

人の処理能力をプログラムから扱うことが出来れば、プログラムが処理可能な領域を広げることができ、より便利となる。しかし、既存のプログラム言語には、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、主にクラウドソーシング等の分野における研究開発 ([3], [6], [7] 等) が主である。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であり、人の能力のうち活用可能なものは計算資源としての機能のみとなっている。また、ただ処理を記述したいだけなのに、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等、複雑な設定をプログラム内に記述しなくてはならなかったり、SQL 文のようなものをプログラム上に直接記述する必要がある。

既存の仕組みでは、プログラムから人力処理を実行するためには、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、家族や職場の人、研究室の人たちに対してプログラムから命令を送るといったことは困難である。また、処理命令のための記法も複雑であり、シンプルな記法で、人の能力を最大限プログラム上で利用できるような枠組みが必要だ。

本論文では、Babascript を提案する。Babascript プログラミング環境は、関数呼び出しによって人に処理命令の通知ができる DSL Babascrtipt(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 Web アプリケーション Babascript Client(図 2) から構成される。Babascript は、クラウドソーシングプラットフォームを利用しない。プログラム上で、命令配信先を指定するだけで、命令を配信することができる。

人力処理実行のためには、Babascript Client をスマートフォンにインストールし、処理実行者に所持させておく必要がある。Babascript による人力処理が実行されると、Babascript Client に命令内容が通知され、処理実行者は命令に従い、処理を実行し、結果を Babascript Client に入力する。

Babascript を利用することで、通常のプログラムと人への処理命令をほぼ同じ記述で実現することができ、専用の記法の知識もほぼ必要ない。非常にシンプルな記法で人の力を借りた処理の記述が可能となる。

```
1  BabaScript = require("babascript");
2  baba = new BabaScript("takumibaba");
3  baba.書類整理をする(function(tesult){
4    // 返り値を得た後の処理を記述する
5  });
```

図 1. Babascript プログラム例

2 BabaScript

2.1 基本仕様

Babascript は、関数呼び出しによって人に処理命令を送れる DSL 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。

命令の実行結果を BabascriptClient から受け取ると、関数の実行を終了し、指定したコールバック関数が実行される。人への処理命令は、通常の関数呼

* 匿名で査読を行うため所属名なし

図 2. Babascript Client インタフェース

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。

人オブジェクトは宣言時に ID を指定することによって、誰に対して命令を配信するかを決定する。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必要がある。また、クライアント側でも同様に、id:baba として宣言する必要がある。

この際、指定した ID を監視するクライアントが複数人だった場合は、後述する broadcast オプションをつかってない場合は、命令を実行していないクライアントへと順番に配信される。

人への処理命令構文の第一引数にはクライアント側に送信するオプション情報を指定し、第二引数には返り値が得られた際に実行するコールバック関数を指定する。コールバック関数実行時には、その引数に返り値と命令情報、実行者情報を格納したオブジェクトが与えられる。

2.2 オプション情報の付加

人への処理命令構文の第一引数に与えるオプション情報には、返り値の型指定などの情報が考えられる。例えば Boolean 型を指定する場合には図 3 のようなプログラムが考えられる。他にも、指定したリストの中から値を選択してほしい、といった命令の場合は、リスト情報を格納した配列を指定する。このオプション情報の処理は基本機能には含まれていないため、ライブラリを利用して実装する応用アプリケーションごとに対応する必要がある。

```
1  BabaScript = require("babascript");
2  baba = new BabaScript("takumibaba");
3  baba.書類整理をする(function(tesult){
4    // 返り値を得た後の処理を記述する
5  });
```

図 3. オプション情報

特別なオプション情報として、broadcast オプションが存在する。broadcast 機能は、指定した ID を監視する全てのクライアントへとタスクを配信し、指定した数の返り値を得られると処理を終了し、コールバック関数を実行するといったものだ。

2.3 サンプルコード

以下にサンプルコードを示す。

```
MEMBER = 3;

members.参加可能な日程を選んでください({
  broadcast: MEMBER
}, function(results) {
  var b, i, work, worker, workers, _k, _ref;
  workers = _.pluck(results, "worker");
  b = [];
  work = [];
  for (i = _k = 0, _ref = workers.length - 1; 0 <= _ref ? _k <= _ref : _k >= _ref; i = 0 <= _ref ? ++_k :
    worker = workers[i];
    b.push((function(_this) {
      return function(callback) {
        var a, date, _l, _len1, _results;
        a = [];
        _results = [];
        for (_l = 0, _len1 = list.length; _l < _len1; _l++) {
          date = list[_l];
          _results.push(worker.参加可能ですか({
            description: date
          }, function(result) {
            a.push(result.value);
            if (result.task.description === "金曜6限") {
              return callback(null, a);
            }
          }));
        }
        return _results;
      };
    })(this));
  }
  return async.parallel(b, function(err, result) {
    var a, c, funcs, k, len, m, _l, _m, _ref1;
    len = result[0].length - 1;
    b = {};
    for (i = _l = 0; 0 <= len ? _l <= len : _l >= len; i = 0 <= len ? ++_l : --_l) {
      a = [];
      for (k = _m = 0, _ref1 = result.length - 1; 0 <= _ref1 ? _m <= _ref1 : _m >= _ref1; k = 0 <= _ref1 ?
        a.push(result[k][i]);
      }
      c = _.reject(a, function(bool) {
        return bool === false;
      });
      b[list[i]] = c.length;
    }
    m = _.max(b, function(n) {
      return n.val;
    });
    if (m[0].length >= MEMBER) {
      return members.以下の日程から開催日時を選んでください({
        description: m
      }, function(result) {
        return process.exit();
      });
    } else {
      funcs = [];
      funcs.push((function(_this) {
        return function(args) {
          var _n, _ref2;
          if (args.length === 1) {
            i = 1;
          }
          for (k = _n = 0, _ref2 = MEMBER - 1; 0 <= _ref2 ? _n <= _ref2 : _n >= _ref2; k = 0 <= _ref2 ? ++
            if (!m[i][k]) {
              members.get(k).以下の日程になりそうですが、調整できませんか({
                description: list[i]
              });
```

図 4. サンプルコード

3 Babasciript Client

Babascript Client は、Babascript からの命令を受け取り、値を返すための Web アプリケーションである。命令を受け取る通信を行うためのライブラリ部と、返り値の入力を受け付ける UI 部で構成される。

3.1 ライブラリ

Babascript Client は、Babascript からの命令受信と処理結果の送信機能を実現するクライアントライブラリだ。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 5 に示す。

```
Client = require("babascript-client");
client = new Client("takumibaba");
client.on("get_task", function(task){
  // プログラムからタスクを受け取った時の
  // 挙動を記述する
});
```

図 5. クライアントライブラリの基本機能

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、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。また、クライアントオブジェクトが持つ returnValue 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。

クライアントライブラリは、UI と完全に分離した実装となっており、かつ利用法もシンプルだ。後述の Web アプリケーションだけでなく、様々なアプリケーションに組み込むことができる。

3.2 Web アプリケーション

Babascript Client が得たタスクをユーザに提示するためのインタフェースとして、Web アプリケーションを実装した。命令において型指定をしておくことによって、型に合った返り値を選択できるよう、ユーザに提示する UI を変化させ、例えば、Boolean 型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。

スマートフォンに最適化した Web アプリケーションとして実装しており、様々な場面において利用可能だ。実世界におけるタスクを処理しながらでも十分に利用可能である。図 2 のような画面を持つ。

4 実装

上記システムは全て Javascript で実装した。Babascript は Node.js 上で動作し、Babascript Client は Web ブラウザ上で動作する。全体図は図 6 のとおりだ。

Babascript と BabascriptClient 間の通信を実現するために、node-linda[2] を利用した。

4.1 処理の流れ

Babascript は、以下のようなフロー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

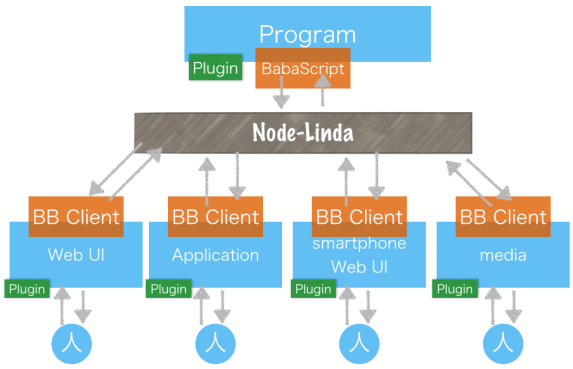

図 6. システム全体図

2. 命令構文の内容に基づき、タスクが生成される
3. タスクが Node-Linda サーバを経由してクライアントへと配信される
4. タスクを受け取ったクライアントがユーザに処理を促す
5. タスク実行者が、処理結果を入力する
6. 処理結果を元に返り値データが生成される
7. Node-Linda サーバを経由して実行元プログラムに返り値が送信される
8. 指定されたコールバック関数が実行され、処理が続く

4.2 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 7 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 7. タスクの JSON オブジェクト

4.3 実行結果の待ち方

人への命令構文に対する返り値が、適切に返り値として戻されるようにするために、命令ごとにユニークな ID を生成している。ユニーク ID は、人への命令構文実行時の UNIX time と、ランダムな数値を結合した文字列である。図 7 における json オブジェクトの'cid' の項目に示されているものだ。このユニーク ID は人への命令構文実行時に生成され、ユニーク ID をこのユニーク ID を伴った返り値が

戻ってくるか、指定したタイムアウト時間に到達するか、タスクキャンセルが発生するまで、

5 応用例

以下のような応用が考えられる。

5.1 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、人の仕事をプログラム化することができる。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。マニュアルで全て記述できるような場合は、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く、その内容の多くはプログラムのようなものであるため、プログラム化は可能であると言える。

プログラム化し、実行することによって、状態をプログラムによって管理することが可能となる。つまり、条件判断や記憶が必要なことは出来るだけコンピュータ側に処理させる、といったことが可能になる。人は細かい条件などを覚えておいたり、自分で判断することなく、ただ指示されたことのみを実行することで、仕事を終わらせることができるようになる。ただ指示されたことのみを実行するだけで良いならば、仕事の引き継ぎなども必要なくなり、人の代替も容易となる。

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。

全ての仕事を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できる。仕事の実行量の定量化や、状況監視、状況の可視化などに活かすことができる。

5.2 人力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、人によって実世界を操作する実世界プログラミングが可能となる。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。

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。

6 考察

Babascript 環境について、以下のように考察する。

6.1 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ノードとなり、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、一処理ノードとして扱うことによって、大きなメリットもある。人とコンピュータ、処理に応じて適切なほうに実行させることができれば、人は人にしかできないことや人がやるべき処理にのみ集中するといったことも実現できる。また、ただやるべきことのみを命令されて動くということは、自分で深く考えて行動する必要がないということだ。とても楽なことでもある。

6.2 タスク実行の遅延と実行保障性

Babascript によってタスク実行を依頼しても、人がすぐにタスク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。この際、Babascript による処理が原因で全体の処理が遅延する可能性がある。

また、労働関係にあるなど、タスク実行に強制力がある場合は、タスク実行が確実に行わ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受信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。

6.3 例外処理

Babascript 環境において、命令文と現実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。

6.4 命令内容の粒度

Babascript では、タスクの命令文の記述には制限がないため、自由となっている。抽象度が高すぎる命令は、あいまいな表記となり、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。抽象度が低すぎる命令は、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。

6.5 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、鍋で煮ている途中で「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、全ての Babascript プログラム中において、一人のタスク実行者は一つのプログラムからのみ、連続してタスク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タスク実行者に示すといった手段によっても解決可能である。

7 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 [**?**] 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 [3]。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 [6]。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 [8]。これらの研究では、クラウドソーシングを利用した問題解決手法の提案をしている。本研究は、不特定多数の群衆をプログラムするためのものではなく、実世界の操作なども含むため、家族や職場の人のような特定可能な人物を対象としている。また、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前提としているが、本研究は、計算資源やデータソースに限らず、実世界への干渉等も対象としている。本研究はより汎用的な枠組みであると言える。

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 [5]。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 [4]。Haptic Turk はゲームでの利用に特化したものだ。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 [10]。人とロボットのタスク実行における協調は、本研究の主眼である「何かの処理を実現するとき、人とコンピュータは相補的に動作するべき」という考えと大きく類似している。

8 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。人をプログラムに組み込み、制御できるようにすることで、

何かの処理を実行していく上で、コンピュータにできることは極力コンピュータに処理させ、人は人にしかできないことや、人がやるべきことだけを処理するというのが本来あるべき処理の切り分け方であるが、現状ではその切り分け方はあいまいである。

Babascript によって、人はプログラム上においてコンピュータと同じ処理ノードとして存在し、関数実行によって処理内容を受け取り、実行して値を返す存在になれる。これによって、プログラマブルになりつつある世界において人自身もその一部になることが可能となる。

また、Babascript 環境によって実現する応用例を示すことによって、人をプログラマブルにした際のメリットを示した。

今後は、議論で述べた Babascript の問題点などを改善していく。

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] 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.

[4] 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.

[5] 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 Mobile Systems, Applications, and Services*, MobiSys '10, pp. 63–76, New York, NY, USA, 2010. ACM.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.